Adaptive Server® Anywhere
Руководство пользователя по SQL

Последнее изменение: ноябрь 2001
Номер выпуска: 38124-01-0800-01

Последнее изменение: ноябрь 2001. Номер выпуска: 38124-01-0800-01.

Содержание

Об этом Руководстве

О чем эта книга?

В этой книге описывается процесс проектирования и создания баз данных. Рассматриваются такие действия, как импортирование, экспортирование и изменение данных, выполнение запросов и формирование хранимых процедур и триггеров.

Для кого предназначена эта книга?

Это Руководство предназначено для всех пользователей Adaptive Server Anywhere.

Перед началом работы

Это Руководство предполагает наличие у читателей определенных базовых знаний о системах управления базами данных (СУБД) и, в частности, Adaptive Server Anywhere. При отсутствии опыта работы с этими системами перед изучением этого Руководства рекомендуется ознакомиться с документом *"Введение в Adaptive Server Anywhere" (Adaptive Server Anywhere Getting Started)*.

Документация по SQL Anywhere Studio

Этот документ входит в состав документации по семейству систем SQL Anywhere. В данном разделе приведен список других книг в составе этой документации с их кратким описанием, а также рекомендации по работе с ними.

Комплект документов по SQL Anywhere Studio

Комплект документов по SQL Anywhere Studio включает в себя следующие книги:

- **Введение в SQL Anywhere Studio (Introducing SQL Anywhere Studio).** В этом документе содержится обзор процесса управления базами данных SQL Anywhere Studio, а также технологий синхронизации. В него также включены разделы для обучения с целью познакомить читателей с каждым из компонентов SQL Anywhere Studio.
- **Новое в SQL Anywhere Studio (What's New in SQL Anywhere Studio).** Этот документ предназначен для пользователей, работавших с предыдущими версиями данного ПО. В нем приводится список новых функций в сравнении с предыдущими выпусками данного продукта и описание процедур обновления.
- **Введение в Adaptive Server Anywhere (Adaptive Server Anywhere Getting Started).** Этот документ предназначен для пользователей, ранее не работавших с реляционными базами данных или не знакомых с Adaptive Server Anywhere. Он содержит краткое руководство, позволяющее немедленно начать использование СУБД Adaptive Server Anywhere, а также вводный материал по проектированию и разработке баз данных и работе с ними.
- **Руководство по администрированию баз данных Adaptive Server Anywhere (Adaptive Server Anywhere Database Administration Guide).** В этом документе представлен обширный набор сведений о работе с базами данных, управлении ими и их конфигурированию.
- **Руководство пользователя SQL Adaptive Server Anywhere (Adaptive Server Anywhere SQL User's Guide).** В этом документе описывается процесс проектирования и создания баз данных, а также такие действия, как импортирование, экспортирование и изменение данных, выполнение запросов и формирование хранимых процедур и триггеров.
- **Справочник по SQL для Adaptive Server Anywhere (Adaptive Server Anywhere SQL Reference Manual).** В этом документе представлено полное справочное руководство по языку SQL, используемому системой Adaptive Server Anywhere. В нем также

содержится описание системных таблиц и процедур Adaptive Server Anywhere.

- **Руководство по программированию Adaptive Server Anywhere (Adaptive Server Anywhere Programming Guide).** Этот документ содержит руководство по разработке и развертыванию приложений баз данных с использованием языков программирования C, C++ и Java. Пользователи таких средств разработки, как Visual Basic и PowerBuilder, могут использовать интерфейсы программирования, предоставляемые этими средствами.
- **Сообщения об ошибках Adaptive Server Anywhere (Adaptive Server Anywhere Error Messages).** В этом документе представлен полный перечень сообщений об ошибках системы Adaptive Server Anywhere вместе с информацией по диагностике.
- **Защита C2 в Adaptive Server Anywhere (Adaptive Server Anywhere C2 Security Supplement).** Правительство США присвоило системе Adaptive Server Anywhere 7.0 определенный уровень защиты C2 по классификации TCSEC (Trusted Computer System Evaluation Criteria, Критерии анализа надежности компьютерных систем). Этот документ может представлять интерес для тех, кто хочет использовать данную версию Adaptive Server Anywhere в стиле приложений среды уровня защиты C2. В этом документе *не* содержится описание функций безопасности, встроенных в продукт после сертификации.
- **Руководство пользователя по системе синхронизации MobiLink (MobiLink Synchronization User's Guide).** В этом документе описываются особенности системы синхронизации данных MobiLink для мобильных вычислений, реализующей совместную работу с данными отдельной БД Oracle, Sybase, Microsoft или IBM и множества БД Adaptive Server Anywhere или UltraLite.
- **Руководство пользователя по SQL Remote (SQL Remote User's Guide).** В этом документе описываются особенности системы репликации данных SQL Remote для мобильных вычислений, реализующей совместную работу с данными отдельной БД Adaptive Server Anywhere или Adaptive Server Enterprise и множества БД Adaptive Server Anywhere через непрямую систему связи, например, электронную почту или службы передачи файлов.
- **Руководство пользователя по UltraLite (UltraLite User's Guide).** В этом документе содержится описание процесса разработки приложений баз данных для небольших устройств, например, карманных органайзеров, с использованием технологии развертывания UltraLite для баз данных Adaptive Server Anywhere.
- **Руководство пользователя по UltraLite для PenRight! MobileBuilder (UltraLite User's Guide for PenRight! MobileBuilder).** Этот документ предназначен для пользователей средства разработки PenRight! MobileBuilder. В нем описывается, как

использовать технологию UltraLite в среде программирования MobileBuilder.

- **Контекстно-зависимая справка.** Этот документ доступен только в режиме online. Он содержит контекстно-зависимую справку для приложений Sybase Central, Interactive SQL и других графических средств.

Кроме этого комплекта документов, приложения SQL Modeler и InfoMaker имеют собственную online-документацию.

Форматы документации

Документация по SQL Anywhere Studio доступна в следующих форматах:

- **Online-книги.** В виде online-книг предоставляется полная документация по SQL Anywhere Studio, включая документы, имеющиеся в печатном виде, а также контекстно-зависимая справка по средствам SQL Anywhere. Содержание online-книг обновляется с выходом каждой новой версии продукта; они являются самым полным и актуальным источником документации на любой момент.

  Для обращения к online-книгам в операционных системах Windows выберите Start (Пуск)➤Programs (Программы)➤Sybase SQL Anywhere 8 ➤ Online Books. Для выполнения навигации по online-книгам можно использовать оглавление HTML-справки, индекс, панель поиска в левой области окна, а также ссылки и меню в правой области окна.

  Для обращения к online-книгам в операционных системах UNIX введите в командной строке и выполните следующую команду:

  ```
  dbbooks
  ```

- **Книги для печати.** Документация по SQL Anywhere представлена также в виде набора файлов формата PDF, которые можно просмотреть в Adobe Acrobat Reader.

  PDF-файлы можно найти на CD-ROM в каталоге *pdf_docs*. Инсталлировать их можно в процессе работы программы установки, выбрав соответствующие пункты.

- **Печатные книги.** В комплект поставки SQL Anywhere Studio включены следующие документы:

  - *Введение в SQL Anywhere Studio (Introducing SQL Anywhere Studio)*;
  - *Введение в Adaptive Server Anywhere (Adaptive Server Anywhere Getting Started)*;
  - *Краткий справочник по SQL Anywhere Studio (SQL Anywhere Studio Quick Reference)*. Перечисленные документы доступны только в печатной форме.

  Полный набор документов доступен в виде комплекта документов по SQL Anywhere в отделе сбыта Sybase или в online-магазине Sybase по адресу http://e-shop.sybase.com/cgi-bin/eshop.storefront/.

Условные обозначения

В данном разделе перечислены символьные и графические условные обозначения, используемые в этой документации.

Условные обозначения синтаксиса

В описаниях синтаксиса SQL используются следующие условные обозначения:

- **Ключевые слова.** Все ключевые слова SQL показаны так же, как слова ALTER TABLE в следующем примере:

  ALTER TABLE [*владедец*.]имя-*таблицы*

- **Метки-заполнители.** Элементы, которые должны быть заменены соответствующими идентификаторами или выражениями, показаны так же, как слова *владелец* и *имя-таблицы* в следующем примере.

  ALTER TABLE [*владелец*.]*имя-таблицы*

- **Повторяющиеся элементы.** Список повторяющихся элементов показан посредством указания элемента этого списка с многоточием после него, так же, как слова *ограничение-столбца* в следующем примере:

  ADD *определение-столбца* [*ограничение-столбца*, ...]

  Конструкция может содержать один и более элементов списка. Если элементов несколько, они должны быть разделены запятыми.

- **Необязательные элементы.** Необязательные элементы оператора заключены в квадратные скобки.

  RELEASE SAVEPOINT [*имя-точки-сохранения*]

  Здесь квадратные скобки означают, что указывать *имя-точки-сохранения* не обязательно. При написании кода квадратные скобки вводить не следует.

- **Параметры.** Если из списка можно выбрать только один элемент (или не выбрать вообще), то эти элементы отделяются вертикальной чертой, а сам список заключается в квадратные скобки.

  [**ASC** | **DESC**]

  В этом примере можно выбрать элемент ASC, элемент DESC или не выбрать ни один из них. При написании кода квадратные скобки вводить не следует.

- **Взаимоисключающие варианты.** Если можно выбрать только один элемент из списка (и выбор должен быть сделан), то такие взаимоисключающие варианты заключены в фигурные скобки.

  [**QUOTES** { **ON** | **OFF** }]

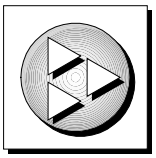
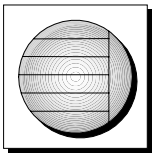
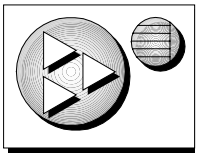
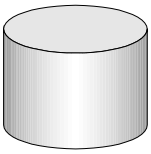
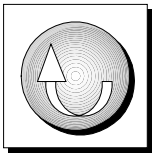
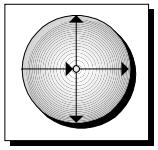
  Если выбран параметр QUOTES, необходимо указать один из вариантов ON или OFF. При написании кода квадратные и фигурные скобки вводить не следует.

- **Один или более параметров.** При выборе более одного параметра следует отделить их друг от друга запятыми.

  {**CONNECT, DBA, RESOURCE** }

Графические значки

В этой документе используются следующие значки:

Значок	Значение
	Клиентское приложение.
	Сервер базы данных, например, Sybase Adaptive Server Anywhere или Adaptive Server Enterprise.
	Приложение и сервер базы данных UltraLite. В системе UltraLite сервер базы данных и приложение являются частью одного и того же процесса.
	База данных. В некоторых сложных диаграммах значок может использоваться для обозначения как самой базы данных, так и сервера базы данных, под управлением которого она находится.
	Микропрограммные средства репликации или синхронизации. Микропрограммные средства являются вспомогательными средствами для обеспечения совместного использования данных различными БД. Это, например, MobiLink Synchronization Server, SQL Remote Message Agent и Replication Agent (Log Transfer Manager), используемые совместно с Replication Server.
	Сервер Sybase Replication Server.
	Интерфейс программирования.

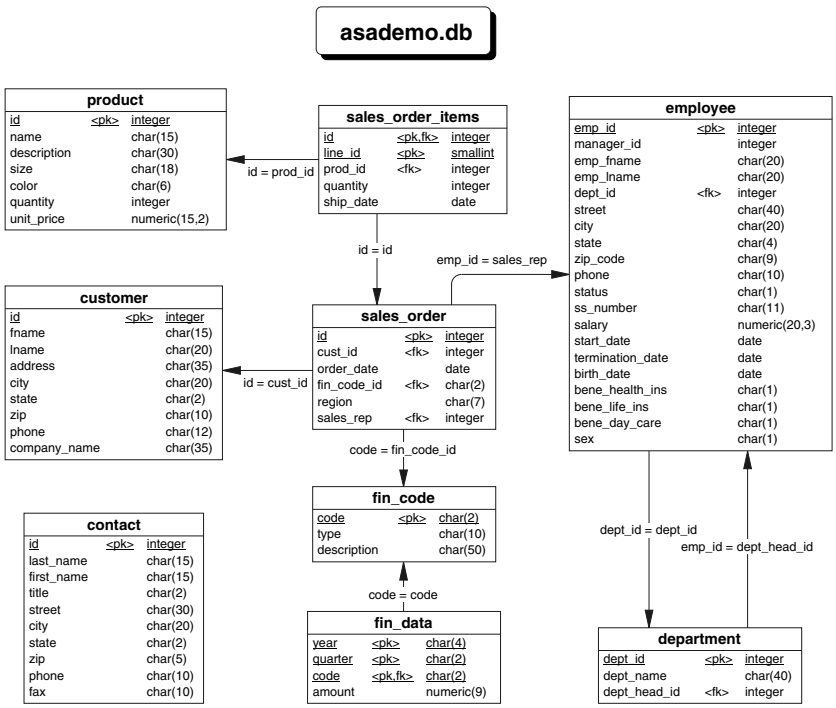
Демонстрационная база данных Adaptive Server Anywhere

Многие из примеров в данном документе относятся к демонстрационной базе данных Adaptive Server Anywhere.

Демонстрационная база данных содержится в файле *asademo.db*, находящемся в папке, где установлена система SQL Anywhere.

Демонстрационная база данных представляет собой базу данных небольшой компании. Она содержит внутреннюю информацию о компании (служащие, отделы, финансовые данные), а также информацию о продукции (продукты), сбыте (заказы на покупку, клиенты, контакты) и финансовую информацию (fin_code, fin_data). Вся информация в базе данных является вымышленной.

На следующем рисунке приведены таблицы в демонстрационной базе данных и их взаимосвязи.



Примечания:

address - адрес
amount - сумма
bene_day_care - номер медицинского полиса
bene_health_ins - номер сертификата о страховании здоровья
bene_life_ins - номер сертификата о страховании жизни
birth_date - дата рождения
city - город
city - город
code - код
color - цвет
company_name - название организации
contact - контакты
cust_id - код клиента
customer - клиенты
departament - отделы
dept_head_id - код начальника отдела
dept_id - код отдела
dept_id - код отдела
dept_name - название отдела
description - описание
description - описание
emp_fname - имя служащего
emp_id - код служащего
emp_lname - фамилия служащего
employee - служащие
fax - номер факса
fin_code_id - код счета
fin_data - финансовые данные
fname - имя
id - код
line id - код строки товара
line id - код строки товара
lname - фамилия
manager_id - код менеджера
name - наименование
order_date - дата заказа
phone - телефон
prod_id - код товара
product - товары
quantity - количество
quantity - количество
quarter - квартал
region - регион
salary - заработная плата
sales_order - заказы на покупку
sales_order_items - заказы на покупку
sales_rep - торговый представитель
sex - пол
ship_date - дата отгрузки
size - размер
ss_number - код социального обеспечения
start_date - дата принятия на работу
state - страна
status - семейное положение
street - улица
termination_date - дата увольнения
title - заголовок
type - тип
unit_price - цена за экземпляр
year - год
zip - почтовый индекс
zip_code - почтовый индекс

Получение дополнительной информации и обратная связь

Нам было бы интересно узнать мнение читателей об этой документации, а также получить предложения и отзывы.

Отправить свой отзыв на эту документацию и программное обеспечение можно с помощью групп новостей, специально предназначенных для обсуждения технологий SQL Anywhere. Эти группы новостей можно найти на сервере групп новостей *forums.sybase.com*.

Имеются следующие группы новостей:

- sybase.public.sqlanywhere.general.
- sybase.public.sqlanywhere.linux.
- sybase.public.sqlanywhere.mobilink.
- sybase.public.sqlanywhere.product_futures_discussion.
- sybase.public.sqlanywhere.replication.
- sybase.public.sqlanywhere.ultralite.

Заявление по группам новостей

Компания iAnywhere Solutions не обязана предоставлять какие-либо решения, информацию или идеи относительно своих групп новостей, а также не обязана обеспечивать наличие обслуживания, кроме системного оператора, контролирующего процесс предоставления обслуживания и обеспечивающего его функционирование и доступность.

Технические консультанты и другой персонал компании iAnywhere Solutions занимаются поддержкой групп новостей при наличии свободного времени. Они предлагают свою помощь на добровольной основе и не занимаются постоянной деятельностью по решению проблем и предоставлению информации. Их возможности по оказанию помощи зависят от рабочей нагрузки.

ЧАСТЬ 1

Принципы использования реляционных баз данных

В этом разделе описываются ключевые принципы и стратегии эффективного использования Adaptive Server Anywhere.

ГЛАВА 1

Проектирование базы данных

Об этой главе

В этой главе вводятся основные принципы проектирования реляционных баз данных и даются пошаговые рекомендации по созданию собственных баз данных. В этой главе используется методика, основанная на целесообразном подходе и известная как концептуальное моделирование данных, основными понятиями которой являются объекты и связи между ними.

Содержание

Введение

Проектирование базы данных – достаточно важная задача, несмотря на то, для баз данных малого или среднего размера это несложно. Неудачно спроектированная база данных может оказаться неэффективной и, возможно, ненадежной. Поскольку клиентские приложения ориентированы на взаимодействие с соответствующими компонентами базы данных и зависят от ее структуры, последующее ее изменение в случае неудачного проектирования может быть затруднительным.

Для получения дополнительной информации можно обратиться к какой-либо книге начального уровня, например, *"Основы баз данных" (A Database Primer),* автор C. J. Date. Интересующимся теорией баз данных рекомендуется превосходное издание – *"Введение в системы баз данных" (An Introduction to Database Systems)* того же автора.

Java-классы в проектировании баз данных

Кроме основных понятий реляционных баз данных, рассмотренных ниже в этой главе, также следует указать возможность использования Java-классов среди других доступных типов данных. В этой главе вопросы проектирования базы данных с применением Java-классов не рассматриваются.

Для получения дополнительной информации о проектировании баз данных с использованием Java-классов как типа данных см. раздел "Проектирование базы данных с использованием Java" (Java database design) на стр. 121 в документе *"Руководство по программированию ASA" (ASA Programming Guide)*.

Основные принципы проектирования базы данных

При проектировании базы данных необходимо определить, сведения о чем она будет содержать, и какого рода будут эти сведения по каждому из включенных в базу данных предмету. Также при этом нужно определить, как эти сведения будут связаны между собой. На общепринятом языке проектирования баз данных то, что создается на этом этапе – это **концептуальная модель базы данных**.

Объекты и связи

Различимые предметы или понятия, сведения о которых будут содержаться в базе данных, называются **объектам**. Отношения между ними называют **связями**. Используя язык описания баз данных, можно представить себе объекты как существительные, а связи – как глаголы.

Концептуальные модели весьма полезны, поскольку они устанавливают четкое различие между объектами и связями. Эти модели не включают подробности реализации проекта базы данных для какой-либо конкретной системы управления базой данных. Они позволяют сосредоточиться на фундаментальной структуре базы данных. Следовательно, они также формируют общий язык для обсуждения вопросов проектирования баз данных.

Диаграммы "объект - связь"

Основной компонент концептуальной модели базы данных - диаграмма, демонстрирующая объекты и связи. Эту диаграмму обычно называют **диаграммой "объект - связь"**. Соответственно, при рассмотрении задач создания концептуальной модели базы данных часто используют понятие моделирования объектов и связей.

Концептуальное проектирование базы данных – это способ разработки базы данных "по нисходящей". В настоящее время существуют мощные инструментальные средства, такие как Sybase PowerDesigner, позволяющие использовать этот метод или другие подходы. Эта глава является вводной, но в ней содержится достаточно информации по проектированию несложных баз данных.

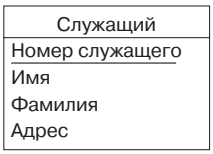
Объекты

С точки зрения языка баз данных, объект – это эквивалент имени существительного. Предметы и понятия, отличимые друг от друга, например, служащие, заказы, отделы, продукты – это все примеры объектов. В базе данных каждый объект представлен таблицей. Объекты, которые встраиваются разработчиком в базу данных, создаются в результате операций, для которых будет использоваться база данных, как, например, отслеживание объемов продаж или обработка информации о служащих.

Атрибуты

Каждый объект имеет набор **атрибутов**. Атрибуты – это определенные характеристики объектов базы данных. Например, для объекта "служащий" целесообразно хранить номер служащего, имя и фамилию, адрес и другую конкретную информацию, которая относится к этому конкретному служащему. Атрибуты также называются свойствами.

Объект изображается как прямоугольное поле. Внутри него перечисляются атрибуты, связанные с этим объектом.



Идентификатор – это один или более атрибутов, от которых зависят все остальные атрибуты. Он однозначно идентифицирует элемент в объекте. Подчеркните названия атрибутов, которые должны стать частью идентификатора.

В примере выше, атрибут "Номер служащего" в объекте "Служащий" однозначно идентифицирует данного служащего. Все остальные атрибуты хранят информацию, которая относится именно к этому служащему. Например, по номеру служащего можно однозначно установить его имя и адрес. Два служащих могут иметь одинаковые имя или адрес, но их идентификаторы не могут совпадать. Атрибут "Номер служащего" подчеркнут, что означает, что он является идентификатором.

Для каждого объекта должен быть создан свой идентификатор. Как будет показано ниже, эти идентификаторы выполняют функцию первичного ключа в таблицах базы данных. Значения первичных ключей должны быть уникальны и не могут быть пустыми (нуль) или неопределенными. Они однозначно идентифицируют каждую запись в таблице и повышают производительность сервера базы данных.

Связи

С точки зрения языка баз данных, связи между объектами – это эквивалент глагола; например, служащий является членом отдела, или же офис расположен в городе. Связи в базе данных могут являться связями между таблицами по внешнему ключу, или непосредственно отдельными таблицами. В этой главе приведены примеры для каждого варианта.

Связи в базе данных – это кодированное представление правил или действий, управляющих данными в объектах. Если в каждом отделе имеется по одному начальнику отдела, можно создать взаимнооднозначные связи между отделами и служащими, чтобы идентифицировать таким образом начальника отдела.

После того, как связи внесены в структуру базы данных, никакой возможности для исключений уже нет; например, невозможно ввести второго начальника отдела. Дублирование элемента отдела повлекло бы за собой дублирование кода отдела, который является идентификатором. Дублирование идентификаторов не допускается.

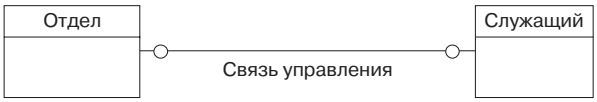
Совет

Строгая структура базы данных полезна, поскольку позволяет устранять возможные несоответствия, такие как появление отдела с двумя управляющими. С другой стороны, разработчику следует создать достаточно гибкую структуру, чтобы имелась возможность произвести некоторое расширение для непредвиденных ранее видов использования. Расширить правильно спроектированную базу данных, как правило, не очень сложно, однако при изменении существующей структуры таблицы может устареть вся база данных вместе с клиентскими приложениями.

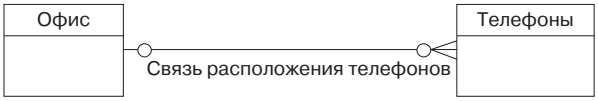
Количество объектов в связи

Существует три вида связей между таблицами. Они соответствуют **количеству объектов** (числу элементов), вовлеченных в связи.

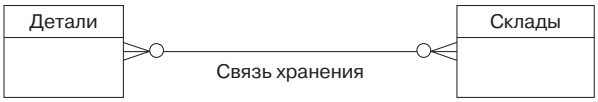
- **Связи "один к одному".** Связь изображается линией между двумя объектами. На линии могут присутствовать другие знаки, такие как два маленьких кружка на рисунке. Назначение этих меток объясняется в последующих разделах. На диаграмме ниже показана ситуация, когда один служащий управляет одним отделом.



- **Связи "один ко многим".** Такая связь показывает, что один элемент, содержащийся в объекте 1, может быть связан со множеством объектов в объекте 2, что обозначено несколькими линиями, указывающими на присоединение к объекту 2. На диаграмме ниже показана ситуация, когда в одном офисе находится много телефонов.



- **Связи "многие ко многим".** В этом случае отображаются несколько линий, обозначающих различные связи обоих объектов. Например, на одном складе может храниться много различных деталей, а деталь одного конкретного типа может храниться на многих складах.



Роли

Каждую связь можно описать с помощью двух **ролей**. Роли можно рассматривать как глаголы или фразы, которые описывают связь с точки зрения каждого ее участника. Например, связи между служащими и отделами могут быть описаны с помощью следующих двух ролей.

1 Служащий - *член* отдела.

2 Отдел *включает* служащего.

Роли очень важны, поскольку они предоставляют удобное и эффективное средство проверки работы.

Совет

Как в случае чтения слева направо, так и справа налево, следующее правило прочтения приводимых диаграмм облегчает их восприятие. Порядок чтения следующий:

1 имя первого объекта,

2 роль сразу после *первого объекта*,

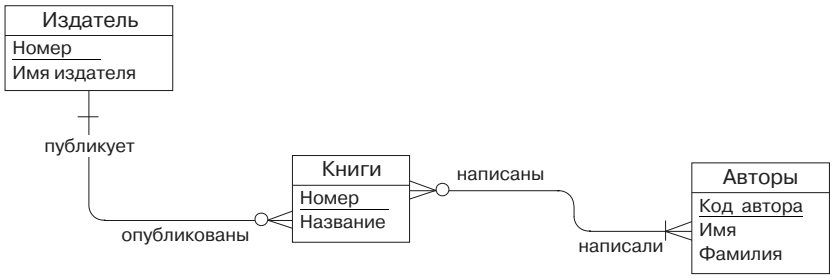
3 количество элементов в связи до *второго объекта*, и

4 имя второго объекта.

Необходимые элементы

Небольшие кружки около каждого конца линии, обозначающей связь, имеют важное значение. Кружок означает, что элемент может существовать в одном объекте без соответствующего элемента в другом объекте.

Если вместо кружка линия перечеркнута, то данный объект должен содержать *по крайней мере один* элемент для каждого элемента в другом объекте. Следующий пример поясняет эти утверждения.



Эта диаграмма соответствует следующим четырем утверждениям.

1 Издатель опубликовывает *нуль или более* книг.

2 Книга опубликована *одним конкретным* издателем.

3 Книга написана *одним или более* авторами.

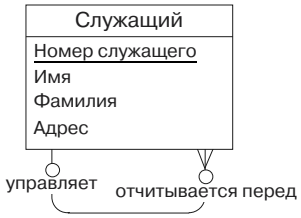
4 Автор пишет *нуль или более* книг.

Совет

Удобно представлять небольшой кружок как цифру 0, а перечеркивание - как цифру один. Кружок означает *по крайней мере нуль.* Перечеркивание означает *по крайней мере один.*

Рефлексивные связи

Иногда связи могут существовать между элементами в одном объекте. В этом случае такие связи называют **рефлексивными**. Оба конца связи присоединяются к одному объекту.



Эта диаграмма соответствует следующим двум утверждениям.

1 Служащий отчитывается перед максимум одним другим служащим.

2 Количество служащих, которыми управляет данных служащий, - нуль или более.

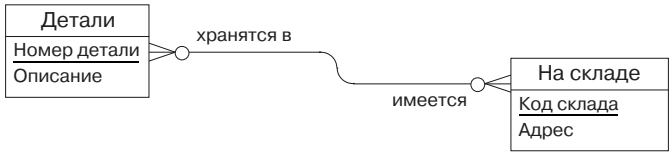
Обратите внимание на существенное обстоятельство: эта связь является необязательной в обоих направлениях. Некоторые служащие не являются управляющими. Аналогично, по крайней мере один служащий должен возглавлять организацию и не отчитываться ни перед кем.

Также было бы разумно обозначить, что служащий не может быть управляющим по отношению к самому себе. Это ограничение является одним из видов *бизнес-правил*. Бизнес-правила рассматриваются ниже в разделе "Процесс проектирования" на стр. 11.

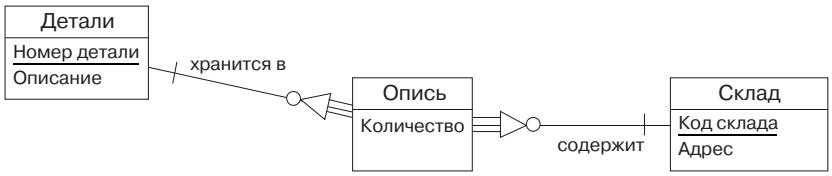
Преобразование связей "многие ко многим" в объекты

При наличии атрибутов, относящихся к *связи*, а не к объекту, можно преобразовать связь в объект. Такая ситуация иногда возникает со связями "многие ко многим", когда имеются атрибуты, специфичные для отдельной связи, вследствие чего их нельзя явно добавить к какому-либо объекту.

Предположим, что определенный набор деталей имеется на нескольких различных складах. Нарисуйте следующую диаграмму.



Требуется создать опись с информацией о количестве каждой детали, хранящейся на каждом конкретном складе. Этот атрибут может быть ассоциирован только со связью. Каждый параметр количества зависит и от деталей, и от рассматриваемого склада. Чтобы представить эту ситуацию, можно перерисовать диаграмму следующим образом:



Обратите внимание на следующие особенности преобразования:

1 Две новые связи соединяют объект связи с каждым из двух исходных объектов. Они наследуют их имена из двух ролей исходных связей: *хранится в* и *содержит* соответственно.

2 Каждому элементу в объекте "Опись" требуется один обязательный элемент в объекте "Детали" и один обязательный элемент в объекте "Склад". Наличие этих связей обязательно, поскольку связь хранения имеет смысл только в том случае, если она ассоциирована с одним конкретным типом детали и одним конкретным складом.

3 Новый объект зависит и от объекта "Детали", и от объекта "Склад". Это означает, что новый объект идентифицируется с использованием идентификаторов обоих этих объектов. В новой диаграмме ниже, один идентификатор из объекта "Детали" и один идентификатор из объекта "Склад" однозначно определяет элемент в объекте "Опись". Треугольники между кружками и несколькими линиями, соединяющими две новых связи и новый объект "Опись", обозначают зависимости.

К объекту "Опись" не требуется добавлять ни "Номер детали", ни "Код склада". Каждый элемент в объекте "Опись" действительно зависит и от конкретной детали и от конкретного склада, и треугольники обозначают эту зависимость более четко.

Процесс проектирования

Процесс проектирования состоит из пяти основных шагов.

Для получения дополнительной информации об реализации проекта базы данных см. раздел "Работы с объектами базы данных" на стр. 27.

Шаг 1: Определение объектов и связей

❖ Определение объектов в проекте базы данных и их связей друг с другом

1 **Определение основных операций** Определите основные операции, для которых будет использоваться создаваемая база данных. Например, можно использовать ее для отслеживания информации о служащих.

2 **Определение объектов** Определите для обозначенных выше операций виды предметов, информацию о которых требуется обрабатывать. Эти предметы станут объектами. Например, это наем *служащих*, распределение их по *отделам* и определение *квалификации*.

3 **Определение связей** Просмотрите операции и определите, какими будут связи между объектами. Например, должна быть связь между деталями и складами. Определите две роли для описания каждой связи.

4 **Разбивка операций** Ранее были определены основные операции. Теперь рассмотрите эти операции более тщательно на предмет проверки того, не могут ли некоторые из них быть преобразованы в действия более низкого уровня. Например, основная операция "*обработка информации о служащих"* может быть разбита на:

- добавление информации о новых служащих;
- изменение информации об имеющемся служащем;
- удаление информации об уволенных служащих.

5 **Определение бизнес-правил** Просмотрите описание деятельности компании и определите, каким бизнес-правилам необходимо следовать. Например, одно бизнес-правило может состоять в том, что отдел имеет одного и только одного начальника отдела. Эти правила будут встроены в структуру базы данных.

Пример объектов и связей

Пример

Корпорация ACME - небольшая компания, имеющая офисы в пяти различных местах.

В настоящий момент в компании ACME работают 75 служащих. Проводится подготовка к расширению компании. Созданы девять отделов, каждый со своим начальником отдела.

С целью содействия в поиске новых служащих, отдел подбора персонала определил 68 профессий, которые, по его мнению, понадобятся компании в будущем штате служащих. При принятии служащего на работу определяется его профессиональный уровень для каждой профессии.

Определение основных операций

Вот некоторые из основных операций для корпорации ACME:

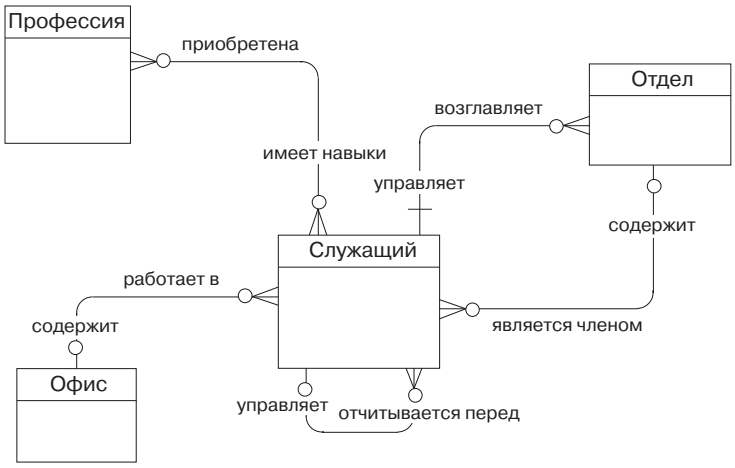
- наем служащих;
- увольнение служащих;
- обработка персональной информации о служащих;
- обработка информации о профессиях, требуемых компании;
- обработка информации о том, какие служащие имеют какие профессии;
- обработка информации об отделах;
- обработка информации об офисах.

Определение объектов и связей

Определите объекты (предметы) и связи (роли), соединяющие их. Постройте диаграмму на основе описания и основных операций.

Используйте поля для обозначения объектов и линии для обозначения связей. Используйте две роли для обозначения каждой связи. Следует также обозначить, какие связи являются связями типа "один ко многим", "один к одному" и "многие ко многим", используя соответствующие значки.

Ниже показана приблизительная диаграмма "объект - связь". Она будет уточняться по мере прохождения главы.



Разбивка основных операций

Из указанных выше основных операций выделяются следующие операции более низкого уровня:

- добавление или удаление служащего;
- добавление или удаление офиса;
- построение списка служащих для отдела;
- добавление профессии к списку профессий;
- определение профессий служащего;
- определение квалификации служащего для каждой профессии;
- определение всех служащих, имеющих одну квалификацию для конкретной профессии;
- изменение квалификации служащего.

Эти операции более низкого уровня могут использоваться для определения потребности в новых таблицах или связях.

Определение бизнес-правил

Бизнес-правила часто представляют связи "один ко многим", "один к одному" и "многие ко многим".

Предполагаемые типы подходящих бизнес-правил включают следующее:

- На данный момент имеется пять офисов; при расширении планируется создать еще максимум пять.
- Служащие могут сменить отдел или офис.
- В каждом отделе имеется один начальник отдела.
- В каждом офисе имеется максимум три номера телефона.
- Каждый номер телефона имеет один или более добавочных номеров.
- После найма служащего определяется квалификация для каждого из нескольких профессий.
- Каждый служащий может иметь от трех до двадцати профессий.
- Служащий может относиться или не относиться к определенному офису.

Шаг 2: Определение требуемых данных

❖ **Определение требуемых данных**

1 Определите базовые данные.

2 Перечислите все данные, которые требуется отслеживать.

3 Установите данные для каждого объекта.

4 Перечислите доступные данные для каждого объекта. Данные, которые описывают объект (предмет), отвечают на вопросы "Кто?", "Что?", "Где?", "Когда?" и "Зачем?".

5 Перечислите все данные, требуемые для каждой связи (действия).

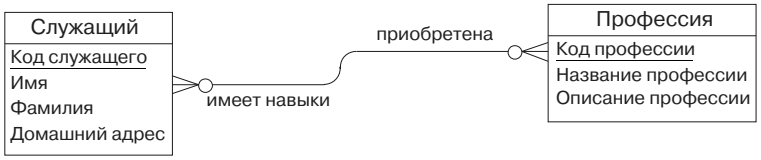
6 Перечислите данные, если таковые имеются, используемые во всех связях.

Определение базовых данных

Базовые данные, которые будут определены, станут именами атрибутов объекта. Например, следующие данные могут применяться к объекту "Служащий", объекту "Профессия" и связи "Специализация".

Служащий	Профессия	Специализация
Код служащего	Код профессии	Квалификация
Имя служащего	Название профессии	Срок приобретения квалификации
Фамилия служащего	Описание профессии	
Отдел служащего		
Офис служащего		
Адрес служащего		

Если создать диаграмму из этих данных, она будет выглядеть примерно так:



Следует заметить, что в эту диаграмму вошли не все перечисленные атрибуты. Отсутствующие элементы относятся к двум категориям:

1 Некоторые содержатся неявно в других связях; например, "Отдел служащего" и "Офис служащего" представлены связями к объектам "Отдел" и "Офис" соответственно.

2 Другие не присутствуют, поскольку они ассоциированы не с каким-либо из этих объектов, а скорее со связями между ними. Вышеприведенная диаграмма неадекватна.

Первая категория элементов будет соответствовать истине после того, как будет нарисована полная диаграмма "объект-связь".

Можно добавить вторую категорию, преобразовав эту связь типа "многие ко многим" в объект.

Новый объект зависит и от объекта "Служащий", и от объекта "Профессия". Он заимствует свои идентификаторы из этих объектов, поскольку зависит от них обоих.

Примечания

- При определении базовых данных необходимо иметь в виду операции, определенные ранее, чтобы видеть, как впоследствии будет осуществляться доступ к данным.

  Например, в некоторых ситуациях, возможно, понадобится перечислить служащих по именам, а в других - по фамилиям. Чтобы выполнить это требование, следует создать атрибуты "Имя" и "Фамилия", а не единственный атрибут, содержащий и имя, и фамилию одновременно. Имея отдельные имена, впоследствии можно создать два индекса, каждый для своей задачи.

- Выберите подходящие не противоречащие друг другу имена. Согласованность данных облегчает обслуживание базы данных и позволяет создавать более легкие для чтения отчеты и окна для вывода информации.

  Например, если предполагается использовать сокращенное имя, такое, как Emp_status для одного атрибута, то не следует использовать полное имя, такое как Employee_ID (Код служащего), для другого атрибута. Вместо этого имена должны быть такими – Emp_status и Emp_ID.

- На этом этапе связь данных с правильным объектом не является критичной. В этом случае можно действовать по интуиции. В следующем разделе описаны тесты, позволяющие проверить принятые решения.

Шаг 3: Нормализация данных

Нормализация – это ряд тестов, позволяющих устранить избыточность в данных и удостовериться, что данные связаны с правильным объектом или связью. Имеется пять тестов. В этом разделе представлены первые три из них. Эти три теста – самые важные и поэтому наиболее часто используемые.

Для чего нужна нормализация?

Цель нормализации состоит в том, чтобы удалить избыточные данные и улучшить совместимость. Например, если адрес клиента хранится во множестве мест, довольно трудно изменить все копии правильно при их передвижении.

Для получения дополнительной информации о тестах нормализации см. документацию по проектированию баз данных.

Уровень нормализации

Имеется несколько тестов на нормализацию данных. После прохождения данными первого теста они считаются имеющими первый уровень нормализации. После прохождения второго теста это второй уровень нормализации, после прохождения третьего – третий уровень нормализации.

❖ **Нормализация данных в базе данных**

1 Перечисление необходимых данных
   - Определите, по крайней мере, по одному ключу для каждого объекта. Каждый объект должен иметь идентификатор.
   - Определите ключи для связей. Ключи для связи являются ключами двух объектов, которые соединены этой связью.
   - Проверьте вычисляемые данные в списке базовых данных. В реляционной базе данных вычисляемые данные обычно не сохраняются.

2 Приведение данных к первому уровню нормализации
   - Если атрибут может иметь несколько различных значений для одного и того же элемента, удалите эти повторяющиеся значения.
   - Создайте один или более объектов или связей с удаляемыми данными.

3 Приведение данных ко второму уровню нормализации
   - Определите объекты и связи с более чем одним ключом.
   - Удалите данные, зависящие только от одной части ключа.
   - Создайте один или более объектов и связей с удаляемыми данными.

4 Приведение данных к третьему уровню нормализации
   - Удалите данные, зависящие от других данных в объекте или связи, но не от ключа.
   - Создайте один или более объектов и связей с удаляемыми данными.

Данные и идентификаторы

Перед началом нормализации (проверки построения базы) просто перечислите данные и определите уникальный идентификатор каждой таблицы. Идентификатор может быть составлен из одной части данных (атрибут) или нескольких (составной идентификатор).

Идентификатор – это набор атрибутов, который однозначно идентифицирует каждую запись в объекте. Например, идентификатором для объекта "Служащий" является атрибут "Код служащего". Идентификатор для связи "Работает в" состоит из атрибутов "Код офиса" и "Код служащего".

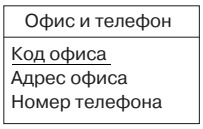
Можно создать идентификатор для каждой связи в базе данных, беря идентификаторы из каждого из объектов, которые она соединяет. В следующей таблице атрибуты, помеченные звездочкой – идентификаторы для объекта или связи.

Объект или *связь*	**Атрибуты**
Офис	*Код офиса Адрес офиса Номер телефона
Работает в	*Код офиса *Код служащего
Отдел	*Код отдела Название отдела
Начальники	*Код отдела *Код служащего
Член	*Код отдела *Код служащего
Профессия	*Код профессии Название профессии Описание профессии
Область специализации	*Код профессии *Код служащего Квалификация Срок приобретения
Служащий	*Код служащего Фамилия Имя Социальное обеспечение (номер полиса) Адрес Номер телефона Дата рождения

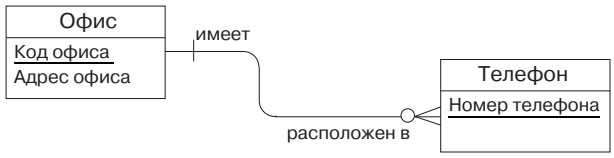
Приведение данных к первому уровню нормализации

- Для проверки данных на соответствие первому уровню нормализации произведите проверку на наличие атрибутов, которые могут иметь повторяющиеся значения.
- Удалите атрибуты в случае, когда несколько значений могут соответствовать одному элементу. Перенесите эти повторяющиеся атрибуты в новый объект.

В объекте ниже, атрибут "Номер телефона" может повторяться – в офисе может быть более одного номера телефона.



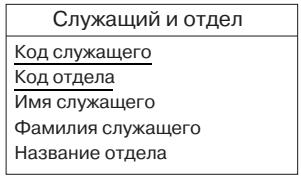
Удалите повторяющийся атрибут и создайте новый объект с названием "Телефон". Установите связь между объектами "Офис" и "Телефон".



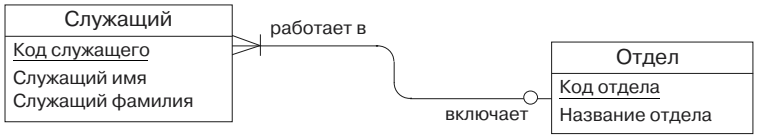
Приведение данных ко второму уровню нормализации

- Удалите данные, не зависящие от всего ключа.
- Обращайте внимание только на те объекты и связи, идентификатор которых состоит из более чем одного атрибута. Для проверки данных на соответствие второму уровню нормализации удалите все те данные, которые зависят не от всего идентификатора. Каждый атрибут должен зависеть от всех атрибутов, составляющих идентификатор.

В этом примере идентификатор объектов "Служащий" и "Отдел" состоит из двух атрибутов. Некоторые данные не зависят от обоих атрибутов идентификатора; например, название отдела зависит только от одного из этих атрибутов – "Код отдела", а имя служащего зависит только от атрибута "Код служащего".



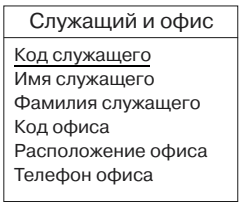
Переместите идентификатор "Код отдела", от которого не зависят другие данные служащего, в его собственный объект с названием "Отдел". Также следует переместить все зависящие от него атрибуты. Создайте связь между объектами "Служащий" и "Отдел".



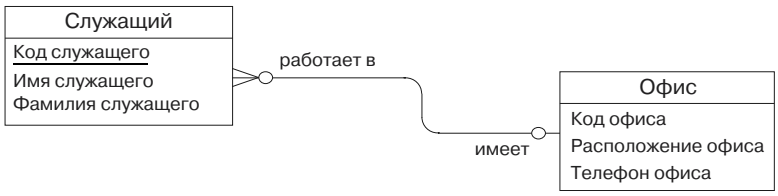
Приведение данных к третьему уровню нормализации

- Удалите данные, которые не зависят непосредственно от ключа.
- Для проверки данных на соответствие третьему уровню нормализации удалите все атрибуты, которые зависят от других атрибутов, а не непосредственно от идентификатора.

В этом примере, объекты "Служащий" и "Офис" содержат некоторые атрибуты, зависящие от идентификатора "Код служащего". Однако такие атрибуты, как "Расположение офиса" и "Телефон офиса" зависят от другого атрибута "Код офиса". Они не зависят непосредственно от идентификатора "Код служащего".



Удалите "Код офиса" и те атрибуты, которые зависят от него. Создайте другой объект с названием "Офис". Затем создайте связи, соединяющие объекты "Служащий" и "Офис".



Шаг 4: Оптимизация связей

По окончании нормализации процесс проектирования почти завершен. Все, что требуется выполнить, – это сгенерировать **физическую модель данных**, которая соответствует концептуальной модели данных. Этот процесс также известен как "оптимизация связей", так как большая часть задачи состоит в преобразовании связей в концептуальной модели в соответствующие таблицы и связи по внешнему ключу.

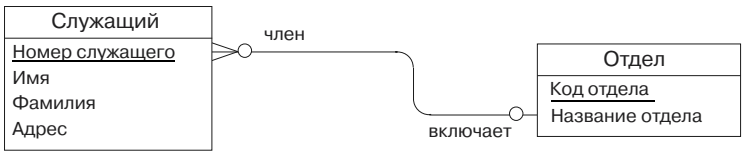
В то время как концептуальная модель в значительной степени независима от подробностей реализации, физическая модель данных жестко привязана к структуре таблицы и опциям, доступным в конкретном приложении базы данных. Это приложение в данном случае – Adaptive Server Anywhere.

Оптимизация связей без данных

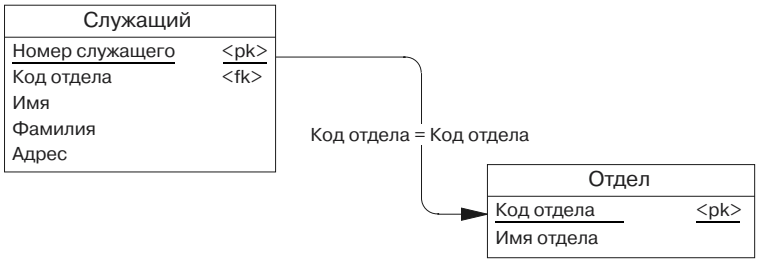
Для реализации связей без данных следует определить внешние ключи. **Внешний ключ** – это столбец или набор столбцов, содержащий значения первичного ключа из другой таблицы. Внешний ключ позволяет обращаться к данным в более чем одной таблице за один раз.

Какое-либо средство разработки баз данных, например, DataArchitect (компонент Sybase PowerDesigner) может сгенерировать для разработчика физическую модель данных. Тем не менее. при самостоятельном выполнении этой операции существуют некоторые основные правила для определения того, где следует разместить ключи.

- **"Один ко многим".** Связь "один ко многим" всегда становится объектом и связью по внешнему ключу.

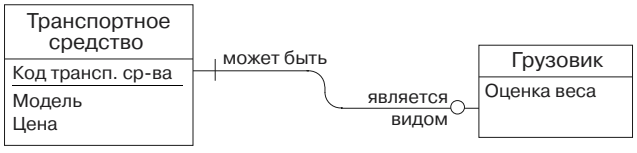


Обратите внимание на то, что объекты становятся таблицами. Идентификаторы в объектах становятся (по крайней мере частично) первичным ключом в таблице. Атрибуты становятся столбцами. В связях "один ко многим" идентификатор в *одном* объекте появится как новой столбец внешнего ключа в *нескольких* таблицах.

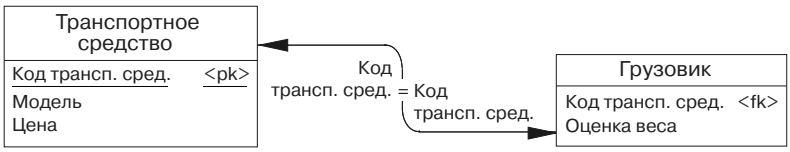


В этом примере *объект* "Служащий" становится *таблицей* "Служащий". Аналогично, объект "Отдел" становится таблицей "Отдел". Внешний ключ с именем "Код отдела" появляется в таблице "Служащий".

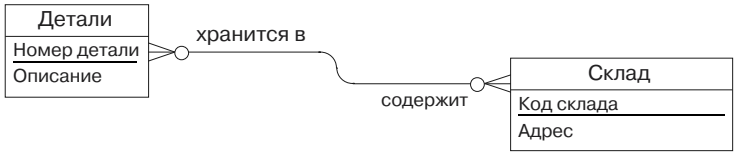
- **"Один к одному".** При связях "один к одному" внешний ключ может войти в любую из таблиц. Если связь является обязательной на одной стороне, но необязательной на другой, она должна перейти на необязательную сторону. В этом примере внешний ключ "Код транспортного средства" следует поместить в таблицу "Грузовик", так как транспортное средство может и не быть грузовиком.



Показанная выше модель "объект-связь" оптимизирует структуру ядра базы данных так, как показано ниже.



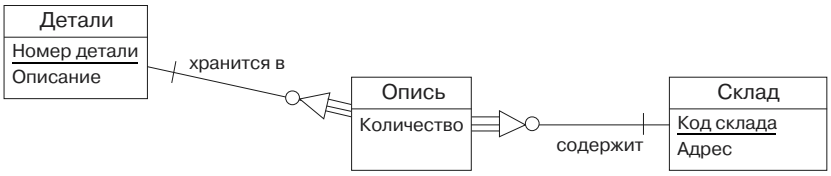
- **"Многие к многим".** Для связей типа "многие к многим" создается новая таблица с двумя внешними ключами. Это преобразование необходимо для обеспечения эффективности базы данных.



Новая таблица "Место хранения" связана с таблицами "Детали" и "Склад".

Оптимизация связей с данными

Некоторые из связей могут содержать данные. Такая ситуация часто возникает в связях "многие к многим".



В таких случаях каждый объект привязывается к таблице. Каждая роль становится внешним ключом, указывающим на другую таблицу.

Объект "Опись" заимствует свои идентификаторы из таблиц "Детали" и "Склад", так как он зависит от обеих этих таблиц. После оптимизации эти заимствованные идентификаторы формируют первичный ключ таблицы "Опись".

Совет

Концептуальная модель данных упрощает процесс проектирования, поскольку не включает множества подробностей. Например, каждая связь "многие к многим " всегда генерирует дополнительную таблицу и две ссылки по внешнему ключу. В рамках концептуальной модели данных, как правило, можно обозначать всю эту структуру единственным соединением.

Шаг 5: Проверка проекта

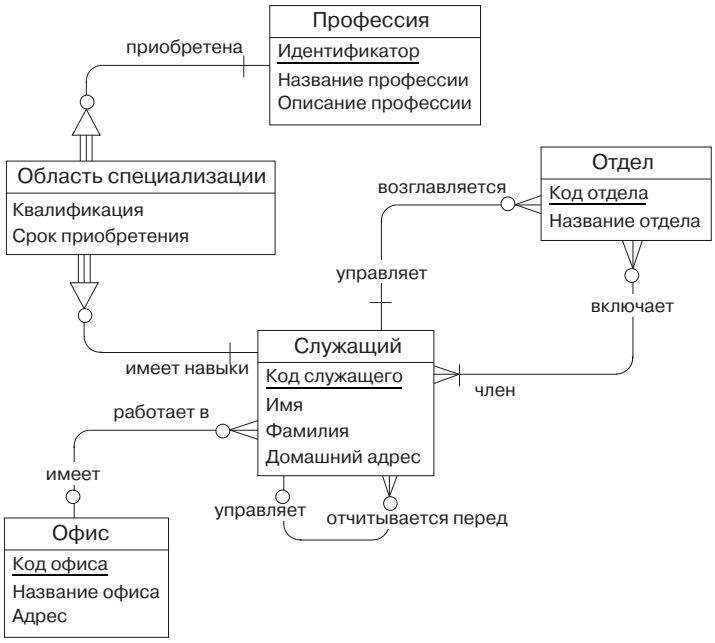
Перед реализацией проекта следует удостовериться, что он соответствует всем требованиям пользователя. Проверьте операции, определенные в начале процесса проектирования и удостоверьтесь, что возможно обращение ко всем данным, которые требуются для производимых операций.

- Удается ли найти путь для получения необходимой информации?
- Отвечает ли проект требованиям?
- Все ли требуемые данные доступны?

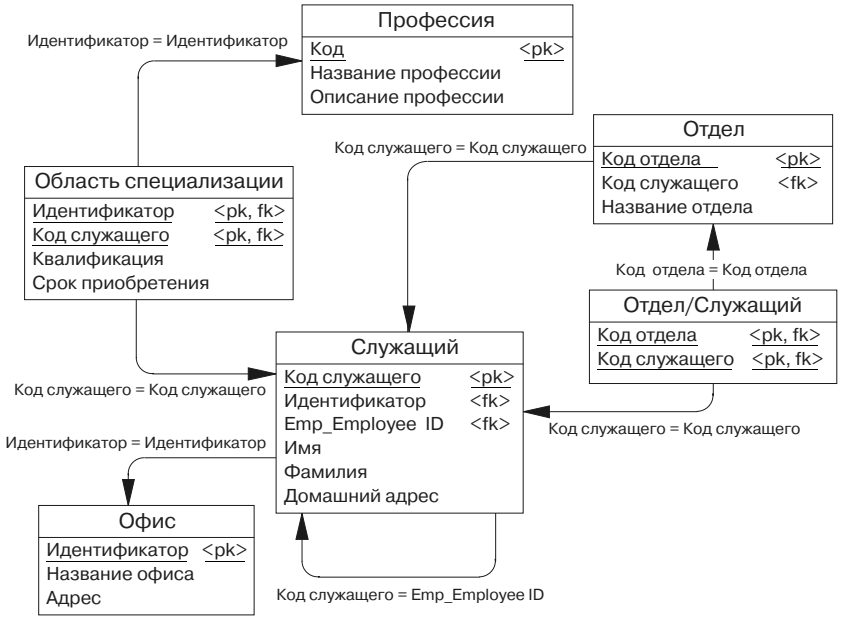
Если ответ на все вышеперечисленные вопросы положительный, то для реализации проекта все готово.

Завершение процесса проектирования

Применение шагов 1–3 к базе данных для небольшой компании дает в результате нижеприведенную диаграмму "объект - связь". Эта база данных теперь соответствует третьему уровню нормализации.



Соответствующая физическая модель данных приведена ниже.



Проектирование свойств таблиц базы данных

Структура базы данных определяет, какие таблицы имеются и какие столбцы содержит каждая таблица. В этом разделе описывается, как указать свойства каждого столбца.

Для каждого столбца необходимо назначить имя столбца, тип данных и размер, разрешены ли пустые значения (NULL), и требуется ли, чтобы значения, разрешенные для столбца, ограничивались в базе данных.

Выбор имен столбцов

Имя столбца может быть произвольным набором букв, цифр или символов. Однако если имя столбца содержит символы, отличные от букв, цифр и знаков подчеркивания, имя не начинается с буквы или совпадает с каким-либо ключевым словом, то следует заключить его в двойные кавычки.

Выбор типов данных для столбцов

Доступные в Adaptive Server Anywhere типы данных включают следующие:

- целочисленные типы данных;
- десятичные типы данных;
- типы данных с плавающей точкой;
- символьные типы данных;
- двоичные типы данных;
- типы данных даты/времени;
- домены (пользовательские типы данных);
- типы данных Java-классов.

Для получения дополнительной информации о типах данных см. раздел "Типы данных SQL" (SQL Data Types) на стр. 51 в документе "*Справочник по SQL для ASA*' *(ASA SQL Reference Manual)*.

Длинный двоичный тип данных (long binary) может использоваться для сохранения такой информации, как изображения (например, растровые рисунки) или документы текстовых редакторов, в базе данных. Эти типы информации обычно называют большими двоичными объектами (BLOBS).

Для получения дополнительной информации о каждом из типов данных см. раздел "Типы данных в SQL" (SQL Data Types) на стр. 51 в документе "*Справочник по SQL для ASA*' *(ASA SQL Reference Manual)*.

NULL и NOT NULL

Если значение столбца обязательно должно быть заполнено, то столбец определяется как NOT NULL. В противном случае столбец может содержать значение NULL, что означает отсутствие какого-либо значения. По умолчанию в SQL значения NULL разрешены, но в то же время, если нет крайней необходимости в разрешении значений NULL, следует явно объявить столбцы как NOT NULL.

Для получения дополнительной информации о значении NULL см. раздел "Значение NULL" (NULL value) на стр. 47 в документе "*Справочник по SQL для ASA*" *(ASA SQL Reference Manual)*. Для получения информации о его использовании в сравнениях см. раздел "Условия поиска" (Search conditions) на стр. 23 в документе "*Справочник по SQL для ASA*" *(ASA SQL Reference Manual)*.

Выбор ограничений

Несмотря на то, что тип данных столбца ограничивает значения, разрешенные в этом столбце (например, только числа или только даты), может потребоваться впоследствии ограничить разрешенные значения. Можно ограничить значения любого столбца, введя ограничение CHECK. Для ограничения разрешенных значений можно использовать любое правильное условие, применимое в разделе WHERE. В большинстве ограничений CHECK используется условие BETWEEN или IN.

Для получения дополнительной информации о допустимых условиях см. раздел "Условия поиска" (Search conditions) на стр. 23 в документе "*Справочник по SQL для ASA*" *(ASA SQL Reference Manual)*. Для получения дополнительной информации о назначении ограничений таблицам и столбцам см. раздел "Обеспечение целостности данных" на стр. 65.

Пример

Демонстрационная база данных содержит таблицу с именем *Department (Отдел)*, в которой имеются столбцы с именами *dept_id*, *dept_name* и *dept_head_id*. Она определена следующим образом:

Столбец	Тип данных	Размер	Null/Not Null	Ограничение
dept_id	integer (целое число)	—	not null	Нет
dept_name	char (символы)	40	not null	Нет
dept_head_id	integer (целое число)	—	null	Нет

Если указать условие NOT NULL, то каждая строка в столбце должна иметь непустое значение.

ГЛАВА 2

Работа с объектами базы данных

Об этой главе

В этой главе описывается механизм создания, изменения и удаления объектов базы данных – таблиц, представлений и индексов.

Содержание

Введение

С помощью инструментальных средств Adaptive Server Anywhere создается файл базы данных, в котором будут храниться данные. После создания этого файла можно начинать использование базы данных. Например, можно добавлять объекты базы данных, такие, как таблицы или пользователи, а также устанавливать общие свойства базы данных.

В этой главе описывается процесс создания базы данных и объектов в ней. Здесь приведены процедуры для Sybase Central, Interactive SQL, а также утилиты, работающие из командной строки. При необходимости получить больше информации о принципах перед началом работы, см. следующие главы:

- "Проектирование базы данных" на стр. 3;
- "Обеспечение целостности данных" на стр. 65;
- "О Sybase Central" (About Sybase Central) на стр. 50 в документе "*Введение в SQL Anywhere Studio*" (*Introducing SQL Anywhere Studio*);
- "Использование Interactive SQL" (Using Interactive SQL) на стр. 75 в документе "*Введение в ASA*" (*ASA Getting Started*).

Совокупность операторов SQL, используемых для выполнения задач в этой главе, называется **языком определения данных** (Data Definition Language, DDL). Определения объектов базы данных формируют схему базы данных, которую можно представлять себе пустую базу данных.

Процедуры и триггеры также являются объектами базы данных, но рассматриваются в разделе "Использование процедур, триггеров и пакетов" на стр. 495.

Содержание главы

Эта глава содержит следующие разделы:

- Введение в работу с объектами базы данных (этот раздел);
- Описание процесса создания самой базы данных и работы с ней;
- Описание процесса создания и изменения таблиц, представлений и индексов.

Работа с базами данных

В этом разделе описывается процесс создания базы данных и работы с ней. Во время чтения этого раздела помните следующие простые положения:

- Создаваемые базы данных (называемые реляционными базами данных) представляют собой набор таблиц, связанных первичными и внешними ключами. Эти таблицы содержат информацию, хранящуюся в базе данных; таблицы и ключи совместно определяют структуру базы данных. База данных может быть размещена в одном или более файлах базы данных, на одном или более устройствах.
- В файле базы данных также хранятся системные таблицы, которые содержат определение схемы, по которой построена база данных.

Создание базы данных

Adaptive Server Anywhere предоставляет несколько способов создания баз данных: в Sybase Central, в Interactive SQL, а также с использованием командной строки. Создание базы данных также называют ее инициализацией. После создания базы данных можно подключаться к ней и формировать таблицы и другие требуемые в этой базе данных объекты.

Журнал транзакций

При создании базы данных необходимо решить, где будет размещаться журнал транзакций. В этом журнале будет храниться информация обо всех изменениях базы данных в порядке их выполнения. В случае повреждения файла базы данных или его носителя, журнал транзакций играет важную роль при восстановлении базы данных. Он также позволяет сделать использование базы данных более эффективным. По умолчанию журнал помещается в ту же папку, что и файл базы данных, но при использовании базы данных как готового продукта это не рекомендуется.

Для получения дополнительной информации о размещении журнала транзакций см. раздел "Конфигурирование базы данных для обеспечения защиты данных" (Configuring your database for data protection) на стр. 313 в документе "*Руководство по администрированию баз данных ASA*" (*ASA Database Administration Guide*).

Совместимость файлов базы данных

База данных Adaptive Server Anywhere представляет собой файл операционной системы. Он может быть скопирован в другое место так же, как и любой другой файл.

Файлы базы данных совместимы со всеми операционными системами, за исключением тех, в которых имеются ограничения по размеру файлов, накладываемые файловой системой, или применяется поддержка Adaptive Server Anywhere для больших файлов. База данных, созданная в любой операционной системе, может использоваться в другой операционной системе после копирования файла(ов) базы данных. Аналогично, база данных, созданная на персональном сервере, может использоваться и на сетевом сервере. Серверы Adaptive Server Anywhere могут работать с базами данных, созданными в более ранних версиях данного программного обеспечения, однако старые серверы не могут работать с более новыми базами данных.

Для получения дополнительной информации об ограничениях см. раздел "Ограничения по размеру и количеству" (Size and number limitations) на стр. 620 в документе "*Руководство по администрированию баз данных ASA*" (*ASA Database Administration Guide*).

Использование для создания базы данных других приложений

Некоторые системы разработки приложений, такие как Sybase PowerBuilder, включают инструментальные средства для создания объектов баз данных. Эти средства создают операторы SQL, которые передаются серверу, как правило, через его ODBC-интерфейс. Если используется одно из этих средств, то при создании таблицы, распределении полномочий и т. д. нет необходимости вручную создавать операторы SQL.

В этой главе описываются операторы SQL, предназначенные для определения объектов базы данных. В случае построения базы данных в интерактивном инструментальном средстве SQL, например, Interactive SQL, эти операторы можно использовать непосредственно. Даже при использовании средств разработки приложений может возникнуть необходимость использовать операторы SQL для добавления к базе данных каких-либо возможностей, не поддерживаемых этим средством разработки.

Для опытных пользователей такие средства проектирования баз данных, как Sybase PowerDesigner, предоставляют более полный и надежный подход к созданию правильно спроектированных баз данных.

Для получения дополнительной информации о проектировании баз данных см. раздел "Проектирование базы данных" на стр. 3.

Создание базы данных (Sybase Central)

Для создания базы данных в Sybase Central можно использовать утилиту Create Database. После создания базы данных она появляется под значком соответствующего сервера в дереве объектов Sybase Central.

Для получения дополнительной информации см. раздел "Создание базы данных (SQL)" на стр. 31 и "Создание баз данных (командная строка)" на стр. 31.

❖ **Создание новой базы данных (Sybase Central)**

1 Выберите Tools ➤Adaptive Server Anywhere 8 ➤Create Database.

2 Выполняйте указания мастера.

❖ **Создание новой базы данных на основе текущего подключения**

1 Выполните подключение к базе данных.

2 Откройте папку Utilities (находится в папке сервера).

3 В правой области окна дважды щелкните Create Database.

4 Выполняйте указания мастера.

Создание баз данных для Windows CE

Создание базы данных для Windows CE выполняется посредством копирования файлов базы данных Adaptive Server Anywhere на устройство. Средства Sybase Central позволяют создавать базы данных для Windows CE без затруднений. Если на машине под управлением Windows или Windows NT имеются установленные службы Windows CE, то имеются и средства, позволяющие создавать базы данных для Windows CE одновременно с созданием баз данных в Sybase Central. Sybase Central обеспечивает соответствие баз данных требованиям для Windows CE и при необходимости копирует полученный в результате файл базы данных на машину с Windows CE.

Создание базы данных (SQL)

В Interactive SQL для создания баз данных можно использовать оператор CREATE DATABASE. Перед использованием этого оператора необходимо подключиться к существующей базе данных.

❖ **Создание новой базы данных (SQL)**

1 Выполните подключение к существующей базе данных.

2 Выполните оператор CREATE DATABASE.

 Для получения дополнительной информации см. раздел "Оператор CREATE DATABASE" (CREATE DATABASE statement) на стр. 257 в документе "*Справочник по SQL для ASA*" *(ASA SQL Reference Manual)*.

Пример

Создайте файл базы данных с именем *temp.db* в папке *c:\temp*.

```
CREATE DATABASE 'c:\\temp\\temp.db
```

Путь к папке указывается относительно сервера базы данных. Задание полномочий, необходимых для выполнения этого оператора из командной строки сервера, выполняется с помощью параметра -gu в командной строке. По умолчанию для настройки требуются полномочия администратора БД.

 Наклонная черта влево в SQL является escape-символом, поэтому ее необходимо удвоить. Для получения дополнительной информации см. раздел "Строки" (Strings) на стр. 8 в документе "*Справочник по SQL для ASA*" (*ASA SQL Reference Manual*).

Создание базы данных (командная строка)

Создать базу данных из командной строки можно с помощью утилиты *dbinit*. При использовании этой утилиты в командной строке можно ввести параметры, задающие различные опции для базы данных.

❖ **Создание новой базы данных (командная строка)**

1 Откройте командную строку.

2 Запустите утилиту *dbinit*. Укажите все необходимые параметры.

Например, чтобы создать базу данных с именем *company.db* и размером страницы 4 КБ, наберите:

```
dbinit -p 4096 company.db
```

Для получения дополнительной информации см. раздел "Утилита инициализации" (The Initialization utility) на стр. 457 в документе "*Руководство по администрированию баз данных ASA*" (*ASA Database Administration Guide*).

Удаление базы данных

При удалении базы данных с диска удалятся все таблицы и данные, включая журнал транзакций, содержащий информацию об изменениях базы данных. Все файлы базы данных доступны только для чтения, с целью предотвращения случайного их изменения или удаления.

В Sybase Central удаление базы данных выполняется с помощью утилиты Erase Database. Для обращения к этой утилите необходимо подключиться к базе данных, однако мастер Erase Database позволяет указать для удаления любую базу данных. Чтобы удалить базу данных, не используемую в данный момент, сервер базы данных должен быть активен.

В Interactive SQL удаление базы данных производится оператором DROP DATABASE. Требуемые полномочия можно задать с помощью параметра `-gu` в командной строке сервера базы данных. По умолчанию для настройки требуются полномочия администратора БД.

Также можно удалить базу данных из командной строки с помощью утилиты *dberase*. При использовании этой утилиты удаляемая база данных не должна использоваться другими пользователями.

❖ Удаление базы данных (Sybase Central)

1. Откройте папку Utilities.
2. В правой области окна дважды щелкните Erase Database.
3. Выполняйте указания мастера.

❖ Удаление базы данных (SQL)

- Выполните оператор DROP DATABASE. Пример:

```
DROP DATABASE 'c:\temp\temp.db'
```

❖ Удаление базы данных (командная строка)

- Запустите из командной строки утилиту *dberase*. Пример:

```
dberase company.db
```

Для получения дополнительной информации см. раздел "Оператор DROP DATABASE" (DROP DATABASE statement) на стр. 368 в документе "*Справочник по SQL для ASA*" (*ASA SQL Reference Manual*), а также раздел "Утилита Erase" (The Erase utility) на стр. 451 в документе "*Руководство по администрированию баз данных ASA*" *(ASA Database Administration Guide)*.

Отключение от базы данных

По окончании работы с базой данных можно отключиться от нее. Adaptive Server Anywhere также предоставляет возможность отключить от текущей базы данных других пользователей; для получения дополнительной информации о выполнении этого в Sybase Central см. раздел "Управление подключенными пользователями" (Managing connected users) на стр. 364 в документе "*Руководство по администрированию баз данных ASA*" (*ASA Database Administration Guide*).

Можно получить *код-подключения* пользователя, применив для запроса кода подключения функцию **connection_property**. Нижеприведенный оператор возвращает код текущего подключения:

```
SELECT connection_property( 'number' )
```

❖ Отключение от базы данных (Sybase Central)

1 Выберите базу данных.

2 Нажмите Tools ➤Disconnect.

❖ Отключение от базы данных (SQL)

- Выполните оператор DISCONNECT.

Пример 1

Использование оператора DISCONNECT в Interactive SQL для отсоединения всех подключений:

```
DISCONNECT ALL
```

Пример 2

Использование оператора DISCONNECT в Embedded SQL:

```
EXEC SQL DISCONNECT :conn_name
```

❖ Отключение других пользователей от базы данных (SQL)

1 Выполните подключение к существующей базе данных с использованием полномочий администратора БД.

2 Выполните оператор DROP CONNECTION.

Пример 3

Нижеприведенный оператор сбрасывает подключение с идентификатором 4.

```
DROP CONNECTION 4
```

Для получения дополнительной информации см. раздел "Оператор DISCONNECT [ESQL] [Interactive SQL]" (DISCONNECT statement [ESQL] [Interactive SQL]) на стр. 365 в документе "*Справочник по SQL для ASA*" (*ASA SQL Reference Manual*) и раздел "Оператор DROP CONNECTION" (DROP CONNECTION statement) на стр. 369 в документе "*Справочник по SQL для ASA*" (*ASA SQL Reference Manual*).

Установка свойств объектов базы данных

Большинство объектов базы данных (в том числе и сама база данных) имеют набор свойств, которые можно просмотреть и/или установить. Некоторые свойства являются неизменяемыми. Они отражают параметры настройки, выбранные при создании базы данных или объекта. Остальные свойства допускают изменение.

Лучший способ просмотра и изменения свойств набора состоит в использовании окон свойств Sybase Central.

Если Sybase Central не используется, свойства могут быть указаны при создании объекта с помощью оператора CREATE. Если объект уже существует, можно изменять его параметры с помощью оператора SET OPTION.

❖ Просмотр и изменение свойств объекта базы данных (Sybase Central)

1. В дереве объектов Sybase Central откройте папку или контейнер, в котором находится объект.
2. Щелкните правой кнопкой мыши на объекте и выберите Properties во всплывающем меню.
3. Отредактируйте требуемые свойства.

Установка параметров базы данных

Параметры базы данных – это редактируемые установки, влияющие на поведение или работу базы данных. В Sybase Central все эти параметры сгруппированы в диалоге Database Options. В Interactive SQL эти параметры устанавливаются с помощью оператора SET OPTION.

❖ Установка параметров базы данных (Sybase Central)

1. Откройте требуемый сервер.
2. Щелкните правой кнопкой мыши на нужной базе данных и выберите Options во всплывающем меню.
3. Отредактируйте требуемые значения.

❖ Установка параметров базы данных (SQL)

- Укажите требуемые свойства в операторе SET OPTION.

Советы

В диалоге Database Options можно также задать параметры базы данных для определенных пользователей и групп (если этот диалог открывается для пользователя или группы, то он имеет название User Options (Параметры пользователя) или Group Options (Параметры группы) соответственно).

При задании параметров всей базы данных, в ней фактически происходит установка параметров для группы PUBLIC, поскольку все пользователи и группы наследуют параметры настройки группы PUBLIC.

Определение консолидированной базы данных

В Sybase Central можно определить консолидированную базу данных для репликации в SQL Remote. Консолидированная база данных – это база данных, функционирующая в установках репликации как "главная" база данных. Консолидированная база данных содержит все данные, которые подлежат репликации, в то время как подчиненные ей удаленные базы данных могут содержать только собственные поднаборы данных. В случае конфликта или несоответствия консолидированная база данных рассматривается как имеющая исходный набор всех данных.

Для получения дополнительной информации см. раздел "Консолидированные и удаленные базы данных" (Consolidated and remote databases) на стр. 21 в документе "*Введение в SQL Anywhere Studio*" (*Introducing SQL Anywhere Studio*).

❖ Определение консолидированной базы данных (Sybase Central)

1. Щелкните правой кнопкой мыши на требуемой базе данных и выберите Properties во всплывающем меню.
2. Щелкните на закладке SQL Remote.
3. Нажмите кнопку Change рядом с текстовым полем Consolidated Database.
4. Установите требуемые параметры.

Отображение системных объектов в базе данных

В базе данных таблицы, представления, хранимые процедуры и домены являются системными объектами. Системные таблицы содержат информацию о самой базе данных, в то время как системные представления, процедуры и домены в значительной степени поддерживают совместимость с Sybase Transact-SQL.

В Interactive SQL нельзя отобразить список всех системных объектов, но есть возможность просмотреть содержимое системной таблицы; для получения дополнительной информации см. раздел "Отображение системных таблиц" на стр. 50.

❖ Отображение системных объектов в базе данных (Sybase Central)

1. Откройте требуемый сервер.
2. Щелкните правой кнопкой мыши на требуемой подключенной базе данных и выберите Filter Objects by Owner (Сортировка объектов по владельцам).
3. Включите **SYS** и **dbo** и нажмите OK.

Системные таблицы, системные представления, системные процедуры и системные домены появляются в соответствующих папках (например, системные таблицы появляются рядом с обычными таблицами в папке Tables).

Для получения дополнительной информации см. раздел "Системные таблицы" (System Tables) на стр. 557 в документе "*Справочник по SQL для ASA*" (*ASA SQL Reference Manual*).

Регистрация операторов SQL при работе с базой данных

Во время работы с базой данных в Sybase Central приложение автоматически генерирует операторы SQL в зависимости от производимых действий. Можно вести журнал появления этих операторов как в отдельном окне, так и в указанном файле.

При работе в Interactive SQL также можно вести журнал выполняемых операторов; для получения дополнительной информации см. раздел "Команды регистрации" (Logging commands) на стр. 90 в документе "*Введение в ASA*" (*ASA Getting Started*).

❖ Регистрация операторов SQL, генерируемых в Sybase Central

1. Щелкните правой кнопкой мыши на базе данных и выберите Log SQL Statements во всплывающем меню.
2. В появившемся диалоге установите требуемые параметры.

Запуск базы данных без подключения

Как в Sybase Central, так и в Interactive SQL можно запустить базу данных, не подключаясь к ней.

❖ Запуск базы данных на сервере без подключения (Sybase Central)

1. Щелкните правой кнопкой мыши на требуемом сервере и выберите Start Database во всплывающем меню.
2. Введите в диалоге Start Database требуемые значения.

База данных отображается под значком сервера как не подключенная.

❖ Запуск базы данных на сервере без подключения (SQL)

1. Запустите Interactive SQL.
2. Выполните оператор START DATABASE.

Пример

Запустите файл базы данных *c:\asa8\sample_2.db* как sam2 на сервере с именем sample.

```
START DATABASE 'c:\asa8\sample_2.db'
AS sam2
ON sample
```

Для получения дополнительной информации см. раздел "Оператор START DATABASE" (START DATABASE statement) на стр. 513 в документе *"Справочник по SQL для ASA"* (*ASA SQL Reference Manual*).

Установка поддержки метаданных jConnect для существующей базы данных

Если база данных была создана без поддержки метаданных jConnect, в дальнейшем можно использовать Sybase Central для ее установки.

❖ **Добавление поддержки метаданных jConnect к существующей базе данных (Sybase Central)**

1 Откройте сервер базы данных.

2 Щелкните правой кнопкой мыши на базе данных и выберите Re-install jConnect Meta-data Support во всплывающем меню.

3 Выполняйте указания мастера.

Работа с таблицами

Если база данных только что создана, то единственные таблицы, имеющиеся в ней – это системные таблицы. Системные таблицы содержат схему базы данных.

В этом разделе описывается процесс создания, изменения и удаления таблиц. Примеры можно выполнять в Interactive SQL, но операторы SQL не зависят от используемых административных средств. При выполнении запросов в Interactive SQL можно редактировать значения в результирующем наборе.

Для получения дополнительной информации см. раздел "Редактирование значений таблиц в Interactive SQL" (Editing table values in Interactive SQL) на стр. 83 в документе "*Введение в ASA*" *(ASA Getting Started)*.

Для более простого воссоздания схемы базы данных при необходимости создайте командные файлы для определения таблиц в базе данных. Командные файлы должны содержать операторы CREATE TABLE и ALTER TABLE.

Для получения дополнительной информации о группах, таблицах и подключении от имени другого пользователя см. раздел "Обращение к таблицам, принадлежащих группам" (Referring to tables owned by groups) на стр. 369 в документе "*Руководство по администрированию баз данных ASA*" (*ASA Database Administration Guide*) и раздел "Имена и префиксы объектов базы данных" (Database object names and prefixes) на стр. 372 в документе "*Руководство по администрированию баз данных ASA*" (*ASA Database Administration Guide*).

Использование редактора таблиц Sybase Central

Редактор таблиц – инструмент, к которому можно обращаться через Sybase Central. Он предоставляет быстрый способ создания и редактирования таблиц и их столбцов.

С помощью редактора таблиц можно создавать, редактировать и удалять столбцы. Можно также добавлять и удалять столбцы в первичном ключе.

❖ Доступ к редактору таблиц для создания новой таблицы

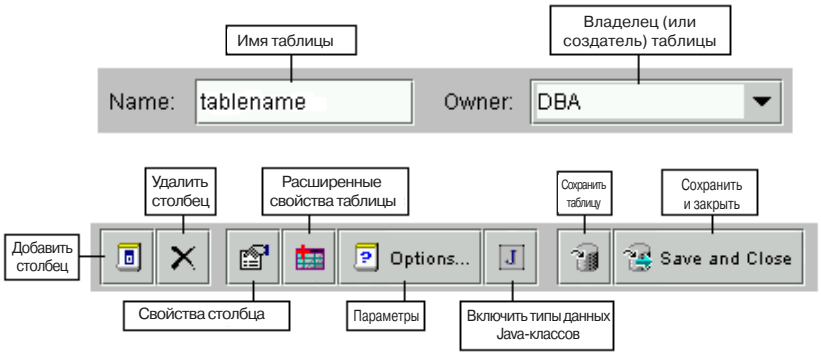
1. Откройте папку Tables.
2. В правой области окна дважды щелкните Add Table.

❖ Доступ к редактору таблиц для редактирования существующей таблицы

1. Откройте папку Tables.
2. Щелкните правой кнопкой мыши на таблице и выберите Edit Columns во всплывающем меню.

Панель инструментов редактора таблиц

После открытия редактора таблиц появляется панель инструментов (показана ниже), имеющая поля и кнопки с общими команд.



В первой половине этой панели отображается имя текущей таблицы и ее владельца (или создателя). Для новой таблицы в обоих этих полях можно указать новые значения. Для существующей таблицы можно ввести новое имя, но нельзя изменить владельца.

Кнопки во второй половине панели используются для следующих действий:

- добавление нового столбца; новый столбец появляется в конце списка существующих столбцов;
- удаление выбранных столбцов;
- просмотр окна свойств выбранного столбца;
- просмотр окна расширенных свойств таблицы для всей таблицы;
- просмотр и изменение параметров редактора таблиц;
- добавление Java-классов к типам данных выбранного столбца; (доступно не всегда.)
- сохранение таблицы без закрытия редактора таблиц;
- сохранение таблицы и последующее закрытие редактора таблиц.

При наведении курсора на ту или иную кнопку появляется всплывающеее описание, которое можно использовать в качестве быстрой справки о функциях этой кнопки.

Использование диалога Advanced Table Properties

В редакторе таблиц можно использовать диалог Advanced Table Properties (Расширенные свойства таблицы) для задания или просмотра типа таблицы или ввода комментария к таблице.

❖ Просмотр и установка расширенных свойств таблицы

1 Откройте редактор таблиц.

2 Нажмите кнопку Advanced Table Properties на панели инструментов.

3 Отредактируйте требуемые значения.

Компоненты диалога Advanced Table Properties

- **Base table (Базовая таблица).** Определяет эту таблицу как базовую (такую, которая постоянно хранит данные вплоть до их удаления). Для существующих таблиц эту установку изменить нельзя; задать тип таблицы можно только при создании новой таблицы.
  - **Dbspace (Размер БД).** Позволяет выбрать используемый таблицей размер (dbspace). Этот параметр доступен только в том случае, если создается новая базовая таблица.
  - **Global temporary table (Глобальная временная таблица).** Определяет эту таблицу как глобальную временную (такую, которая хранит данные только для одного подключения). Для существующих таблиц эту установку изменить нельзя; задать тип таблицы можно только при создании новой таблицы.
  - **Delete rows (Удаление строк).** Включает удаление строк глобальной временной таблицы при выполнении команды COMMIT.
  - **Preserve rows (Сохранение строк).** Включает сохранение строк глобальной временной таблицы при выполнении команды COMMIT.
- **Primary key maximum hash size (Максимальный размер хэша первичного ключа).** Размер хэша - это количество байтов, используемых для размещения значения в индексе. Выбирать размер хэша можно только для баз данных Adaptive Server Anywhere версии 7.
- **Comment (Комментарий).** Предоставляет место для ввода комментария (текстового описания) данного объекта. Например, можно использовать эту область для описания назначения объекта в системе.

Совет

Тип таблицы и комментарий также появляются в окне свойств таблицы.

Создание таблиц

Если база данных только что создана, то единственные таблицы, имеющиеся в ней - системные таблицы, содержащие схему базы данных. После этого можно создавать новые таблицы для хранения актуальных данных, как с помощью операторов SQL в Interactive SQL, так и с использованием Sybase Central.

Можно создавать таблицы двух типов:

- **Базовая таблица.** Таблица, содержащая постоянные данные. Таблица и ее данные продолжают существовать, пока не будет произведено явное удаление этих данных или удаление таблицы. Она называется базовой таблицей, в отличие от временных таблиц и представлений.

- **Временная таблица.** Данные во временной таблице хранятся только для одного подключения. Определения глобальной временной таблицы (но не данных) хранятся в базе данных вплоть до удаления. Определения локальной временной таблицы и данных существуют только в течение одного подключения.

Для получения дополнительной информации о временных таблицах см. раздел "Временные таблицы" на стр. 63.

Таблицы состоят из строк и столбцов. Каждый столбец содержит определенный вид информации, например, номер телефона или имя, в то время как каждая строка определяет конкретный элемент.

❖ Создание таблицы (Sybase Central)

1. Выполните подключение к базе данных.
2. Откройте папку Tables.
3. В правой области окна дважды щелкните Add Table.
4. В редакторе таблиц:
   - Введите имя новой таблицы.
   - Выберите владельца таблицы.
   - Создайте столбцы, используя кнопку Add Column на панели инструментов.
   - Задайте свойства столбца.
5. Сохраните таблицу и закройте редактор таблицы по окончании работы.

❖ Создание таблицы (SQL)

1. Выполните подключение к базе данных с использованием полномочий администратора БД.
2. Выполните оператор CREATE TABLE.

Пример

Следующий оператор создает новую таблицу, в которой будет описана квалификация каждого служащего компании. В таблице имеются столбцы для идентификатора, названия и типа (технический или административный) для каждой профессии.

```
CREATE TABLE skill (
   skill_id INTEGER NOT NULL,
   skill_name CHAR( 20 ) NOT NULL,
   skill_type CHAR( 20 ) NOT NULL
)
```

Для получения дополнительной информации см. раздел "Оператор CREATE TABLE" (CREATE TABLE statement) на стр. 321 в документе "*Справочник по SQL для ASA*" (*ASA SQL Reference Manual*) и раздел "Использование редактора таблиц Sybase Central" (Using the Sybase Central Table Editor) на стр. 38.

Изменение таблиц

В этом разделе описывается процесс изменения структуры или определения столбцов в таблице. Например, можно добавлять столбцы, изменять различные атрибуты столбца или полностью удалить столбец.

В Sybase Central это выполняется с использованием кнопок на панели инструментов редактора таблиц. В Interactive SQL это выполняется с помощью оператора ALTER TABLE.

При работе с Sybase Central можно также управлять столбцами (добавлять или удалять их из первичного ключа, изменять их свойства, удалять их) с помощью команд меню, предварительно выбрав столбец в папке Columns.

Для получения информации об изменении свойств объектов базы данных см. раздел "Установка свойств объектов базы данных" (Setting properties for database objects) на стр. 34.

Для получения информации о предоставлении и возобновлении полномочий на таблицы см. раздел "Получение полномочий на таблицы" (Granting permissions on tables) на стр. 355 документа "*Руководство по администрированию баз данных ASA*" (*ASA Database Administration Guide*) и раздел "Возобновление полномочий пользователей" (Revoking user permissions) на стр. 362 в документе "*Руководство по администрированию баз данных ASA*" *(ASA Database Administration Guide)*.

Изменение таблиц (Sybase Central)

В Sybase Central можно изменять таблицы, используя редактор таблиц. Например, можно добавлять или удалять столбцы, изменять определения столбцов и изменять свойства столбца или таблицы.

❖ Изменение существующей таблицы (Sybase Central)

1. Выполните подключение к базе данных.
2. Откройте папку Tables.
3. Щелкните правой кнопкой мыши на таблице, которую требуется изменить, и выберите Edit Columns во всплывающем меню.
4. Выполните необходимые изменения с помощью редактора таблиц.

Советы

Добавить столбец можно, открыв папку Columns нужной таблицы и дважды щелкнув Add Column.

Удалить столбец можно, открыв папку Columns нужной таблицы, щелкнув правой кнопкой мыши на столбце и выбрав Delete во всплывающем меню.

Для получения дополнительной информации см. раздел "Использование редактора таблиц Sybase Central" (Using the Sybase Central Table Editor) на стр. 38 и раздел "Оператор ALTER TABLE" (ALTER TABLE statement) на стр. 219 в документе "*Справочник по SQL для ASA*" *(ASA SQL Reference Manual)*.

Изменение таблиц (SQL)

В Interactive SQL таблицы можно изменять с помощью оператора ALTER TABLE.

❖ Изменение существующей таблицы (SQL)

1 Выполните подключение к базе данных с использованием полномочий администратора БД.

2 Выполните оператор ALTER TABLE.

Примеры

Нижеприведенная команда добавляет столбец к таблице *skill* (Профессия), чтобы создать пространство для дополнительного описания профессии:
ALTER TABLE skill.

```
ADD skill_description
CHAR( 254 )
```

Этот оператор добавляет столбец с именем *skill_description*, в котором может содержаться несколько предложений, описывающих данную профессию.

С помощью оператора ALTER TABLE можно также изменить атрибуты столбца. Следующий оператор уменьшает максимальную ширину столбца *skill_description* демонстрационной базы данных с 254 символов до 80:

```
ALTER TABLE skill
MODIFY skill_description CHAR( 80 )
```

Любые имеющиеся элементы длиннее 80 символов сокращаются до 80-символьного предела, о чем выдается предупреждение.

Следующий оператор изменяет имя столбца *skill_type* на имя *classification:*

```
ALTER TABLE skill
RENAME skill_type TO classification
```

Следующий оператор удаляет столбец *classification*.

```
ALTER TABLE skill
DROP classification
```

Следующий оператор изменяет имя всей таблицы:

```
ALTER TABLE skill
RENAME qualification
```

Эти примеры демонстрируют процесс изменения структуры базы данных. Используя оператор ALTER TABLE, можно изменять практически все, относящееся к таблице – добавить или удалить внешние ключи, изменить тип столбца с одного на другой и т.д. Во всех этих случаях, после какого-либо изменения хранимые процедуры, представления и любые другие элементы, связанные с этой таблицей, работать не будут.

Для получения дополнительной информации см. раздел "Оператор ALTER TABLE" (ALTER TABLE statement) на стр. 219 в документе "*Справочник по SQL для ASA*" раздел "Обеспечение целостности данных" на стр. 65.

Удаление таблиц

В этом разделе описывается процесс удаления таблиц из базы данных. Для выполнения этого действия можно воспользоваться как Sybase Central, так и Interactive SQL. В Interactive SQL удаление таблицы также называется ее сбросом.

Удалить таблицу нельзя, если она используется как статья в публикации SQL Remote. Если попытаться выполнить это действие в Sybase Central, появится сообщение об ошибке.

❖ Удаление таблицы (Sybase Central)

1. Выполните подключение к базе данных.
2. Откройте папку Tables соответствующей базы данных.
3. Щелкните правой кнопкой мыши на таблице и выберите Delete во всплывающем меню.

❖ Удаление таблицы (SQL)

1. Выполните подключение к базе данных с использованием полномочий администратора БД.
2. Выполните оператор DROP TABLE.

Пример

Следующая команда DROP TABLE удаляет все записи в таблице *skill* и затем удаляет определение таблицы *skill* из базы данных.

```
DROP TABLE skill
```

Подобно оператору CREATE, оператор DROP автоматически выполняет оператор COMMIT перед сбросом таблицы и после него. В результате все изменения в базе данных, произошедших после выполнения последней команды COMMIT или ROLLBACK, становятся постоянными. Оператор сброса сбрасывает также все индексы в таблице.

Для получения дополнительной информации см. раздел "Оператор DROP" (DROP statement) на стр. 366 в документе "*Справочник по SQL для ASA*" (*ASA SQL Reference Manual*).

Просмотр информации в таблицах

Для просмотра данных, содержащихся в таблицах базы данных, можно использовать как Sybase Central, так и Interactive SQL

При работе в Sybase Central просмотреть данные в таблице можно щелчком на закладке Data в правой области окна.

При работе в Interactive SQL выполните следующий оператор:

```
SELECT * FROM table-name
```

Редактировать данные в таблице можно на закладке Results (Результаты) Interactive SQL, или из Sybase Central.

Управление первичными ключами

Первичный ключ – это уникальный идентификатор, состоящий из столбца или комбинации столбцов со значениями, которые не изменяются на протяжении всего времени существования данных в строке. Поскольку уникальность существенна для качественной разработки базы данных, лучше всего задавать первичный ключ при определении таблицы.

В этом разделе описывается процесс создания и редактирования первичных ключей в базе данных. Для выполнения этих действий можно воспользоваться как Sybase Central, так и Interactive SQL.

Порядок столбцов в первичных ключах на несколько столбцов

Порядок столбцов первичного ключа определяется в соответствии с порядком столбцов во время создания таблицы. Он не основывается на порядке столбцов, определенном в объявлении первичного ключа.

Управление первичными ключами (Sybase Central)

В Sybase Central первичный ключ таблицы появляется в нескольких местах:

- на панели Columns окна свойств таблицы;
- в папке Columns таблицы;
- в редакторе таблиц.

Столбцы первичного ключа имеют специальные значки для их отличия от не ключевых столбцов. Списки в окне свойств таблицы и в папке Columns отображают столбцы первичного ключа (наряду с не ключевыми столбцами) в том порядке, в котором они были созданы в базе данных. Это может зависеть от текущего расположения столбцов в первичном ключе.

❖ Создание и редактирование первичного ключа с использованием окна свойств

1. Откройте папку Tables.
2. Щелкните правой кнопкой мыши на таблице и выберите Properties во всплывающем меню.
3. Щелкните на закладке Columns окна свойств.
4. Используйте кнопки для изменения первичного ключа.

❖ **Создание и редактирование первичного ключа с использованием закладки Columns**

1 Откройте папку Tables и дважды щелкните на таблице.

2 Откройте папку Columns для этой таблицы и щелкните правой кнопкой мыши на столбце.

3 Во всплывающем меню выберите одно из следующих действий:

- Add to Primary Key (Добавление к первичному ключу), если столбец еще не является частью первичного ключа и требуется добавить его в ключ;
- Remove From Primary Key (Удаление из первичного ключа), если столбец является частью первичного ключа, и его требуется удалить из ключа.

❖ **Создание и редактирование первичного ключа с использованием редактора таблиц**

1 Откройте папку Tables.

2 Щелкните правой кнопкой мыши на таблице и выберите Edit Columns во всплывающем меню.

3 В редакторе таблиц щелкните на значке в поле Key (Ключ) (в крайней левой области редактора таблиц), чтобы добавить столбец к первичному ключу или удалить его из ключа.

Управление первичными ключами (SQL)

В Interactive SQL можно создавать и изменять первичные ключи, используя операторы CREATE TABLE и ALTER TABLE. Эти операторы позволяют устанавливать многие атрибуты таблицы, включая ограничения столбца, а также проверки.

Столбцы в первичном ключе не могут содержать пустые (NULL) значения. Столбец первичного ключа необходимо определить как NOT NULL.

❖ **Изменение первичного ключа существующей таблицы (SQL)**

1 Выполните подключение к базе данных с использованием полномочий администратора БД.

2 Выполните оператор ALTER TABLE.

Пример 1

Следующий оператор создает такую же таблицу *skill*, как и ранее, за исключением того, что он добавляет первичный ключ:

```
CREATE TABLE skill (
   skill_id INTEGER NOT NULL,
   skill_name CHAR( 20 ) NOT NULL,
   skill_type CHAR( 20 ) NOT NULL,
   primary key( skill_id )
)
```

Значения первичного ключа должны быть уникальны для каждой строки в таблице, что в данном случае означает следующее: в таблице должна быть только одна строка, имеющая данный *skill_id* (код таблицы). Каждая строка в таблице однозначно определяется ее первичным ключом.

Пример 2

Следующий оператор добавляет столбцы *skill_id* и *skill_type* к первичному ключу таблицы *skill*:

```
ALTER TABLE skill (
  ADD PRIMARY KEY ( "skill_id", "skill_type" )
)
```

Если в операторе ALTER TABLE присутствует выражение PRIMARY KEY, у таблицы уже не должно быть первичного ключа, который был создан оператором CREATE TABLE или другим оператором ALTER TABLE.

Пример 3

Следующий оператор удаляет все столбцы из первичного ключа в таблице *skill*. Перед удалением первичного ключа убедитесь, что последствия этого действия для базы данных точно известны.

```
ALTER TABLE skill
   DELETE PRIMARY KEY
```

Для получения дополнительной информации см. раздел "Оператор ALTER TABLE" (ALTER TABLE statement) на стр. 219 в документе "*Справочник по SQL для ASA*" и раздел "Управление первичными ключами (Sybase Central)" на стр. 45.

Управление внешними ключами

В этом разделе описывается процесс создания и редактирования внешних ключей в базе данных. Для выполнения этих действий можно воспользоваться как Sybase Central, так и Interactive SQL.

Внешние ключи используются для связи значений в дочерней таблице (или внешней таблице) с соответствующими им в родительской таблице (или первичной таблице). Таблица может иметь множество внешних ключей, которые ссылаются на множество родительских таблиц, связывая, таким образом, различные типы информации.

Управление внешними ключами (Sybase Central)

В Sybase Central внешний ключ таблицы появляется в папке Foreign Keys (Внешние ключи) (расположенной в контейнере таблицы).

Создать внешний ключ в таблице нельзя, если таблица содержит такие значения для внешних столбцов, которые не могут соответствовать значениям в первичном ключе первичной таблицы.

После создания внешнего ключа, можно отслеживать их в папке Referenced By (Ссылки) каждой таблицы; эта закладка отображает все внешние таблицы, которые ссылаются в настоящий момент на выбранную таблицу.

❖ **Создание нового внешнего ключа в данной таблице (Sybase Central)**

1 Откройте таблицу в папке Tables.

2 Откройте папку Foreign Keys для этой таблицы.

3 В правой области окна дважды щелкните Add Foreign Key.

4 Выполняйте указания мастера.

❖ **Удаление внешнего ключа (Sybase Central)**

1 Откройте таблицу в папке Tables.

2 Откройте папку Foreign Keys для этой таблицы.

3 Щелкните правой кнопкой мыши на внешнем ключе, который требуется удалить, и выберите Delete во всплывающем меню.

❖ **Отображение информации о таблицах, имеющих внешние ключи со ссылкой на данную таблицу (Sybase Central)**

1 Откройте требуемую таблицу.

2 Откройте папку Referenced By.

Советы

При создании внешнего ключа с использованием мастера можно задать свойства внешнего ключа. Чтобы задать или изменить свойства после того, как внешний ключ был создан, щелкните правой кнопкой мыши на внешнем ключе и выберите Properties во всплывающем меню.

Можно просмотреть свойства таблицы, ссылающейся на данную таблицу, щелкнув правой кнопкой мыши на таблице и выбрав Properties во всплывающем меню.

Управление внешними ключами (SQL)

В Interactive SQL можно создавать и изменять внешние ключи, используя операторы CREATE TABLE и ALTER TABLE. Эти операторы позволяют устанавливать многие атрибуты таблицы, включая ограничения столбца, а также проверки.

Таблица может иметь только один первичный ключ, но внешних ключей может быть столько, сколько необходимо.

❖ **Изменение внешнего ключа существующей таблицы (SQL)**

1 Выполните подключение к базе данных с использованием полномочий администратора БД.

2 Выполните оператор ALTER TABLE.

Пример 1

Можно создать таблицу с именем *emp_skill*, которая будет содержать описание квалификации каждого служащего для каждой профессии, в котором они имеют квалификацию, как показано ниже:

```
CREATE TABLE emp_skill(
   emp_id INTEGER NOT NULL,
   skill_id INTEGER NOT NULL,
   "skill level" INTEGER NOT NULL,
   PRIMARY KEY( emp_id, skill_id ),
   FOREIGN KEY REFERENCES employee,
   FOREIGN KEY REFERENCES skill
   )
```

Определение таблицы *emp_skill* имеет первичный ключ, состоящий из двух столбцов: *emp_id* и *skill_id*. Служащий может иметь более одной профессии, и вследствие этого появляться в нескольких строках; аналогично, несколько служащих могут иметь данную профессию, и *skill_id* может появиться несколько раз. В то же время не может быть больше одного элемента с конкретной комбинацией служащего и профессии.

Таблица *emp_skill* также имеет два внешних ключа. Элементы внешнего ключа указывают, что столбец *emp_id* должен содержать допустимое число из таблицы *employee*, и *skill_id* должен содержать допустимый элемент из таблицы *skill*.

Пример 2

Можно добавить внешний ключ с именем *foreignkey* к существующей таблице *skill* и связать этот внешний ключ с первичным ключом в таблице *contact*, как показано ниже:

```
ALTER TABLE skill
ADD FOREIGN KEY "foreignkey" ("skill_id")
REFERENCES "DBA"."contact" ("id")
```

Этот пример создает связь между столбцом *skill_id* таблицы *skill* (внешняя таблица) и столбцом *id* таблицы *contact* (первичная таблица). Пункт "DBA" показывает владельца таблицы *contact*.

Пример 3

Можно задать свойства внешнего ключа при его создании. Например, следующий оператор создает такой же внешний ключ, как в примере 2, но кроме этого определяет внешний ключ как NOT NULL наряду с ограничениями на обновление или удаление.

```
ALTER TABLE skill
ADD NOT NULL FOREIGN KEY "foreignkey" ("skill_id")
REFERENCES "DBA"."contact" ("id")
ON UPDATE RESTRICT
ON DELETE RESTRICT
```

В Sybase Central можно также задать свойства в мастере создания внешнего ключа или в окне свойств внешнего ключа.

Для получения дополнительной информации см. раздел "Оператор ALTER TABLE" (ALTER TABLE statement) на стр. 219 в документе "*Справочник по SQL для ASA*" (*ASA SQL Reference Manual*) и раздел "Управление внешними ключами (Sybase Central)" на стр. 47.

Копирование таблиц или столбцов в пределах/между базами данных

При использовании Sybase Central можно копировать существующие таблицы или столбцы и вставлять их в другое местоположение в той же самой базе данных или в абсолютно другую базу данных.

Если Sybase Central не используется, см. один из следующих разделов:

- вставка результатов выполнения оператора SELECT в определенное местоположение: см. раздел "Оператор SELECT" (SELECT statement) на стр. 490 в документе "*Справочник по SQL для ASA*" (*ASA SQL Reference Manual*);
- вставка строки или набора строк из какого-либо местоположения в базе данных в таблицу: см. "Оператор INSERT" (INSERT statement) на стр. 431 в документе "*Справочник по SQL для ASA*" (*ASA SQL Reference Manual*).

Отображение системных таблиц

В базе данных таблица, представление, хранимая процедура и домен являются системными объектами. **Системные таблицы** содержат схему базы данных или информацию о самой базе данных. Системные представления, процедуры и домены в значительной степени поддерживают совместимость Sybase Transact-SQL.

Вся информация о таблицах в базе данных появляется в системных таблицах. Информация распределена между несколькими таблицами.

Для получения дополнительной информации см. раздел "Системные таблицы" (System Tables) на стр. 557 в документе "*Справочник по SQL для ASA*" (*ASA SQL Reference Manual*).

❖ Отображение системных таблиц (Sybase Central)

1. Щелкните правой кнопкой мыши на требуемой подключенной базе данных и выберите Filter Objects by Owner во всплывающем меню.
2. Выберите переключатель около пункта SYS и нажмите OK.

   Системные таблицы, системные представления, системные процедуры и системные домены появляются в соответствующих папках (например, системные таблицы появляются рядом с обычными таблицами в папке Tables).

❖ Просмотр системных таблиц (SQL)

1. Выполните подключение к базе данных.
2. Выполните оператор SELECT, указав системную таблицу, которую требуется просмотреть. Владельцем системных таблиц является пользователь с кодом SYS.

Пример

Отображение содержимого таблицы *sys.systable* на закладке Results в окне Results.

```
SELECT *
FROM SYS.SYSTABLE
```

Работа с представлениями

Представления – это вычисляемые таблицы. Представления можно использовать в тех случаях, когда требуется предоставлять информацию пользователям базы данных именно так, как необходимо разработчику, и в полностью настраиваемом формате.

Сходные черты представлений и базовых таблиц

Представления во многом подобны постоянным таблицам базы данных (постоянные таблицы также называют базовыми таблицами):

- можно назначать полномочия на доступ к представлениям так же, как и к базовым таблицам;
- к представлениям можно выполнять запросы SELECT;
- для некоторых представлений можно выполнять операции UPDATE, INSERT и DELETE;
- на основе представлений можно создавать другие представления.

Различия между представлениями и таблицами

Представления и постоянные таблицы имеют ряд отличий:

- по представлениям нельзя создавать индексы;
- нельзя выполнять операции UPDATE, INSERT или DELETE для всех представлений;
- нельзя назначать представлениям ключи и применять по отношению к ним средства контроля за целостностью;
- представления ссылаются на информацию в базовых таблицах, но не содержат копий этой информации; при каждом вызове представления она вычисляется заново.

Преимущества разграничения доступа

Представления позволяют разграничить доступ к данным в базе данных. Разграничение доступа служит для нескольких целей:

- **Улучшенная защита.** Предоставляется доступ только к той информации, которая является необходимой в данном случае.
- **Повышенное удобство пользования.** Данные представляются пользователям и разработчикам приложений в более наглядной форме, чем в базовых таблицах.
- **Повышенная согласованность.** В базе данных производится централизация определений общих запросов.

Создание представлений

Во время просмотра данных оператор SELECT обрабатывает одну или более таблиц и возвращает результирующий набор, который также является таблицей. Так же как и базовая таблица, результирующий набор запроса SELECT организован в столбцы и строки. Представление присваивает конкретному запросу имя и хранит его определение в системных таблицах базы данных.

Предположим, что часто требуется получить список служащих в каждом отделе. Этот список можно получить с помощью следующего оператора:

```
SELECT dept_ID, count(*)
FROM employee
GROUP BY dept_ID
```

Можно создать представление, содержащее результаты выполнения этого оператора, используя как Sybase Central, так и Interactive SQL.

❖ Создание нового представления (Sybase Central)

1 Выполните подключение к базе данных.

2 Откройте папку Views этой базы данных.

3 В правой области окна дважды щелкните Add View (Wizard).

4 Выполняйте указания мастера. По окончании работы автоматически открывается редактор кода.

5 Дополните код, введя таблицу и столбцы, которые будут использоваться. Для вышеприведенного примера введите **employee** и **dept_ID**.

6 В меню File редактора кода выберите Save/Execute in Database.

Новые представления появляются в папке Views.

❖ Создание нового представления (SQL)

1 Выполните подключение к базе данных.

2 Выполните оператор CREATE VIEW.

Пример

Создайте представление с именем *DepartmentSize*, которое будет содержать результаты выполнения оператора SELECT, приведенного в начале этого раздела:

```
CREATE VIEW DepartmentSize AS
SELECT dept_ID, count(*)
FROM employee
GROUP BY dept_ID
```

Так как информация в представлении не хранится отдельно в базе данных, обращение к представлению вызывает выполнение ассоциированного оператора SELECT с целью получения соответствующих данных.

С одной стороны, это удобно, поскольку означает, что при изменении кем-либо таблицы *employee* информация в представлении *DepartmentSize* автоматически обновляется. С другой стороны, сложные операторы SELECT могут увеличить время, которое требуется SQL для поиска правильной информацию каждый раз при использовании представления.

Для получения дополнительной информации см. раздел "Оператор CREATE VIEW" (CREATE VIEW statement) на стр. 342 в документе "*Справочник по SQL для ASA*" *(ASA SQL Reference Manual)*.

Использование представлений

Ограничения на операторы SELECT

При создании представлений существуют некоторые ограничения на использование операторов SELECT. В частности, в запросе SELECT нельзя использовать раздел ORDER BY. Особенностью реляционных таблиц является то, что порядок строк или столбцов не учитывается, а использование раздела ORDER BY выстроило бы строки представления в определенном порядке. В определениях представлений можно использовать раздел GROUP BY, подзапросы и соединения.

Чтобы разработать представление, запрос SELECT следует отладить отдельно, так, чтобы он возвращал именно те результаты, которые необходимы, и в требуемом формате. После того, как запрос SELECT будет отлажен полностью, можно будет создавать представления, ставя перед запросом соответствующие выражения. Пример:

```
CREATE VIEW viewname AS
```

Обновление представлений

Операторы UPDATE, INSERT и DELETE разрешены для одних представлений, но не разрешены для других; это зависит от ассоциированного с представлением оператора SELECT.

Нельзя обновлять представления, содержащие агрегатные функции, например,COUNT(*). Также нельзя обновлять представления, содержащие раздел GROUP BY в операторе SELECT, или представления, содержащие операцию UNION. Во всех этих случаях невозможно применить UPDATE как действие над используемыми таблицами.

Копирование представлений

В Sybase Central можно копировать представления между базами данных. Для этого необходимо выбрать представление в правой области окна Sybase Central и перетащить его в папку Views другой подключенной базы данных. В результате этого создается новое представление, и код исходного представления копируется в него.

Следует отметить, что в новое представление копируется только код представления. Другие свойства представления, например, полномочия, не копируются.

Использование раздела WITH CHECK OPTION

Даже если операторы INSERT и UPDATE разрешены для применения к представлению, возможно, что вставленные или измененные строки в используемых таблицах могут оказаться не соответствующими требованиям самого представления. Например, в представлении не появляется новых строк, даже если действие INSERT или UPDATE изменило используемые таблицы.

Примеры с использованием раздела WITH CHECK OPTION

Следующий пример иллюстрирует эффективность применения раздела WITH CHECK OPTION. Этот дополнительный раздел является заключительным разделом в операторе CREATE VIEW.

❖ **Создание представления, отображающего информацию о служащих в отделе сбыта (SQL)**

◆ Введите следующие операторы:

```
CREATE VIEW sales_employee
AS SELECT
   emp_id, emp_fname,
   emp_lname,
   dept_id
FROM employee
WHERE dept_id = 200
```

Содержание этого представления следующее:

```
SELECT *
FROM sales_employee
```

Они появляются в Interactive SQL следующим образом:

Emp_id	Emp_fname	Emp_lname	Dept_id
129	Philip	Chin	200
195	Marc	Dill	200
299	Rollin	Overbey	200
467	James	Klobucher	200
...	...	...	...

◆ **Переместите Филиппа Чина (Philip Chin) в отдел маркетинга.** Данное изменение представления приводит к удалению из него элемента, поскольку элемент теперь не соответствует критерию отбора представления.

```
UPDATE sales_employee
SET dept_id = 400
WHERE emp_id = 129
```

◆ **Создайте список всех служащих в отделе сбыта.** Просмотрите представление.

```
SELECT *
FROM sales_employee
```

Emp_id	emp_fname	emp_lname	dept_id
195	Marc	Dill	200
299	Rollin	Overbey	200
467	James	Klobucher	200
641	Thomas	Powell	200
...	...	...	...

При создании представления с использованием раздела WITH CHECK OPTION любой применяемый к представлению оператор UPDATE или INSERT проходит проверку с целью обеспечения соответствия новой строки условиям представления. При отсутствии соответствия данная операция вызывает ошибку и отклоняется.

Следующее измененное представление sales_employee отклоняет оператор обновления и генерирует соответствующее сообщение об ошибке:

Invalid value for column 'dept_id' in table 'employee' (Недопустимое значение для столбца 'dept_id' в таблице 'employee')

- **Создайте представление, отображающее информацию о служащих в отделе сбыта (вторая попытка).**
  В этот раз используйте WITH CHECK OPTION.

```
CREATE VIEW sales_employee

AS SELECT emp_id, emp_fname, emp_lname, dept_id
  FROM employee
  WHERE dept_id = 200
WITH CHECK OPTION
```

Параметр проверки наследуется

Если представление (скажем, "*V2*") определено для представления *sales_employee*, любое обновление или вставка в "*V2*", не соответствующие критериям проверки WITH CHECK OPTION в применении к *sales_employee*, отклоняется, даже если "*V2*" определено без параметра проверки.

Изменение представлений

Изменять представление можно с использованием как Sybase Central, так и Interactive SQL. В этом случае нельзя непосредственно переименовать существующее представление. Вместо этого необходимо создать новое представление с новым именем, скопировать в него код предыдущего, а затем удалить старое представление.

В Sybase Central для редактирования кодов представлений, процедур и функций имеется редактор кода. В Interactive SQL для изменения представления можно использовать оператор ALTER VIEW. Оператор ALTER VIEW заменяет определение представления новым определением, но сохраняет полномочия представления.

Для получения информации об изменении свойств объектов базы данных см. раздел "Установка свойств объектов базы данных" на стр. 34.

Для получения дополнительной информации о настройке полномочий см. раздел "Получение полномочий на таблицы" (Granting permissions on tables) на стр. 355 в документе "Руководство по администрированию баз данных ASA" (ASA Database Administration Guide) и раздел "Получение полномочий на представления" (Granting permissions on views) на стр. 357 в документе "Руководство по администрированию баз данных ASA" (ASA Database Administration Guide). Для получения информации о возобновлении полномочий см. раздел "Восстановление полномочий пользователей" (Revoking user permissions) на стр. 362 в документе "Руководство по администрированию баз данных ASA" (ASA Database Administration Guide).

❖ **Редактирование определения представления (Sybase Central)**

1 Откройте папку Views.

2 Щелкните правой кнопкой мыши на требуемом представлении и выберите Edit во всплывающем меню.

3 Отредактируйте код представления в редакторе кода.

4 Для выполнения кода в базе данных выберите File ➤ Save View.

❖ **Редактирование определения представления (SQL)**

1 Выполните подключение к базе данных с использованием полномочий администратора БД или как владелец представления.

2 Выполните оператор ALTER VIEW.

Пример

Переименуйте имена столбцов представления *DepartmentSize* (описанного в разделе "Создание представлений" на стр. 52 этого раздела) так, чтобы они имели более информативные имена.

```
ALTER VIEW DepartmentSize
   (Dept_ID, NumEmployees)
AS
   SELECT dept_ID, count(*)
   FROM Employee
   GROUP BY dept_ID
```

Для получения дополнительной информации см. раздел "Оператор ALTER VIEW" (ALTER VIEW statement) на стр. 226 в документе "*Справочник по SQL для ASA*" *(ASA SQL Reference Manual)*.

Удаление представлений

Удалить представление можно как в Sybase Central, так и в Interactive SQL.

❖ **Удаление представления (Sybase Central)**

1 Откройте папку Views.

2 Щелкните правой кнопкой мыши на требуемом представлении и выберите Delete во всплывающем меню.

❖ **Удаление представления (SQL)**

1 Выполните подключение к базе данных с использованием полномочий администратора БД или как владелец представления.

2 Выполните оператор DROP VIEW.

Примеры

Удалите представление с именем *DepartmentSize*.

```
DROP VIEW DepartmentSize
```

Для получения дополнительной информации см. раздел "Оператор DROP" (DROP statement) на стр. 366 в документе "*Справочник по SQL для ASA*" (*ASA SQL Reference Manual*).

Просмотр информации в представлениях

Для просмотра данных в представлениях можно воспользоваться утилитой Interactive SQL. Эта утилита позволяет выполнять запросы с целью идентификации данных, которые требуется просмотреть. Для получения дополнительной информации об использовании таких запросов см. раздел "Запросы: выбор данных в таблице" на стр. 179.

При работе в Sybase Central можно щелкнуть правой кнопкой мыши на представлении, на которое имеются полномочия, и выбрать View Data in Interactive SQL во всплывающем меню. Эта команда открывает Interactive SQL, где на закладке Results в окне Results отображается содержимое представления. Для просмотра представления Interactive SQL выполняет оператор

```
select * from <owner>, <view>.
```

Представления в системных таблицах

Вся информация о представлениях в базе данных содержится в системной таблице SYS.SYSTABLE. Информация представлена в более удобочитаемом формате в системном представлении SYS.SYSVIEWS. Для получения дополнительной информации о них см. раздел "Системная таблица SYSTABLE" (SYSTABLE system table) на стр. 617 в документе "*Справочник по SQL для ASA*" (*ASA SQL Reference Manual*) и раздел "Системное представление SYSVIEWS" (SYSVIEWS system view) на стр. 638 в документе "*Справочник по SQL для ASA*" (*ASA SQL Reference Manual*).

Для просмотра информации в этих таблицах можно воспользоваться Interactive SQL. Чтобы увидеть все столбцы в представлении SYS.SYSVIEWS, введите в окне SQL Statements следующий оператор:

```
SELECT *
FROM SYS.SYSVIEWS
```

Для получения текстового файла, содержащего определение конкретного представления, используйте оператор следующим образом:

```
SELECT viewtext FROM SYS.SYSVIEWS
   WHERE viewname = 'DepartmentSize';
OUTPUT TO viewtext.SQL
FORMAT ASCII
```

Работа с индексами

Производительность является важным фактором, который необходимо иметь в виду при проектировании и создании базы данных. Применение индексов может значительно повысить производительность операторов поиска указанной строки или заданного поднабора строк.

В каких случаях используют индексы

Индекс предоставляет информацию о порядке следования в столбцах таблицы. Индекс подобен телефонной книге, информация о людях в которой на первом уровне отсортирована по фамилиям, а на втором – по именам людей с идентичными фамилиями. Такой порядок позволяет ускорить поиск номера телефона по известной фамилии, однако не помогает при поиске номера телефона по определенному адресу. Аналогично, индекс базы данных полезен только при поиске по определенному столбцу или столбцам.

Сервер базы данных автоматически пытается использовать индекс, если подходящий индекс существует, и в случае его использования производительность возрастает.

Преимущество использования индексов растет с увеличением размера таблицы. С увеличением размера телефонной книги растет и среднее время поиска номера телефона по данному адресу, однако поиск номера телефона, скажем, К. Камински в большой телефонной книге требует ненамного больше времени, чем в маленькой.

Используйте индексы для столбцов, по которым чаще всего производится поиск

Индексы требуют дополнительного пространства и могут немного уменьшить производительность операторов, изменяющих данные в таблице, например, INSERT, UPDATE и DELETE. Однако они могут заметно повысить производительность поиска и настоятельно рекомендуются к использованию в случае необходимости частого выполнения поиска данных.

Для получения дополнительной информации о производительности см. раздел "Использование индексов" на стр. 144.

Если столбец является первичным или внешним ключом, то скорость поиска по нему уже достаточно высока, поскольку Adaptive Server Anywhere автоматически индексирует ключевые столбцы. Таким образом, ручное создание индекса ключевого столбца не требуется и, как правило, не рекомендуется. Если столбец – только часть ключа, то индекс может оказаться полезным.

В каких случаях могут быть полезны индексы первичных ключей

Индекс ключевого столбца может повысить производительность, если на первичный ключ ссылается большое количество внешних ключей.

Adaptive Server Anywhere автоматически использует индексы для улучшения производительности любого оператора базы данных всякий раз, когда это возможно. После создания индекса нет необходимости ссылаться на него. Кроме того, индекс автоматически обновляется при удалении, изменении или вставке строк.

Для получения информации об изменении свойств объектов базы

данных см. раздел "Установка свойств объектов базы данных" на стр. 34.

Создание индексов

Индексы создаются по заданной таблице. Создать индекс по представлению нельзя. Для создания индекса можно использовать как Sybase Central, так и Interactive SQL.

❖ **Создание нового индекса для заданной таблицы (Sybase Central)**

1 Откройте папку Tables.

2 Дважды щелкните на таблице.

3 Откройте папку Indexes для этой таблицы.

4 В правой области окна дважды щелкните Add Index.

5 Выполняйте указания мастера.

Все индексы появляются в папке Indexes соответствующей таблицы.

❖ **Создание нового индекса для заданной таблицы (SQL)**

1 Выполните подключение к базе данных с использованием полномочий администратора БД или как владелец таблицы, по которой создается индекс.

2 Выполните оператор CREATE INDEX.

Пример

С целью ускорить поиск по фамилиям служащих в демонстрационной базе данных можно создать индекс с именем *EmpNames* с помощью следующего оператора:

```
CREATE INDEX EmpNames
ON employee (emp_lname, emp_fname)
```

Для получения дополнительной информации см. раздел "Оператор CREATE TABLE" (CREATE TABLE statement) на стр. 280 в документе "*Справочник по SQL для ASA*" (*ASA SQL Reference Manual*) и раздел "Контроль и повышение производительности" на стр. 141.

Проверка правильности индексов

Можно произвести проверку правильности индекса для проверки того, что каждая внесенная в индекс строка действительно существует в таблице. Для индексов внешнего ключа проверка правильности также обеспечивает то, что соответствующая строка существует в первичной таблице, и значения их хэшей совпадают. Эта проверка дополняет проверку правильности, выполняемую оператором VALIDATE TABLE.

❖ **Проверка правильности индекса (Sybase Central)**

1 Выполните подключение к базе данных с использованием полномочий администратора БД или как владелец таблицы, по которой создается индекс.

2 Откройте папку Tables.

3 Дважды щелкните на таблице.

4 Откройте папку Indexes для этой таблицы.

5 Щелкните правой кнопкой мыши на индексе и выберите Validate во всплывающем меню.

❖ **Проверка правильности индекса (SQL)**

1 Выполните подключение к базе данных с использованием полномочий администратора БД или как владелец таблицы, по которой создается индекс.

2 Выполните оператор VALIDATE INDEX.

❖ **Проверка правильности индекса (командная строка)**

1 Откройте командную строку.

2 Запустите утилиту *dbvalid*.

Примеры

Проверьте правильность индекса с именем *EmployeeIndex*. Если вместо имени индекса указать имя таблицы, то будет проверен индекс первичного ключа.

```
VALIDATE INDEX EmployeeIndex
```

Проверьте правильность индекса с именем *EmployeeIndex*. Ключ **-i** указывает, что каждое заданное имя объекта является индексом.

```
dbvalid -i EmployeeIndex
```

Для получения дополнительной информации см. раздел "Оператор VALIDATE INDEX" на стр. 547 в документе "*Справочник по SQL для ASA*" (*ASA SQL Reference Manual*) и раздел "Утилита Validation" (The Validation utility) на стр. 513 в документе "*Руководство по администрированию баз данных ASA*" *(ASA Database Administration Guide)*.

Удаление индексов

Если индекс больше не требуется, можно удалить его из базы данных с помощью Sybase Central или Interactive SQL.

❖ **Удаление индекса (Sybase Central)**

1 Откройте папку Indexes для требуемой таблицы.

2 Щелкните правой кнопкой мыши на требуемом индексе и выберите Delete во всплывающем меню.

❖ **Удаление индекса (SQL)**

1 Выполните подключение к базе данных с использованием

полномочий администратора БД или как владелец таблицы, индекс которой удаляется.

2 Выполните оператор DROP INDEX.

Пример

Следующий оператор удаляет индекс из базы данных:

```
DROP INDEX EmpNames
```

Для получения дополнительной информации см. раздел "Оператор DROP" (DROP statement) на стр. 366 в документе "*Справочник по SQL для ASA*" (*ASA SQL Reference Manual*).

Индексы в системных таблицах

Вся информация об индексах в базе данных содержится в системных таблицах SYS.SYSINDEX и SYS.SYSIXCOL. Информация представлена в более удобочитаемом формате в системном представлении SYS.SYSINDEXES. Для просмотра информации в этих таблицах можно использовать Sybase Central или Interactive SQL.

Временные таблицы

Временные таблицы, локальные и глобальные, служат для одной цели – временного хранения данных. Однако различия между ними и преимущества каждого типа происходят из времени существования каждой из таблиц.

Локальная временная таблица существует только в течение одного сеанса подключения или, если она указана в составном операторе, во время выполнения составного оператора.

Определение **глобальной временной** таблицы хранится в базе данных постоянно, но строки существуют только в пределах данного подключения. После закрытия подключения к базе данных данные в глобальной временной таблице исчезают. Однако определение таблицы сохраняется в базе данных, и к ней можно обратиться при следующем открытии базы данных.

Временные таблицы хранятся во временном файле. Подобно любому другому dbspace, страницы из временного файла могут кэшироваться. Операции с временными таблицами не регистрируются в журнале транзакций.

ГЛАВА 3

Обеспечение целостности данных

Об этой главе

Встраивание средств контроля за целостностью непосредственно в базу данных – лучший способ обеспечить правильное хранение данных. В этой главе описываются средства Adaptive Server Anywhere, предназначенные для обеспечения правильности и надежности данных в базе данных.

Можно установить несколько видов контроля за целостностью. Например, можно обеспечить правильность отдельных элементов, применяя к таблицам и столбцам ограничения и условия CHECK. Установка свойств столбца путем выбора соответствующего типа данных или задания соответствующих значений по умолчанию также содействует достижению этой цели.

Операторы SQL в данной главе включают операторы CREATE TABLE и ALTER TABLE, основные формы которых были представлены в разделе "Работа с объектами баз данных" на стр. 27.

Содержание

Обзор вопросов целостности данных

Если данные имеют целостность, то данные правильны – корректны и точны, - и реляционная структура базы данных не повреждена. Средства контроля за ссылочной целостностью сохраняют и поддерживают реляционную структуру базы данных. Эти правила обеспечивают согласованность данных между таблицами.

Adaptive Server Anywhere поддерживает хранимые процедуры, которые предоставляют возможность детального контроля над вводом данных в базу данных. Кроме того, можно создавать триггеры, а также пользовательские хранимые процедуры, вызываемые автоматически при начале некоторого действия, например, обновления определенного столбца.

Для получения дополнительной информации о процедурах и триггерах см. раздел "Использование процедур, триггеров и пакетов" на стр. 495.

Причины возникновения недопустимости данных

Ниже приведено несколько примеров того, как данные в базе данных могут стать недопустимыми, если не выполнены надлежащие проверки. Возникновение ситуаций, подобных любому из этих примеров, можно предотвратить с помощью средств, описанных в этой главе.

Недопустимые данные

- Оператор неправильно ввел дату коммерческой транзакции.
- Заработок служащего становится меньше в десять раз из-за того, что оператор пропустил цифру.

Дублированные данные

- Двое различных пользователей добавляют один и тот же новый отдел (с **dept_id** 200) в таблицу отделов базы данных организации.

Непроверенные связи по внешнему ключу

- Отдел с идентификатором **dept_id** 300 закрывается, и одна запись служащего случайно остается не перенесенной ни в один новый отдел.

Средства контроля за целостностью базы данных

Чтобы обеспечить правильность данных в базе данных, следует сформировать проверки и определить критерии допустимых и недопустимых данных, а также создать правила, которым данные должны удовлетворять (так называемые бизнес-правила). Проверки и правила в совокупности являются **средствами контроля**.

Встраивание средств контроля за целостностью в базу данных

Средства контроля, которые встроены в саму базу данных, более надежны, чем средства, встроенные в клиентские приложения или устно разъясненные пользователям базы данных. Встроенные в базу данных средства контроля становятся частью определения самой базы данных, и база данных реализуется их последовательно через все приложения. Однократная установка средства контроля в базе данных налагает его на всю последующую работу с базой данных.

Напротив, средства контроля, встроенные в клиентские приложения, становятся уязвимыми каждый раз при замене ПО, и их, возможно, потребуется установить в нескольких приложениях или в нескольких местах в одном клиентском приложении.

Процесс изменения содержимого базы данных

Изменения данных в таблицах базы данных происходят при выполнении операторов SQL из клиентских приложений. Фактически только несколько операторов SQL изменяют информацию в базе данных. Возможны следующие действия:

- обновление информации в строке таблицы оператором UPDATE;
- удаление существующей строки таблицы оператором DELETE;
- Вставка новой строки в таблицу оператором INSERT.

Инструментальные средства обеспечения целостности данных

С целью поддержания целостности данных можно использовать значения по умолчанию, ограничения данных, а также средства контроля, направленные на поддержание ссылочной структуры базы данных.

Значения по умолчанию

Назначение столбцам значений по умолчанию применяется для повышения надежности определенных способов ввода данных. Например:

- Столбец может иметь значение даты по умолчанию, равное текущей дате, для записи даты выполнения транзакции с любым пользователем или действием клиентского приложения.
- Другие типы значений по умолчанию позволяют значениям столбцов автоматически приращиваться без какого-либо специального пользовательского действия, кроме ввода новой строки. Использование этой возможности обеспечивает то, что элементы (например, заказы на поставку) имеют уникальные последовательные номера.

Для получения дополнительной информации о этом и других значениях столбца по умолчанию см. раздел "Использование значений столбца по умолчанию" на стр. 70.

Средства контроля

К данным в отдельных столбцах или таблицах можно применить несколько типов средств контроля; например:

- Ограничение NOT NULL не позволяет внести в столбец пустой элемент.
- Условие CHECK, назначенное столбцу, обеспечивает соответствие каждого элемента в столбце определенному условию. Например, можно обеспечить то, что элементы столбца заработной платы будут находиться в пределах указанного диапазона, и таким образом защитить систему от пользовательской ошибки при вводе новых значений.

- Условия CHECK могут быть наложены на связанные значения в разных столбцах. Например, можно обеспечить то, что элемент *date_returned* (дата возврата) в базе данных библиотеки будет содержать более позднюю дату, чем элемент *date_borrowed* (дата получения).
- С помощью триггеров можно реализовать более сложные условия CHECK. Для получения дополнительной информации о триггерах см. раздел "Использование процедур, триггеров и пакетов" на стр. 495.

Кроме того, ограничения столбца могут быть унаследованы из доменов. Для получения дополнительной информации об этих и других ограничениях таблиц и столбцов см. раздел "Использование ограничений таблиц и столбцов" на стр. 75.

Целостность объектов и ссылочная целостность

Связи, определенные первичными и внешними ключами, объединяют информацию в таблицах реляционной базы данных. Эти связи следует встраивать непосредственно в базу данных. Нижеприведенные правила целостности поддерживают структуру базы данных:

- **Целостность объекта.** Отслеживание первичных ключей. Обеспечивается то, что каждая строка данной таблицы может быть однозначно определена первичным ключом, который имеет непустое (NOT NULL) значение.
- **Ссылочная целостность.** Отслеживание внешних ключей, определяющих связи между таблицами. Обеспечивается то, что все значения внешнего ключа либо совпадают с соответствующими значениями в соответствующем первичном ключе, либо содержат значение NULL в случае, если их установками разрешено такое значение.

Для получения дополнительной информации об обеспечении ссылочной целостности см. раздел "Обеспечение целостности объектов и ссылочной целостности" на стр. 82. Для получения дополнительной информации о проектировании связей по первичному и внешнему ключам см. раздел "Проектирование базы данных" на стр. 3.

Использование триггеров для поддержки целостности

Для обеспечения целостности данных можно также использовать триггеры. **Триггер** – это процедура, хранящаяся в базе данных и выполняемая автоматически каждый раз при изменении информации в определенной таблице. Триггеры представляют собой мощный механизм обеспечения надежности данных, предоставляемый администраторам и разработчикам базы данных.

Для получения дополнительной информации о триггерах см. раздел "Использование процедур, триггеров и пакетов" на стр. 495.

Операторы SQL для реализации средств контроля за целостностью

Следующие операторы SQL реализуют функции контроля за целостностью:

- **Оператор CREATE TABLE.** Этот оператор реализует функции контроля за целостностью во время создания базы данных.
- **Оператор ALTER TABLE.** Этот оператор добавляет средства контроля за целостностью к существующей базе данных или изменяет такие средства для существующей базы данных.
- **Оператор CREATE TRIGGER.** Этот оператор создает триггеры, которые реализуют более сложные бизнес-правила.

Для получения дополнительной информации о синтаксисе этих операторов см. раздел "Операторы SQL" (SQL Statements) на стр. 189 в документе "*Справочник по SQL для ASA*" (*ASA SQL Reference Manual*).

Использование значений столбца по умолчанию

Значения столбца по умолчанию автоматически назначают указанное значение определенным столбцам всякий раз при вводе новой строки в таблицу базы данных. Имеющееся значение по умолчанию не требует никаких действий со стороны клиентского приложения, однако если клиентское приложение все же определяет значение для столбца, то новое значение заменяет значение столбца по умолчанию.

Значения столбца по умолчанию могут быстро и в автоматическом режиме заполнить столбцы информацией, такой как дата или время в момент вставки строки, или код пользователя, вводящего информацию. Использование значений столбца по умолчанию способствует сохранению целостности данных, но не обеспечивает ее. Клиентские приложения всегда могут подставить другие значения вместо значений по умолчанию.

Поддерживаемые значения по умолчанию

SQL поддерживает следующие значения по умолчанию:

- строка, указанная в операторе CREATE TABLE или ALTER TABLE;
- число, указанное в операторе CREATE TABLE или ALTER TABLE;
- автоматически приращиваемое число: на единицу большее, чем предыдущее самое высокое значение в столбце;
- текущая дата, время или штамп времени;
- текущий код пользователя базы данных;
- значение NULL;
- постоянное выражение, если оно не ссылается на объекты базы данных.

Создание значений столбца по умолчанию

Для создания значений столбца по умолчанию при создании таблицы используется оператор CREATE TABLE, а для добавления таких значений в дальнейшем – оператор ALTER TABLE.

Пример

Следующий оператор добавляет условие к существующему столбцу с именем **id** в таблице **sales_order**, после чего значения в нем автоматически приращиваются (если клиентское приложение не предоставляет другого значения):

```
ALTER TABLE sales_order
MODIFY id DEFAULT AUTOINCREMENT
```

Таким же образом назначается любое другое значение по умолчанию. Для получения дополнительной информации см. раздел "Оператор ALTER TABLE" (ALTER TABLE statement) на стр. 219 в документе "*Справочник по SQL для ASA*" (*ASA SQL Reference Manual*) и раздел "Оператор CREATE TABLE" (CREATE TABLE statement) на стр. 321 в документе "*Справочник по SQL для ASA*" (*ASA SQL Reference Manual*).

Изменение и удаление значений столбца по умолчанию

Изменить или удалить значения столбца по умолчанию можно, используя ту же форму оператора ALTER TABLE, который использовался при создании этих значений. Следующий оператор меняет значение по умолчанию столбца с именем **order_date** с его текущей установки на CURRENT DATE (Текущая дата):

```
ALTER TABLE sales_order
MODIFY order_date DEFAULT CURRENT DATE
```

Можно удалить значения столбца по умолчанию, изменив их на NULL. Следующий оператор удаляет значение по умолчанию из столбца **order_date**:

```
ALTER TABLE sales_order
MODIFY order_date DEFAULT NULL
```

Работа со значениями столбца по умолчанию в Sybase Central

В Sybase Central можно добавлять, изменять и удалять значения столбца по умолчанию, используя закладку Value окна свойств столбца.

❖ **Отображение окна свойств столбца**

1. Выполните подключение к базе данных.
2. Откройте папку Tables этой базы данных.
3. Дважды щелкните на таблице, содержащей столбец, который требуется изменить.
4. Откройте папку Columns этой таблицы.
5. Щелкните правой кнопкой мыши на столбце и выберите Properties во всплывающем меню.

Текущая дата и время по умолчанию

В столбцах с типами данных DATE (Дата), TIME (Время) или TIMESTAMP (Штамп времени) можно по умолчанию использовать текущую дату, текущее время или текущий штамп времени. Выбранные значения по умолчанию должны быть совместимы с типом данных столбца.

Полезные примеры установки текущей даты по умолчанию

Текущая дата по умолчанию может быть полезна при записи:

- дат телефонных звонков в базе данных контактов;
- дат заказов в базе данных продаж;
- дата получения книги посетителем в базе данных библиотеки.

Текущий штамп времени

Текущий штамп времени подобен текущей дате по умолчанию, но обеспечивает большую точность. Например, пользователь приложения управления контактами может иметь несколько контактов с одним клиентом в один день: текущий штамп времени по умолчанию помогает различать эти контакты.

Поскольку штамп времени делает запись даты и времени с точностью до миллионных долей секунды, он также может оказаться полезным в тех случаях, когда для базы данных важна последовательность событий.

Для получения дополнительной информации о штампах времени, а также о значениях времени и даты см. раздел "Типы данных SQL" на стр. 51 в документе "*Справочник по SQL для ASA*" (*ASA SQL Reference Manual*).

Код пользователя по умолчанию

Назначение столбцу пользователя по умолчанию (DEFAULT USER) – это простой и надежный способ идентифицировать пользователя, создающего элемент в базе данных. Эта информация может требоваться, например, когда торговые представители работают над поручениями.

Встраивание кода пользователя по умолчанию в первичный ключ таблицы полезно при наличии подключаемых время от времени пользователей, так как это позволяет предотвращать конфликты во время обновления информации. Такой пользователь может сделать копию таблиц, необходимых для работы, на своем портативном компьютере и внести изменения, не будучи подключенным к многопользовательской базе данных, и затем по возвращении передать серверу журнал транзакций.

Значение по умолчанию AUTOINCREMENT

Значение по умолчанию AUTOINCREMENT (Автоприращение) используется для числовых полей данных, где величина самого числа может не иметь значения. Эта функция назначает каждой новой строке значение на единицу большее, чем предыдущее самое высокое значение в столбце. Столбцы со значением AUTOINCREMENT можно использовать для записи номеров заказов на поставку, идентификации звонков в службу поддержки клиентов и других элементов, где требуется идентификационный номер.

Столбцы с автоприращением – это обычно столбцы первичного ключа или столбцы, ограничения которых требуют наличия только уникальных значений (см. раздел "Обеспечение целостности объектов" на стр. 82). Например, автоприращение по умолчанию эффективно, если столбец является первым столбцом индекса, поскольку сервер использует индекс или определение ключа для поиска наибольшего значения.

Хотя использование автоприращения по умолчанию возможно и в других случаях, это может неблагоприятно повлиять на производительность базы данных. Например, в тех случаях, когда следующее значение для каждого столбца хранится как целое число (4 байта), использование значений выше 2^{31}-1, а также больших двойных или числовых значений может вызвать перенос к отрицательным значениям.

Получить самое последнее значение, вставленное в столбец функцией автоприращения, можно с помощью глобальной переменной *@@identity*. Для получения дополнительной информации см. раздел "Глобальная переменная @@identity" (@@identity global variable) на стр. 44 в документе "*Справочник по SQL для ASA*" (*ASA SQL Reference Manual*).

Автоприращение и отрицательные числа

Функция автоприращения предназначена для работы с положительными целыми числами.

Начальное значение автоприращения при создании таблицы устанавливается в 0. Если в столбец вставляют отрицательные значения в явном виде, то нуль остается наивысшей назначенной величиной. После вставки без указания какого-либо значения функция автоприращения генерирует значение 1, после чего любые другие сгенерированные значения обязательно будут положительными. Если бы были вставлены только отрицательные значения, и база данных была бы остановлена и перезапущена, то произошло бы повторное вычисление самого высокого значения в столбце, и после этого генерировались бы отрицательные значения.

В приложениях UltraLite значение автоприращения не устанавливается в 0 при создании таблицы, и в случае использования в столбце соответствующего типа данных функция автоприращения генерирует отрицательные числа.

С целью предотвращения использования отрицательных значений столбцы с автоприращением следует настроить на использование беззнаковых чисел (unsigned).

Автоприращение и столбец IDENTITY

☞ Столбец со значением по умолчанию AUTOINCREMENT рассматривается в приложениях Transact-SQL как столбец IDENTITY. Для получения информации о столбцах IDENTITY см. раздел "Специальный столбец IDENTITY" на стр. 387.

Значение по умолчанию NULL

Для столбцов, в которых разрешено использование значений NULL, определение значения по умолчанию NULL означает то же, что и отсутствие определения значения по умолчанию. Если вставляющий строку клиент не указывает значение явно, то строка автоматически получает значение NULL.

Использовать значения по умолчанию NULL можно в тех случаях, когда информация для некоторых столбцов является необязательной или не всегда доступной, или когда не требуется правильность данных в базе данных.

☞ Для получения дополнительной информации о значении NULL см. раздел "Значение NULL" (NULL value) на стр. 47 в документе "*Справочник по SQL для ASA*" *(ASA SQL Reference Manual)*.

Строковые и численные значения по умолчанию

Значение по умолчанию может быть специальной строкой или числом, если столбец поддерживает строковый или числовой тип данных. Необходимо обеспечить возможность преобразования значения по умолчанию в значение, имеющее тот же тип, что и данные в столбце.

Строки и числа по умолчанию полезны в тех случаях, когда имеется типичный элемент для данного столбца. Например, если организация имеет два офиса, штаб в **city_1** (город 1) и небольшой офис в **city_2** (город 2), для упрощения ввода данных можно задать элемент по умолчанию для столбца местоположения как **city_1**.

Постоянные выражения по умолчанию

Значением по умолчанию может быть постоянное выражение, если оно не ссылается на объекты базы данных. Постоянные выражения позволяют вносить в значения столбца по умолчанию такие элементы, как *дата через пятнадцать дней*, которые вводятся как

```
... DEFAULT ( dateadd( day, 15, getdate() ) )
```

Использование ограничений таблиц и столбцов

Наряду с основной структурой таблицы (номер, имя и тип данных столбца, имя и местоположение таблицы), операторы CREATE TABLE и ALTER TABLE могут определять множество различных атрибутов таблицы, которые позволяют контролировать целостность данных.

> ***Предостережение***
>
> *Изменение таблиц может повлиять на работу других пользователей базы данных. Хотя выполнить оператор ALTER TABLE можно и в то время, когда активны другие подключения, в то же время его нельзя выполнить, если какое-либо подключение использует таблицу, подлежащую изменению. В случае больших таблиц выполнение оператора ALTER TABLE занимает довольно много времени, и во время его работы любые другие запросы к изменяемой таблице запрещены.*

В этом разделе описывается использование ограничений в целях поддержки обеспечения точности данных в таблице.

Использование условий CHECK для столбцов

Условие CHECK используется для того, чтобы значения в столбце удовлетворяли некоторому определенному критерию или правилу. Например, эти правила или критерии могут просто требоваться для того, чтобы данные были целесообразными, или это могут быть более жесткие правила, отражающие организационные нормы и процедуры.

Условия CHECK для отдельных значений столбцов используются в тех случаях, когда для столбца допустима только ограниченная область значений.

Пример 1

- Можно применить определенное требование к форматированию. Например, если таблица имеет столбец для телефонных номеров, требуется обеспечить то, что пользователи вводят их одним и тем же способом. Для североамериканских номеров телефона это может быть ограничение такого вида:

```
ALTER TABLE customer
MODIFY phone
CHECK ( phone LIKE '(___) ___-____' )
```

Пример 2

- Можно обеспечить соответствие элемента одному из нескольких значений. Например, чтобы обеспечить то, что столбец **city** (город) содержит только один из нескольких разрешенных номеров городов (скажем, тех, где организация имеет офисы), можно использовать ограничение следующего вида:

```
ALTER TABLE office
MODIFY city
CHECK ( city IN ( 'city_1', 'city_2', 'city_3' ) )
```

- По умолчанию в строковых сравнениях регистр не учитывается, если база данных не создана явно как база данных с учетом регистра.

Пример 3

- Можно обеспечить нахождение даты или числа в определенной области значений. Например, можно обеспечить то, что значения в столбце **start_date** (начальная дата) таблицы служащих находились между датой создания организации и текущей датой, используя следующее ограничение:

```
ALTER TABLE employee
MODIFY start_date
CHECK ( start_date BETWEEN '1983/06/27'
                   AND CURRENT DATE )
```

- Можно использовать несколько форматов даты. Формат ГГГГ/ММ/ДД в этом примере удобен тем, что он всегда будет распознан, независимо от текущих параметров настройки.

Проверка CHECK завершается неудачно, только если условие возвращает значение FALSE. Если условие возвращает значение UNKNOWN, то изменение разрешается.

Условия CHECK для столбцов в доменах

Можно привязывать условия CHECK к доменам. Столбцы, определенные для этих типов данных, наследуют условия CHECK. Условие CHECK, явно указанное для столбца, имеет приоритет над условием в домене.

Любой определенный столбец, использующий тип данных **posint**, принимает только положительные целых числа, если сам столбец не имеет явно указанного условия CHECK. В нижеприведенном примере домен принимает только положительные целые числа. Так как любая переменная с префиксом @ при применении условия CHECK заменяется именем столбца, любое имя переменной с префиксом @ может использоваться вместо **@col**.

```
CREATE DATATYPE posint INT
CHECK ( @col > 0 )
```

Оператор ALTER TABLE с разделом DELETE CHECK удаляет все условия CHECK из определения таблицы, включая унаследованные из доменов.

Для получения дополнительной информации о доменах см. раздел "Домены" (Domains) на стр. 74 в документе "*Справочник по SQL для ASA*" (*ASA SQL Reference Manual*).

Работа с ограничениями таблиц и столбцов в Sybase Central

В Sybase Central добавление, изменение и удаление ограничений столбцов выполняется на закладке Constraints окна свойств столбца или таблицы.

❖ **Управление ограничениями таблицы**

1. Откройте папку Tables.
2. Щелкните правой кнопкой мыши на таблице и выберите Properties во всплывающем меню.
3. Щелкните на закладке Constraints.
4. Выполните соответствующие изменения.

❖ **Управление ограничениями столбца**

1. Откройте папку Tables и дважды щелкните на таблице, чтобы открыть ее.
2. Откройте папку Columns.
3. Щелкните правой кнопкой мыши на столбце и выберите Properties во всплывающем меню.
4. Щелкните на закладке Constraints.
5. Выполните соответствующие изменения.

Использование условий CHECK в таблицах

Наложенное на таблицу условие CHECK обычно обеспечивает то, что два значения в добавленной или измененной строке имеют надлежащую связь друг с другом. Условия CHECK для столбца, напротив, хранятся индивидуально в системных таблицах, и их можно индивидуально заменить или удалить. Поскольку такое поведение более гибко, рекомендуется использовать условия CHECK для отдельных столбцов везде, где это возможно.

Например, в базе данных библиотеки элемент **date_borrowed** (дата получения) должен быть меньше, чем **date_returned** (дата возврата).

```
ALTER TABLE loan
ADD CHECK(date_returned >= date_borrowed)
```

Изменение и удаление условий CHECK

Есть несколько способов изменить существующий набор условий CHECK для таблицы.

- Можно добавить новое условие CHECK к таблице или к отдельному столбцу, как описано выше.

- Можно удалить условие CHECK для столбца, установив его в NULL. Например, нижеприведенный оператор удаляет условие CHECK столбца **phone** в таблице **customer**:

```
ALTER TABLE customer
MODIFY phone CHECK NULL
```

- Можно заменить условие CHECK для столбца тем же самым способом, каким выполнялось бы добавление этого условия. Например, нижеприведенный оператор добавляет или заменяет условие CHECK в столбце **phone** в таблице **customer**:

```
ALTER TABLE customer
MODIFY phone
CHECK ( phone LIKE '___-___-____' )
```

- Имеется два способа изменить условие CHECK, определенное для таблицы (в отличие от условия CHECK, определенного для столбца):
  - Можно добавить новое условие CHECK с помощью оператора ALTER TABLE с разделом ограничения таблицы ADD.
  - Можно удалить все существующие условия CHECK (включая условия CHECK столбца и условия CHECK, унаследованные из доменов), используя конструкцию ALTER TABLE DELETE CHECK, и затем добавить новые условия CHECK.

Использование оператора ALTER TABLE с разделом DELETE CHECK:

```
ALTER TABLE table_name
DELETE CHECK
```

Удаление столбца из таблицы не удаляет связанные со столбцом условия CHECK, хранящиеся в ограничении таблицы. Отсутствие действия удаления ограничения приводит к сообщению об ошибке "column not found" ("столбец не найден") после любой попытки вставить или даже просто запросить данные из таблицы.

Проверка CHECK заканчивается неудачно, только если условие возвращает значение FALSE. Если условие возвращает значение UNKNOWN, то изменение разрешается.

Использование доменов

Домен – это определяемый пользователем тип данных, который вместе с другими атрибутами может ограничивать область приемлемых значений или предоставлять значения по умолчанию. Домен расширяет один из встроенных типов данных. Область допустимых значений обычно определяется ограничением на формат данных. Кроме того, домен может определять значение по умолчанию и разрешать или запрещать пустые значения.

Определение собственных доменов может потребоваться по ряду причин.

- Можно предотвратить множество распространенных ошибок, если будет отсутствовать возможность ввода недопустимых значений. Помещенное в домен ограничение обеспечивает то, что все столбцы и переменные будут хранить значения только в нужной области или формате. Например, один из типов данных позволяет обеспечить то, что все номера кредитной карточки, введенные в базу данных, имеют правильное количество цифр.
- Использование доменов может сделать приложения и структуру базы данных намного более понятными.
- Создание доменов повышает удобство пользования базой данных. Например, можно потребовать, чтобы все идентификаторы таблицы были положительными целыми числами, которые по умолчанию подлежат автоприращению. Это ограничение можно осуществить вводом соответствующих ограничений и значений по умолчанию каждый раз при определении новой таблицы, но гораздо удобнее определить новый домен и просто указать, что идентификатор может принимать значения только из указанного домена.

Для получения дополнительной информации о доменах см. раздел "Типы данных SQL" (SQL Data Types) на стр. 51 в документе "*Справочник по SQL для ASA*" (*ASA SQL Reference Manual*).

Создание доменов (Sybase Central)

Для создания домена или назначения его столбцу можно использовать Sybase Central.

❖ Создание нового домена (Sybase Central)

1. Откройте папку Domains.
2. В правой области окна дважды щелкните Add Domain.
3. Выполняйте указания мастера.

Все домены появляются в папке Domains в Sybase Central.

❖ Назначение доменов столбцам (Sybase Central)

1. Откройте папку Columns для требуемой таблицы.
2. Щелкните правой кнопкой мыши на требуемом столбце и выберите Properties во всплывающем меню.
3. На закладке Data Type окна свойств столбца назначьте домен.

Создание доменов (SQL)

Для создания и определения доменов используется оператор CREATE DOMAIN.

❖ **Создание нового домена (SQL)**

1 Выполните подключение к базе данных.

2 Выполните оператор CREATE DOMAIN.

Пример 1: простые домены

Некоторые столбцы в базе данных должны использоваться для хранения имен, а другие – для хранения адресов. Тогда можно определить следующие домены:

```
CREATE DOMAIN persons_name CHAR(30)
CREATE DOMAIN street_address CHAR(35)
```

Определив эти домены, можно использовать их так же, как и встроенные типы данных. Например, можно использовать эти определения, чтобы определить таблицы следующим образом.

```
CREATE TABLE customer (
   id INT  DEFAULT AUTOINCREMENT  PRIMARY KEY
   name persons_name
   address street_address
)
```

Пример 2: значения по умолчанию, ограничения на формат данных и идентификаторы

В вышеприведенном примере первичный ключ таблицы определен как имеющий целый тип (integer). Подобные идентификаторы могут потребоваться для многих таблиц. Вместо того, чтобы определять, что они являются целыми числами, намного удобнее создать домен идентификаторов для использования в этих приложениях.

При создании домена можно определить значение по умолчанию и предоставить ограничение на формат данных с целью обеспечить то, что никакие недопустимые значения не будут введены ни в один столбец этого типа.

Идентификаторы таблиц обычно являются целыми числами. Для уникальных идентификаторов используются положительные целые числа. Поскольку такие идентификаторы, скорее всего, будут использоваться во многих таблицах, можно определить следующий домен.

```
CREATE DOMAIN identifier INT
DEFAULT AUTOINCREMENT
CHECK ( @col > 0 )
```

Это ограничение на формат данных использует переменную @col. Используя это определение, можно переписать определение таблицы клиентов, приведенной выше.

```
CREATE TABLE customer (
   id identifier PRIMARY KEY
   name persons_name
   address street_address
)
```

Пример 3: встроенные домены

Adaptive Server Anywhere поставляется с некоторыми встроенными доменами. Эти встроенные домены можно использовать так же, как и домен, созданный самостоятельно. Например, следующий денежно-кредитный домен уже создан заранее.

```
CREATE DOMAIN MONEY NUMERIC(19,4)
NULL
```

Для получения дополнительной информации см. раздел "Оператор CREATE DOMAIN" (CREATE DOMAIN statement) на стр. 265 в документе "*Справочник по SQL для ASA*" *(ASA SQL Reference Manual)*.

Удаление доменов

Для удаления домена можно воспользоваться как Sybase Central, так и оператором DROP DOMAIN.

Удалить домен может только администратор БД или пользователь, который создал этот домен. Кроме того, домен не может быть удален, если какая-либо переменная или столбец в базе данных является экземпляром домена. В этом случае перед удалением домена требуется удалить все столбцы или переменные этого типа.

❖ Удаление домена (Sybase Central)

1. Откройте папку Domains.
2. Щелкните правой кнопкой мыши на требуемом домене и выберите Delete во всплывающем меню.

❖ Удаление домена (SQL)

1. Выполните подключение к базе данных.
2. Выполните оператор DROP DOMAIN.

Пример

Следующий оператор удаляет домен *customer_name*.

```
DROP DOMAIN customer_name
```

Для получения дополнительной информации см. раздел "Оператор DROP" (DROP statement) на стр. 366 в документе "*Справочник по SQL для ASA*" (*ASA SQL Reference Manual*).

Обеспечение целостности объектов и ссылочной целостности

Реляционная структура базы данных позволяет персональному серверу идентифицировать информацию в пределах базы данных и обеспечивает то, что все строки в каждой таблице поддерживают связи между таблицами (определенными в структуре базы данных).

Обеспечение целостности объектов

При вставке или удалении строки пользователем сервер базы данных проверяет допустимость первичного ключа для таблицы, а также то, что каждая строка в таблице однозначно определена первичным ключом.

Пример 1

Таблица **employee** в демонстрационной базе данных использует код служащего как первичный ключ. При добавлении нового служащего в таблицу сервер базы данных проверяет, что новое значение кода служащего уникально и не является NULL.

Пример 2

Таблица **sales_order_items** в демонстрационной базе данных использует два столбца для определения первичного ключа.

Эта таблица содержит информацию о заказах. Один столбец содержит **id,** определяющий заказ, но могут существовать и несколько элементов в каждом заказе, поэтому этот столбец сам по себе не может быть первичным ключом. Дополнительный столбец **line_id** идентифицирует строку, соответствующую элементу. Столбцы **id** и **line_id**, взятые вместе, однозначно определяют элемент и формируют первичный ключ.

Нарушение целостности объектов клиентским приложением

Целостность объектов подразумевает уникальность значений первичных ключей в пределах таблицы и отсутствие в ней значений NULL. Если клиентское приложение пытается вставить или обновить значение первичного ключа, передавая не уникальные значения, то это может нарушить целостность объекта. Нарушение целостности объекта препятствует добавлению новой информации в базу данных, поэтому клиентскому приложению посылается сообщение об ошибке.

Разработчик приложения должен решить, как представить эту информацию пользователю и позволить ему предпринять соответствующие действия. Соответствующие действия, как правило, заключаются в предложении пользователю предоставить другое, уникальное значение для первичного ключа.

Обеспечение целостности объектов посредством первичных ключей

Как только будет определен первичный ключ для каждой таблицы, обеспечение целостности объекта не потребует никаких дальнейших действий ни от разработчика клиентского приложения, ни от администратора базы данных.

Владельцы таблицы определяют первичный ключ для таблицы при ее создании. Если они изменяют структуру таблицы позднее, они могут также переопределить первичный ключ.

Некоторые системы разработки приложений и средства проектирования баз данных позволяют создавать и изменять таблицы базы данных. Если используется такая система, то, скорее всего, не придется вводить операторы CREATE TABLE или ALTER TABLE явно: приложение может сгенерировать оператор непосредственно из предоставленной информации.

Для получения дополнительной информации о создании первичных ключей см. раздел "Управление первичными ключами" на стр. 45. Синтаксис оператора CREATE TABLE подробно описан в разделе "Оператор CREATE TABLE" (CREATE TABLE statement) на стр. 321 в документе "*Справочник по SQL для ASA*" (*ASA SQL Reference Manual*). Для получения информации об изменении структуры таблицы см. раздел "Оператор ALTER TABLE" (ALTER TABLE statement) на стр. 219 в документе "*Справочник по SQL для ASA*" (*ASA SQL Reference Manual*).

Обеспечение ссылочной целостности

Внешний ключ (составленный из определенного столбца или комбинации столбцов) связывает информацию в одной таблице (**внешняя** таблица) с информацией в другой (**связанной** или **первичной**) таблицей. Чтобы связи внешнего ключа были правильными, элементы во внешнем ключе должны соответствовать значениям первичного ключа строки в связанной таблице. Иногда вместо первичного ключа можно использовать ссылку на другую уникальную комбинацию столбцов.

Пример 1

Демонстрационная база данных содержит таблицу служащих и таблицу отделов. Первичный ключ для таблицы служащих - код служащего, а первичный ключ для таблицы отделов - код отдела. В таблице служащих код отдела называется **внешним ключом** для таблицы отделов, так как каждый код отдела в таблице служащих соответствует определенному коду отдела в таблице отделов.

Связи по внешнему ключу являются связями "один ко многим". Несколько элементов в таблице служащих имеют один и тот же код отдела, но код отдела является первичным ключом для таблицы отделов и поэтому уникален. Если бы внешний ключ мог ссылаться на столбец в таблице отделов, содержащей повторяющиеся записи или элементы со значением NULL, то не было бы никакой возможности узнать, на какую строку в таблице отделов должна быть ссылка. Это обязательный внешний ключ.

Пример 2

Предположим, что база данных также содержит таблицу офисов с местоположениями офисов.

Таблица служащих могла бы иметь внешний ключ для таблицы офисов, которая указывает, в каком городе находится офис того или иного служащего. Разработчик базы данных может оставить местоположение офиса неопределенным во время найма служащего, например, потому что он еще не был привязан к определенному офису или не работал в офисе. В этом случае для внешнего ключа разрешены значения NULL, и данных ключ является необязательным.

Обеспечение ссылочной целостности посредством внешних ключей

Подобно первичным ключам, для создания внешних ключей используются операторы CREATE TABLE и ALTER TABLE. После создания внешнего ключа столбец или столбцы в ключе могут содержать только те значения, которые присутствуют как значения первичного ключа в таблице, связанной с внешним ключом.

☞ Для получения дополнительной информации о создании внешних ключей см. раздел "Управление первичными ключами" на стр. 45.

Нарушение ссылочной целостности

Ссылочная целостность базы данных может быть нарушена в результате следующего:

- изменения или удаления значения первичного ключа; все внешние ключи, связанные с этим первичным ключом, становятся недопустимыми;
- добавления новой строки во внешнюю таблицу и ввода значения для внешнего ключа, не имеющего соответствующего значения в первичном ключе; данные в базе данных становятся недопустимыми.

Adaptive Server Anywhere предоставляет защиту против обоих типов нарушения целостности.

Нарушение ссылочной целостности клиентским приложением

Если клиентское приложение обновляет или удаляет значение первичного ключа в таблице, и если внешний ключ связан с этим значением первичного ключа в другом месте в базе данных, то появляется опасность нарушения ссылочной целостности.

Пример

Если сервер разрешил обновление или удаление первичного ключа и не сделал никаких изменений связанных с ним внешних ключей, то ссылка по внешнему ключу становится недопустимой. Любая попытка использовать ссылку по внешнему ключу, например, в операторе SELECT с использованием раздела KEY JOIN, будет неудачной, поскольку никакого соответствующего значения в связанной таблице не существует.

В то время как Adaptive Server Anywhere поддерживает защиту целостности объекта прямым способом, просто отказываясь вводить данные и возвращая сообщение об ошибках, потенциальные нарушения целостности связей становятся более сложными. Имеется несколько опций (так называемые действия ссылочной целостности), предназначенных для поддержки ссылочной целостности.

Действия ссылочной целостности

Поддержание ссылочной целостности при изменении или удалении первичного ключа, на который имеются ссылки, может быть достигнуто вводом запрета на изменение или удаление. Часто, однако, с целью поддержки ссылочной целостности можно также выполнить специальное действие по каждому внешнему ключу. Операторы CREATE TABLE и ALTER TABLE позволяют администраторам базы данных и владельцам таблиц определять, какой действие произвести над внешними ключами, связанными с измененным первичным ключом, при возникновении нарушения.

Можно определить каждое из доступных действий ссылочной целостности отдельно при изменении и удалении первичного ключа:

- **RESTRICT.** Генерирует ошибку и предотвращает изменение при попытках изменения значения первичного ключа. Это действие ссылочной целостности, выполняемое по умолчанию.
- **SET NULL.** Устанавливает все внешние ключи, связанные с измененным первичным ключом, в NULL.
- **SET DEFAULT.** Устанавливает все внешние ключи, связанные с измененным первичным ключом, в значение столбца по умолчанию (определенное в определении таблицы).
- **CASCADE.** При использовании с ON UPDATE это действие обновляет все внешние ключи, связанные с измененным первичным ключом, в новое значение. При использовании с ON DELETE это действие удаляет все строки, содержащие внешние ключи, связанные с удаленным первичным ключом.

Действия ссылочной целостности реализуются посредством системных триггеров. Триггер, определенный в первичной таблице, выполняется с использованием полномочий владельца *вторичной* таблицы. Это поведение означает, что каскадные операции могут иметь место между таблицами с различными владельцами, без необходимости предоставления дополнительных полномочий.

Проверка ссылочной целостности

Для внешних ключей, определенных для операций запрета, которые могли бы нарушить ссылочную целостность, во время выполнения оператора производятся проверки по умолчанию. Если задано выражение CHECK ON COMMIT, то проверки происходят только при совершении транзакций.

Использование параметров базы данных для управления

Установка параметра базы данных WAIT_FOR_COMMIT управляет поведением системы, если для запрета операций, которые могли бы нарушить ссылочную целостность, определен внешний ключ. Раздел CHECK ON COMMIT имеет приоритет над этим параметром.

Если WAIT_FOR_COMMIT по умолчанию установлен в OFF, то операции, которые могли бы сделать базу данных некорректной, выполняться не будут. Например, попытка удалить отдел, в котором все еще имеются служащие, отклоняется. Следующий оператор вызывает ошибку "primary key for row in table 'department' is referenced in another table" ("первичный ключ строки в таблице 'department' связан с другой таблицей"):

```
DELETE FROM department
WHERE dept_id = 200
```

Установка WAIT_FOR_COMMIT в ON откладывает проверку ссылочной целостности до выполнения подтверждения. Если база данных несогласованна, это приводит к отказу в подтверждении и появлению сообщения об ошибке. В этом режиме пользователь базы данных может удалить отдел со служащими в нем, однако пользователь не может подтвердить изменение базы данных, пока не произойдет:

- удаление или переназначение служащих, принадлежащих этому отделу;
- повторное выполнение условия поиска в операторе SELECT для выбора строк, в которых имеется нарушение ссылочной целостности;
- вставка **dept_id** строку обратно в таблицу **department**;
- откат транзакции для отмены операции DELETE.

Правила целостности в системных таблицах

Вся информация о проверках целостности базы данных и правилах хранится в следующих системных таблицах:

Системная таблица	Описание
SYS.SYSTABLE	Столбец **view_def** в SYS.SYSTABLE содержит условия CHECK. Для представлений **view_def** содержит команду CREATE VIEW, с помощью которой было создано представление. Можно проверить, является ли конкретная таблица базовой таблицей или представлением, посмотрев на столбец **table_type**, содержа-щий BASE или VIEW (соответственно).
SYS.SYSTRIGGER	SYS.SYSTRIGGER содержит действия ссылочной целостности. Столбец **referential_action** содержит один символ, указывающий, является ли действие каскадным (C), удалением (D), заданием NULL (N) или ограничением (R). Столбец **event** содержит один символ, определяю-щий событие, приведшее к действию: удаление (D), вставка (I), изменение (U) или обновление списка-столбцов (C). Столбец **trigger_time** показывает, происходит ли действие после (A) или до (B) события вызова.
SYS.SYSFOREIGNKEYS	Это представление показывает информа-цию о внешних ключах из двух таблиц - SYS.SYSFOREIGNKEY и SYS.SYSFKCOL в более удобочитаемом формате.
SYS.SYSCOLUMNS	Это представление показывает информа-цию из таблицы SYS.SYSCOLUMN в более удобочитаемом формате. Оно включает настройки по умолчанию и информацию о первичных ключах для столбцов.

Для получения дополнительной информации о содержании каждой системной таблицы см. раздел "Системные таблицы" (System Tables) на стр. 557 в документе "*Справочник по SQL для ASA*" (*ASA SQL Reference Manual*).

Для просмотра этих таблиц и представлений можно использовать как Sybase Central, так и Interactive SQL.

ГЛАВА 4

Использование транзакций и уровней изоляции

Об этой главе

Можно группировать операторы SQL в транзакции, которые обладают следующим свойством: либо выполняются все операторы, либо ни один оператор не выполняется. Нужно проектировать каждую транзакцию для выполнения задачи, изменяющей базу данных от одного состояния согласованности к другому.

В этой главе описываются транзакции и их использование в приложениях. Также рассматривается то, как в Adaptive Server Anywhere можно установить уровни изоляции для ограничения влияния параллельных транзакций.

Содержание

Введение в транзакции

Для обеспечения целостности данных важно, чтобы можно было идентифицировать состояния, в которых информация в базе данных является **согласованной**. Понятие согласованности лучше всего иллюстрируется при помощи следующего примера:

Пример согласованности

Предположим, что база данных используется для обработки финансовых счетов, и нужно переводить денежные средства со счета одного клиента на другой. База данных находится в состоянии согласованности и до, и после того, как средства переводятся; но она находится в состоянии несогласованности после того, как сняли денежные средства с одного счета, и прежде, чем положили их на второй. Во время перевода денежных средств база данных находится в состоянии согласованности, когда общая сумма на счетах клиентов такая же, как и до перевода средств. Когда денежные средства переведены наполовину, база данных находится в состоянии несогласованности. Данные дебета и кредита должны быть либо обработаны все, либо не обработаны вообще.

Транзакции как логические блоки

Транзакция представляет собой логический блок выполнения. Каждая транзакция является последовательностью логически связанных команд, служащих для выполнения одной задачи и преобразования базы данных из одного состояния согласованности в другое. Характер состояния согласованности зависит от базы данных.

Операторы внутри транзакции обрабатываются как неделимый блок: либо все выполнены, либо ни один не выполнен. По завершении каждой транзакции необходимо **подтверждать** выполненные изменения, чтобы сделать их постоянными. Если по какой либо причине некоторые команды в транзакции обрабатываются неправильно, тогда любые промежуточные изменения отменяются, или выполняется **откат транзакции**. Другими словами, эти транзакции являются **атомарными**.

Группировка операторов в транзакции является ключевой как при защите согласованности данных (даже в случае ошибки носителей или отказа системы), так и при управлении параллельными операциями базы данных. Транзакции могут безопасно чередоваться, и выполнение каждой транзакции отмечает точку, в которой информация в базе данных является согласованной.

В случае отказа системы или сбоя базы данных в ходе нормальной работы Adaptive Server Anywhere выполняет автоматическое восстановление данных при следующем запуске базы данных. В процессе автоматического восстановления восстанавливаются все завершенные транзакции, и выполняется откат всех транзакций, которые не были подтверждены, когда произошел отказ. Атомарный характер транзакций обеспечивает то, что базы данных восстанавливаются к состоянию согласованности.

Для получения дополнительной информации о резервном копировании базы данных и восстановлении данных см. раздел “Резервное копирование и восстановление данных” (Backup and Data Recovery) на стр. 295 в документе “ *Руководство по администрированию баз данных ASA” (ASA Database Administration Guide)*.

Для получения дополнительной информации о параллельном использовании базы данных см. раздел “Введение в параллельную обработку” на стр. 92.

Использование транзакций

Adaptive Server Anywhere позволяет группировать команды в транзакции. Знание того, какие команды или действия выражают начало или конец транзакции, позволяет воспользоваться всем преимуществом этой функции.

Начало транзакций

Транзакции начинаются с одного из следующих событий:

- первый оператор после подключения к базе данных;
- первый оператор, следующий за концом транзакции.

Завершение транзакций

Транзакции завершаются одним из следующих событий:

- Оператор COMMIT сохраняет изменения в базе данных.
- Оператор ROLLBACK отменяет все изменения, выполненные транзакцией.
- Выполняется оператор с функцией автоматического подтверждения (команды определения данных, такие как ALTER, CREATE, COMMENT и DROP, имеют функцию автоматического подтверждения).
- При отключении от базы данных выполняется неявный откат.
- ODBC и JDBC позволяют установку автоматического подтверждения, которая выполняет действие COMMIT после каждого оператора. По умолчанию ODBC и JDBC требуют включения автоматического подтверждения, а каждый оператор рассматривается как отдельная транзакция. Для того чтобы воспользоваться возможностями проектирования транзакции, необходимо выключить функцию автоматического подтверждения.

  Для получения дополнительной информации об автоматическом подтверждении см. раздел “Установка режима автоматического или ручного подтверждения” (Setting autocommit or manual commit mode) на стр. 45 в документе “*Руководство по программированию ASA*” (*ASA Programming Guide)*.

- Установка в OFF параметра CHAINED базы данных подобна назначению автоматического подтверждения после каждого оператора. По умолчанию для соединений, которые используют jConnect или приложения Open Client, параметр CHAINED установлен в OFF.

  Для получения дополнительной информации см. раздел “Установка режима автоматического или ручного подтверждения” (Setting autocommit or manual commit mode) на стр. 45 в документе “*Руководство по программированию ASA*” (*ASA Programming Guide)* и раздел “Параметр CHAINED” (CHAINED option) на стр. 543 в документе “*Руководство по администрированию баз данных ASA*” *(ASA Database Administration Guide)*.

Параметры в SQL

Interactive SQL позволяет управлять тем, когда и как завершать транзакции из приложения Interactive:

- Если параметр AUTO_COMMIT установлен в ON, Interactive SQL автоматически подтверждает результаты после каждого успешного выполнения оператора и автоматически выполняет ROLLBACK после каждой неудачной попытки выполнения оператора.
- Установка параметра COMMIT_ON_EXIT управляет тем, что произойдет с неподтвержденными изменениями при выходе из Interactive SQL. Если этот параметр установлен в ON (значение по умолчанию), Interactive SQL выполняет оператор COMMIT; в противном случае он отменяет неподтвержденные изменения оператором ROLLBACK.

☞ Adaptive Server Anywhere также поддерживает команды Transact-SQL, такие как BEGIN TRANSACTION, для обеспечения совместимости с Sybase Adaptive Server Enterprise. Для получения дополнительной информации см. раздел “Совместимость с Transact-SQL” на стр. 371.

Введение в параллельную обработку

Параллельная обработка - это способность сервера базы данных обрабатывать несколько транзакций в одно и то же время. Параллельные транзакции, используемые не для специальных механизмов внутри сервера базы данных, могут мешать друг другу, производя несогласованные и неправильные данные.

Пример

Базы данных в универмаге должна быть построена так, чтобы разрешить многим клеркам изменять счета клиентов одновременно. Каждый клерк должен иметь возможность изменить состояние счетов при работе с клиентом и не могут позволить себе ждать момента, когда база данных будет свободна для использования.

Кто должен знать о параллельной обработке

Параллельная обработка транзакций касается всех администраторов базы данных и разработчиков. Даже в случае однопользовательской базы данных существует возможность столкнуться с параллельностью при обработке команд от нескольких приложений или даже нескольких подключений от одного приложения. Эти приложения и подключения могут мешать друг другу, так же как и несколько пользователей в сети.

Влияние размера транзакции на параллельную обработку

Способ группировки операторов SQL в транзакции может оказать значительное воздействие на целостность данных и производительность системы. Если транзакция слишком короткая, и она не содержит весь логический блок действий, то в базе данных могут появиться несогласованные данные. Если транзакция является слишком длинной и содержит несколько несвязанных действий, то есть большая вероятность того, что при выполнении ROLLBACK будут удалены те данные, которые, возможно, были бы переданы в базу данных совершенно безопасно.

Если транзакции длинные, они могут уменьшить степень параллельной обработки, препятствуя одновременному выполнению других транзакций.

Есть много факторов, которые определяют соответствующую длину транзакции в зависимости от типа приложения и среды.

Точки сохранения внутри транзакций

Можно идентифицировать важные состояния внутри транзакции и возвращаться к ним выборочно, используя **точки сохранения** для разделения групп связанных операций.

Оператор SAVEPOINT определяет промежуточную точку в течение выполнения транзакции. Можно отменить все изменения после этой точки, используя оператор ROLLBACK TO SAVEPOINT. После выполнения оператора RELEASE SAVEPOINT или завершения транзакции использовать точку сохранения невозможно.

Команды RELEASE SAVEPOINT или ROLLBACK TO SAVEPOINT не приводят к снятию каких-либо блокировок; блокировки удаляются только по завершении транзакции.

Именование и вложение точек сохранения

Именование и вложение точек сохранения позволяет использовать много активных точек сохранения внутри транзакции. Изменения между SAVEPOINT и RELEASE SAVEPOINT может быть отменена откатом назад к предыдущей точке сохранения или откатом всей транзакции. Изменения внутри транзакции становятся постоянными в базе данных только после подтверждения транзакции. После завершения транзакции все точки сохранения удаляются.

Точки сохранения не могут использоваться в режиме массовых операций. При использовании точек сохранения возникают лишь небольшие издержки.

Уровни изоляции и согласованность

Четыре уровня изоляции

Adaptive Server Anywhere позволяет управлять тем, насколько операции в одной транзакции видны операциям в других параллельных транзакциях. Это выполняется установкой параметра базы данных под названием **уровень изоляции (isolation level)**. Adaptive Server Anywhere предоставляет четыре различных уровня изоляции (пронумерованные от 0 до 3), используемые для предотвращения некоторых или всех несогласованных действий. Уровень 3 предоставляет самый высокий уровень изоляции. Более низкие уровни обеспечивают меньшую согласованность, но, как правило, лучшую производительность. Уровень 0 является установкой по умолчанию.

Все уровни изоляции обеспечивают то, что любая транзакция либо выполнится полностью, либо вообще не выполнится, и что никакие обновления не будут потеряны.

Основные виды несогласованности

Есть три основных вида несогласованности, которые могут возникнуть при выполнении параллельных транзакций. Этот список не исчерпывающий, поскольку также возможно возникновение несогласованности других видов. Эти три вида указаны в стандарте ISO SQL/92 и важны, так как определяют поведение при более низких уровнях изоляции.

- **Неверное чтение.** Транзакция A изменяет строку, но не подтверждает или выполняет откат изменения. Транзакция B читает измененную строку. Транзакция A либо продолжает изменять строку перед выполнением COMMIT, либо выполняет откат изменения. В любом случае транзакция B работает со строкой в состоянии, которое не было подтверждено.

  Для получения дополнительной информации о том, как уровни изоляции создают проблему неверного чтения, см. раздел “Учебный раздел по неверному чтению” на стр. 106.

- **Чтение без повторения.** Транзакция A читает строку. Транзакция B затем изменяет или удаляет строку и выполняет COMMIT. Если транзакция A затем попытается прочитать ту же самую строку снова, строка будет изменена или удалена.

  Для получения дополнительной информации о чтении без повторения см. раздел “Учебный раздел по чтению без повторения” на стр. 109.

- **Фантомная строка.** Транзакция A читает набор строк, которые удовлетворяют некоторому условию. Транзакция B затем выполняет INSERT или UPDATE для строки, которая предварительно не удовлетворяла условию A. Транзакция B подтверждает эти изменения. Эти подтвержденные строки теперь удовлетворяют условию. Транзакция A затем повторяет начальное чтение и получает различный набор строк.

Для получения дополнительной информации о фантомных строках см. раздел “Учебный раздел по фантомным строкам” на стр. 113.

Также могут существовать другие виды несогласованности. Эти три вида были выбраны для стандарта ISO SQL/92, так как они являются типовыми проблемами, и с их помощью было удобно описывать блокировки между транзакциями.

Уровни изоляции и типы чтения, чтение без повторения и фантомные строки

Уровни изоляции различаются согласно видам несогласованных действий и неверного поведения, которые возможны в Adaptive Server Anywhere. "**x**" означает, что поведение предотвращено, а отсутствие "**x**" означает, что поведение возможно.

Уровень изоляции	Неверное чтение	Чтение без повторения	Фантомные строки
0			
1	x		
2	x	x	
3	x	x	x

Эта таблица демонстрирует два принципа:

- Каждый уровень изоляции устраняет один из трех основных видов несогласованности.
- Каждый уровень устраняет виды несогласованности, устраняемые при всех нижних уровнях.

Эти четыре уровня изоляции имеют различные ODBC-имена. Эти имена основаны на названиях видов несогласованности, которые они предотвращают, и описаны в разделе “Параметр ValuePtr” на стр. 98.

Неустойчивость курсора

Другим важным видом несогласованности является **неустойчивость курсора**. Когда присутствует несогласованность этого вида, транзакция может изменить строку, на которую ссылается курсор другой транзакции. Стабильность курсора обеспечивает то, что приложения, использующие курсоры, не становятся причиной несогласованности данных в базе данных.

Пример

Транзакция A читает строку с использованием курсора. Транзакция B изменяет эту строку.

Не зная, что строка была изменена, транзакция A изменяет ее, обрабатывая данные изменяемой строки неправильно.

Устранение неустойчивости курсора

Adaptive Server Anywhere достигает состояния **стабильности курсора** посредством установки уровней изоляции 1, 2 и 3. Стабильность курсора обеспечивает то, что никакие другие транзакции не могут изменить информацию, которая содержится в текущей строке курсора. Информация в строке курсора может быть копией информации, содержащейся в конкретной таблице, или комбинацией данных из различных строк нескольких таблиц. Каждый раз, когда используется объединение или выделение в операторе SELECT, вероятно, что будет задействоваться более чем одна таблица.

Для получения информации о программировании процедур SQL и курсоров см. раздел “Использование процедур, триггеров и пакетов” на стр. 495.

Курсоры применяются только в том случае, когда Adaptive Server Anywhere используется через другое приложение. Для получения дополнительной информации см. раздел “Использование SQL в приложениях” (Using SQL in Applications) на стр. 9 в документе “*Руководство по программированию ASA*” (*ASA Programming Guide)*.

Другая проблема для приложений, использующих курсоры, состоит в том, являются ли изменения основных данных видимыми приложению. Управлять изменениями, которые являются видимыми приложениям, можно посредством указания чувствительности курсора.

Для получения дополнительной информации о чувствительности курсора см. раздел “Курсоры Adaptive Server Anywhere” (Adaptive Server Anywhere cursors) на стр. 28 в документе “*Руководство по программированию ASA*” (*ASA Programming Guide)*.

Установка уровня изоляции

Каждое подключение к базе данных имеет свой собственный уровень изоляции. Кроме того, база данных может сохранить заданный по умолчанию уровень изоляции для каждого пользователя или группы. Установка PUBLIC разрешает установить единственный заданный по умолчанию уровень изоляции для групп всей базы данных.

Уровень изоляции является параметром базы данных. Изменение параметров базы данных осуществляется с использованием оператора SET OPTION. Например, следующая команда устанавливает уровень изоляции для текущего пользователя в 3, самый высокий уровень.

```
SET OPTION ISOLATION_LEVEL = 3
```

Используя команду SET OPTION, можно изменить изоляцию определенного подключения и заданного по умолчанию уровня, связанного с кодом пользователя. При наличии соответствующих полномочий можно также изменить уровень изоляции для других пользователей или групп.

❖ **Установка уровня изоляции для текущего кода пользователя**

- Выполните оператор SET OPTION. Например, следующий оператор устанавливает уровень изоляции 3 для текущего пользователя:

```
SET TEMPORARY OPTION ISOLATION_LEVEL = 3
```

❖ **Установка уровня изоляции для пользователя или группы**

1 Выполните подключение к базе данных с использованием полномочий администратора БД.

2 Выполните оператор SET OPTION, добавляя имя группы и точку перед ISOLATION_LEVEL. Например, следующая команда устанавливает заданную по умолчанию изоляцию для специальной группы PUBLIC в 3.

```
SET OPTION PUBLIC.ISOLATION_LEVEL = 3
```

❖ **Установка уровня изоляции только для текущего сеанса**

Выполните оператор SET OPTION, используя ключевое слово TEMPORARY. Например, следующий оператор устанавливает уровень изоляции 3 на протяжении всего подключения:

```
SET TEMPORARY OPTION ISOLATION_LEVEL = 3
```

При отключении уровень изоляции возвращается к своему предыдущему значению.

Уровень изоляции по умолчанию

При подключении к базе данных сервер базы данных определяет начальный уровень изоляции следующим образом:

1 Уровень изоляции по умолчанию может быть установлен для каждого пользователя и группы. Если уровень сохранен в базе данных для определенного кода пользователя, сервер базы данных использует его.

2 Если нет, сервер базы данных проверяет группы, к которым принадлежит данный пользователь, пока уровень не будет найден. Все пользователи являются членами специальной группы PUBLIC. Если не найдена другая установка, то Adaptive Server Anywhere будет использовать уровень, назначенный этой группе.

Для получения дополнительной информации о пользователях и группах см. раздел “Управление кодами и полномочиями пользователей” (Managing User Ids and Permissions) на стр. 347 в документе “*Руководство по администрированию баз данных ASA” (ASA Database Administration Guide)*.

Для получения дополнительной информации о синтаксисе оператора SET OPTION см. раздел “Оператор SET OPTION” (SET OPTION statement) на стр. 503 в документе “*Справочник по SQL для ASA” (SQL ASA Reference Manual)*.

Можно изменять уровень изоляции в пределах транзакции, если, например, только одна таблица или группа таблиц требуют сериализованного доступа. Для получения информации об изменении уровня изоляции в пределах транзакции см. раздел “Изменение уровня изоляции в пределах транзакции” на стр. 98.

Установка уровня изоляции из приложения с поддержкой ODBC

Приложения с поддержкой ODBC вызывают **SQLSetConnectAttr** с атрибутом **Attribute**, установленным в SQL_ATTR_TXN_ISOLATION, и параметром **ValuePtr**, установленным согласно соответствующему уровню изоляции:

Параметр ValuePtr

ValuePtr	Уровень изоляции
SQL_TXN_READ_UNCOMMITTED	0
SQL_TXN_READ_COMMITTED	1
SQL_TXN_REPEATABLE_READ	2
SQL_TXT_SERIALIZABLE	3

Изменение уровня изоляции через ODBC

Можно изменить уровень изоляции подключения через ODBC, используя функцию SQLSetConnectOption из библиотеки *ODBC32.dll*.

Функция **SQLSetConnectOption** читает три параметра: значение дескриптора ODBC-подключения, факт того, что нужно устанавливать уровень изоляции, и значение, соответствующее уровню изоляции. Эти значения отражены в таблице ниже.

Строка	Значение
SQL_TXN_ISOLATION	108
SQL_TXN_READ_UNCOMMITTED	1
SQL_TXN_READ_COMMITTED	2
SQL_TXN_REPEATABLE_READ	4
SQL_TXT_SERIALIZABLE	8

Пример

Следующий вызов функции устанавливает уровень изоляции подключения MyConnection в 2:

```
SQLSetConnectOption(MyConnection.hDbc, SQL_TXN_ISOLATION,
SQL_TXN_REPEATABLE_READ)
```

ODBC использует возможность изоляции, чтобы поддержать различные параметры блокировки базы данных. Например, в PowerBuilder можно использовать атрибут Lock (Блокировка) объекта транзакции, чтобы установить уровень изоляции при подключении к базе данных. Атрибут Lock является строкой и устанавливается следующим образом:

```
SQLCA.lock = "RU"
```

Параметр Lock применяется только в момент выполнения CONNECT. Изменения атрибута Lock после выполнения CONNECT не отражаются на подключении.

Изменение уровня изоляции в пределах транзакции

Иногда бывает, что различные уровни изоляции являются подходящими для различных частей одной транзакции. Adaptive Server Anywhere разрешает изменять уровень изоляции базы данных в середине транзакции.

При изменении параметра ISOLATION_LEVEL в пределах транзакции новая установка будет выполнена только для следующего:

- любых курсоров, открытых после изменения;
- любых операторов, выполненных после изменения.

Изменение уровня изоляции в течение транзакции позволяет сразу управлять несколькими блокировками в пределах транзакции. В некоторых случаях для выполнения транзакции необходимо прочитать большую таблицу, но произвести конкретные действия только с несколькими из строк. Если возможная несогласованность не будет иметь серьезных последствий для этой транзакции, можно установить изоляцию на низкий уровень, на время сканирования большой таблицы, чтобы не задерживать работу других.

Можно также изменять уровень изоляции в течение транзакции, если, например, только одна таблица или группа таблиц требуют сериализованного доступа.

Пример, в котором уровень изоляции изменяется в пределах транзакции, см. в разделе “Учебный раздел по фантомным строкам” на стр. 113.

Просмотр уровня изоляции

Проверить уровень изоляции текущего подключения можно с использованием функции CONNECTION_PROPERTY.

❖ Просмотр уровня изоляции для текущего подключения

- Выполните следующий оператор:

```
SELECT CONNECTION_PROPERTY('ISOLATION_LEVEL')
```

Блокировка и взаимоблокировка транзакций

В процессе выполнения транзакции сервер базы данных устанавливает блокировку на строки, чтобы препятствовать использование этих строк другими транзакциями. **Блокировки** управляют количеством и типами разрешенного вмешательства.

Adaptive Server Anywhere использует **блокировку транзакции**, чтобы разрешить транзакциям выполняться одновременно без вмешательства, либо с ограниченным вмешательством. Блокировка может быть установлена для любой транзакции с целью воспрепятствовать другим параллельным транзакциям изменять или даже обращаться к конкретной строке. Эта схема блокировки транзакции всегда предотвращает определенные типы вмешательства. Например, транзакция, которая изменяет конкретную строку таблицы, всегда устанавливает блокировку этой строки для обеспечения того, что отсутствует возможность одновременного изменения или удаления этой строки другой транзакцией.

Блокировка транзакций

Когда транзакция пытается выполнять операцию, которая запрещена блокировкой, установленной другой транзакцией, возникает конфликт, и выполнение транзакции, пытавшейся выполнить эту операцию, отменяется или блокируется.

☞ В разделе “Двухфазная блокировка” на стр. 131 описывается взаимоблокировку, которая происходит, когда две или больше транзакции блокированы друг другом таким образом, что ни одна не может продолжить работу.

☞ Иногда набор транзакций достигает состояния, когда ни одна из них не может продолжить работу. Для получения дополнительной информации см. раздел “Взаимоблокировка” на стр. 101.

Параметр BLOCKING

Если каждая из двух транзакций устанавливает блокировку чтения одной строки, поведение в случае, когда одна из них попытается изменить эту строку, зависит от установки базы данных BLOCKING. Чтобы изменить строку, эта транзакция должна блокировать другую транзакцию, однако она не может сделать этого, пока эта строка блокирована другой транзакцией.

- Если параметр BLOCKING установлен в ON (значение по умолчанию), то транзакция, которая пытается произвести запись, ждет, пока другая транзакция не устранит блокировку чтения. После этого выполняется запись.
- Если параметр BLOCKING был установлен в OFF, то транзакция, которая пытается произвести запись, получает сообщение об ошибке.

Когда параметр BLOCKING установлен в OFF, вместо ожидания выполнение транзакции завершается, и любые произведенные ей изменения не сохраняются. В этом случае попробуйте выполнить транзакцию снова, позже.

Наличие блокировки более вероятно при более высоких уровнях изоляции, так как для этих уровней характерно наличие большого количества блокировок и проверок. На более высоких уровнях изоляции, как правило, возникает меньше случаев параллельной обработки. Степень наличия параллельности зависит от характера конкретных параллельных транзакций.

Для получения дополнительной информации о параметре BLOCKING см. раздел "Параметр BLOCKING" (BLOCKING option) на стр. 542 в документе "*Руководство по администрированию баз данных ASA" (ASA Database Administration Guide)*.

Взаимоблокировка

Блокировка транзакции может привести к **взаимоблокировке**, ситуации, в которой набор транзакций достигает состояния, где ни одна из них не может продолжить работу.

Причины взаимоблокировок

Взаимоблокировка может возникнуть по двум причинам:

- **Конфликт циклической блокировки.** Транзакция A блокирована транзакцией B, и транзакция B блокирована транзакцией A. Ясно, со временем проблема не решится, и одна из транзакций должна быть отменена, разрешая другой продолжить работу. Та же самая ситуация может возникнуть с более чем двумя транзакциями, блокированными друг другом.
- **Блокировка всех активных потоков базы данных.** Когда транзакция становится блокированной, ее поток базы данных не освобождается. Если база данных сконфигурирована с тремя потоками, и транзакции A, B, и C блокированы транзакцией D, которая в настоящий момент не выполняет запрос, то возникает ситуация взаимоблокировки, и с этого момента доступные потоки отсутствуют.

Adaptive Server Anywhere автоматически отменяет последнюю транзакцию, которая была блокирована (устранение ситуации взаимоблокировки), и возвращает ошибку этой транзакции, в которой указывается тип произошедшей взаимоблокировки.

Количество потоков базы данных, которое использует сервер, зависит от настроек конкретной базы данных. Для получения информации об установке количества потоков базы данных см. раздел "Параметр THREAD_COUNT" (THREAD_COUNT option) на стр. 584 в документе "*Руководство по администрированию баз данных ASA" (ASA Database Administration Guide)* и раздел "Параметр –ge командной строки" (-ge command-line option) на стр. 146 в документе "*Руководство по администрированию баз данных ASA" (ASA Database Administration Guide)*.

Определение объекта блокировки

Для определения того, какие подключения блокированы какими подключениями, используется системная процедура *sa_conn_info*. Эта процедура возвращает результирующий набор, состоящий из одной строки для каждого подключения. Один столбец результирующего набора отображает, блокировано ли подключение, и если да, то какое другое подключение его блокирует.

Для получения дополнительной информации см. раздел "Системная процедура sa_conn_info" (sa_conn_info system procedure) на стр. 648 в документе "*Справочник по SQL для ASA" (SQL ASA Referece Manual)*.

Выбор уровней изоляции

Выбор уровня изоляции зависит от вида задачи, которую выполняет приложение. Этот раздел дает некоторые рекомендации для выбора уровней изоляции.

При выборе соответствующего уровня изоляции необходимо учитывать как потребность в согласованности, так и потребность в беспрепятственном выполнении параллельных транзакций. Если транзакция задействует только одно или два специфичных значения в одной таблице, то маловероятно, что она столкнется с другими процессами, в отличие от транзакции, которая производит поиск во множестве больших таблиц, должна блокировать много строк или все таблицы и может занять длительное время для завершения работы.

Например, если транзакция осуществляет перевод денежных средств между банковскими счетами или проверяет баланс счета, скорее всего потребуется сделать все возможное для обеспечения того, что возвращаемые данные были правильны. С другой стороны, если требуется получить грубую оценку пропорции неактивных счетов, то можно не учитывать время ожидания транзакции во время выполнения других транзакций и немного пожертвовать точностью, чтобы избежать столкновения с другими пользователями базы данных.

Кроме того, перевод может воздействовать только на две строки, которые содержат балансы двух счетов, в то время как для вычисления оценки должны быть прочитаны все счета. Поэтому перевод, скорее всего, задержит другие транзакции.

Adaptive Server Anywhere предоставляет четыре уровня изоляции: уровни 0, 1, 2 и 3. Уровень 3 предоставляет полную изоляцию и обеспечивает то, что транзакции чередуются таким способом, что расписание предоставляет возможность сериализации.

Сериализуемые расписания

Чтобы обрабатывать транзакции одновременно, сервер базы данных должен выполнить некоторые операторы компонента одной транзакции, затем некоторые из других транзакций, перед продолжением обработки дальнейших операций из первой. Порядок, в котором чередуются операции компонентов различных транзакций, называется **расписанием**.

Параллельная обработка транзакций этим способом может привести ко многим возможным исходам, включая три основных вида несогласованности, описанных в предыдущем разделе. Иногда конечное состояние базы данных также может быть достигнуто выполнением транзакций последовательно, т. е. когда одна транзакция всегда завершается полностью перед запуском следующей. Расписание называют **сериализуемым** при выполнении транзакций последовательно, в некотором порядке, что оставляет базу данных в том же состоянии, что и фактическое расписание.

Для получения дополнительной информации о том, как Adaptive Server Anywhere обрабатывает сериализацию, см. раздел “Двухфазная блокировка” на стр. 131.

Сериализуемость является широко распространенным критерием

правильности. Сериализуемое расписание рассматривается как правильное, так как база данных не находится под влиянием параллельной обработки транзакций.

Уровень изоляции влияет на сериализуемость транзакции. На уровне изоляции 3 все расписания сериализуемы. Значением по умолчанию является уровень 0.

Сериализуемость означает отсутствие последствий параллельной обработки

Даже в том случае, когда транзакции выполнены последовательно, конечное состояние базы данных может зависеть от порядка, в котором эти транзакции выполнены. Например, если одна транзакция заносит в определенную ячейку значение 5, а другая заносит число 6, то конечное значение ячейки определяется той транзакцией, которая выполняется последней.

Сериализуемость расписания не приводит к определению того, какой порядок транзакций был бы наиболее эффекитвным, а скорее указывает на отсутствие последствий параллельной обработки транзакций. Результаты, которые могут быть достигнуты последовательным выполнением набора транзакций в некотором порядке, предположительно рассматриваются как правильные.

Несериализуемые расписания ведут к несогласованности

Типичными проблемами, возникающими при использовании несериализуемого расписания, являются виды несогласованности, приведенные в разделе "Основные виды несогласованности" на стр. 94. В каждом случае несогласованность появляется из-за того, что операторы чередовались таким образом, что привели к результату, который не был бы возможен, если бы все транзакции выполнялись последовательно. Например, неверное чтение может произойти только тогда, когда одна транзакция выбирает строки, в то время как другая транзакция находится в процессе вставки или обновления данных в той же самой строке.

Типовые транзакции при различных уровнях изоляции

Для различных типов задач подходят различные уровни изоляции. Используйте информацию ниже для помощи при решении того, какой уровень лучше всего подходит для каждой конкретной операции.

Типовые транзакции уровня 0

Транзакции, в которых выполняется просмотр или ввод данных, могут длиться несколько минут и читать большое количество строк. Если используется уровень изоляции 2 или 3, это ограничит параллельную обработку транзакций. Поэтому для этого вида транзакций обычно используется уровень изоляции 0 или 1.

Например, для приложения поддержки принятия решений, которое производит чтение больших объемов информации из базы данных для выполнения статистических обзоров, не может оказать значительное воздействие то, что несколько прочитанных им строк будут впоследствии изменены. Если для такого приложения требуется высокая изоляция, оно может произвести блокировку чтения при большом количестве данных, не разрешая другим приложениям доступ к ним для записи.

Типовые транзакции уровня 1

Уровень изоляции 1 особенно полезен при использовании с курсорами, так как эта комбинация обеспечивает стабильность курсора без значительного увеличения требований блокировки. Adaptive Server Anywhere достигает этого преимущества благодаря раннему снятию блокировок чтения, произведенных для текущей строки курсора. Эти блокировки должны сохраниться до окончания транзакции либо на уровне 2, либо на уровне 3 для обеспечения чтения с повторением.

Например, транзакция, которая изменяет уровни описи при помощи курсора, подходит для этого уровня, потому что каждая корректировка уровней описи, такая как прием и продажа товаров, не была бы потеряна, и эти частые корректировки будут иметь минимальное воздействие на другие транзакции.

Типовые транзакции уровня 2

При уровне изоляции 2 строки, которые соответствуют установленному критерию, не могут быть изменены другими транзакциями. Можно использовать этот уровень, когда нужно читать строки более одного раза, полагаясь на то, что строки, содержавшиеся в первом результирующем наборе, не будут изменены.

Из-за относительно большого количества требуемых блокировок чтения нужно с осторожностью использовать этот уровень изоляции. Как с транзакциями уровня 3, тщательное проектирование базы данных и индексов уменьшает количество применяемых блокировок и, следовательно, может значительно улучшить производительность базы данных.

Типовые транзакции уровня 3

Уровень изоляции 3 соответствует тем транзакциям, которые требуют наибольшей защиты. Устранение фантомных строк позволяет выполнять многошаговые операции с набором строк без опасения того, что в результате этих операций появятся новые строки и повлияют на результат транзакции.

Однако из-за высокой степени целостности, которую предоставляет уровень изоляции 3, он должен использоваться экономно в больших системах, которым необходимо поддерживать большое количество параллельных транзакций. Adaptive Server Anywhere устанавливает на этом уровне больше блокировок, чем на любом другом, вызывая вероятность того, что одна транзакция будет препятствовать выполнению многих других.

Улучшение параллельной обработки на уровнях изоляции 2 и 3

Уровни изоляции 2 и 3 используют много блокировок, и поэтому для баз данных, которые постоянно используют эти уровни изоляции, большое значение имеет правильное проектирование. Когда нужно использовать сериализуемые транзакции, важно, чтобы при проектировании базы данных, в особенности индексов, учитывались бизнес-правила проекта. Можно также улучшить производительность, разбивая большие транзакции на несколько меньших, и таким образом сокращая отрезок времени, когда строки блокированы.

Хотя велика вероятность того, что сериализуемые транзакции будут блокировать другие транзакции, они не всегда являются менее эффективными. При обработке этих транзакций Adaptive Server Anywhere может выполнить определенную оптимизацию с целью увеличения

производительности, несмотря на увеличенное количество блокировок. Например, так как каждое чтение строк должно быть блокировано проверкой того, действительно ли они соответствуют критерию поиска, сервер базы данных может комбинировать операции чтения строк и установки блокировок.

Уменьшение воздействия блокировок

Там, где это уместно, следует избегать выполнения транзакций уровня изоляции 3. Такие транзакции ведут к установке большого количества блокировок и, следовательно, могут существенно влиять на выполнение других параллельных транзакций.

Когда характер операции требует использование на уровне изоляции 3, можно уменьшить его воздействие на параллельную обработку, уменьшив количество строк, которые должны быть прочитаны при запросе, насколько возможно. Эти шаги позволят транзакции уровня 3 выполняться быстрее и, что более важно, уменьшат количество блокировок, которые она устанавливает.

Например, добавление индекса может значительно ускорить выполнение транзакций, особенно тогда, когда по крайней мере одна из них должна выполняться на уровне изоляции 3. Индекс может дать два преимущества:

- индекс размещает строки эффективным способом;
- при выполнении поиска с использованием индекса будет установлено меньшее количество блокировок.

Для получения дополнительной информации о методах блокировки, используемых в Adaptive Server Anywhere, см. раздел “Принципы блокировки” на стр. 121.

Для получения дополнительной информации о производительности и о том, как Adaptive Server Anywhere организовывает доступ к информации для выполнения команд, см. раздел “Контроль и повышение производительности” на стр. 141.

Учебный раздел по уровням изоляции

Различные уровни изоляции ведут себя различными способами, и то, какой из них будет использоваться, зависит от базы данных и от выполняемых операций. Следующие учебные разделы помогут определить, какие уровни изоляции являются подходящими для различных задач.

Учебный раздел по неверному чтению

В этом учебном разделе показан один тип несогласованности, которая может возникнуть при одновременном выполнении нескольких транзакций. Два служащих в маленькой торговой компании обращаются к корпоративной базе данных в одно и то же время. Первый служащий является менеджером по продажам этой компании. Второй служащий является бухгалтером.

Менеджеру по продажам требуется увеличить цену футболок, продаваемых этой фирмой, на 0,95 доллара, но он не вполне владеет синтаксисом языка SQL. В то же самое время бухгалтер пытается вычислить значение розничных продаж для текущей описи, чтобы включить в отчет, который он предполагал принести на следующее заседание правления. Это неизвестно менеджеру по продажам.

Совет

Перед следующим изменением базы данных целесообразно протестировать изменение, используя SELECT вместо UPDATE.

В этом примере пользователь играет роль двух человек, использующих демонстрационную базу данных одновременно.

1 Запустите Interactive SQL.

2 Выполните подключение к демонстрационной базе данных как менеджер по продажам (Sales Manager):

- В диалоге Connect выберите в качестве источника данных ODBC *ASA 8.0 Sample*.
- На закладке Advanced введите следующую строку, чтобы упростить идентификацию окна:

```
ConnectionName=Sales Manager
```

- Нажмите кнопку OK для подключения.

3 Запустите второй экземпляр Interactive SQL.

4 Выполните подключение к демонстрационной базе данных как бухгалтер (Accountant):

- В диалоге Connect выберите в качестве источника данных ODBC *ASA 8.0 Sample*.

- На закладке Advanced введите следующую строку, чтобы упростить идентификацию окна:

```
ConnectionName=Accountant
```

- Нажмите кнопку OK для подключения.

5 Как менеджер по продажам, поднимите цену всех футболок на 0,95 долл.:

- В окне, помеченном как Sales Manager, выполните следующие команды:

```
SELECT id, name, unit_price
FROM product;
UPDATE PRODUCT
SET unit_price = unit_price + 95
WHERE NAME = 'Tee Shirt'
```

Результат:

id	name	unit_price
300	Tee Shirt	104.00
301	Tee Shirt	109.00
302	Tee Shirt	109.00
400	Baseball Cap	9.00
...	...	...

Пользователь тут же понимает, что следовало ввести 0,95 вместо 95, но прежде, чем он успевает исправить ошибку, бухгалтер обращается к базе данных из другого офиса.

6 Бухгалтера интересует количество денежных средств, отраженных в описи. Как бухгалтер, выполните следующие команды, чтобы вычислить общее значение розничной продажи всех товаров на складе:

```
SELECT SUM( quantity * unit_price )
  AS inventory
FROM product
```

Результат:

inventory
21453.00

К сожалению, это вычисление неточно. Менеджер по продажам случайно увеличил цену товара до 95 $, и результаты отражают эту ошибочную цену. Эта ошибка демонстрирует один из типовых видов несогласованности, известный как **неверное чтение**. Бухгалтер обратился к данным, которые менеджер по продажам ввел, но еще не подтвердил.

Неверное чтение и другие виды несогласованности, описанные в разделе “Уровни изоляции и согласованность” на стр. 94, можно устранить.

7 Как менеджер по продажам, устраните ошибку, выполнив откат первых изменений и введя правильно команду UPDATE. Проверьте, что новые значения правильны.

```
ROLLBACK;
UPDATE product
SET unit_price = unit_price + 0.95 WHERE NAME = 'Tee
Shirt';
```

id	name	unit_price
300	Tee Shirt	9.95
301	Tee Shirt	14.95
302	Tee Shirt	14.95
400	Baseball Cap	9.00
...	...	...

8 Бухгалтер не знает, что количество, которое он вычислил, было ошибочным. Правильное значение можно получить, снова выполнив оператор SELECT в окне бухгалтера.

```
SELECT SUM( quantity * unit_price )
  AS inventory
FROM product;
```

inventory
6687.15

9 Завершите транзакцию в окне менеджера по продажам. Менеджер по продажам должен ввести оператор COMMIT, чтобы сделать изменения постоянными, но вместо этого можно ввести ROLLBACK, чтобы избежать изменения копии демонстрационной базы данных на машине.

```
ROLLBACK;
```

Бухгалтер получает ошибочную информацию из базы данных, не зная об этом, так как сервер базы данных обрабатывает операции менеджера по продажам и бухгалтера одновременно.

Учебный раздел по чтению без повторения

Пример в разделе "Учебный раздел по неверному чтению" на стр. 106 описывает первый вид несогласованности, а именно неверное чтение данных. В этом примере бухгалтер произвел расчет в тот момент, когда менеджер по продажам производил обновление цены. В расчете бухгалтера была использована ошибочная информация, которую менеджер по продажам ввел ранее, а теперь исправил ошибку.

Следующий пример демонстрирует другой вид несогласованности: чтение без возможности повторения. В этом примере будут показаны роли тех же двух человек, одновременно использующих демонстрационную базу данных. Менеджер по продажам хочет предложить новую отпускную цену на пластмассовые козырьки. Бухгалтер хочет проверить цены некоторых товаров, содержащихся в последнем заказе.

Этот пример начинается с уровня изоляции 1 для обоих подключений, а не уровня изоляции 0, являющегося значением по умолчанию для демонстрационной базы данных Adaptive Server Anywhere. Установкой уровня изоляции 1 устраняется вид несогласованности, описанный в предыдущем примере, а именно неверное чтение.

1 Запустите Interactive SQL.

2 Выполните подключение к демонстрационной базе данных как менеджер по продажам.

- В диалоге Connect выберите источник данных ODBC *ASA 8.0 Sample*.
- На закладке Advanced введите следующую строку, чтобы упростить идентификацию окна:

```
ConnectionName=Sales Manager
```

- Нажмите кнопку OK для подключения.

3 Запустите второй экземпляр Interactive SQL.

4 Выполните подключение к демонстрационной базе данных как бухгалтер.

- В диалоге Connect выберите источник данных ODBC *ASA 8.0 Sample*.
- На закладке Advanced введите следующую строку, чтобы упростить идентификацию окна:

```
ConnectionName=Accountant
```

- Нажмите кнопку OK для подключения.

5 Установите уровень изоляции для подключения бухгалтера на 1, выполнив следующую команду.

```
SET TEMPORARY OPTION ISOLATION_LEVEL = 1
```

6 Установите уровень изоляции 1 в окне менеджера по продажам, выполнив следующую команду:

```
SET TEMPORARY OPTION ISOLATION_LEVEL = 1
```

7 Бухгалтер решает получить список цен на козырьки. Как бухгалтер, выполните следующую команду:

```
SELECT id, name, unit_price FROM product
```

id	name	unit_price
300	Tee Shirt	9.00
301	Tee Shirt	14.00
302	Tee Shirt	14.00
400	Baseball Cap	9.00
...	...	...

8 Менеджер по продажам решает представить новую отпускную цену на пластмассовые козырьки. Как менеджер по продажам, выполните следующую команду:

```
SELECT id, name, unit_price FROM product
WHERE name = 'Visor';
UPDATE product
SET unit_price = 5.95 WHERE id = 501;
COMMIT;
```

id	name	unit_price
500	Visor	7.00
501	Visor	5.95

9 Сравните цену козырька в окне менеджера по продажам с ценой того же самого козырька в окне бухгалтера. Окно бухгалтера все еще отображает старую цену, даже после того, как менеджер по продажам ввел новую цену и подтвердил изменение.

Такая несогласованность называется **чтением без повторения**, так как в случае, если бы бухгалтер произвел такую же операцию SELECT во второй раз в *той же самой транзакции*, он не получил бы тех же самых результатов. Проверьте это самостоятельно. Как бухгалтер, выполните команду выбора еще раз. Заметьте, что теперь отображается отпускная цена, заданная менеджером по продажам.

```
SELECT id, name, price
FROM product
```

id	name	unit_price
300	Tee Shirt	9.00
301	Tee Shirt	14.00
302	Tee Shirt	14.00
400	Baseball Cap	9.00
...	...	...

Конечно, если бухгалтер завершил свою транзакцию, например, выполнив команду COMMIT или ROLLBACK перед повторным использованием SELECT, то это был бы другой случай. База данных

доступна для одновременного использования многими пользователями, и изменение кем-либо каких-либо значений полностью допустимо как до, так и после транзакции бухгалтера. Изменение в результатах некорректно только потому, что оно происходит в середине его транзакции. Такое событие делает расписание несериализуемым.

10 Бухгалтер замечает такое поведение и решает, что с этого момента он не хочет, чтобы цены изменялись, пока он просматривает их. Неповторяющиеся чтения устраняются на уровне изоляции 2. В роли бухгалтера выполните:

```
SET TEMPORARY OPTION ISOLATION_LEVEL = 2;

SELECT id, name, unit_price
FROM product
```

11 Менеджер по продажам решает, что лучше задержать продажу пластмассовых козырьков до следующей недели, чтобы ему не пришлось назначать меньшую цену на большой заказ, прибытие которого ожидается завтра. Попробуйте выполнить следующие операторы в соответствующем окне. Команда начнет выполняться, после чего его окно окажется зафиксированным.

```
UPDATE product
SET unit_price = 7.00
WHERE id = 501
```

Сервер базы данных должен обеспечить возможность повтора чтения на уровне изоляции 2. С этой целью он устанавливает блокировку чтения на каждую строку таблицы продуктов, которую читает бухгалтер. Если менеджер по продажам пытается вернуть предыдущую цену, то ее транзакция должна получить блокировку записи на строку пластмассовых козырьков таблицы продуктов. Поскольку блокировки являются исключительными, его транзакция должна дождаться отмены транзакцией бухгалтера блокировки чтения.

12 Бухгалтер закончил просмотр цен. Он не хочет создавать риск случайного изменения базы данных, поэтому он завершает свою транзакцию оператором ROLLBACK.

```
ROLLBACK
```

Заметьте, что, как только сервер базы данных выполняет этот оператор, транзакция менеджера по продажам завершается.

id	name	unit_price
500	Visor	7.00
501	Visor	7.00

13 Менеджер по продажам может теперь закончить работу. Он хочет подтвердить свое изменение, чтобы восстановить исходную цену.

```
COMMIT
```

Типы блокировок и различные уровни изоляции

При изменении уровня изоляции бухгалтера с 1 на 2 сервер базы данных использовал блокировки чтения там, где до этого не было установлено ни одной блокировки. В целом, каждый уровень изоляции характеризуется типами необходимых блокировок и тем, как обрабатываются блокировки, установленные другими транзакциями.

На уровне изоляции 0 серверу базы данных требуются только блокировки записи. Эти блокировки используются для обеспечения того, что никакие две транзакции не будут производить изменений, способных вызвать конфликт. Например, транзакция уровня 0 устанавливает блокировку записи строки перед ее изменением или удалением и вставляет любые новые строки с уже установленной блокировкой записи.

Транзакции уровня 0 не выполняют проверки читаемых ими строк. Это означает, что при чтении строки транзакцией уровня 0 не проверяется наличие или отсутствие блокировок, установленных другими транзакциями. Поскольку никаких проверок не требуется, транзакции уровня 0 выполняются довольно быстро. Эта скорость достигается в ущерб согласованности данных. Всякий раз при чтении строки, блокированной на запись другой транзакцией, существует возможность возвращения ими неверных данных.

На уровне 1 при выполнении транзакции строка проверяется на наличие блокировки записи.

Такие транзакции обеспечивают подтверждение всех читаемых ими данных, несмотря на то, что при этом требуется выполнить еще одну операцию. Попробуйте повторить первый учебный раздел, установив уровень изоляции на 1 вместо 0. Окажется, что расчет бухгалтера не может быть выполнен, пока, пока транзакция менеджера по продажам, обновляющего цену футболок, не будет завершена.

Когда бухгалтер поднял свой уровень изоляции до 2, сервер базы данных начал использовать блокировки чтения. С этого момента он устанавливает блокировку чтения для транзакции бухгалтера на каждую выбранную им строку.

Блокировка транзакции

В вышеприведенном учебном разделе окно менеджера по продажам было зафиксировано на время выполнения менеджером по продажам команды UPDATE. Сервер базы данных начал выполнять эту команду, затем обнаружил, что транзакцией бухгалтера установлена блокировка чтения на строку, которую требовалось изменить менеджеру по продажам. В этот момент сервер базы данных просто приостановил выполнение команды UPDATE. Как только бухгалтер завершил свою транзакцию командой ROLLBACK, сервер базы данных автоматически снял установленные им блокировки. Не встретив никаких других препятствий, он перешел к завершению выполнения команды UPDATE менеджера по продажам.

Как правило, конфликт блокировок происходит тогда, когда одна транзакция пытается монопольно заблокировать строку, на которую уже установлена блокировка другой транзакции, или установить совместную блокировку на строку, на которой уже установлена монопольная блокировка другой транзакции. Одна транзакция должна дождаться завершения другой транзакции. В этом случае говорят, что ожидающая транзакция **блокирована** другой транзакцией.

Когда сервер базы данных распознает конфликт блокировок, не позволяющий немедленно продолжить транзакцию, он может либо приостановить выполнение транзакции, либо прервать транзакцию,

откатить назад все изменения и выдать ошибку. Управление этими действиями производится настройкой параметра BLOCKING. Если параметр BLOCKING установлен в ON, то вторая транзакция ожидает, как в учебном разделе выше.

Для получения дополнительной информации о параметре блокировки см. раздел "Параметр BLOCKING" на стр. 100.

Учебный раздел по фантомным строкам

В данном учебном разделе описано продолжение того же сценария. В данном случае бухгалтер просматривает таблицу отделов, в то время как менеджер по продажам создает новый отдел. Будет рассмотрен процесс появления фантомной строки.

Выполните шаги 1-4 из предыдущего раздела "Учебный раздел по чтению без повторения" на стр. 109 так, чтобы имелось два экземпляра Interactive SQL.

1 В окне менеджера по продажам установите уровень изоляции на 2, выполнив следующую команду.

```
SET TEMPORARY OPTION ISOLATION_LEVEL = 2;
```

2 Установите уровень изоляции на 2 для окна бухгалтера, выполнив следующую команду.

```
SET TEMPORARY OPTION ISOLATION_LEVEL = 2;
```

3 В окне бухгалтера введите следующую команду для получения списка всех отделов.

```
SELECT * FROM department
ORDER BY dept_id;
```

dept_id	dept_name	dept_head_id
100	R & D	501
200	Sales	902
300	Finance	1293
400	Marketing	1576
...	...	...

4 Менеджер по продажам решает создать новый отдел, который будет специализироваться на иностранном рынке. Возглавит новый отдел служащий Филипп Чин (Philip Chin) с кодом emp_id 129.

```
INSERT INTO department
   (dept_id, dept_name, dept_head_id)
   VALUES(600, 'Foreign Sales', 129);
```

Последняя команда создает новую запись для нового отдела. Она появляется как новая строка внизу таблицы в окне менеджера по продажам.

5 Однако бухгалтер не знает о новом отделе. На уровне изоляции 2 сервер базы данных устанавливает блокировки для обеспечения отсутствия каких-либо изменений строк, но не устанавливает блокировок, запрещающих другим транзакциям вставку новых строк.

Бухгалтер обнаружит новую строку только после повторного выполнения команды SELECT. Выполните оператор SELECT в окне бухгалтера еще раз. Появится добавленная в таблицу новая строка.

```
SELECT * FROM department ORDER BY dept_id;
```

dept_id	dept_name	dept_head_id
100	R & D	501
200	Sales	902
300	Finance	1293
500	Shipping	703
...	...	...

Появившаяся новая строка называется **фантомной строкой**, поскольку с точки зрения бухгалтера она появляется как призрак, будто бы ниоткуда. Бухгалтер подключен на уровне изоляции 2. На этом уровне сервер базы данных устанавливает блокировки только на те строки, которые в данный момент используются. Другие строки остаются без изменений, и, следовательно, ничто не мешает менеджеру по продажам вставить новую строку.

6 Бухгалтер предпочел бы избежать таких неожиданностей в будущем, поэтому он поднимает уровень изоляции своей текущей транзакции до 3. Как бухгалтер, введите следующие команды.

```
SET TEMPORARY OPTION ISOLATION_LEVEL = 3
SELECT *
FROM department
ORDER BY dept_id
```

7 Менеджер по продажам хотел бы создать второй отдел для поддержки инициатив продаж, направленных на крупных корпоративных партнеров. Выполните следующую команду в окне менеджера по продажам.

```
INSERT INTO department
   (dept_id, dept_name, dept_head_id)
VALUES(700, 'Major Account Sales', 902)
```

В окне менеджера по продажам выполнение операции приостановится из-за установленных бухгалтером блокировок. Нажмите кнопку Interrupt the SQL Statement на панели инструментов (или выберите Stop в меню SQL), чтобы прервать выполнение операции.

8 Чтобы избежать изменения демонстрационной базы данных Adaptive Server Anywhere, следует выполнить откат вставки новых отделов. Выполните следующую команду в окне менеджера по продажам:

```
ROLLBACK
```

После того, как бухгалтер повысил свой уровень изоляции до 3 и снова выделил все строки в таблице отделов, сервер базы данных установил блокировку на вставку для каждой строки в таблице и одну дополнительную фантомную блокировку для предотвращения вставки в конец таблицы. Когда менеджер по продажам попытался вставить новую строку в конец таблицы, именно эта последняя блокировка прервала выполнение его команды.

Обратите внимание, что команда менеджера по продажам была блокирована даже при том, что менеджер по продажам все еще подключен на уровне изоляции 2. Сервер базы данных устанавливает блокировки на вставку подобно блокировкам чтения, как это предусмотрено уровнем изоляции и операторами каждой транзакции. После установки блокировок они должны учитываться любыми другими параллельными транзакциями.

Для получения дополнительной информации о блокировках см. раздел "Принципы блокировки" на стр. 121.

Учебный раздел по практическому применению блокировок

В данном разделе описано продолжение того же сценария. В этом разделе бухгалтер и менеджер по продажам должны выполнить задачи, включающие заказ на покупку и таблицы деталей заказа на покупку. Бухгалтеру необходимо проверить количества комиссионных чеков, которые были выплачены служащим на основе продаж, сделанных ими в течение апреля 2001 года. Менеджер по продажам замечает, что несколько заказов не были внесены в базу данных, и хочет добавить их.

Их работа демонстрирует применение фантомной блокировки. Фантомная блокировка – это совместная блокировка, установленная в позиции сканирования по индексу для предотвращения появления фантомных строк. Когда транзакция на уровне изоляции 3 выбирает строки, соответствующие определенному критерию, сервер базы данных устанавливает блокировки на вставку во избежание вставки другими транзакциями совпадающих строк. Количество блокировок, помещенных от имени пользователя, зависит как от критерия поиска, так и от структуры базы данных.

Выполните шаги 1-3 из предыдущего учебного раздела, где описано, как запустить два экземпляра Interactive SQL.

1 Установите уровень изоляции на 2 и в окне менеджера по продажам, и в окне бухгалтера, выполнив следующую команду.

```
SET TEMPORARY OPTION ISOLATION_LEVEL = 2
```

2 Каждый месяц торговым представителям выплачивается комиссия, которая рассчитывается как процент от их продаж в течение этого месяца. Бухгалтер подготавливает комиссионные чеки за апрель 2001 года. Его первая задача состоит в том, чтобы вычислить полные продажи каждого представителя в течение этого месяца.

Введите следующую команду в окне бухгалтера. Цены, информация

о заказе на покупку и данные служащих хранятся в отдельных таблицах. Соедините эти таблицы, используя связи по внешнему ключу, чтобы собрать необходимые части информации.

```
SELECT emp_id, emp_fname, emp_lname,
   SUM(sales_order_items.quantity * unit_price)
      AS "April sales"
FROM employee
   KEY JOIN sales_order
   KEY JOIN sales_order_items
   KEY JOIN product
WHERE '2001-04-01' <= order_date
   AND order_date < '2001-05-01'
GROUP BY  emp_id, emp_fname, emp_lname
```

emp_id	emp_fname	emp_lname	Продажи в апреле
129	Philip	Chin	2160.00
195	Marc	Dill	2568.00
299	Rollin	Overbey	5760.00
467	James	Klobucher	3228.00
...	...	...	...

3 Менеджер по продажам замечает, что большой заказ, продажа которого выполнена Филиппом Чином (Philip Chin), не был введен в базу данных. Филипп предпочитает получать свои комиссионные сразу, поэтому менеджер по продажам вводит пропущенный заказ, который был сделан 25 апреля.

В окне менеджера по продажам введите следующие команды. Заказ на покупку и детали заказов вводятся в отдельные таблицы, поскольку один заказ может содержать много товаров. Следует создать запись для заказа на покупку до внесения в него элементов. В целях сохранения ссылочной целостности сервер базы данных позволяет транзакции добавлять элементы в заказ только в том случае, если этот заказ уже существует.

```
INSERT into sales_order
VALUES ( 2653, 174, '2001-04-22', 'r1',
      'Central', 129);

INSERT into sales_order_items
VALUES ( 2653, 1, 601, 100, '2001-04-25' );

COMMIT;
```

4 Бухгалтеру не может знать, что менеджер по продажам только что добавил новый заказ. Если бы новый заказ был введен ранее, он был бы включен в расчет продаж Филиппа Чина за апрель.

В окне бухгалтера заново вычислите общие количества продаж за апрель. Используйте ту же самую команду и заметьте, что сумма продаж Филиппа Чина за апрель изменяется на 4560.00 долл.

emp_id	emp_fname	emp_lname	Продажи за апрель
129	Philip	Chin	4560.00
195	Marc	Dill	2568.00
299	Rollin	Overbey	5760.00
467	James	Klobucher	3228.00
...	...	...	...

Представьте, что бухгалтер теперь отмечает все заказы, сделанные в апреле, чтобы указать, что комиссия выплачена. Заказ, который только что ввел менеджер по продажам, мог бы быть найден при втором поиске и отмечен как выплаченный, при этом даже не будучи включенным в полные продажи Филиппа за апрель!

5 На уровне изоляции 3 сервер базы данных устанавливает блокировки на вставку для обеспечения того, что никакие другие транзакции не смогут добавить строку, соответствующую критерию поиска или выбора.

Прежде всего, откатите вставку пропущенного заказа Филиппа: выполните следующий оператор в окне менеджера по продажам.

```
ROLLBACK
```

6 В окне бухгалтера выполните следующие два оператора.

```
ROLLBACK;
SET TEMPORARY OPTION ISOLATION_LEVEL = 3;
```

7 В окне менеджера по продажам выполните следующие операторы для удаления нового заказа.

```
DELETE
FROM sales_order_items
WHERE id = 2653;
DELETE
FROM sales_order
WHERE id = 2653;
COMMIT;
```

8 В окне бухгалтера выполните тот же самый запрос, что и ранее.

```
SELECT emp_id, emp_fname, emp_lname,
   SUM(sales_order_items.quantity * unit_price)
      AS "April sales"
FROM employee
   KEY JOIN sales_order
   KEY JOIN sales_order_items
   KEY JOIN product
WHERE '2001-04-01' <= order_date
   AND order_date < '2001-05-01'
GROUP BY  emp_id, emp_fname, emp_lname
```

Поскольку был установлен уровень изоляции 3, сервер базы данных автоматически устанавливает блокировки на вставку для обеспечения невозможности вставки элементов апрельского заказа менеджером по продажам до того момента, пока бухгалтер не завершит свою транзакцию.

9 Вернитесь к окну менеджера по продажам. Вновь попытайтесь ввести пропущенный заказ Филиппа Чина.

```
INSERT INTO sales_order
VALUES ( 2653, 174, '2001-04-22',
         'r1','Central', 129)
```

Окно менеджера по продажам зависнет; операция не будет завершена. Нажмите кнопку *Interrupt the SQL Statement* на панели инструментов (или выберите Stop в меню SQL), чтобы прервать выполнение операции.

10 Менеджеру по продажам сейчас запрещено вводить апрельский заказ, но может показаться, что он все еще может ввести его в мае.

Измените дату команды на 05 мая и попробуйте снова.

```
INSERT INTO sales_order
VALUES ( 2653, 174, '2001-05-05', 'r1',
       'Central', 129)
```

Окно менеджера по продажам снова зависнет. Нажмите кнопку *Interrupt the SQL Statement* на панели инструментов (или выберите Stop в меню SQL), чтобы прервать выполнение операции. Несмотря на то, что сервер базы данных устанавливает для предотвращения вставки не больше блокировок, чем необходимо, эти блокировки могут повлиять на большое количество других транзакций.

Сервер базы данных помещает блокировки в индексы таблиц. Например, он размещает фантомную блокировку в индексе, поэтому новая строка не может быть вставлена прямо перед индексом. Однако если подходящего индекса нет, ему придется заблокировать все строки в таблице.

В некоторых ситуациях наличие блокировок на вставку может привести к запрету вставки в таблицу для одних и разрешению для других.

11 Менеджеру по продажам требуется добавить второй элемент в заказ 2651. Используйте следующую команду:

```
INSERT INTO sales_order_items
VALUES ( 2651, 2, 302, 4, '2001-05-22' )
```

Все идет нормально, и менеджер по продажам решает добавить следующий элемент и в заказ 2652.

```
INSERT INTO sales_order_items
VALUES ( 2652, 2, 600, 12, '2001-05-25' )
```

Окно менеджера по продажам зависнет. Нажмите кнопку *Interrupt the SQL Statement* на панели инструментов (или выберите Stop в меню SQL), чтобы прервать выполнение операции.

12 Завершите этот учебный сценарий, отменив все изменения во избежание изменения демонстрационной базы данных. Введите следующую команду в окне менеджера по продажам.

```
ROLLBACK
```

Введите ту же самую команду в окне бухгалтера.

```
ROLLBACK
```

Теперь можно закрыть оба окна.

Принципы блокировки

Когда сервер базы данных обрабатывает транзакцию, он может заблокировать одну или более строк таблицы. Блокировки позволяют сохранить надежность информации, содержащейся в базе данных, путем предотвращения параллельного доступа к ним других транзакций. При их использовании также повышается точность результатов запросов, так как учитывается факт обновления информации в данный момент.

Сервер базы данных устанавливает эти блокировки автоматически; явный ввод инструкций не требуется. Сервер хранит все блокировки, установленные транзакцией, вплоть до ее завершения, например, операторами COMMIT или ROLLBACK, за единственным исключением, отмеченным в разделе "Ранний сброс блокировок чтения (исключение)" на стр. 132.

Транзакцию, которая обращается к строке, обычно называют транзакцией, установившей блокировку. В зависимости от типа блокировки, другие транзакции получают либо ограниченный доступ к блокированной строке, либо не имеют доступа вообще.

Получение информации о блокировках в таблице

Для получения информации о блокировках, содержащихся в базе данных, можно использовать системную процедуру *sa_locks*. Для получения дополнительной информации см. раздел "Системная процедура sa_locks" (sa_locks system procedure) на стр. 656 в документе "*Справочник по SQL для ASA*" (*ASA SQL Reference Manual*).

Можно также просмотреть блокировки в Sybase Central. Откройте папку Tables соответствующей базы данных. В правой области окна появится закладка с именем Locks. Для каждой блокировки эта закладка отображает код подключения, код пользователя, имя таблицы, тип блокировки и имя блокировки.

Объекты, которые могут быть заблокированы

Adaptive Server Anywhere устанавливает блокировки на следующие объекты.

- **Строки в таблицах.** Транзакция может блокировать определенную строку с целью предотвратить ее изменение другой транзакцией. Если транзакции требуется изменить строку, то она должна установить на эту строку блокировку записи.
- **Точки ввода между строками.** Транзакции обычно сканируют строки с использованием определенного индексом порядка или последовательно. В обоих случаях блокировка может быть помещена в позицию сканирования. Например, установка блокировки в индексе может запретить другой транзакции вставить строку с определенным значением или областью значений.
- **Схемы таблиц.** Транзакция может блокировать схему таблицы, запретив другим транзакциям изменять структуру этой таблицы.

Среди этих объектов наличие строк представляется наиболее понятным. Очевидно, что транзакция, производящая чтение, обновление, удаление или вставку строки, должна ограничить параллельный доступ других транзакций. Аналогично, транзакция, изменяющая структуру таблицы (возможно, вставляя новый столбец), может сильно повлиять на другие транзакции. В этом случае для предотвращения ошибок особенно важно ограничить доступ других транзакций.

Порядок строк

Для упорядочения строк можно использовать индекс, основанный на конкретном критерии, который был определен при создании индекса.

Если индекса нет, Adaptive Server Anywhere сортирует строки в соответствии с их физическим размещением на диске. В случае последовательного сканирования создается специальный порядок строк, определяемый внутренними потребностями сервера базы данных. Не следует полагаться на порядок строк при последовательном сканировании. Тем не менее, с точки зрения сканирования строк, Adaptive Server Anywhere обрабатывает запрос подобно сканированию по индексу, несмотря на использование порядка строк по собственному выбору. На сканируемые позиции могут устанавливаться блокировки, как при использовании индекса.

Посредством блокировки позиции сканирования транзакция предотвращает некоторые действия других транзакций, касающиеся определенной области значений в этом порядке строк. Блокировка вставки и блокировка на вставку всегда устанавливаются для позиций сканирования.

Например, транзакция могла бы удалить строку, таким образом удалив и определенное значение первичного ключа. Пока эта транзакция не подтвердит изменение или не откатит его назад, она должна сохранять свое право на одно из этих действий. В случае удаленной строки, она должна иметь гарантию, что никакая другая транзакция не сможет вставить строку, используя то же самое значение первичного ключа и, следовательно, делая невозможной операцию отката. Блокировка позиции сканирования этой строки резервирует это право, при этом минимизируя влияние на другие транзакции.

Типы блокировок

Adaptive Server Anywhere использует четыре различных типа блокировок для реализации своей схемы блокирования и обеспечения соответствующих уровней изоляции между транзакциями:

- **блокировка чтения** (совместная);
- **фантомная блокировка или блокировка на вставку** (совместная);
- **блокировка записи** (монопольная);
- **антифантомная блокировка или блокировка вставки** (совместная).

Помните, что сервер базы данных размещает эти блокировки автоматически, и явный ввод инструкций не требуется.

Каждая из этих блокировок имеет свою собственную цель, и все они используются вместе. Каждая из них предотвращает определенный набор типов несогласованности, которая могла бы возникнуть при их отсутствии. В зависимости от уровня изоляции следует выбрать, будет ли сервер базы данных использовать все или некоторые из блокировок для поддержки требуемой степени согласованности данных.

Вышеуказанные типы блокировок имеют следующие области использования:

- Транзакция устанавливает **блокировку записи** всякий раз при вставке, обновлении и удалении строки данной транзакцией. Если установлена блокировка записи, то никакая другая транзакция уже не может установить на эту строку ни блокировку чтения, ни блокировку записи. Блокировка записи представляет собой монопольную блокировку.

- Транзакция устанавливает **блокировку чтения** при чтении строки данной транзакцией. Несколько транзакций могут установить блокировку чтения на одну и ту же строку (блокировка чтения является совместной, или не монопольной, блокировкой). Если на строку установлена блокировка чтения, то никакая другая транзакция уже не может установить на нее блокировку записи. Таким образом, установившая блокировку чтения транзакция получает гарантию того, что никакая другая транзакция не изменит и не удалит строку.

- **Блокировка на вставку**, или **фантомная блокировка**, является совместно используемой блокировкой, помещаемой в позицию сканирования по индексу с целью предотвращения появления фантомных строк. Она запрещает другим транзакциям вставлять строку в таблицу прямо перед строкой, блокированной на вставку. Блокировки на вставку для операций поиска с использованием индексов требуют наличия блокировки чтения на каждую читаемую строку и одной дополнительной блокировки чтения для предотвращения вставки в индекс в конце результирующего набора. Фантомные строки для операций поиска, не использующих индексы, требуют наличия блокировки чтения на все строки в таблице с целью запретить вставкам изменять результирующий набор и, таким образом, могут оказать негативное влияние на параллельную обработку транзакций.

- **Блокировка вставки**, или **антифантомная блокировка**, является совместно используемой блокировкой, помещаемой в позицию сканирования по индексу с целью резервирования права на вставку строки. Как только одна транзакция устанавливает блокировку вставки на строку, никакая другая транзакция уже не может установить на эту строку блокировку на вставку. Блокировка чтения на соответствующую строку всегда устанавливается вместе с блокировкой вставки для обеспечения того, что никакой другой процесс не сможет обновить или уничтожить строку, совершив, таким образом, обход блокировки вставки.

Adaptive Server Anywhere использует эти четыре типа блокировок по мере необходимости для обеспечения требуемого уровня согласованности данных. Явного запроса на использование определенной блокировки не требуется. Вместо этого применяется управление уровнем согласованности, как описывается в следующем разделе. Наличие сведений о типах блокировок поможет при выборе уровней изоляции и позволит лучше понять степень воздействия каждого уровня на производительность.

Монопольные и совместные блокировки

Каждый из этих четырех типов блокировок входит в одну из двух категорий:

- **Монопольные блокировки.** Монопольную блокировку на строку таблицы в конкретный момент может установить только одна транзакция. Никакая транзакция не может установить монопольную блокировку, если какая-либо другая транзакция уже установила блокировку любого типа на ту же самую строку. Как только транзакция устанавливает монопольную блокировку, запросы на блокировку строки другими транзакциями будут отклонены.

  Блокировки записи являются монопольными.

- **Совместные блокировки.** Такие блокировки могут быть установлены в одно и то же время любым количеством транзакций на любую строку. Совместные блокировки иногда называются немонопольными блокировками.

  Блокировки чтения, вставки и блокировки на вставку являются совместными.

Только одна транзакция должна иметь право на изменение строки в одной операции. В противном случае две одновременно выполняемые транзакции могут попытаться изменить одно значение на два различных новых. Следовательно, важно, чтобы блокировка записи была монопольной.

Напротив, не возникает никаких затруднений, если более чем одной транзакции требуется прочитать строку. Так как ни одна из них не изменяет строку, конфликта не происходит. Следовательно, блокировки чтения могут использоваться совместно.

Аналогичное рассуждение можно привести относительно блокировок на вставку и блокировок вставки. Много транзакций могут предотвратить вставку строки в определенной позиции сканирования путем получения каждой из них блокировки на вставку. Та же логика относится и к блокировкам вставки. Если для той или иной транзакции требуется монопольный доступ, она может легко получить его, установив блокировки на вставку и блокировки вставки на одну и ту же строку. В случае, когда такие блокировки устанавливается одной транзакцией, конфликта не возникает.

Какие блокировки вызывают конфликты?

Следующая таблица показывает конфликтные комбинации блокировок.

	чтение	**запись**	**на вставку**	**вставка**
чтение		конфликт		
запись	конфликт	конфликт		
на вставку				конфликт
вставка			конфликт	

Эти конфликты возникают только в тех случаях, когда блокировки устанавливаются различными транзакциями. Например, одна транзакция может установить блокировку на вставку и блокировку вставки на одну позицию сканирования с целью получения монопольного доступа к этой позиции.

Блокировка при обработке запросов

Блокировки, используемые Adaptive Server Anywhere при вводе пользователем оператора SELECT, зависят от уровня изоляции транзакции.

Операторы SELECT на уровне изоляции 0

При выполнении оператора SELECT на уровне изоляции 0 никаких операций блокировки не требуется. Ни одна транзакция не защищена от изменений, производимых другими транзакциями. Ответственность за интерпретацию результатов этих запросов лежит на программисте или пользователе базы данных, который должен помнить об этих ограничениях.

Операторы SELECT на уровне изоляции 1

Может показаться неожиданным то, что Adaptive Server Anywhere использует блокировки при выполнении транзакции на уровне изоляции 1 практически не в большей степени, чем на уровне изоляции 0. Действительно, сервер базы данных изменяет свое поведение только в двух отношениях.

Первое различие в выполнении операций относится скорее не к установке блокировок, а к их учету. На уровне изоляции 0 транзакция может читать любую строку, независимо от того, установила ли другая транзакция блокировку записи на эту строку. Напротив, перед чтением каждой строки на уровне изоляции 1 транзакция должна проверить, не установлена ли в этом месте блокировка записи. Она не может читать строки, имеющие блокировку записи, так как выполнение этой операции могло бы повлечь за собой неверное чтение данных.

Второе различие в выполнении операций заключается в обеспечении устойчивости курсора. Устойчивость курсора достигается установкой блокировки чтения на текущую строку в позиции курсора. Такая блокировка чтения снимается при перемещении курсора. Если курсор установлен на результате соединения, может быть затронута более чем одна строка. В этом случае сервер базы данных устанавливает блокировки чтения на все строки, информация из которых вошла в текущую строку курсора, и удаляет все эти блокировки, как только текущей выбирается другая строка курсора. Блокировка чтения, устанавливаемая для обеспечения устойчивости курсора - это единственный тип блокировки, не сохраняющейся до конца транзакции.

Операторы SELECT на уровне изоляции 2

На уровне изоляции 2 Adaptive Server Anywhere изменяет свои действия, чтобы обеспечить возможность повторения операций чтения. Если команда SELECT возвращает значения из каждой строки в таблице, то сервер базы данных устанавливает блокировку чтения на каждую строку таблицы при ее чтении. Если, напротив, оператор SELECT содержит раздел WHERE или другое условие, которое ограничивает работу выбранными строками, то сервер базы данных вместо этого читает каждую строку, проверяет значения в строке на соответствие этому критерию и затем устанавливает блокировку чтения на строку в случае ее соответствия критерию.

Как и на всех уровнях изоляции, блокировки, установленные на уровне 2, включают все блокировки, заданные на уровнях 1 и 0. Таким образом, устойчивость курсора обеспечена и здесь, и неверное чтение не допускается.

Операторы SELECT на уровне изоляции 3

При применении уровня изоляции 3 Adaptive Server Anywhere должен обеспечить сериализуемость всех расписаний. В частности, в дополнение к требованиям, наложенным на каждом из более низких уровней, должны быть ликвидированы фантомные строки.

В целях выполнения этого требования сервер базы данных использует блокировки чтения и блокировки на вставку. Когда производится выбор, сервер базы данных устанавливает блокировку чтения на каждую строку, информация из которой входит в результирующий набор. Это обеспечивает то, что никакие другие транзакции не могут изменить эти данные до окончания их использования.

Это требование аналогично процедурам, используемым сервером базы данных на уровне изоляции 2, но отличается тем, что блокировка должна быть установлена на каждое обращение к строке для чтения, *независимо от того, соответствует она или нет какому-либо примененному критерию.* Например, если выбираются имена всех служащих в отделе сбыта, то сервер должен установить блокировку на все строки, содержащие информацию о специалистах по продажам, если транзакция выполняется на уровне изоляции 2 или 3. На уровне изоляции 3, однако, он также должен установить блокировку чтения на каждую из строк служащих, не находящихся в отделе сбыта. В противном случае, кто-либо другой, имеющий доступ к базе данных, потенциально может переместить в отдел продаж другого служащего, в то время как пользователь все еще использует старые результаты.

Факт того, что блокировка чтения должна быть установлена на каждую строку независимо от ее соответствия примененному критерию, имеет два важных следствия.

- Серверу базы данных, возможно, потребуется разместить намного больше блокировок, чем было бы необходимо на уровне изоляции 2.
- Сервер базы данных может работать несколько более эффективно. Он может немедленно установить блокировку чтения на каждую строку при ее чтении, так как блокировки должны быть установлены независимо от того, принята ли информация из этой строки.

Количество блокировок на вставку, устанавливаемых сервером, может значительно изменяться; оно зависит от критериев и от доступных в таблице индексов. Предположим, что выбирается информация о служащем с кодом служащего 123. Если код служащего является первичным ключом таблицы служащих, то сервер базы данных может минимизировать количество своих операций. Он может использовать индекс, автоматически создающийся для первичного ключа, для эффективного определения местонахождения строки. Кроме того, нет опасности изменения кода другого служащего на 123 другой транзакцией, поскольку значения первичного ключа должны быть уникальны. Сервер может обеспечить то, что этот идентификатор не будет присвоен второму служащему, путем установки блокировки чтения только на одну строку, содержащую информацию о служащем с этим идентификатором.

Напротив, сервер базы данных установил бы больше блокировок, если бы были выделены все служащие в отделе сбыта. Так как в отдел может быть добавлено любое количество служащих, серверу скорее всего будет необходимо считать каждую строку в таблице служащих и проверить, не работает ли каждый конкретный человек в отделе сбыта. Если дело обстоит именно так, то на каждую строку должны быть установлены как блокировки чтения, так и блокировки на вставку.

Блокировка при выполнении вставок

Операции INSERT создают новые строки. Для обеспечения целостности данных Adaptive Server Anywhere использует следующую процедуру.

Для получения дополнительной информации об использовании блокировок при выполнении вставок см. раздел "Блокировки на вставку" на стр. 130.

1 Выделение места в памяти для размещения новой строки. Местоположение изначально скрыто от остальной части базы данных, поэтому пока нет необходимости учитывать возможность обращения к ней другой транзакцией.

2 Заполнение новой строки любыми предоставленными значениями.

3 Установка блокировки записи на новую строку.

4 Размещение блокировки вставки в таблице, в которую добавляется строка. Помните, что блокировки вставки являются монопольными блокировками, поэтому, как только блокировка вставки установлена, никакая другая транзакция уже не может блокировать вставку посредством установки блокировки на вставку.

5 Вставка строки в таблицу. После этого другие транзакции впервые могут обнаружить, что новая строка существует. Тем не менее, изменение или удаление строки другими транзакциями невозможно из-за ранее установленной блокировки записи.

6 Обновление всех затронутых индексов и проверка ссылочной целостности и уникальности, где необходимо. Проверка ссылочной целостности означает получение гарантии того, что ни один внешний ключ не указывает на несуществующий первичный ключ. Значения первичного ключа должны быть уникальны. Другие столбцы также могут быть определены так, чтобы содержать только уникальные значения, и если такие столбцы существуют, то проверяется уникальность.

7 Транзакция может быть выполнена тогда, когда ссылочная целостность не будет нарушена вследствие следующего действия: записи операции в файл журнала транзакций и сброса всех блокировок.

8 Вставка других строк (при необходимости), если был выбран каскадный параметр, и запуск триггеров.

Уникальность

Можно проверить, что все значения в определенном столбце или комбинации столбцов являются уникальными. Сервер базы данных всегда выполняет эту задачу путем создания индекса для уникального столбца, даже если он не создается явно.

В частности, все значения первичного ключа должны быть уникальны. Сервер базы данных автоматически создает индекс для первичного ключа каждой таблицы. Таким образом, не следует ставить серверу базы данных задачу создать индекс первичного ключа, поскольку это действие было бы избыточным.

Висячие ключи и ссылочная целостность

Внешний ключ – это ссылка на первичный ключ, обычно в другой таблице. Если этот первичный ключ не существует, то неверный внешний ключ называют **висячим ключом**.

Adaptive Server Anywhere автоматически обеспечивает отсутствие в базе данных висячих ключей. Этот процесс называется **проверкой ссылочной целостности**. Сервер базы данных проверяет ссылочную целостность путем подсчета висячих ключей.

WAIT FOR COMMIT

Можно указать серверу базы данных задержать проверку ссылочной целостности до конца транзакции. В этом режиме можно вставить одну строку, содержащую внешний ключ, затем вставить вторую строку, содержащую отсутствующий первичный ключ. Следует выполнить обе операции в одной транзакции. В противном случае сервер базы данных не разрешит эти операции.

Чтобы сервер базы данных задержал проверку ссылочной целостности до момента подтверждения транзакции, установите значение параметра WAIT_FOR_COMMIT в ON. По умолчанию этот параметр установлен в OFF. Чтобы включить его, дайте следующую команду:

```
SET OPTION WAIT_FOR_COMMIT = ON;
```

Перед подтверждением транзакции сервер базы данных удостоверяется, что ссылочная целостность соблюдена, путем проверки количества созданных транзакцией висячих ключей. В конце каждой транзакции это число должно быть нулевым.

Даже если необходимый первичный ключ существует во время операции вставки строки, сервер базы данных должен обеспечить его существование при подтверждении результатов. Это реализуется посредством установки блокировки чтения на соответствующую строку. При установленной блокировке чтения любая другая транзакция все еще может читать эту строку, но не может удалить или изменить ее.

Блокировка при обновлении

Сервер базы данных изменяет информацию, содержащуюся в определенной записи, используя следующую процедуру.

1 Установка блокировки записи на нужную строку.

2 Если какие-либо из измененных элементов входят в индекс, удаление каждой записи индекса, соответствующей старым значениям. Таким образом регистрируются любые создавшиеся висячие ключи.

3 Обновление каждого из затронутых значений.

4 Если индексированные значения были изменены, добавление новых записей в индекс. Проверка уникальности, где необходимо, и ссылочной целостности, если был изменен какой-либо первичный или внешний ключ.

5 Транзакция может быть подтверждена, если ссылочная целостность не будет нарушена вследствие следующего действия: записи операции в файл журнала транзакций (включая предыдущие значения всех элементов в строке) и сброса всех блокировок.

6 Каскадирование операций вставки или удаления, если был выбран этот параметр, и это повлияло на первичные или вторичные ключи.

Может показаться неожиданным то, что кажущаяся простой операция изменения значения в таблице может требовать довольно большого количества действий. Объем работы, которую должен выполнить сервер базы данных, заметно снижается, если изменяемое значение не является частью первичного или внешнего ключа. Он снижается еще больше, если значение не содержится в индексе, явно или неявно, поскольку этот атрибут был объявлен уникальным.

Операция проверки ссылочной целостности во время операции UPDATE не менее проста, чем во время операции INSERT. Фактически при изменении значения первичного ключа могут возникнуть висячие ключи. При вставке заменяющего значения сервер базы данных должен провести проверку на наличие висячих ключей еще раз.

Блокировка при удалении

Операция DELETE состоит почти из тех же шагов, что и операция INSERT, но следующих в обратном порядке.

1 Установка блокировки записи на нужную строку.

2 Удаление всех записей индекса, соответствующих каким-либо значениям в строке.

Непосредственно перед удалением каждой записи индекса, установка одной или более блокировок на вставку по мере необходимости, чтобы предотвратить вставку другой транзакцией такого же элемента до момента завершения удаления. В целях проверки ссылочной целостности сервер базы данных также отслеживает все висячие ключи, возникающие вследствие удаления.

3 Удаление строки из таблицы, после чего она уже не будет видна другим транзакциям. Строка не может быть уничтожена до момента подтверждения транзакции, поскольку такое действие сделает невозможным откат транзакции.

4 Транзакция может быть подтверждена, если ссылочная целостность не будет нарушена вследствие следующего действия: записи операции в файл журнала транзакций (включая предыдущие значения всех элементов в строке), сброса всех блокировок и уничтожения строки.

5 Каскадирование операции удаления, если был выбран этот параметр, и это повлияло на первичные или внешние ключи.

Блокировки на вставку

Сервер базы данных должен предоставлять возможность отката операции DELETE. Частично он обеспечивает это посредством установки блокировок на вставку. Эти блокировки не являются монопольными; однако они запрещают другим транзакциям действия по вставке строки, вследствие которых исчезает возможность отката операции DELETE. Например, удаленная строка, возможно, содержала значение первичного ключа, или другое уникальное значение.

Если другой транзакции было разрешено вставить строку с тем же значением, то операция DELETE не может быть отменена без нарушения уникальности.

Adaptive Server Anywhere реализует функции контроля за уникальностью посредством индексов. В случае простой таблицы с первичным ключом с одним атрибутом может быть достаточно одной фантомной блокировки. В других ситуациях может резко увеличиться количество требуемых блокировок. Например, таблица может не иметь первичного ключа или другого индекса, связанного с любым из атрибутов. Так как строки в таблице не имеют базового порядка их расположения, единственный способ предотвратить вставки может состоять в том, чтобы установить блокировку на вставку на всю таблицу.

Удаление строки может привести к установке большого количества блокировок. Влияние на параллельную обработку в базе данных можно минимизировать множеством способов. Как описано выше, индексы и первичные ключи уменьшают количество необходимых блокировок, поскольку они создают определенный порядок строк в таблице. Сервер базы данных автоматически пользуется наличием определенного порядка. Вместо установки блокировок на каждую строку в таблице, сервер может просто блокировать *следующую* строку. Без индекса строки не имеют порядка расположения, и понятие следующей строки не имеет смысла.

Сервер базы данных устанавливает блокировки на вставку на строку, следующую за удаленной строкой. Если будет удалена последняя строка таблицы, то сервер базы данных просто установит блокировку на вставку на невидимую конечную строку. Фактически, если таблица не содержит индекса, то количество требуемых блокировок на вставку на единицу больше количества строк в таблице.

Блокировки на вставку и блокировки чтения

В то время как одна или более блокировок на вставку исключает возможность блокировки вставки, и одна или более блокировок чтения исключают возможность блокировки записи, между блокировками вставки/на вставку и блокировками чтения/записи взаимодействия нет. Например, хотя блокировка записи не может быть установлена на строку, содержащую блокировку чтения, она может быть установлена на строку, имеющую только блокировку на вставку. Вследствие такой гибкости серверу базы данных становится доступно больше опций, но в тоже время это означает, что сервер всегда должен принимать дополнительные меры по установке блокировки чтения при установке блокировки на вставку. В противном случае другая транзакция может удалить строку.

Двухфазная блокировка

Общая информация о блокировке, приведенная в разделах выше, часто достаточна для удовлетворения потребностей пользователя. Бывают, однако, случаи, когда требуется более глубокое знание того, что происходит в сервере базы данных при выполнении основных видов операций. Это знание закладывает более глубокую основу для понимания и предсказания потенциальных проблем, с которыми могут встретиться пользователи базы данных.

Двухфазная блокировка важна в контексте обеспечения сериализуемости расписаний. **Протокол двухфазной блокировки** определяет процедуру, которой следует каждая транзакция.

Этот протокол имеет большое значение, поскольку в случае его соблюдения всеми транзакциями он обеспечивает наличие сериализуемого и, следовательно, корректного расписания. Он также способствует пониманию того, почему при определенных способах блокировки разрешены некоторые типы несогласованности.

Протокол двухфазной блокировки

1 Перед работой с той или иной строкой транзакция должна установить блокировку на эту строку.

2 После снятия блокировки транзакция больше не должна устанавливать никаких блокировок.

Практически, транзакция обычно сохраняет блокировки до своего завершения оператором COMMIT или ROLLBACK. Снятие блокировки до окончания транзакции приводит к невозможности отката изменений всякий раз, когда после работы со строками требуется возвратить их к исходному состоянию.

Протокол двухфазной блокировки позволяет сформулировать следующую важную теорему:

Теорема двухфазной блокировки

Если все транзакции подчиняются протоколу двухфазной блокировки, то все возможные чередующиеся расписания сериализуемы.

Другими словами, если все транзакции следуют протоколу двухфазной блокировки, то не возможен ни один из типов несогласованности, указанных выше.

Этот протокол определяет операции, необходимые для обеспечения полной совместимости данных. Однако может оказаться так, что в конкретной базе данных некоторые типы несогласованности допустимы при выполнении некоторых операций. Полное устранение несогласованности часто означает снижение эффективности базы данных.

Блокировки записи устанавливаются на изменяемые, вставляемые и удаляемые строки независимо от уровня изоляции. Они всегда сохраняются до момента подтверждения и отката.

Блокировки чтения на разных уровнях изоляции

Уровень изоляции	Блокировки чтения
0	Нет
1	На строках, которые появляются в результирующем наборе; сохраняются только на время нахождения курсора на строке.
2	На строках, которые появляются в результирующем наборе; сохраняются, пока пользователь не выполнит COMMIT или ROLLBACK.
3	На всех читаемых строках и всех точках ввода, пересекающихся при вычислении результирующего набора.

Для получения дополнительной информации см. раздел "Сериализуемые расписания" на стр. 102.

Подробную информацию о блокировках имеет смысл разделить на две части: что происходит при операциях INSERT, UPDATE, DELETE и SELECT, и как различные уровни изоляции влияют на размещение блокировок чтения, блокировок на вставку и блокировок вставки.

Несмотря на то, что можно управлять количеством устанавливаемых блокировок в пределах сервера базы данных путем задания уровня изоляции, существуют также блокировки, которые действует на всех уровнях, даже на уровне 0. Эти блокировки являются базовыми. Например, пока одна транзакция обновляет строку, никакая другая транзакция не может изменить ту же самую строку до завершения первой транзакции. Без этой предосторожности нельзя было бы выполнить откат первой транзакции.

Операции блокировки, выполняемые сервером базы данных на уровне изоляции 0, – самые подходящие для начального изучения, поскольку они представляют основу. Другие уровни вводят новые средства блокировки, но не ликвидируют существующие на более низких уровнях. Таким образом, движение к более высокому уровню изоляции добавляет операции, которых нет на более низких уровнях.

Раннее снятие блокировок чтения (исключение)

На уровне изоляции 3 транзакция устанавливает блокировку чтения на каждую строку, которую она читает. При нормальной работе транзакция никогда не снимает блокировку до момента своего завершения. Действительно, при необходимости обеспечить сериализуемость расписания транзакции не следует снимать блокировку слишком рано.

Adaptive Server Anywhere всегда сохраняет блокировки записи вплоть до завершения транзакции. Если бы блокировка была снята слишком рано, то другая транзакция могла бы изменить эту строку и сделать невозможным откат первой транзакции.

Блокировки чтения снимаются только в одном особом случае. На уровне изоляции 1 транзакции устанавливают блокировку чтения на строку только тогда, когда она становится текущей строкой курсора. На уровне изоляции 1, однако, блокировка снимается, когда эта строка перестает быть текущей. Такое поведение приемлемо, поскольку серверу базы данных не требуется обеспечивать возможность повтора чтения на уровне изоляции 1.

Для получения дополнительной информации об уровнях изоляции см. раздел "Выбор уровней изоляции" на стр. 102.

Специальная оптимизация

В предыдущих разделах рассмотрены блокировки, устанавливаемые тогда, когда все транзакции выполняются на заданном уровне изоляции. Например, если все транзакции выполняются на уровне изоляции 2, то блокировка устанавливается так, как описано в соответствующем разделе выше.

Однако существует возможность того, что в базе данных должны будут обрабатываться множественные транзакции, выполняемые на различных уровнях. Некоторые транзакции, такие как перемещения денежных средств между счетами, должны быть сериализуемы и, следовательно, выполняться на уровне изоляции 3. Для других операций, например, обновления адреса или вычисления средних ежедневных продаж, часто может быть удовлетворительным более низкий уровень изоляции.

В то время, когда на сервере базы данных не выполняются транзакции на уровне 3, происходит оптимизация некоторых операций в целях повышения производительности. Например, для поддержания транзакций уровня 3 часто необходима установка большого количества дополнительных блокировок на вставку и блокировок вставки. При некоторых обстоятельствах, сервер базы данных может не выполнять установку или проверку некоторых типов блокировок, если транзакции уровня 3 отсутствуют.

Например, блокировки на вставку применяются сервером базы данных при следующих обстоятельствах:

1 Обеспечение возможности отката операций удаления в таблицах с уникальными атрибутами.

2 Устранение фантомных строк в транзакциях уровня 3.

Если в конкретной таблице не используются транзакции уровня 3, то серверу базы данных не требуется устанавливать блокировки на вставку в индексе таблицы, не содержащей уникальных атрибутов. Если, однако, имеется хотя бы одна транзакция уровня 3, то все транзакции, даже уровня 0, должны устанавливать блокировки на вставку так, чтобы транзакции уровня 3 могли учитывать операции, выполняемые другими транзакциями.

Конечно, когда сервер базы данных предпринимает попытку оптимизации описанных выше типов, он всегда добавляет к таблице примечания. Если неожиданно запускается транзакция уровня 3, то можно быть уверенным, что для нее будут установлены необходимые блокировки.

Возможно, будет трудно управлять таким количеством используемых одновременно уровней изоляции, зависящих от конкретных операций, которые нужно выполнять различным пользователям базы данных. При наличии возможности, однако, может быть целесообразно выбрать время, в течение которого выполняются транзакции уровня 3, поскольку они потенциально могут вызвать существенное замедление операций базы данных. Это воздействие усиливается, так как сервер базы данных вынужден выполнять дополнительные действия для операций низшего уровня.

Некоторые аспекты параллельной обработки

В этом разделе рассматриваются следующие аспекты параллельной обработки операций:

- "Генерация первичного ключа" на стр. 135;
- "Операторы определения данных и параллельная обработка" на стр. 136.

Генерация первичного ключа

Вожможны ситуации, при которых база данных должна автоматически сгенерировать уникальный номер. Например, при построении таблицы для размещения счетов продаж предпочтительным может оказаться автоматическое назначение базой данных уникальных номеров счетов, а не назначение этих номеров служащими отдела сбыта.

Существует несколько способов генерации таких номеров.

Пример

Например, новые номера счетов можно получать прибавлением 1 к предыдущему номеру счета. Применение этого способа невозможно, если добавлением счетов в базу данных занимается более чем один пользователь. Два пользователя могут решить использовать один и тот же номер счета.

Существует несколько решений этой проблемы:

- Назначение диапазона номеров счетов каждому пользователю, добавляющему новые счета.

  Эта схема реализуется путем создания таблицы со столбцами *user name* (имя пользователя) и *invoice number* (номер счета). Таблица будет иметь по одной строке для каждого пользователя, добавляющего счета. Каждый раз при добавлении счета пользователем номер в таблице будет увеличиваться и использоваться для нового счета. Для обработки всех таблиц в базе данных эта таблица должна иметь три столбца: имя таблицы, имя пользователя и последнее значение ключа. Следует периодически проверять наличие достаточного количества номеров у каждого человека.

- Создание таблицы со столбцами *table name* (имя таблицы) и *last key value* (последнее значение ключа).

  Одна строка в этой таблице содержит последний использованный номер счета. Каждый раз, когда пользователь добавляет счет, выполняется новое подключение, номер в таблице увеличивается, и изменение немедленно подтверждается. Увеличенный номер может использоваться для нового счета. Другие пользователи будут способны использовать номера счета, так как только что произошло обновление строки отдельной транзакцией, которая была практически моментальной.

- Вероятно, лучшее решение состоит в использовании столбца со значением AUTOINCREMENT (автоприращение) по умолчанию.

Например:

```
CREATE TABLE orders (
   order_id INTEGER NOT NULL DEFAULT AUTOINCREMENT,
   order_date DATE,
   primary key( order_id )
)
```

При вставке в таблицу, если значение для столбца автоприращения не определено, генерируется уникальное значение. Если значение определено, то оно и будет использоваться. Если значение является большим, чем текущее максимальное значение для столбца, то это значение будет использоваться как начальная точка для последующих вставок. Значение последней вставленной в столбец автоприращения строки доступно как глобальная переменная **@@identity**.

Уникальные значения в реплицированных базах данных

При репликации базы данных и наличии возможности добавления элементов более чем одним пользователем должны использоваться другие способы.

Для получения дополнительной информации см. раздел "Репликация и параллельная обработка" на стр. 137.

Операторы определения данных и параллельная обработка

Операторы определения данных, изменяющие всю таблицу, такие как CREATE INDEX, ALTER TABLE и TRUNCATE TABLE, отклоняются всякий раз, когда таблица оператора в данный момент используется другим подключенным пользователем. Эти операторы определения данных могут выполняться в течение длительного времени, и сервер базы данных не будет обрабатывать запросы к этой таблице во время обработки команд.

Оператор CREATE TABLE не приводит к возникновению конфликтов параллельной обработки.

Операторы GRANT, REVOKE и SET OPTION также не вызывают конфликтов параллельной обработки. Эти команды затрагивают любые новые операторы SQL, посланные серверу базы данных, но не затрагивают существующие отдельные операторы.

Операторы GRANT and REVOKE недоступны для пользователя, если он подключен к базе данных.

Операторы определения данных и реплицированные базы данных

Использование операторов определения данных в реплицированных базах данных требует особой осторожности. Для получения дополнительной информации см. отдельное руководство "*Репликация данных с помощью SQL Remote*" (*Data Replication with SQL Remote*).

Репликация и параллельная обработка

Некоторые компьютеры в сети могут быть портативными компьютерами, которые часто выносятся из офиса или подключаются к сети время от времени. Возможна ситуация, когда для таких компьютеров требуется использовать несколько приложений баз данных вне подключения к сети.

Репликация базы данных - идеальное решение этой проблемы. Используя SQL Remote или синхронизацию MobiLink, можно опубликовать информацию в консолидированной, или главной, базе данных, доступной для любого количества других компьютеров. Можно полностью управлять информацией, скопированной на любой конкретный компьютер. Любой пользователь может получить отдельные таблицы или даже части строк или столбцов таблицы. Определяя информацию, которую будет получать каждый, можно обеспечить то, что их копии базы данных содержат не больше информации, чем им необходимо.

Подробная информация относительно репликации SQL Remote и синхронизации MobiLink представлена в отдельных руководствах "*Руководство пользователя по SQL Remote*" (*SQL Remote User's Guide*) и "*Руководство пользователя по MobiLink*" (*MobiLink User's Guide*). Информация в данном разделе не является полной. Здесь, скорее, представлены понятия, непосредственно связанные с вопросами параллельной обработки и блокировки.

SQL Remote и MobiLink позволяют проводить обновление реплицированных баз данных из центральной консолидированной базы данных, а также обновлять эти же центральные данные по результатам транзакций, обработанных на удаленной машине. Так как обновление может происходить в обоих направлениях, эта возможность называется **двунаправленной репликацией**.

Поскольку результаты транзакций могут затронуть консолидированную базу данных независимо от того, обработаны они на центральной машине или на удаленной, необходимо разрешение выполнения параллельных транзакций.

Транзакции могут выполняться одновременно на различных машинах. Они даже могут использовать одни и те же данные. Тем не менее, в этом случае машины могут быть физически не связанными. Удаленная машина может не иметь возможности подключиться к консолидированной базе данных и получить информацию о каком-либо виде блокировки или изменении строк. Таким образом, установка блокировок не может предотвратить несогласованности так, как это происходит в случае обработки всех транзакций на одном сервере.

Еще одна трудность обусловлена тем, что не каждая удаленная машина может хранить полную копию базы данных. Рассмотрим транзакцию, выполненную непосредственно в главной консолидированной базе данных. Она может затронуть строки в двух или более таблицах. Та же транзакция может не выполниться в удаленной базе данных, поскольку не обеспечено наличие копий одной или обеих затронутых таблиц на этой машине. Даже если такие таблицы существуют, они могут и не содержать точно ту же информацию – это зависит от того, насколько давно синхронизировалась информация в этих двух базах данных.

С целью учета вышеуказанных ограничений репликация основана не на транзакциях, а скорее на операциях. **Операция** – это изменение одной строки в таблице. Это изменение может быть результатом выполнения оператора UPDATE, INSERT или DELETE. Операция, вызванная операторами UPDATE или DELETE, идентифицирует начальные значения каждого столбца, а транзакция, вызванная операторами INSERT или UPDATE, записывает конечные значения.

Транзакция может вызвать одну, более одной или ни одной операции. Одна операция не может быть результатом двух или более транзакций. Если две транзакции изменяют таблицу, то будут выполнены две или более соответствующих операции.

Если операция следует из транзакции, обработанной на удаленном компьютере, то она должна быть передана консолидированной базе данных так, чтобы информация могла быть объединена. Если, с другой стороны, операция следует из транзакции на компьютере с консолидированной базой данных, то эта операция, возможно, должна быть передана только *некоторым* удаленным узлам, а не всем. Так как каждый удаленный узел может содержать точную копию части полной базы данных, SQL Remote передает операцию удаленному узлу только в том случае, если она затрагивает эту часть базы данных.

Репликация на основе журнала транзакций

SQL Remote использует механизм репликации **на основе журнала транзакций**. При активизации SQL Remote на машине выполняется сканирование журнала транзакций с целью определения операций, которые требуется передать, и подготавливается одно или более сообщений.

SQL Remote может передать эти сообщения между компьютерами разными способами. Программа может создать файлы, содержащие сообщения, и разместить их в определенной папке. Другим способом является использование SQL Remote для передачи сообщений посредством любого из наиболее распространенных протоколов обмена сообщениями. Скорее всего, можно использовать существующую почтовую систему.

При объединении операций из удаленных узлов в консолидированную базу данных могут возникнуть конфликты. Например, два пользователя, каждый на отдельном удаленном узле, возможно, изменили одно и то же значение в одной и той же таблице. Принимая во внимание, что встроенные в Adaptive Server Anywhere средства блокировки могут устранить конфликт между параллельными транзакциями, обрабатываемыми на одном сервере, в то же время следует отметить, что невозможно автоматически устранить все конфликты между двумя удаленными пользователями, если они оба имеют полномочия на изменение одного и того же значения.

Администратор базы данных может избежать этой потенциальной проблемы, соответствующим образом спроектировав базу данных или создав алгоритм разрешения конфликтов. Например, можно решить, что только один пользователь будет ответственен за обновление определенной области значений в конкретной таблице. Если такое ограничение непрактично, то вместо этого можно использовать средства разрешения конфликтов SQL Remote и реализовать триггеры и процедуры, которые разрешают конфликты способом, подходящим для обрабатываемых данных.

SQL Remote предоставляет инструментальные средства и средства программирования, которыми следует воспользоваться для максимально полного использования возможностей репликации баз данных. Для получения дальнейшей информации см. документы "*Руководство пользователя по SQL Remote*" (*SQL Remote User's Guide*) и "*Руководство пользователя по MobiLink*" (*MobiLink User's Guide*).

Резюме

Транзакции и блокировки являются лишь второстепенными по важности в отношении связей между таблицами. Целостность и производительность любой базы данных могут выиграть от разумного использования блокировок и аккуратного построения транзакций. Оба этих параметра существенны для создания баз данных, которые должны выполнять большое количество команд одновременно.

Транзакции группируют операторы SQL в логические блоки. Любой из них можно завершить либо откатом всех произведенных изменений, либо подтверждением этих изменений, делая их, таким образом, постоянными.

Транзакции важны для восстановления данных в случае отказа системы. Они также играют основную роль при сочетании операторов из параллельных транзакций.

Для повышения производительности одновременно должно выполняться большое количество транзакций. Каждая транзакция состоит из компонентов операторов SQL. Если две или больше транзакции должны быть выполнены одновременно, сервер базы данных должен спланировать выполнение индивидуальных операторов. Параллельные транзакции потенциально могут представить новые, несогласованные результаты, которые не могли возникнуть в случае последовательного выполнения тех же транзакций.

Существует много типов возможной несогласованности, но особенно важны четыре характерных типа. Эти типы упомянуты в стандарте ISO SQL/92 и на их основе определены уровни изоляции.

- **Неверное чтение.** Одна транзакция читает данные, измененные, но еще не подтвержденные другой транзакцией.
- **Чтение без повторения.** Транзакция читает одну и ту же строку дважды и получает различные значения.
- **Фантомная строка.** Транзакция дважды выбирает строки, используя некоторый критерий, и находит новые строки во втором результирующем наборе.
- **Потеря обновления.** Изменения строки, произведенные одной транзакцией, полностью потеряны, потому что другой транзакции было разрешено сохранить изменение, основанное на более ранних данных.

Расписание называют сериализуемым в том случае, когда результат выполнения операторов согласно расписанию совпадает с тем, который мог быть достигнут выполнением каждой из транзакций последовательно. Расписания обычно называются **корректными**, если они являются сериализуемыми. Выполнение сериализуемого расписания не приводит к появлению ни одного из вышеуказанных типов несогласованности.

Блокировка определяет количество и типы разрешенного взаимодействия. Adaptive Server Anywhere предоставляет четыре уровня блокировки: уровни изоляции 0, 1, 2 и 3. При самой высокой изоляции, уровня 3, Adaptive Server Anywhere обеспечивает сериализуемость расписания, что означает, что результат выполнения всех транзакций является эквивалентным результату их последовательного выполнения.

Однако блокировки, установленные одной транзакцией, могут препятствовать работе других транзакций. Вследствие этой проблемы рекомендуется использовать более низкие уровни изоляции всякий раз, когда наличие несогласованности допустимо.

Высокая изоляция, примененная в целях увеличения согласованности данных, часто ведет к замедлению параллельной обработки, снижению эффективности базы данных при обработке параллельных транзакций. Поэтому часто требуется найти правильное соотношение важности требований по совместимости и производительности, чтобы определить наиболее подходящий уровень изоляции для каждой операции.

Противоречивые требования к блокировке между различными транзакциями могут привести к полному блокированию или взаимоблокировке. Adaptive Server Anywhere содержит механизмы для работы в таких ситуациях и предоставляет средства управления ими.

Транзакции на более высоких уровнях изоляции, однако, *не всегда* влияют на параллельную обработку. Препятствия для других транзакций возникают только в тех случаях, когда им требуется доступ к блокированным строкам. Увеличить эффективность параллельной обработки можно путем аккуратного проектирования базы данных и транзакций. Например, можно сократить время сохранения блокировки, разделив одну транзакцию на две более коротких, или может оказаться, что добавление индекса позволяет транзакции работать на более высоких уровнях изоляции с меньшим количеством блокировок.

Широкое распространение портативных компьютеров означает, что базу данных, возможно, потребуется реплицировать. Репликация - чрезвычайно удобное средство Adaptive Server Anywhere, но по отношению к параллельной обработке оно требует отдельного рассмотрения. Эти темы рассматриваются в отдельном руководстве.

ГЛАВА 5

Контроль и повышение производительности

Об этой главе

В этой главе описывается процесс контроля и повышения производительности базы данных.

Содержание

Советы по достижению наивысшей производительности

Adaptive Server Anywhere автоматически обеспечивает высокую производительность. Однако следующие советы помогут добиться наибольшей эффективности работы продукта.

Использование журнала транзакций

Может возникнуть впечатление, что без журнала транзакций быстродействие Adaptive Server Anywhere повысится из-за необходимости поддержания меньшего количества информации на диске. На самом же деле, все наоборот. Журнал транзакций не только в значительной степени обеспечивает защиту, но и радикально повышает производительность.

При работе без журнала транзакций Adaptive Server Anywhere приходится устанавливать контрольные точки по завершении каждой транзакции. Запись этих изменений требует значительных ресурсов.

При наличии журнала транзакций Adaptive Server Anywhere записывает только примечания, подробно описывающие изменения по мере их возникновения. Adaptive Server Anywhere записывает информацию обо всех новых страниц базы данных за наиболее короткое возможное время. Контрольные точки позволяют убедиться, что информация введена в файл базы данных, что она достоверна и актуальна.

Совет

Всегда используйте журнал транзакций. Он помогает защитить данные и значительно повысить производительность.

Если имеется возможность хранить журнал транзакций на ином физическом устройстве, нежели содержащем основной файл базы данных, это еще больше повысит производительность. Как правило, дополнительной головке дискового накопителя не требуется стремиться к поиску конца журнала транзакций.

Увеличение размера кэша

Adaptive Server Anywhere сохраняет недавно использованные страницы в кэше. Если при запросе необходимо обратиться к странице больше одного раза, или если другой подключенный пользователь запрашивает ту же страницу, то ее можно обнаружить уже присутствующей в памяти, избежав необходимости считывания информации с диска. Это особо относится к базам данных, использующим шифрование и потому требующим большего размера кэша, чем базы данных без шифрования.

Если размер кэша слишком мал, то Adaptive Server Anywhere не может хранить страницы в памяти достаточно долго, чтобы воспользоваться вышеуказанными преимуществами.

В UNIX, Windows NT, Windows 95/98 и других 32-разрядных операционных системах Windows сервер базы данных динамически изменяет размер кэша до необходимого уровня. Однако размер кэша все-таки ограничен объемом физически доступной памяти и объемом памяти, используемой другими приложениями.

В Windows CE и Novell NetWare размер кэша задается в командной строке при запуске сервера базы данных. Проверьте, что для кэша базы данных выделен максимально возможный объем памяти, с учетом требований параллельно запущенных приложений и процессов. Например, производительность баз данных, использующих Java-объекты, при большом размере кэша значительно увеличивается. При использовании в базе данных Java минимальный размер кэша составляет 8 Мб.

Совет

Увеличение размера кэша, как правило, значительно повышает производительность, поскольку извлечение информации из памяти происходит во много раз быстрее, чем при ее считывании с диска. Для увеличения размера кэша может быть целесообразным приобретение RAM большего объема.

Для получения дополнительной информации см. раздел "Использование кэша для повышения производительности" на стр. 150.

Нормализация табличной структуры

Как правило, информация в каждом столбце таблицы должна зависеть исключительно от значения первичного ключа. Если это не так, тогда в одной таблице может содержаться несколько копий одинаковых данных. В этом случае таблицу необходимо нормализовать.

При нормализации уменьшается процент дублирования в реляционной базе данных. Представьте, например, что служащие компании работают в нескольких разных офисах. Для нормализации базы данных попробуйте разместить информацию об офисах (например, адреса и основные номера телефонов) в отдельной таблице, вместо копирования всей этих сведений для каждого служащего.

Однако положительные стороны нормализации не следует переоценивать. Если дублированной информации немного, то, возможно, удобнее сохранить ее в таком виде и не нарушить целостности данных, используя триггеры или другие ограничения.

Эффективное использование индексов

При выполнении запроса Adaptive Server Anywhere выбирает способ обращения к каждой таблице. Использование индексов позволяет значительно ускорить процедуру доступа. Если сервер базы данных не может найти подходящий индекс, то он прибегает к последовательному сканированию всей таблицы – процессу, занимающему порой очень много времени.

Представьте, например, что нужно найти информацию о человеке в очень большой базе данных, а известно только его имя или только фамилия. Если индекса нет, то Adaptive Server Anywhere сканирует всю таблицу целиком. Однако если созданы два индекса (в одном из которых первой идет фамилия, а в другом – имя), то Adaptive Server Anywhere сначала сканирует индексы и, как правило, находит нужную информацию быстрее.

Использование индексов

Несмотря на то, что индексы позволяют Adaptive Server Anywhere очень эффективно находить требуемые данные, соблюдайте определенную осторожность при их добавлении. Наличие индексов требует выполнения дополнительных операций всякий раз при вставке, удалении или обновлении строки, поскольку Adaptive Server Anywhere также должен обновить все задействованные индексы.

Добавлять индексы следует только тогда, когда они действительно обеспечат Adaptive Server Anywhere более эффективный доступ к данным. Например, добавление индекса целесообразно, если он позволит избежать ненужного последовательного сканирования большой таблицы. Однако если при добавлении строк в таблицу эффективность важнее, чем оперативный поиск информации, использование индексов должно быть сведено к минимуму.

Для получения дополнительной информации см. раздел "Индексы" на стр. 329.

Использование подходящего размера страниц

При использовании больших размеров страниц Adaptive Server Anywhere считывает информацию баз данных более эффективно. Например, большие размеры страниц можно использовать, если имеется большая база данных, или при последовательном обращении к информации. Большие размеры страниц предоставляют также и другие преимущества, включая повышение коэффициента разветвления индексов, уменьшение числа индексных уровней и возможность создания таблиц с большим числом столбцов.

Размер страниц существующей базы данных изменить нельзя. В этом случае необходимо создать новую базу данных и использовать флаг `-p` *dbinit* для указания размера страницы. Например, следующая команда создает базу данных со страницами на 4 Кб каждая.

```
dbinit -p 4096 new.db
```

С другой стороны, преимущества, предоставляемые меньшими размерами страниц, часто остаются незамеченными. Действительно, страницы меньшего размера содержат меньше информации и могут привести к неэффективному использованию свободного места, особенно в случаях, когда вставляются строки по размеру немногим больше половины страницы. Однако маленькие размеры страницы позволяют Adaptive Server Anywhere задействовать меньший объем ресурсов, поскольку в кэше того же размера можно хранить большее число страниц. Страницы небольшого размера особенно полезны, если база данных ведется на небольших компьютерах с ограниченным объемом памяти. Они могут также помочь в ситуациях, когда база данных в основном используется для получения разрозненных сведений из разных местоположений.

Для получения дополнительной информации о больших размерах страниц см. раздел "Установка максимального размера страницы" (Setting a maximum page size) на стр. 13 в документе "*Руководство по администрированию баз данных ASA" (ASA Database Administration Guide)*.

Хранение разных файлов на разных устройствах

Дисководы функционируют гораздо медленнее, чем современные процессоры или оперативная память. Часто простое ожидание считывания или записи страниц является причиной замедления работы сервера базы данных.

При размещении разных физических файлов базы данных на разных физических устройствах почти всегда происходит повышение производительности базы данных. Например, в то время как один дисковод занят подкачкой страниц базы данных в кэш и обратно, другое устройство может вести запись в файл журнала.

Следует отметить, что для получения этих преимуществ устройства должны быть независимы. Единственный диск, разделенный на меньшие логические диски, вряд ли обеспечит какие-либо преимущества.

Adaptive Server Anywhere использует четыре типа файлов:

1. база данных (*.db*),
2. журнал транзакций (*.log*),
3. зеркальная копия журнала транзакций (*.mlg*),
4. временный файл (*.tmp*).

Файл базы данных хранит все содержимое базы данных. В одном файле может содержаться одна база, либо можно добавить до 12 dbspaces, являющихся дополнительными файлами, содержащими ту же самую базу данных. Выберите для него соответствующее местоположение.

Файл журнала транзакций требуется для восстановления информации в базе данных в случае сбоя. Для дополнительной защиты копию этого файла можно хранить в третьем типе файла, называемом **файлом зеркальной копии журнала транзакций**.

Adaptive Server Anywhere записывает одну и ту же информацию одновременно в каждый из этих файлов.

Совет

Помещением файла зеркальной копии журнала транзакций (если он используется) на физически отдельный диск обеспечивается лучшая защита от дискового сбоя, и Adaptive Server Anywhere работает быстрее, поскольку имеется возможность эффективной записи в журнал транзакций и в его файл зеркальной копии. Для указания местоположения файлов журнала транзакций и зеркальной копии журнала транзакций используйте утилиту *dblog* командной строки или утилиту Change Log File Settings в Sybase Central.

Возможно, Adaptive Server Anywhere нуждается в большем количестве свободного пространства, нежели доступно в кэше, для таких операций, как сортировка и формирование объединений. Когда серверу необходимо это пространство, он интенсивно его использует. Общая производительность базы данных становится сильно зависимой от быстродействия устройства, содержащего четвертый тип файлов, - **временные файлы**.

Совет

Если временный файл находится на устройстве с высоким быстродействием и физически отдельном от устройства, содержащего файл базы данных, то Adaptive Server Anywhere будет работать быстрее. Это происходит из-за того, что многие операции, требующие обязательного использования временного файла, также запрашивают большой объем информации из базы данных. Размещение информации на двух отдельных дисках обеспечивает одновременное выполнение этих операций.

В Windows Adaptive Server Anywhere проверяет следующие переменные среды (в указанном порядке) для определения каталога, в который будет помещен временный файл.

1 ASTMP

2 TMP

3 TMPDIR

4 TEMP

Если ни одна из них не определена, Adaptive Server Anywhere помещает временный файл в текущий каталог, который нельзя назвать подходящим местоположением для наивысшей производительности.

В UNIX Adaptive Server Anywhere проверяет переменную среды ASTMP, чтобы определить каталог для размещения временного файла.

Если переменная среды ASTMP не определена, Adaptive Server Anywhere помещает временный файл в каталог */tmp/.SQLAnywhere*.

Если на компьютере установлено достаточное количество быстродействующих устройств, то производительность может быть повышена посредством размещения каждого из этих файлов на отдельном устройстве. Можно даже разделить базу данных на множественные dbspaces, размещенные на отдельных устройствах. В этом случае сгруппируйте таблицы в отдельных dbspaces так, чтобы общие операции соединения считывали информацию из различных файлов.

Подобная же стратегия включает в себя размещение временных файлов и файлов базы данных на RAID-устройстве или в наборе томов с чередованием Windows NT. Хотя такие устройства работают как логический диск, они значительно повышают эффективность, распределяя файлы по нескольким физическим дискам и обращаясь к информации при помощи многосекционной головки.

Для получения дополнительной информации о рабочих таблицах см. раздел "Использование рабочих таблиц при обработке запросов" на стр. 157.

Для получения информации о восстановлении данных см. раздел "Резервное копирование и восстановление данных" (Backup and Data Recovery) на стр. 295 в документе "*Руководство по администрированию баз данных ASA" (ASA Database Administration Guide)*.

Для получения информации о журналах транзакций и утилите *dbcc* см. раздел "Параметры утилиты журнала транзакций" (Transaction log utility options) на стр. 497 в документе "*Руководство по администрированию баз данных ASA" (ASA Database Administration Guide)*.

Выключение режима автоматического подтверждения

Если приложение выполняется в режиме автоматического подтверждения, то Adaptive Server Anywhere рассматривает каждый оператор как отдельную транзакцию. Фактически это эквивалентно добавлению оператора COMMIT в конце каждой из команд.

Вместо применения режима автоматического подтверждения попробуйте сгруппировать команды так, чтобы каждая группа выполняла одну логическую задачу. Если режим автоматического подтверждения выключен, то после каждой логической группы команд необходимо выполнять явное подтверждение. Также помните, что если логические транзакции большие, то могут иметь место блокировка и взаимная блокировка.

Затраты ресурсов при применении режима автоматического подтверждения особенно высоки, если не используется файл журнала транзакций. Каждый оператор вызывает создание контрольной точки, что включает запись на диск большого количества страниц с информацией.

В каждом прикладном интерфейсе имеется свой собственный способ установки режима автоматического подтверждения. Для интерфейсов Open Client, ODBC и JDBC режим автоматического подтверждения установлен по умолчанию.

Для получения дополнительной информации об автоматическом подтверждении см. раздел "Установка режима автоматического или ручного подтверждения" (Setting autocommit or manual commit mode) на стр. 45 в документе "Р*уководство по программированию ASA" (ASA Programming Guide)*.

Проверка фрагментации файлов, таблиц и индексов

Производительность снижается при избыточной фрагментации файлов, таблиц или индексов. Это становится все более важным по мере увеличения размера базы данных. Adaptive Server Anywhere содержит хранимые процедуры, генерирующие информацию о фрагментации файлов, таблиц и индексов.

Если производительность заметно снизилась, попробуйте перестроить базу данных так, чтобы фрагментация таблиц и/или индексов была уменьшена. Также для уменьшения фрагментации файла саму базу данных можно поместить на логический диск, либо периодически запускать одну из доступных утилит Windows.

Иногда может потребоваться повысить производительность базы данных без ее полной перестройки. Например, полная перестройка может оказаться невозможной вследствие требований непрерывного доступа к базе данных. Для дефрагментации строк в таблице или сжатия индексов, разреженных вследствие применения команды DELETE, можно использовать оператор REORGANIZE TABLE. Реорганизация таблицы может уменьшить общее число страниц, используемых для сохранения таблицы и ее индексов, а также число уровней в индексном дереве.

Для получения дополнительной информации об обнаружении и исправлении фрагментации файлов, таблиц и индексов см. раздел "Фрагментация" на стр. 165.

Использование методов массовых операций

При загрузке в базу данных информации очень большого объема удобно пользоваться специальным инструментарием, предусмотренным для этих задач.

При загрузке больших файлов индексы к таблице более эффективно создавать после того, как данные загружены.

Для получения информации о повышении производительности массовых операций см. раздел "Учет производительности при перемещении данных" на стр. 410.

Использование параметра WITH EXPRESS CHECK при проверке правильности таблиц

Если оказывается, что подтверждение правильности больших баз данных с кэшем маленького размера занимает слишком много времени, то можно воспользоваться одним из двух параметров для сокращения времени выполнения данной операции. Скорость проверки правильности таблиц может значительно увеличиться при использовании параметра WITH EXPRESS CHECK с оператором VALIDATE TABLE или параметра `-fx` с утилитой проверки правильности.

Для получения информации о повышении производительности при проверке правильности баз данных см. раздел "Повышение производительности при проверке правильности баз данных" (Improving performance when validating databases) на стр. 323 в документе "*Руководство по администрированию баз данных ASA" (ASA Database Administration Guide)*.

Использование средств сжатия Adaptive Server Anywhere

Активизирование средств сжатия для одного или всех подключений и установка минимального размера, до которого пакет может быть сжат, при некоторых условиях значительно повышают производительность Adaptive Server Anywhere.

При активизировании средств сжатия увеличивается объем информации, хранящейся в пакетах данных, и уменьшается, таким образом, число пакетов, требующихся для передачи определенного набора данных. При уменьшении числа пакетов данные можно передавать быстрее.

Указание порога сжатия позволяет выбрать минимальный размер пакетов данных, которые необходимо сжать. На выбор оптимального значения порога сжатия могут влиять разные факторы, включая тип и скорость передачи используемой сети.

Для экономии времени и во избежание возможного разочарования, перед проведением сжатия передаваемых данных в рабочей среде следует провести анализ быстродействия в сети с использованием конкретного приложения.

Для получения дополнительной информации см. раздел "Параметр сжатия подключения" (Compress connection parameter) на стр. 175 в документе "*Руководство по администрированию баз данных ASA" (ASA Database Administration Guide)* и раздел "Параметр подключения CompressionThreshold" (CompressionThreshold connection parameter) на стр. 176 в документе "*Руководство по администрированию баз данных ASA" (ASA Database Administration Guide)*.

Использование кэша для повышения производительности

Кэш базы данных – это область памяти, используемая сервером базы данных для хранения страниц базы данных с целью быстрого повторного доступа. Чем больше страниц доступно в кэше, тем меньшее количество раз серверу базы данных приходится считывать данные с диска. Поскольку чтение данных с диска происходит медленно, объем доступного кэша часто является ключевым фактором при определении производительности.

Управлять размером кэша базы данных можно с использованием командной строки сервера базы данных при запущенной базе данных.

Динамическое изменение размеров кэша

Adaptive Server Anywhere обеспечивает автоматическое изменение размеров кэша базы данных. В разных операционных системах возможности такого изменения различны. В операционных системах Windows NT, Windows 95/98 и UNIX кэш увеличивается и сжимается. В других операционных системах кэш может только увеличиваться в размере. Более подробная информация представлена в разделах ниже.

Полное **динамическое изменение размеров кэша** позволяет обеспечить то, что на производительность сервера базы данных не влияет неправильное распределение памяти. Кэш расширяется, если сервер базы данных может с пользой задействовать его больший объем, до тех пор, пока доступна память, и уменьшается, когда он не требуется, с тем, чтобы сервер базы данных не оказывал негативное влияние на работу других приложений в системе. Конечно, эффективность динамического изменения размеров кэша ограничена физической памятью, доступной в системе.

Динамическое изменение размеров кэша во многих ситуациях устраняет необходимость явного конфигурирования кэша базы данных, что упрощает работу с Adaptive Server Anywhere.

Динамического изменения размеров кэша на Windows CE, Novell NetWare или Linux не предусмотрено. При использовании кэша Address Windowing Extensions (AWE) функция динамического изменения размеров кэша отключена.

Для получения дополнительной информации о кэшах AWE см. раздел "Параметр –cw командной строки" (–cw command-line option) на стр. 139 в документе "*Руководство по администрированию баз данных ASA" (ASA Database Administration Guide)*.

Ограничение памяти, используемой при кэшировании

Начальные минимальный и максимальный размеры кэша контролируются из командной строки сервера базы данных.

- **Начальный размер кэша.** Начальный размер кэша задается посредством указания серверу базы данных параметра -c командной строки. Значения по умолчанию следующие:

- **Windows CE.** Используется следующая формула:

  ```
  max( 2M, min( размер БД , физическая-память ) )
  ```

  где *размер БД* - полный размер файла базы данных или запущенных файлов, и *физическая-память* - 25 % физической памяти на машине.

- **Windows NT, Windows 95/98, Netware.** Используется следующая формула:

  ```
  max( 2M, min( размер БД , физическая-память ) )
  ```

  где *размер БД* - полный размер файла базы данных или запущенных файлов, и *физическая-память* - 25 % физической памяти на машине.

  Если на Windows 2000 или Windows XP используется кэш AWE, используется следующая формула:

  ```
  min( 100% доступной памяти-128MB, размер БД )
  ```

  Кэш AWE не используется, если это значение меньше, чем 3 Гб - 128 Мб.

  Для получения информации о кэше AWE см. раздел "Параметр –cw командной строки" (–cw command-line option) на стр. 139 в документе "*Руководство по администрированию баз данных ASA" (ASA Database Administration Guide)*.

- **UNIX.** Минимум 8 Мб.

Для получения информации о начальном размере кэша для UNIX см. раздел "Динамическое изменение размера кэша (UNIX)" на стр. 152.

- **Максимальный размер кэша.** Максимальным размером кэша можно управлять указанием серверу базы данных ключа -c командной строки. Значение по умолчанию основано на эвристике, зависящей от физической памяти машины.
- **Минимальный размер кэша.** Минимальный размер кэша задается посредством указания серверу базы данных параметра `-cl` командной строки. По умолчанию минимальный размер кэша такой же, как начальный размер кэша.

Можно также отключить функцию динамического изменения размеров кэша, используя параметр `-ca` командной строки.

Для получения дополнительной информации о параметрах командной строки см. раздел "Сервер базы данных" (The database server) на стр. 128 в документе "*Руководство по администрированию баз данных ASA" (ASA Database Administration Guide)*.

Динамическое изменение размеров кэша (Windows NT, Windows 95/98)

В Windows NT и Windows 95/98 сервер базы данных раз в минуту оценивает состояние кэша, анализирует операционную статистику и рассчитывает оптимальный размер кэша. Сервер вычисляет целевой размер кэша, использующий всю незадействованную в настоящий момент физическую память, за исключением приблизительно 5 Мб, оставляемых свободными для системных нужд. Целевой размер кэша не может быть меньше, чем заданный явно или неявно минимальный размер кэша. Целевой размер кэша не может превышать заданный явно или неявно максимальный размер кэша, либо сумму размеров всех открытых файлов базы данных и временных файлов.

Во избежание колебаний размера кэша сервер базы данных увеличивает размер кэша посредством приращения. Вместо немедленной корректировки размера кэша до целевого значения при каждой настройке размер кэша изменяется на 75% разницы между текущим и целевым размером кэша.

Для поддержки больших размеров кэша Windows 2000 и Windows XP могут использовать Address Windowing Extensions (AWE) путем указания параметра -cw командной строки при запуске сервера базы данных. Кэши AWE не поддерживают динамическое изменение размеров кэша.

Для получения дополнительной информации см. раздел "Параметр –cw командной строки" (–cw command-line option) на стр. 139 в документе "*Руководство по администрированию баз данных ASA" (ASA Database Administration Guide)*.

Динамическое изменение размеров кэша (UNIX)

В UNIX для управления размером кэша сервер базы данных использует область свопинга и память. Область свопинга - системный ресурс на большинстве (но не на всех) операционных систем UNIX. В этом разделе сумма памяти и области свопинга называется **системными ресурсами**. Для получения подробной информации см. документацию к операционной системе.

При запуске база данных распределяет указанный максимальный размер кэша из системных ресурсов. Часть кэша загружается в память (начальный размер кэша), а остаток сохраняется как область свопинга.

Общая сумма системных ресурсов, используемых сервером базы данных, является постоянной, пока сервер базы данных не прекращает работу, но само соотношение сохраняется в записях об изменении памяти. Сервер базы данных каждую минуту анализирует кэш и операционную статистику. Если сервер базы данных занят и требует ресурсов, он может переместить страницы кэша из области свопинга в память. Если ресурсы не требуются, то сервер может переместить их из памяти в область свопинга.

Начальный размер кэша

По умолчанию начальный размер кэша назначен с использованием эвристики, основанной на доступных системных ресурсах. Начальный размер кэша всегда в 1,1 раза меньше, чем общий размер базы данных.

Если начальный размер кэша больше, чем 3/4 доступных системных ресурсов, то сервер базы данных завершает работу и выводит сообщение об ошибке “Not Enough Memory” (Недостаточно памяти).

Максимальный размер кэша

Максимальный кэш должен быть меньше доступных системных ресурсов на машине. По умолчанию максимальный размер кэша назначается с использованием эвристики, основанной на доступных системных ресурсах и общей физической памяти на машине.

При указании максимального размера кэша, превышающего доступные системные ресурсы, сервер базы данных завершает работу и выводит сообщение об ошибке “Not Enough Memory” (Недостаточно памяти). При указании максимального размера кэша, превышающего доступную память, сервер предупреждает о снижении производительности, но не завершает работу.

Сервер базы данных распределяет весь *максимальный* размер кэша из системных ресурсов и не освобождает его до завершения работы. Следует убедиться в том, что выбирается максимальный размер кэша, обеспечивающий высокую эффективность работы Adaptive Server Anywhere, оставляя место для других приложений. Формула для заданного по умолчанию максимального размера кэша - эвристика, где делается попытка достичь этого равновесия. Требуется только настроить значение, если значение по умолчанию не подходит конкретной системе.

При указании максимального размера кэша менее 8 Мб запустить Java-приложения невозможно. Низкие максимальные размеры кэша будут влиять на производительность.

Минимальный размер кэша

По умолчанию минимальный размер кэша на UNIX - 8 Мб.

Контроль размера кэша

Следующая статистика была добавлена к средствам контроля производительности NT и к функциям свойств базы данных.

- **CurrentCacheSize** - текущий размер кэша в килобайтах;
- **MinCacheSize** - минимально допустимый размер кэша в килобайтах;
- **MaxCacheSize** - максимально допустимый размер кэша в килобайтах;
- **PeakCacheSize** - пиковый размер кэша в килобайтах.

Для получения дополнительной информации об этих свойствах см. раздел "Свойства на уровне сервера" (Server-level properties) на стр. 608 в документе “*Руководство по администрированию баз данных ASA” (ASA Database Administration Guide)*.

Для получения информации о контроле производительности см. раздел "Контроль производительности базы данных" на стр. 159.

Использование ключей для повышения эффективности запросов

Первичные и внешние ключи, изначально используемые в целях проверки правильности, также могут повысить производительности базы данных.

Пример

Следующий пример иллюстрирует действие первичных ключей, направленное на ускорение выполнения запросов.

```
SELECT *
FROM employee
WHERE emp_id = 390
```

Самым простым способом для выполнения сервером этого запроса будет просмотр всех 75 строк в таблице **employee** и проверка номера служащего в каждой строке, чтобы убедиться, что он равен 390. Это не займет много времени, поскольку служащих всего 75, однако последовательное сканирование таблиц с несколькими тысячами записей может продлиться довольно долго.

Средства контроля за ссылочной целостностью, выраженные каждым первичным или внешним ключом, реализованы в Adaptive Server Anywhere принудительно через справку того или иного индекса, неявно созданную при объявлении каждого первичного или внешнего ключа. Столбец *emp_id* является первичным ключом для таблицы *employee*. Соответствующий индекс первичного ключа позволяет выполнить быстрый поиск служащего с номером 390. Этот быстрый поиск занимает почти одинаковое количество времени при просмотре 100 или 1 000 000 строк в таблице *employee*.

Использование первичных ключей для повышения эффективности запросов

С использованием первичных ключей эффективность повышается по следующему оператору:

SELECT * FROM employee WHERE emp_id = 390

Информация на закладке Plan

Закладка Plan в окне Results содержит следующую информацию:

```
employee <employee>
```

Каждый раз, когда имя в круглых скобках в описании PLAN закладки Plan совпадает с названием таблицы, это означает, что для повышения производительности для таблицы используется первичный ключ.

Использование внешних ключей для повышения эффективности запросов

В следующем запросе перечислены заказы от клиента с кодом клиента 113:

```
SELECT *
FROM sales_order WHERE cust_id = 113
```

Информация на закладке Plan

Закладка Plan в окне Results содержит следующую информацию:

```
sales_order <ky_so_customer>
```

Здесь *ky_so_customer* ссылается на внешний ключ, который имеется в таблице *sales_order* для таблицы *Customer*.

Разделение индексов первичных и внешних ключей

Для первичных и внешних ключей автоматически создаются отдельные индексы. Эта установка позволяет Adaptive Server Anywhere более эффективно выполнять большое количество операций. Эта функция была впервые представлена в версии 7.0.

Сортировка результатов запроса

Множество запросов имеют раздел ORDER BY, который позволяет расположить строки в предсказуемом порядке. Запрос на информацию быстро выполняется посредством индексов. Например,

```
SELECT *
FROM customer
ORDER BY customer.lname
```

может использовать индекс в столбце *lname* таблицы клиентов, чтобы обращаться к строкам таблицы клиентов в алфавитном порядке по фамилии.

Запросы с разделами WHERE и ORDER BY

При одновременном наличии в запросе разделов WHERE и ORDER BY могут возникнуть осложнения.

```
SELECT *
FROM customer
WHERE id > 300
ORDER BY company_name
```

Сервер должен выбрать одну из двух стратегий:

1 Сканирование всей таблицы клиентов в порядке по названию организации с проверкой каждой строки на наличие кода клиента, превышающего 300.

2 Использование ключ в столбце *id* только для просмотра организаций с кодом, превышающим 300. После этого результаты должны быть отсортированы по названию организации.

Если значений кода, превышающих 300, очень мало, то вторая стратегия предпочтительнее, поскольку потребуется просканировать и отсортировать только несколько строк. Если большинство значений кода превышает 300, то предпочтительнее использовать первую стратегию, потому что она не требует сортировки.

Для получения дополнительной информации о сортировке см. раздел "Раздел ORDER BY: сортировка результатов запроса" на стр. 216 или раздел "Раздел GROUP BY: группирование результатов запроса " на стр. 209.

Использование рабочих таблиц при обработке запросов

Рабочие таблицы содержат данные временных результирующих наборов, создаваемых в процессе выполнения запроса. Рабочие таблицы используются, когда Adaptive Server Anywhere сочтет использование одной из них более целесообразным, чем использование альтернативных стратегий. Обычно при использовании рабочей таблицы времени на вывод первых нескольких строк требуется больше, однако временные затраты на вывод всех строк в некоторых случаях могут быть значительно снижены. Из-за этого различия Adaptive Server Anywhere выбирает различные стратегии, основанные на параметре OPTIMIZATION_GOAL. Значением по умолчанию является первая строка. При установке “первая строка” Adaptive Server Anywhere пытается избежать использования рабочих таблиц. При установке “все строки” Adaptive Server Anywhere использует рабочие таблицы, если они снижают затраты ресурсов на выполнение запроса.

Рабочие таблицы используются в следующих случаях:

Использование рабочих таблиц

- Если запрос имеет раздел ORDER BY, GROUP BY, OR DISTINCT, и Adaptive Server Anywhere не использует индекс для сортировки строк. Если существует подходящий индекс, и параметр OPTIMIZATION_GOAL установлен в значение “первая строка”, то Adaptive Server Anywhere не использует рабочие таблицы. Однако если OPTIMIZATION_GOAL установлен в значение "все строки", то затраты ресурсов могут быть значительнее при выводе всех строк запроса с использованием индекса, чем при построении рабочей таблицы и сортировке строк. Если OPTIMIZATION_GOAL установлен в значение "все строки", Adaptive Server Anywhere выбирает стратегию с наименьшими затратами. Для разделов GROUP BY and DISTINCT рабочие таблицы используются для алгоритмов на основе хэша, но обычно они более эффективны при выборке всех строк запроса.
- Когда выбран алгоритм соединения хэша, рабочие таблицы используются для хранения промежуточных результатов (если ввод не может быть размещен в памяти). Кроме того, одна рабочая таблица используется для сохранения результатов соединения.
- Если курсор открыт с чувствительными значениями, рабочая таблица создается для хранения идентификаторов строк и первичных ключей базовых таблиц. Эта рабочая таблица заполняется в прямом направлении по мере того, как строки выводятся из запроса. Однако если выбрана последняя строка курсора, то заполняется вся таблица.
- Если курсор открыт с нечувствительной семантикой, рабочая таблица заполняется результатами запроса, когда запрос открыт.
- Если UPDATE выполняется для нескольких строк, и обновляемый столбец появляется в разделе WHERE обновления или в индексе, используемом для обновления.
- Если UPDATE или DELETE для нескольких строк имеют подзапрос в разделе WHERE, который ссылается на обновляемую таблицу.

- При выполнении INSERT из оператора SELECT оператор SELECT ссылается на таблицу вставки.
- Если INSERT, UPDATE или DELETE выполняются для нескольких строк, и для таблицы определен соответствующий триггер, который может быть запущен во время операции.

В этих случаях записи, задействованные при выполнении операции, входят в рабочую таблицу. При определенных обстоятельствах, например, при управляемых клавиатурой курсорах, в рабочей таблице строится временный индекс. Операция помещения требуемых записей в рабочую таблицу может потребовать длительного времени, прежде чем появятся результаты запроса. Создание индексов, которые могут использоваться для сортировки в первом случае, указанном выше, сокращает время получения первых нескольких строк. Однако общее время вывода всех строк может увеличиться при использовании рабочих таблиц, поскольку они допускают алгоритмы запроса, основанные на хэшировании и сортировке со слиянием. Эти алгоритмы используют последовательный ввод/вывод с более высоким быстродействием, чем произвольный ввод/вывод, используемый при сканировании индекса.

Оптимизатор запросов в сервере базы данных анализирует каждый запрос для установления факта повышения производительности при использовании рабочей таблицы. Расширения к оптимизатору в новых версиях Adaptive Server Anywhere могут усовершенствовать план доступа к запросам. Для применения этих средств оптимизации никаких действий со стороны пользователя не требуется.

Примечания

Случаи INSERT, UPDATE и DELETE, описанные выше, как правило, не влияют на производительность, поскольку являются единовременными операциями. Однако при возникновении ошибок можно перефразировать команду так, чтобы избежать конфликта и формирования рабочей таблицы. Но это не всегда возможно.

Контроль производительности базы данных

Adaptive Server Anywhere предоставляет набор статистики, которую можно использовать для контроля производительности базы данных. Клиентские приложения, доступные из Sybase Central, могут обращаться к статистике с использованием функций. Кроме того, сервер делает эти статистические данные доступными для системного монитора Windows NT.

В данном разделе описываются процедуры получения доступа к статистике производительности и другим связанным статистическим данным из клиентских приложений, способы контроля производительности базы данных с использованием Sybase Central и системного монитора Windows NT, а также способы проверки фрагментации файлов, таблиц и индексов.

Получение статистики базы данных из клиентского приложения

Adaptive Server Anywhere предоставляет набор системных функций, используемых для доступа к информации по принципам "по подключению", "по базе данных", "по всему серверу". Множество типов доступной информации охватывает как статическую информацию (такую, как имя сервера), так и подробную статистику, связанную с производительностью (например, использование диска и памяти).

Функции для получения системной информации

Следующие функции могут использоваться для получения системной информаци:

- **Функция property** - выводит значение определенного свойства для всего процессора базы данных;
- **Функция connection_property** - выводит значение определенного свойства для определенного подключения или для текущего подключения (по умолчанию);
- **Функция db_property** - выводит значение определенного свойства для определенной базы данных или для текущей базы данных (по умолчанию).

В качестве аргумента нужно указать только название свойства, которое необходимо получить. Функции возвращают значение для текущего сервера, подключения или базы данных.

Для получения дополнительной информации см. раздел “Функция PROPERTY” (PROPERTY function) на стр. 159 в документе “*Справочник по SQL для ASA*” (*ASA SQL Reference Manual*), раздел “Функция CONNECTION_PROPERTY” (CONNECTION_PROPERTY function) на стр. 111 в документе “*Справочник по SQL для ASA*” (*ASA SQL Reference Manual*) и раздел “Функция DB_PROPERTY” (DB_PROPERTY function) на стр. 124 в документе “*Справочник по SQL для ASA*” (ASA *SQL Reference Manual*).

Полный список свойств, доступных для системных функций, см. в разделе “Системные функции” (System functions) на стр. 98 в документе “*Справочник по SQL для ASA*” (*ASA SQL Reference Manual*).

Примеры

Следующий оператор устанавливает переменную *server_name* в имя

текущего сервера:

```
SET server_name = property( 'name' )
```

Следующий запрос возвращает код пользователя для текущего подключения:

```
SELECT connection_property( 'userid' )
```

Следующий запрос возвращает имя файла для корневого файла текущей базы данных:

```
SELECT db_property( 'file' )
```

Повышение эффективности запросов

Для повышения производительности клиентское приложение, контролирующее работу базы данных, должно использовать функцию *property_number*, чтобы идентифицировать поименованное свойство и затем использовать данный номер для получения статистики в дальнейшем. Следующий набор операторов иллюстрирует процесс из Interactive SQL:

```
CREATE VARIABLE propnum INT ;
CREATE VARIABLE propval INT ;
SET propnum = property_number( 'cacheread' );
SET propval = property( propnum )
```

Полученные таким образом имена свойств доступны для многих других статистических данных базы данных, от количества операций записи страницы журнала транзакций и количества выполненных контрольных точек до количества прочтений базовых страниц индекса из кэша памяти.

Многие из этих видов статистики можно просмотреть в форме графа из инструментального средства управления базой данных Sybase Central.

Контроль статистики базы данных из Sybase Central

При помощи системного монитора Sybase Central статистика любого сервера базы данных Adaptive Server Anywhere, к которому можно подключиться в Sybase Central, представляется в виде графа. Все статистические данные в Sybase Central представлены в папке Statistics.

Возможности системного монитора включают:

- обновление в режиме реального времени (в корректируемых интервалах);
- создание легенды с цветовыми обозначениями и изменяемым размером;
- настройку свойств внешнего вида.

При работе с системным монитором следует отметить, что он использует фактические запросы к серверу для сбора статистических данных, поэтому сам монитор влияет на некоторые статистические данные (например, на количество обращений к кэшу в секунду). В качестве альтернативного варианта повышенной точности статистические данные сервера можно представить в виде графа, используя системный монитор Windows NT.

Для получения информации об установке свойств см. раздел "Установка свойств объектов базы данных" на стр. 34.

Открытие системного монитора Sybase Central

Системный монитор можно вывести в правой области окна Sybase Central в открытой папке Statistics.

❖ Открытие системного монитора

1 Откройте папку Statistics требуемого сервера.

2 В правой области окна нажмите закладку Performance Monitor.

Примечание

Системный монитор представляет в виде графа только предварительно добавленные статистические данные.

См. также разделы:

- "Добавление и удаление статистических данных" на стр. 161;
- "Настройка системного монитора Sybase Central" на стр. 162;
- "Контроль статистики базы данных из системного монитора Windows NT" на стр. 163.

Добавление и удаление статических данных

❖ Добавление статистических данных в системный монитор Sybase Central

1 Откройте папку Statistics.

2 Проверьте, что отображена страница Statistics Items в правой области окна.

3 Щелкните правой кнопкой мыши на статистических данных, не отображаемых в текущий момент в виде графа, и выберите Add to Performance Monitor во всплывающем меню.

❖ Удаление статистических данных из системного монитора Sybase Central

1 Выполните одну из следующих операций.

- При работе в папке Statistics (с отображением закладки Statistics Items в правой области окна) щелкните правой кнопкой мыши на статистических данных, представляемых в текущий момент в виде графа.
- При работе в системном мониторе щелкните правой кнопкой мыши на нужных статистических данных в легенде.

2 Выберите Remove from Performance Monitor во всплывающем меню.

Совет

Статистические данные можно добавлять и удалять из системного монитора в окне свойств статистики.

См. также разделы:

- "Открытие системного монитора Sybase Central" на стр. 161;
- "Настройка системного монитора Sybase Central" на стр. 162;
- "Контроль статистики базы данных из системного монитора Windows NT" на стр. 163.

Настройка системного монитора Sybase Central

Конфигурацию системного монитора Sybase Central можно изменить; можно выбрать тип используемого им графа и интерва времени между обновлениями, вносимыми в граф.

❖ Выбор типа графа

1 Выберите Tools ➤ Options.

2 В диалоге Options нажмите закладку Chart.

3 Выберите тип графа.

❖ Установка интервала обновления

1 Выберите Tools ➤ Options.

2 В диалоге Options нажмите закладку Chart.

3 Переместите слайдер для отражения нового значения времени (или введите значение непосредственно в имеющееся текстовое поле).

См. также разделы:

- "Открытие системного монитора Sybase Central" на стр. 161;
- "Добавление и удаление статистических данных" на стр. 161;
- "Контроль статистики базы данных из системного монитора Windows NT" на стр. 163.

Контроль статистики базы данных из системного монитора Windows NT

В качестве альтернативы использованию системного монитора Sybase Central можно использовать системный монитор Windows NT (включенный в Windows NT).

Монитор Windows NT работает только при установке NT-к-NT и имеет два преимущества:

- Он предлагает больше статистических данных о производительности (в основном связанных с сетевыми подключениями).
- В отличие от монитора Sybase Central, монитор Windows NT неинтрузивен. Вместо выполнения запросов к серверу в нем применена схема разделяемой памяти, поэтому он не оказывает влияния на сами статистические данные.

Полный список статистики производительности представлен в разделе "Статистика системного монитора" (Performance Monitor statistics) на стр. 594 в документе "*Руководство по администрированию баз данных ASA*" (*ASA Database Administration Guide*).

❖ Использование системного монитора Windows NT

1. С запущенными процессором Adaptive Server Anywhere или базой данных запустите Performance Monitor:
   - Выберите Start (Пуск)➤Programs (Программы)➤Administrative Tools (Common) (Администрирование (Общее))➤ Performance monitor (Системный монитор).
2. Выберите Edit ➤ Add To Chart или нажмите кнопку с изображением знака "плюс" на панели инструментов. Появляется диалог Add to Chart.
3. Из списка объектов выберите один из следующих объектов:
   - **Adaptive Server Anywhere Connection** (Подключение Adaptive Server Anywhere). Контроль производительности для одного подключения. Выберите подключение для контроля из отображенного списка.
   - **Adaptive Server Anywhere Database** (База данных Adaptive Server Anywhere). Контроль производительности для одной базы данных. Выберите базу данных для контроля из отображенного списка.
   - **Adaptive Server Anywhere Engine** (Процессор Adaptive Server Anywhere). Контроль производительности всего сервера.

В поле Counter отображается список статистических данных, доступных для просмотра.

4. В списке Counter нажмите на статистические данные для их просмотра. Для выбора множественных статистических данных при нажатии кнопки мыши удерживайте нажатыми клавиши CTRL или SHIFT.
5. Если выбран пункт "Adaptive Server Anywhere Connection" или "Adap-

tive Server Anywhere Database", выберите экземпляр из поля Instance.

6 Для просмотра описания выбранного счетчика нажмите Explain.

7 Для отображения счетчика нажмите Add.

8 Нажмите Done, когда будут выделены все необходимые для просмотра счетчики.

☞ Для получения дополнительной информации о системном мониторе Windows NT см. интерактивную справку программы.

Фрагментация

По мере внесения изменений в базу данных ее файл, таблицы и индексы фрагментируются. Фрагментация может стать причиной снижения производительности. Adaptive Server Anywhere предоставляет информацию, которую можно использовать для просмотра уровня фрагментации в файлах, таблицах и индексах.

В этом разделе описывается процедура проверки фрагментации в файлах, таблицах и индексах, а также процедура дефрагментации файлов и таблиц.

Фрагментация файлов

Избыточная фрагментация жесткого диска может привести к снижению производительности. Фрагментация диска становится все более важной по мере увеличения размера базы данных.

Сервер базы данных определяет количество фрагментов файла в каждом dbspace при запуске базы данных на Windows NT. При количестве фрагментов, превышающем 1, сервер отображает следующую информацию в окне сообщений сервера:

```
Database file "mydatabase.db" consists of nnn fragments
(Файл базы данных "mydatabase.db" состоит из nnn фрагментов)
```

Можно также получить число фрагментов файла базы данных, используя свойство базы данных *DBFileFragments*.

Для получения дополнительной информации см. раздел "Свойства на уровне базы данных" (Database-level properties) на стр. 612 в документе "*Руководство по администрированию баз данных ASA*" (*ASA Database Administration Guide*).

❖ Устранение ошибок фрагментации файлов

- Поместите саму базу данных в дисковый раздел.
- Периодически запускайте одну из утилит дефрагментации диска, доступных в Windows.

Фрагментация таблиц

Если строки не расположены в базе данных непрерывно или разделены по нескольким страницам, производительность снижается, поскольку эти строки требуют выполнения дополнительных обращений к страницам. Фрагментация таблиц отличается от фрагментации файлов.

Adaptive Server Anywhere резервирует дополнительный участок памяти на каждой странице на случай добавления информации в строки. Если в результате обновления строка не умещается на первоначально выделенном ей участке, то она разбивается, и в первоначальном местоположении строки появляется указатель на другую страницу, где строка размещена полностью. Например, заполнение пустых строк операторами UPDATE или вставка новых столбцов в таблицу могут привести к чрезмерному разбиению строки. Чем больше строк размещается на отдельных страницах, тем больше времени требуется при обращении к дополнительным страницам.

Для получения информации о степени фрагментации таблиц базы данных можно воспользоваться хранимой процедурой *sa_table_fragmentation*. Для запуска этой процедуры необходимо обладать полномочиями администратора БД. Следующий оператор вызывает хранимую процедуру *sa_table_fragmentation*:

```
CALL sa_table_fragmentation (['table_name'
[,'owner_name']])
```

Если параметры не заданы, появляется информация обо всех таблицах; в противном случае процедура выполняется только для поименованной таблицы.

В окне Interactive SQL Results результирующий набор для таблицы показан следующим образом:

TableName, rows, row_segments, segs_per_row

Дефрагментация таблиц

Следующие процедуры полезны при обнаружении пониженной производительности из-за высокой степени фрагментации таблиц. Выгрузка и перезагрузка базы данных является наиболее комплексным методом, поскольку происходит дефрагментация всех таблиц, включая системные. Для дефрагментации заданных таблиц или частей таблиц запустите REORGANIZE TABLE. Реорганизация таблиц не прерывает обращения к базе данных.

❖ Дефрагментация всех таблиц в базе данных

1 Выгрузите базу данных.

2 Перезагрузите базу данных для восстановления дискового пространства и повышения производительности.

Для получения дополнительной информации о выгрузке базы данных см. раздел "Утилита dbunload командной строки" (The dbunload command-line utility) на стр. 502 в документе "*Руководство по администрированию баз данных ASA*" (*ASA Database Administration Guide*).

Для получения дополнительной информации о восстановлении базы данных см. раздел "Утилита REBUILD" (The REBUILD utility) на стр. 486 в документе "*Руководство по администрированию баз данных ASA*" (*ASA Database Administration Guide*).

❖ Дефрагментация отдельных таблиц

- Выполните оператор REORGANIZE TABLE.

Для получения дополнительной информации см. раздел "Оператор REORGANIZE TABLE" (REORGANIZE TABLE statement) на стр. 472 в документе "*Справочник по SQL для ASA*" (*ASA SQL Reference Manual)*.

Фрагментация индексов

Индексы предназначены для ускорения поиска в заданных столбцах, однако они могут стать фрагментированными, если в таблице с индексом выполняется большое количество операторов DELETE. При частом обращении к индексу и недостаточном для хранения всего индекса размере кэша это может привести к снижению производительности.

Хранимая процедура *sa_index_density* предоставляет информацию о степени фрагментации в индексах базы данных. Для выполнения этой процедуры необходимо обладать полномочиями администратора БД. Следующий оператор вызывает хранимую процедуру *sa_index_density*:

```
CALL sa_index_density (['table_name'[,'owner_name']])
```

Если параметры не заданы, то появляется информация для всех таблиц. В противном случае процедура выполняется только для поименованной таблицы.

В окне Interactive SQL Results результирующий набор для таблицы показан следующим образом:

TableName (имя таблицы), IndexName (имя индекса), LeafPages (базовые страницы), Density (плотность)

Density (плотность) – это доля между 0 и 1. Для индексов с большим количеством базовых страниц предпочтительны более высокие значения плотности.

Если индекс сильно фрагментирован, можно запустить REORGANIZE TABLE. Индекс также можно удалить и восстановить заново. Однако если индекс является первичным ключом, то потребуется также удалить и восстановить индексы внешнего ключа.

Для получения дополнительной информации см. раздел "Оператор REORGANIZE TABLE" (REORGANIZE TABLE statement) на стр. 472 в документе "*Справочник по SQL для ASA*" (*ASA SQL Reference Manual)*.

Для получения дополнительной информации об удалении индекса см. раздел "Удаление индексов" на стр. 61.

Контроль эффективности запросов

Adaptive Server Anywhere предоставляет инструментальные средства для проверки эффективности запросов. Полная документация о каждом инструментальном средстве представлена в файле *Readme.txt*, помещенным в ту же папку, что и инструментальное средство.

fetchtst

Функция: определяет время, необходимое для получения результирующего набора.

Местоположение: *SQL Anywhere 8\Samples\Asa\PerformanceFetch*

odbcfet

Функция: определяет время, необходимое для получения результирующего набора.

Эта функция подобна *festchtst*, но с меньшим количеством функциональных возможностей.

Местоположение: *SQL Anywhere 88\Samples\Asa\PerformanceFetch*

instest

Функция: определяет время, необходимое для вставки строк в таблицу.

Местоположение: *SQL Anywhere 8\Samples\Asa\PerformanceInsert*

trantest

Функция: измеряет загрузку, которая может быть обработана в данной конфигурации сервера при данных структуре базы данных и наборе транзакций.

Местоположение: *SQL Anywhere 8\Samples\Asa\PerformanceTransaction*

Для получения информации о системных процедурах, измеряющих время выполнения запросов, см. раздел "Системная процедура sa_get_request_profile" (sa_get_request_profile system procedure) на стр. 653 в документе "*Справочник по SQL для ASA*" (*ASA SQL Reference Manual*) и раздел "Системная процедура sa_get_request_times" (sa_get_request_times system procedure) на стр. 654 в документе "*Справочник по SQL для ASA*" (*ASA SQL Reference Manual*).

Профилирование процедур базы данных

Профилирование процедур показывает время выполнения хранимых процедур, функций, событий и триггеров. Также можно узнать время выполнения по каждой строке процедуры. Используя данные профилирования базы данных, можно определить процедуры для изменения настроек в целях повышения производительности базы данных.

При включенном профилировании Adaptive Server Anywhere контролирует используемые хранимые процедуры, функции, события и триггеры, следит за временем их выполнения и количеством вызовов каждого.

Данные профилирования сохраняются в памяти сервером, и их можно просмотреть в Sybase Central на закладке Profile или в Interactive SQL. С момента включения профилирования база данных собирает данные профилирования до момента его отключения, либо до тех пор, пока не будет выключен сервер.

Для получения дополнительной информации о получении данных профилирования в Interactive SQL см. раздел "Просмотр данных профилирования процедур в Interactive SQL" на стр. 175.

Включение профилирования процедур

Включить профилирование можно либо в Sybase Central, либо в Interactive SQL. Для включения и использования профилирования процедур необходимо обладать полномочиями администратора БД.

❖ **Включение профилирования (Sybase Central)**

1. Выполните подключение к базе данных с использованием полномочий администратора БД.
2. Выберите базу данных в левой области окна.
3. В меню File выберите Properties. Появляется окно свойств базы данных.
4. На закладке Profiling выберите Enable Profiling on This Database.
5. Для закрытия окна свойств нажмите OK.

Примечание

Для включения профилирования можно также щелкнуть правой кнопкой мыши на базе данных в Sybase Central.

Выберите Profiling ➤ Profile во всплывающем меню.

❖ **Включение профилирования (SQL)**

1. Выполните подключение к базе данных с использованием полномочий администратора БД.
2. Вызовите хранимую процедуру *sa_server_option* с параметром ON.

Например, введите:

```
CALL sa_server_option ( 'procedure_profiling', 'ON')
```

Сброс профилирования процедур

При сбросе профилирования в базе данных удаляются старые данные, и немедленно начинается сбор новых данных о процедурах, функциях, событиях и триггерах.

В следующих разделах предполагается, что пользователь уже выполнил подключение к базе данных с использованием полномочий администратора БД и включил профилирование процедур.

❖ **Сброс профилирования (Sybase Central)**

1. Выберите базу данных в левой области окна.
2. В меню File выберите Properties. Появляется окно свойств базы данных.
3. На закладке Profiling нажмите Reset Now.
4. Для закрытия окна свойств нажмите OK.

Примечание

Для сброса профилирования можно также щелкнуть правой кнопкой мыши на базе данных в Sybase Central.

Выберите Profiling ➤ Reset Profiling во всплывающем меню.

❖ **Сброс профилирования (SQL)**

1. Вызовите хранимую процедуру *sa_server_option* с параметром RESET. Например, введите:

```
CALL sa_server_option ('procedure_profiling',
'RESET')
```

Выключение профилирования процедур

По окончании работы с данными профилирования можно выполнить либо выключение, либо очистку результатов. При выключении профилирования в базе данных прекращается сбор данных профилирования, и информация, собранная до этого момента, остается на закладке Profile в Sybase Central. При очистке результатов профилирования в базе данных профилирование выключается, и удаляются все данные профилирования на закладке Profile в Sybase Central.

❖ **Выключение профилирования (Sybase Central)**

1 Выберите базу данных в левой области окна.

2 В меню File выберите Properties. Появляется окно свойств базы данных.

3 На закладке Profiling снимите флаг Enable Profiling on This Database.

4 Для закрытия окна свойств нажмите OK.

Примечание

Для выключения профилирования можно также нажать правой кнопкой мыши на базе данных в Sybase Central.
Выберите Profiling ➤ Profile во всплывающем меню.

❖ **Выключение профилирования (SQL)**

1 Вызовите хранимую процедуру *sa_server_option* с параметром OFF. Например, введите:

```
CALL sa_server_option ('procedure_profiling',
'OFF')
```

❖ **Очистка результатов профилирования (Sybase Central)**

1 Выберите базу данных в левой области окна.

2 В меню File выберите Properties.

Появляется окно свойств базы данных.

3 На закладке Profiling нажмите Clear Profiling.

Если профилирование включено, то можно только очистить его результаты.

4 Для закрытия окна свойств нажмите OK.

Примечание

Для очистки результатов профилирования можно также нажать правой кнопкой мыши на базе данных в Sybase Central.
Выберите Profiling ➤ Clear Profiling во всплывающем меню.

❖ **Выключение профилирования (SQL)**

1 Вызовите хранимую процедуру *sa_server_option* с параметром CLEAR. Например, введите:

```
CALL sa_server_option ('procedure_profiling',
'CLEAR')
```

Просмотр данных профилирования процедур в Sybase Central

При профилировании процедур предоставляется различная информация: информация по всей базе данных, об особых типах объектов или об определенной процедуре. Эта информация может быть отображена следующими способами:

- подробно для всех профилированных объектов в пределах базы данных;
- подробно для всех хранимых процедур и функций;
- подробно для всех событий;
- подробно для всех триггеров;
- подробно для отдельных профилированных объектов;

Для просмотра данных профилирования необходимо выполнить подключение к базе данных и включить профилирование.

При просмотре данных профилирования для всей базы данных появляются следующие столбцы:

- **Name** (Имя). Указывает имя объекта.
- **Owner** (Владелец). Указывает владельца объекта.
- **Milliseconds** (Миллисекунды). Указывает общее время выполнения для каждого объекта.
- **Calls** (Вызовы). Указывает количество вызовов каждого объекта.
- **Table** (Таблица). Указывает, к какой таблице принадлежит триггер (этот столбец появляется на закладке Profile базы данных).

В этих столбцах показаны общие данные профилирования для всех процедур, выполненных в пределах базы данных. Поскольку одна процедура может вызвать другие процедуры, в списке может содержаться больше элементов, чем было непосредственно вызвано пользователями.

❖ Просмотр общих данных профилирования для хранимых процедур и функций

1. Разверните базу данных в левой области окна.
2. Выберите папки Procedures и Functions в левой области окна.
   Список всех хранимых процедур и функций базы данных появляется на закладке Details в правой области окна.
3. Нажмите закладку Profile в правой области окна.
   Данные профилирования обо всех хранимых процедурах и функциях в пределах базы данных появляются на закладке Profile.

❖ Просмотр общих данных профилирования для событий

1. Разверните базу данных в левой области окна.
2. Выберите папку Events в левой области окна.
   Список всех событий базы данных появляется на закладке Details в правой области окна.
3. Нажмите закладку Profile в правой области окна.
   Данные профилирования обо всех событиях в пределах базы данных появляются на закладке Profile.

❖ **Просмотр общих данных профилирования для триггеров**

1 Разверните базу данных в левой области окна.

2 Выберите папку Tables в левой области окна.

Список всех таблиц базы данных появляется на закладке Details в правой области окна.

3 В левой области окна откройте таблицу, для которой нужно выполнить профилирование триггеров.

4 Откройте папку Triggers в левой области окна.

Список всех триггеров для таблицы появляется на закладке Details.

5 Нажмите закладку Profile в правой области окна.

Данные профилирования обо всех триггерах таблицы появляются на закладке Profile.

Просмотр данных профилирования для заданной процедуры

Adaptive Server Anywhere предоставляет информацию профилирования процедур об отдельных хранимых процедурах, функциях, событиях и триггерах. В Sybase Central отображаемая информация об отдельных процедурах и обо всех хранимых процедурах, функциях, событиях или триггерах в пределах базы данных различается.

При просмотре данных профилирования для заданной процедуры появляются следующие столбцы:

- **Calls** (Вызовы). Указывает количество вызовов объекта.
- **Milliseconds** (Миллисекунды). Указывает общее время выполнения для каждого объекта.
- **Line** (Строка). Указывает номер строки рядом с каждой строкой процедуры.
- **Source** (Источник). Построчно отображает процедуру SQL.

Процедура разбита построчно, так что представлено время выполнения для каждой строки, по которому можно понять, какие из строк требуется изменить с целью уменьшения времени выполнения процедуры. Для получения доступа к данным профилирования процедуры необходимо выполнить подключение к базе данных, включить профилирование с использованием полномочий администратора БД.

❖ **Просмотр данных профилирования для хранимой процедуры или функции**

1 Разверните базу данных в левой области окна.

2 Выберите папки Procedures и Functions в левой области окна.

Список всех хранимых процедур и функций базы данных появляется на закладке Details в правой области окна.

3 Выберите хранимую процедуру или функцию, для которой необходимо выполнить профилирование, в левой области окна.

4 Нажмите закладку Profile в правой области окна.

Данные профилирования о заданной хранимой процедуре или

функции появляются на закладке Profile в правой области окна.

❖ **Просмотр данных профилирования для события**

1 Разверните базу данных в левой области окна.

2 Выберите папку Events в левой области окна.

Список всех событий в пределах базы данных появляется на закладке Details в правой области окна.

3 Выберите событие для профилирования в левой области окна.

4 Нажмите закладку Profile в правой области окна.

Данные профилирования о заданном событии появляются на закладке Profile в правой области окна.

❖ **Просмотр данных профилирования для триггеров**

1 Разверните базу данных в левой области окна.

2 Выберите таблицу, для которой необходимо выполнить профилирование триггеров, в левой области окна.

3 Откройте папку Triggers в левой области окна.

Список всех триггеров в пределах таблицы появляется на закладке Details в правой области окна.

4 Выберите триггер, для которого необходимо выполнить профилирование, в левой области окна.

5 Нажмите закладку Profile в правой области окна.

Данные профилирования о заданном триггере появляются на закладке Profile в правой области окна.

Просмотр данных профилирования процедур в Interactive SQL

Для просмотра данных профилирования процедур используются хранимые процедуры. В Sybase Central и Interactive SQL просматриваемая информация профилирования одинакова.

Хранимая процедура *sa_procedure_profile_summary* предоставляет информацию обо всех процедурах в пределах базы данных. Эту процедуру можно использовать для просмотра данных профилирования для хранимых процедур, функций, событий и триггеров в пределах того же результирующего набора. Следующие параметры ограничивают строки, возвращаемые процедурой.

- **p_object_name.** Определяет имя объекта для профилирования.
- **p cowner_name.** Определяет владельца, для объектов которого необходимо выполнить профилирование.
- **p_table_name.** Определяет таблицу, для которой необходимо выполнить профилирование триггеров.
- **p_object_type.** Определяет тип объекта для профилирования. Можно выбрать четыре следующих параметра. Выбор одного из этих значений

ограничивает результирующий набор только одним объектом указанного типа.

- **P -** хранимая процедура
- **F** - функция
- **T** - триггер
- **E** - событие

- **p_ordering.** Определяет порядок сортировки результирующего набора.

Помните, что перечисленных элементов может быть больше, чем непосредственно вызванных пользователями, поскольку одна процедура может вызывать другую процедуру.

В следующих разделах предполагается, что пользователь выполнил подключение к базе данных с использованием полномочий администратора БД и включил профилирование процедур.

❖ Просмотр общих данных профилирования для всех процедур

1 Выполните хранимую процедуру *sa_procedure_profile_summary*. Например, введите:

```
CALL sa_procedure_profile_summary
```

2 В меню SQL выберите Execute.

Результирующий набор с информацией обо всех процедурах в базе данных появляется на закладке Results в окне Results.

Для получения дополнительной информации о хранимой процедуре sa_procedure_profile_summary см. раздел "Системная процедура sa_procedure_profile_summary" (sa_procedure_profile_summary system procedure) на стр. 659 в документе "*Справочник по SQL для ASA*" (*ASA SQL Reference Manual*).

Просмотр данных профилирования для заданной процедуры в Interactive SQL

Хранимая процедура *sa_procedure_profile* предоставляет информацию об отдельных строках в пределах заданных процедур. Результирующий набор включает номер строки, время выполнения и процент от полного времени выполнения для строк в пределах процедур. Для ограничения строк, возвращаемых процедурой, можно использовать следующие параметры:

- **p_object_name.** Определяет имя объекта для профилирования.
- **p_owner_name.** Определяет владельца, для объектов которого необходимо выполнить профилирование.
- **p_table_name.** Определяет таблицу, для которой необходимо выполнить профилирование триггеров. Если параметры в запросе отсутствуют, то процедура возвращает данные профилирования для всех вызванных процедур.

❖ Просмотр данных профилирования для заданных строк в пределах процедур

1 Выполните хранимую процедуру *sa_procedure_profile*. Например,

введите:

```
CALL sa_procedure_profile
```

2 В меню SQL выберите Execute.

Результирующий набор с данными профилирования для отдельных строк процедуры появляется на закладке Results в окне Results.

Для получения дополнительной информации о хранимой процедуре sa_procedure_profile см. раздел "Системная процедура sa_procedure_profile" (sa_procedure_profile system procedure) на стр. 658 в документе "*Справочник по SQL для ASA*" (*ASA SQL Reference Manual*).

ЧАСТЬ 2

Работа с базами данных

В данной части Руководства описывается механизм выполнения общих задач в Adaptive Server Anywhere.

ГЛАВА 6

Запросы: выбор данных в таблице

Об этой главе

При помощи оператора SELECT из базы данных можно извлечь данные. Этот оператор можно использовать для извлечения поднабора строк из одной или более таблиц, а также для извлечения поднабора столбцов из одной или нескольких таблиц.

В этой главе представлены основные сведения об использовании операторов SELECT для одной таблицы. Более подробное описание применения оператора SELECT содержится далее в этом Руководстве.

Содержание

Обзор запросов

При помощи запросов можно затребовать данные из базы данных и получить результаты. Этот процесс также называется извлечением данных. Все SQL-запросы выражены с использованием оператора SELECT.

Разделы в запросах

Операторы SELECT строятся из разделов. В следующем синтаксисе SELECT каждая новая строка является отдельным разделом. Здесь приведены только наиболее общие разделы.

SELECT список-выбора
[**FROM** выражение-таблицы]
[**WHERE** условие-поиска]
[**GROUP BY** имя-столбца]
[**HAVING** условие-поиска]
[**ORDER BY** { выражение | целое число }]

В операторе SELECT имеются следующие разделы:

- Раздел SELECT определяет столбцы, из которых необходимо извлечь данные. Это единственный раздел, наличие которого обязательно при выполнении оператора SELECT.
- Раздел FROM определяет таблицы, из которых извлекаются столбцы. Он требуется во всех запросах, в которых данные извлекаются из таблиц. Операторы SELECT без разделов FROM имеют другое значение, поэтому в данной главе они не рассматриваются.
- Раздел ON определяет способ соединения таблиц в разделе FROM. Он используется только для запросов данных из нескольких таблиц и в данной главе не рассматривается.
- Раздел WHERE определяет строки в таблицах, которые необходимо просмотреть.
- Раздел GROUP BY позволяет объединять данные.
- Раздел HAVING определяет строки, в которых собираются агрегированные данные.
- Раздел ORDER BY сортирует строки в результирующем наборе. (По умолчанию строки возвращаются из реляционных баз данных без определенного порядка.)

Большинство разделов является дополнительным, но если они включены, тогда они должны появляться в правильном порядке.

Для получения дополнительной информации о синтаксисе оператора SELECT см. раздел "Оператор SELECT" (SELECT statement) на стр. 490 в документе *"Справочник по SQL для ASA"* (*ASA SQL Reference Manual*).

В данной главе рассматривается только следующий набор запросов:

- Запросы только с одной указанной таблицей в разделе FROM. Для получения информации о запросах для нескольких таблиц см. раздел "Соединения: извлечение данных из нескольких таблиц" на стр. 223.
- Запросы без разделов GROUP BY, HAVING или ORDER BY. Для получения информации об этих запросах см. раздел "Сведение, группирование и сортировка результатов запроса" на стр. 203.

SQL-запросы

В данном Руководстве операторы SELECT и другие операторы SQL показаны с каждым разделом на отдельной строке и с ключевыми словами SQL в верхнем регистре. Последнее, однако, не обязательно. Ключевые слова SQL можно набрать в любом регистре в любом случае, и строки можно прерывать в любой точке.

Ключевые слова и строки

Например, следующий оператор SELECT находит в таблице Contact имена и фамилии контактных лиц, проживающих в Калифорнии.

```
SELECT first_name, last_name
FROM Contact
WHERE state = 'CA'
```

Данный оператор можно ввести и так (этот вариант, впрочем, менее "читабелен"):

```
SELECT first_name,
last_name from contact
where state
 = 'CA'
```

Учет регистра в строках и идентификаторах

В базах данных Adaptive Server Anywhere ввод идентификаторов (имен таблиц, имен столбцов и т.д.) не зависит от выбранного регистра.

В строках регистр не учитывается по умолчанию, поэтому обозначения "CA", "ca", "cA" и "Ca" эквивалентны, но если база данных создается как зависимая от регистра, тогда важно соблюдать регистр строки. В демонстрационной базе данных регистр не учитывается.

Уточнение идентификаторов

Имена идентификаторов базы данных можно **уточнить** в случае возникновения двусмысленности относительно того, на какой объект указывает ссылка. Например, демонстрационная база данных содержит несколько таблиц со столбцом, названным city, поэтому может потребоваться соотнести ссылки на *"city"* с именем таблицы. В большой базе данных для идентификации таблицы может потребовать использовать имя владельца таблицы.

```
SELECT DBA.contact.city
FROM contact
WHERE state = 'CA'
```

Поскольку в примерах данной главы показаны запросы для одной таблицы, то имена столбцов в моделях синтаксиса и примерах обычно не соотносятся с именами таблиц или их владельцев.

Эти элементы опускаются для обеспечения "прозрачности" оператора, однако само по себе включение уточняющих элементов целесообразно.

В следующих разделах этой главы синтаксис оператора SELECT рассматривается более подробно.

Раздел SELECT: определение столбцов

Список выбора

Список выбора обычно содержит ряд имен столбцов, разделенных запятыми или звездочкой, и используется в качестве краткой записи всех столбцов.

В более широком смысле список выбора может включать одно или более выражений, разделенных запятыми. Общий синтаксис списка выбора выглядит следующим образом:

SELECT *выражение [, выражение]...*

Если имя какой-либо таблицы или столбца в списке не отвечает правилам допустимых идентификаторов, тогда идентификатор необходимо заключить в двойные кавычки.

Выражения в списке выбора могут включать знак * (все столбцы), список имен столбцов, символьные строки, заголовки столбцов, а также выражения, включающие арифметические операторы. Можно также включать агрегатные функции, рассматриваемые в разделе "Сведение, группирование и сортировка результатов запроса" на стр. 203.

Для получения дополнительной информации о том, что может входить в выражения, см. раздел "Выражения" (Expressions) на стр. 14 в документе *"Справочник по SQL для ASA" (ASA SQL Reference Manual)*.

В следующих разделах представлены примеры выражений различных типов, которые можно использовать в списке выбора.

Выделение всех столбцов в таблице

В операторах SELECT знак * имеет особое значение. Он замещает все имена столбцов во всех таблицах, указанных в разделе FROM. Его можно использовать при необходимости просмотра всех столбцов в таблице с целью сокращения времени ввода и избежания ошибок при вводе.

При использовании SELECT * столбцы возвращаются в порядке, в котором они были определены при создании таблицы.

Синтаксис выделения всех столбцов в таблице следующий:

```
SELECT *
FROM выражение-таблицы
```

SELECT * находит все столбцы, имеющиеся в настоящий момент в таблице, так, что изменения в структуре таблицы, такие, как добавления, удаления или переименование столбцов, автоматически изменяют результаты SELECT *. Составление списка отдельных столбцов позволяет с большей точностью осуществлять контроль результатов.

Пример

Следующий оператор извлекает все столбцы в таблице *department* (Отделы). Раздел WHERE не включен, поэтому данный оператор извлекает каждую строку в таблице:

```
SELECT *
FROM department
```

Результаты выглядят следующим образом:

ept_id	dept_name	dept_head_id
100	R & D	501
200	Sales	902
300	Finance	1293
400	Marketing	1576
...	...	...

Те же результаты получаются путем перечисления имен всех столбцов по порядку в таблице после ключевого слова SELECT:

```
SELECT dept_id, dept_name, dept_head_id
FROM department
```

Подобно имени столбца, знак "*" может быть соотнесен с именем таблицы, как показано в следующем запросе:

```
SELECT department.*
FROM department
```

Выбор отдельных столбцов из таблицы

Для выбора отдельных столбцов в таблице используется следующий синтаксис:

SELECT *имя_столбца [, имя_столбца]...*
FROM *имя-таблицы*

Имя каждого столбца должно быть отделено запятой от имени столбца, следующего за ним, например:

```
SELECT emp_lname, emp_fname
FROM employee
```

Перегруппировка столбцов

Порядок, в котором перечислены имена столбцов, определяет порядок, в котором столбцы будут отображены. Два примера ниже показывают определение порядка столбцов при отображении. В обоих примерах найдены и отображены названия отделов и коды из всех пяти строк таблицы отделов, но в другом порядке.

```
SELECT dept_id, dept_name
FROM department
```

dept_id	dept_name
100	R & D
200	Sales
300	Finance
400	Marketing
...	...

```
SELECT dept_name, dept_id
FROM department
```

dept_name	dept_id
R & D	100
Sales	200
Finance	300
Marketing	400
...	...

Переименование столбцов с использованием псевдонимов в результатах запроса

Результаты запроса состоят из набора столбцов. По умолчанию заголовок каждого столбца является выражением, предоставленным в списке выбора.

При отображении результатов запроса заголовок каждого столбца по умолчанию является именем, присвоенным ему во время создания. Одним из следующих способов можно определить другой заголовок столбца или **псевдоним:**

SELECT *имя-столбца* **AS** *псевдоним*
SELECT *имя-столбца псевдоним*
SELECT *псевдоним = столбец-имя*

Использование псевдонимов обеспечивает удобочитаемые результаты. Например, в списке отделов можно изменить имя столбца dept_name на Department следующим образом:

```
SELECT dept_name AS Department,
   dept_id AS "Identifying Number"
FROM department
```

Department	Identifying Number
R & D	100
Sales	200
Finance	300
Marketing	400
...	...

Использование пробелов и ключевых слов в псевдониме

Псевдоним *Identifying Number* для *dept_id* заключен в двойные кавычки, так как он является идентификатором. Двойные кавычки можно также использовать при указании ключевых слов в псевдонимах. Например, следующий запрос недействителен без кавычек:

```
SELECT dept_name AS Department,
   dept_id AS "integer"
FROM department
```

Если необходимо обеспечить совместимость с Adaptive Server Enterprise, то нужно использовать псевдонимы в кавычках размером до 30 байт.

Символьные строки в результатах запроса

Рассмотренные операторы SELECT выдают результаты, состоящие исключительно из данных таблиц в разделе FROM. Строки символов могут также быть отображены в результатах запроса путем заключения их в одиночные кавычки и отделения от других элементов списка выбора запятыми.

Для включения символа кавычки в строку ему должен предшествовать другой символ кавычки.

Например:

```
SELECT 'The department''s name is' AS " ",
   Department = dept_name
FROM department
```

	Department
The department's name is (Название отдела)	R & D
The department's name is	Sales
The department's name is	Finance
The department's name is	Marketing
The department's name is	Shipping

Вычисление значений в списке выбора

Выражения в списке выбора могут быть более сложными, чем просто имена столбцов или строки. Например, в списке выбора можно выполнить расчеты с данными из числовых столбцов.

Арифметические операции

Чтобы проиллюстрировать операции с числами, которые можно выполнять в списке выбора, начнем с перечисления имен, количеств товаров на складе и цен за единицу товара в демонстрационной базе данных.

```
SELECT name, quantity, unit_price
FROM product
```

name	quantity	unit_price
Tee Shirt	28	9
Tee Shirt	54	14
Tee Shirt	75	14
Baseball Cap	112	9
...	...	...

Предположим, что существует практика пополнения товарных запасов, когда на складе остается 10 единиц товара. В следующем запросе перечислено количество каждого товара, который должен быть продан перед повторным заказом:

```
SELECT name, quantity - 10
   AS "Sell before reorder"
FROM product
```

name	Sell before reorder
Tee Shirt	18
Tee Shirt	44
Tee Shirt	65
Baseball Cap	102
...	...

Значения в столбцах можно объединять. В следующем запросе перечислены общие количества каждого товара на складе:

```
SELECT name,
   quantity * unit_price AS "Inventory value"
FROM product
```

name	Inventory value
Tee Shirt	252
Tee Shirt	756
Tee Shirt	1050
Baseball Cap	1008
...	...

Порядок выполнения арифметических операций

Если в выражении присутствуют несколько арифметических операций, то сначала рассчитываются умножение, деление и значение по модулю, после чего следует очередь вычитания и сложения. Если все арифметические операции в выражении имеют тот же самый уровень предшествования, то порядок выполнения - слева направо.

Выражения в круглых скобках имеют приоритет выполнения перед всеми остальными операциями.

Например, следующий оператор SELECT вычисляет общее количества каждого товара в описи и затем вычитает пять долларов из этого значения.

```
SELECT name, quantity * unit_price - 5
FROM product
```

Во избежание недоразумений рекомендуется использовать круглые скобки. Следующий запрос имеет то же значение и выдает те же результаты, что и предыдущий, но он может показаться проще для понимания:

```
SELECT name, ( quantity * unit_price ) - 5
FROM product
```

Для получения дополнительной информации о порядке выполнения операций см. раздел "Порядок выполнения операций" (Operator precedence) на стр. 12 в документе *"Справочник по SQL для ASA" (ASA SQL Reference Manual)*.

Строковые операции

Строки можно связывать с помощью операции конкатенации строк. В качестве обозначения операции конкатенации можно использовать знак ‖ (также в SQL/92) или знак + (поддерживается в Adaptive Server Enterprise).

Следующий пример иллюстрирует использование операции конкатенации строк в списке выбора:

```
SELECT emp_id, emp_fname || ' ' || emp_lname AS Name
FROM employee
```

emp_id	Name
102	Fran Whitney
105	Matthew Cobb
129	Philip Chin
148	Julie Jordan
...	...

Операции даты и времени

Можно использовать операции в столбцах даты и времени, однако это требует применения функций. Для получения информации о функциях SQL см. раздел *"Функции SQL" (SQL Functions)* на стр. 91 в документе *"Справочник по SQL для ASA" (ASA SQL Reference Manual)*.

Удаление дублированных результатов запроса

Дополнительное ключевое слово DISTINCT удаляет дублирующиеся строки из результатов выполнения оператора SELECT.

Если ключевое слово DISTINCT не определено, то строки могут извлекаться с дублированием. Для извлечения всех строк перед списком выбора можно определить ALL. Для обеспечения совместимости с другими реализациями SQL синтаксис Adaptive Server позволяет использовать ALL для явного запроса всех строк. ALL является значением по умолчанию.

Например, если необходимо найти все города (sity) в таблице contact, то без ключевого слова DISTINCT извлекаются 60 строк:

```
SELECT city
FROM contact
```

С использованием DISTINCT можно удалить дублированные записи. Следующий запрос возвращает только 16 строк:

```
SELECT DISTINCT city
FROM contact
```

Дублированные значения NULL

Ключевое слово DISTINCT рассматривает значения NULL как дубликаты друг друга. Другими словами, если ключевое слово DISTINCT включено в оператор SELECT, в результатах возвращается только NULL, независимо от того, сколько раз встретилось значение NULL.

Раздел FROM: определение таблиц

Раздел FROM требуется в каждом операторе SELECT, обрабатывающем данные из таблиц или представлений.

✍ Раздел FROM может включать условия JOIN, связывающие две или более таблиц, а также соединения с другими запросами (производные таблицы). Для получения информации об этих возможностях см. раздел "Соединения: извлечение данных из нескольких таблиц" на стр. 223.

Уточнение имен таблиц

В разделе FROM для таблиц и представлений всегда разрешен синтаксис имен следующего типа:

```
SELECT список-выбора
FROM владелец.имя_таблицы
```

Уточнение имен таблиц и представлений необходимо только в том случае, если существует возможность перепутать имена.

Использование корреляционных имен

В целях экономии времени на ввод имени таблицы можно назначить корреляционное имя. Корреляционное имя назначается в разделе FROM путем его ввода после имени таблицы, как показано ниже:

```
SELECT d.dept_id, d.dept_name
FROM Department d
```

Все другие ссылки на таблицу *Department*, например, в разделе WHERE, *должны* использовать корреляционное имя. Корреляционные имена должны соответствовать правилам для допустимых идентификаторов.

Раздел WHERE: определение строк

Раздел WHERE в операторе SELECT определяет условия поиска, т.е. то, какие именно строки будут извлечены. Общий формат:

SELECT *список_выбора*
FROM *список_таблиц*
WHERE *условие-поиска*

Условия поиска, также называемые уточнениями или предикатами, в разделе WHERE могут включать следующее:

- **Операции сравнения** (=, < ,> и т. д.). Например, можно перечислить всех служащих, заработная плата которых превышает 50 000 долл.:

```
SELECT emp_lname
FROM employee
WHERE salary > 50000
```

- **Области значений** (BETWEEN и NOT BETWEEN). Например, можно перечислить всех служащих, заработная плата которых варьируется между 40 000 и 60 000 долл.:

```
SELECT emp_lname
FROM employee
WHERE salary BETWEEN 40000 AND 60000
```

- **Списки** (IN, NOT IN). Например, можно перечислить всех клиентов в Онтарио, Квебеке или Манитобе:

```
SELECT company_name , state
FROM customer
WHERE state IN( 'ON', 'PQ', 'MB')
```

- **Символьные пары** (LIKE и NOT LIKE). Например, можно перечислить всех клиентов, номера телефонов которых начинаются с 415 (номер телефона хранится в базе данных как строка):

```
SELECT company_name , phone
FROM customer
WHERE phone LIKE '415%'
```

- **Неизвестные значения** (IS NULL и IS NOT NULL). Например, можно перечислить все отделы с менеджерами:

```
SELECT dept_name
FROM Department
WHERE dept_head_id IS NOT NULL
```

- **Комбинации** (AND, OR). Например, можно перечислить всех служащих, заработная плата которых превышает 50 000 долл., и имена начинаются с буквы A.

```
SELECT emp_fname, emp_lname
FROM employee
WHERE salary > 50000
AND emp_fname like 'A%'
```

Кроме того, ключевое слово WHERE может представить следующее:

- **Условия соединения Transact-SQL**. Соединения описаны в разделе "Соединения: извлечение данных из нескольких таблиц" на стр. 223.

☞ Для получения дополнительной информации об условиях поиска см. раздел "Условия поиска" (Search conditions) на стр. 23 в документе *"Справочник по SQL для ASA" (ASA SQL Reference Manual)*.

В следующих разделах описано использование разделов WHERE.

Использование операций сравнения в разделе WHERE

В разделе WHERE можно использовать операторы сравнения. Операции следуют синтаксису:

***WHERE** выражение операция-сравнения выражение*

☞ Для получения дополнительной информации об операциях сравнения см. раздел "Операции сравнения" (Comparison operators) на стр. 9 в документе "*Справочник по SQL для ASA" (ASA SQL Reference Manual)*. Для получения информации о том, из чего может состоять выражение, см. раздел "Выражения" (Expressions) на стр. 14 в документе *"Справочник по SQL для ASA" (ASA SQL Reference Manual)*.

Примечания о сравнениях

- **Порядок сортировки.** При сравнении символьных данных знак < означает "ранее в порядке сортировки", а знак > означает "позже в порядке сортировки". Порядок сортировки определяется установленным порядком сортировки при создании базы данных. Порядок сортировки можно проверить с помощью утилиты *dbinfo* командной строки в базе данных:

  ```
  dbinfo -c "uid=DBA;pwd=SQL"
  ```

  Порядок сортировки также можно проверить в Sybase Central. Информация о порядке сортировке находится на закладке Extended Information окна свойств базы данных.

- **Конечные пробелы.** При создании базы данных для целей сравнения указывается режим игнорирования или не игнорирования конечных пробелов.

  По умолчанию конечные пробелы в базах данных не игнорируются. Например, 'Dirk' - не то же самое, что 'Dirk '. Можно создавать базы данных с дополнением пробелов, так, чтобы конечные пробелы игнорировались. В базах данных Adaptive Server Enterprise конечные пробелы игнорируются по умолчанию .

- **Сравнение дат.** При сравнении дат знак < означает "ранее", а знак > означает "позже".

- **Учет регистра.** При создании базы данных указывается зависимость (или независимость) строковых сравнений от регистра.

По умолчанию в создаваемых базах данных регистр не учитывается. Например, 'Dirk' - то же самое, что и 'DIRK'. Можно установить учет регистра в базе данных. Учет регистра для баз данных Adaptive Server Enterprise является поведением по умолчанию.

Ниже приведено несколько операторов SELECT с использованием операций сравнения:

```
SELECT *
FROM product
WHERE quantity < 20
SELECT E.emp_lname, E.emp_fname
FROM employee E
WHERE emp_lname > 'McBadden'
SELECT id, phone
FROM contact
WHERE state  != 'CA'
```

Операция NOT

Операция NOT инвертирует какое-либо выражение. В любом из следующих двух запросов будут найдены все футболки и бейсболки стоимостью до 10 долл. Однако обратите внимание на различие в положении между отрицательной логической операцией (NOT) и отрицательной операцией сравнения (!>).

```
SELECT id, name, quantity
FROM product
WHERE (name = 'Tee Shirt' OR name = 'BaseBall Cap')
AND NOT unit_price > 10
SELECT id, name, quantity
FROM product
WHERE (name = 'Tee Shirt' OR name = 'BaseBall Cap')
AND unit_price !> 10
```

Использование области значений (BETWEEN и NOT BETWEEN) в разделе WHERE

Ключевое слово BETWEEN определяет диапазон поиска значений, причем поиск ведется по самому маленькому, самому большому значениям, а также по всем значениям между ними.

❖ **Составление списка всех товаров ценой от 10 до 15 долл. включительно**

♦ Введите следующий запрос:

```
SELECT name, unit_price
FROM product
WHERE unit_price BETWEEN 10 AND 15
```

name	unit_price
Tee Shirt	14
Tee Shirt	14
Baseball Cap	10
Shorts	15

Можно использовать NOT BETWEEN для нахождения всех строк, не входящих в указанную область значений.

❖ **Составление списка всех товаров дешевле 10 долл. и дороже 15 долл.**

♦ Выполните следующий запрос:

```
SELECT name, unit_price FROM product
WHERE unit_price NOT BETWEEN 10 AND 15
```

name	unit_price
Tee Shirt	9
Baseball Cap	9
Visor	7
Visor	7
...	...

Использование списков в разделе WHERE

Ключевое слово IN позволяет выбрать значения, соответствующие только одному значению в списке значений. Выражение может быть константой или именем столбца, и список может быть набором констант или, что встречается более часто, подзапросом.

Например, если необходимо составить список имен и штатов всех контактных лиц, проживающих в Онтарио, Манитобе или Квебеке без IN, можно ввести следующий запрос:

```
SELECT company_name , state
FROM customer
WHERE state = 'ON' OR state = 'MB' OR state = 'PQ'
```

Однако при использовании IN результаты будут теми же. Элементы, следующие за ключевым словом IN, должны быть разделены запятыми и заключены в скобки. Заключите значение символа, даты или времени в одиночные кавычки. Например:

```
SELECT company_name , state
FROM customer
WHERE state IN( 'ON', 'MB', 'PQ')
```

Возможно, наиболее важное применение ключевого слова IN - это использование данного слова во вложенных запросах, также называемых подзапросами.

Соответствие символьных строк в разделе WHERE

Ключевое слово LIKE указывает на то, что следующая символьная строка соответствует заданному образцу. Ключевое слово LIKE используется с символьными, двоичными данными или данными даты и времени.

Для LIKE используется следующий синтаксис:

{**WHERE** | **HAVING** } выражение [**NOT**] **LIKE** соответствующее-выражение

Выражение для соответствия будет сравниваться со значением "соответствующее-выражение", которое может включать следующие символы:

Символы	Значение
%	Соответствует любой строке с 0 или более символов.
_	Соответствует любому символу.
[спецификатор]	Спецификатор в квадратных скобках может принимать следующие формы: ♦ **Диапазон**. Диапазон в форме rangespec1-rangespec2, где rangespec1 обозначает начало диапазона символов, дефис обозначает диапазон, и rangespec2 обозначает конец диапазона символов. ♦ **Набор.** Набор может состоять из любого дискретного набора значений, расположенных в любом порядке, например, [a2bR]. Обратите внимание, что диапазон [a-f] и наборы [abc-def] и [fcbdae] возвращают тот же набор значений.
[^спецификатор]	Символ вставки (^), идущий перед спецификатором, означает невключение. [^a-f] означает "не в диапазоне a-f"; [^a2bR] означает "не a, 2, b или R.

Данные столбцов можно сопоставить с константами, переменными или с другими столбцами, содержащими групповые символы, отображенные в таблице. При использовании констант строки соответствия и символьные строки должны быть заключены в одиночные кавычки.

Примеры

Все следующие примеры используют LIKE со столбцом *last_name* в таблице *Contact*. Запросы имеют следующую форму:

```
SELECT last_name
FROM contact
WHERE last_name LIKE соответствующее-выражение
```

Первый пример был бы введен так:

```
SELECT last_name
FROM contact
WHERE last_name LIKE 'Mc%'
```

Соответствующее выражение	Описание	Возвращаемые данные
'Mc%	'Поиск любого имени, начинающегося с букв **Mc.**	McEvoy
'%er	'Поиск любого имени, заканчивающегося на **er**.	Brier, Miller, Weaver, Rayner
'%en%	'Поиск любого имени, содержащего буквы **en**.	Pettengill, Lencki, Cohen
'_ish	'Поиск имени из четырех букв, заканчивающегося на ish.	Fish
'Br[iy][ae]r	'Поиск Brier, Bryer, Briar или Bryar.	Brier
'[M-Z]owell	'Поиск всех имен, заканчивающихся на **owell**, начинающихся на буквы от M до Z.	Powell
'M[^c]%	'Поиск всех имен, начинающихся с **M**, но вторая буква которых - не **c**.	Moore, Mulley, Miller, Masalsky

LIKE и групповые символы

Групповые символы, используемые без LIKE, интерпретируются как **литералы**, а не как образец: они точно представляют их собственные значения. В следующем запросе производится попытка нахождения телефонных номеров, состоящих только из четырех символов. Запрос не находит телефонные номера, начинающиеся с 415.

```
SELECT phone
FROM Contact
WHERE phone = '415%'
```

Использование LIKE со значениями даты и времени

LIKE можно использовать в полях даты и времени, а также в символьных данных. При использовании LIKE со значениями даты и времени даты конвертируются в стандартный формат DATETIME, а затем в VARCHAR.

Одной из особенностей использования LIKE при поиске значений DATETIME является необходимость аккуратного написания выражения проверки равенства, поскольку записи даты и времени могут содержать разные части даты.

Например, если в столбец *arrival_time* (время прибытия) вставляется значение 9:20 и текущая дата, то раздел

```
WHERE arrival_time = '9:20'
```

не находит этого значения, поскольку запись содержит как дату, так и время. Однако раздел ниже найдет значение 9:20:

```
WHERE arrival_time LIKE '%09:20%'
```

Использование NOT LIKE

При работе с NOT LIKE можно использовать те же групповые символы, что и с LIKE. Для нахождения всех телефонных номеров с междугородним кодом 415 в таблице *Contact* можно воспользоваться любым из нижеприведенных запросов:

```
SELECT phone
FROM Contact
WHERE phone NOT LIKE '415%'
SELECT phone
FROM Contact
WHERE NOT phone LIKE '415%'
```

Символьные строки и кавычки

При вводе или поиске символьных данных и дат их необходимо заключить в одиночные кавычки, как показано в следующем примере.

```
SELECT first_name, last_name FROM contact WHERE
first_name = 'John'
```

Если параметр *quoted_identifier* базы данных установлен в OFF (по умолчанию используется ON), то символьные данные или данные даты можно заключить в двойные кавычки.

❖ **Отключение параметра quoted_identifier для текущего кода пользователя**

- Введите следующую команду:

```
SET OPTION quoted_identifier = 'OFF'
```

Параметр *quoted_identifier* предназначен для совместимости с Adaptive Server Enterprise. По умолчанию Adaptive Server Enterprise отключает параметр quoted_identifier, а в Adaptive Server Anywhere параметр *quoted_identifier* установлен в ON.

Кавычки в строках

Существует два способа определения литеральных кавычек в рамках символьной записи. Первый метод заключается в использовании двух последовательных кавычек. Например, если символьная запись начинается с одиночной кавычки, и необходимо включить одиночную кавычку как часть записи, то использовать нужно две одиночных кавычки:

```
'I don''t understand.'
```

С двойными кавычками (quoted_identifier выключен):

```
"He said, ""It is not really confusing."""
```

Второй метод, применяемый только с *quoted_identifier*, установленным на OFF, - заключить кавычки в кавычки другого типа. Другими словами, окружите запись, содержащую двойные кавычки, одиночными кавычками, или наоборот. Вот некоторые примеры:

```
'George said, "There must be a better way."'
"Isn't there a better way?"
'George asked, "Isn''t there a better way?"'
```

Неизвестные значения: NULL

NULL в столбце означает, что ни пользователь, ни приложение не занесли запись в этот столбец. Значение данных для столбца неизвестно или недоступно.

Значение NULL не означает "нуль" (числовые значения) или "пробел" (символьные значения). Значения NULL, скорее, позволяют отличать преднамеренную запись нуля для числовых столбцов, пробела - для символьных и отсутствие записи - NULL - как для числовых, так и символьных столбцов.

Ввод NULL

Как определено в операторе CREATE TABLE, ввести NULL в столбец, где разрешены значения NULL, можно двумя способами:

- **Значение по умолчанию.** Если данные не введены, и столбец не имеет другой настройки по умолчанию, то вводится NULL.
- **Явный ввод.** Значение NULL можно ввести явно путем ввода слова "NULL" (без кавычек).

Если слово NULL введено в кавычках в символьный столбец, то оно рассматривается как данные, а не как нулевое значение.

Например, столбец *dept_head_id* таблицы отдела допускает нулевые значения. Для отделов без начальников можно ввести две строки следующим образом:

```
INSERT INTO department (dept_id, dept_name)
VALUES (201, 'Eastern Sales')
INSERT INTO department
VALUES (202, 'Western Sales', null)
```

При извлечении значений NULL

При извлечении значений NULL отображение результатов запроса в Interactive SQL показывает "NULL" в соответствующей позиции:

```
SELECT *
FROM department
```

dept_id	dept_name	dept_head_id
100	R & D	501
200	Sales	904
300	Finance	1293
400	Marketing	1576
500	Shipping	703
201	Eastern Sales	(NULL)
202	Western Sales	(NULL)

Проверка столбца на NULL

Можно использовать IS NULL в условиях поиска для сравнения значений столбцов со значением NULL и для их выбора и проведения определенной операции, основанной на результатах сравнения. Выбираются или заканчиваются определенным действием только столбцы, возвращающие значение TRUE; столбцы, возвращающие FALSE или UNKNOWN, не выбираются.

В следующем примере выбраны только строки, для которых *unit_price* меньше, чем 15 долл. или NULL:

```
SELECT quantity , unit_price
FROM product
WHERE unit_price < 15
OR unit_price IS NULL
```

Результатом сравнения любого значения с NULL является UNKNOWN, поскольку невозможно определить, равен NULL (или не равен) заданному значению или другому значению NULL.

Существуют определенные условия, которые никогда не возвращают TRUE. Поэтому запросы, использующие эти условия, не возвращают результирующих наборов. Например, следующее сравнение никогда не может быть TRUE, поскольку NULL означает наличие неизвестного значения:

```
WHERE column1 > NULL
```

Эта логика также применяется при использовании двух имен столбцов в разделе WHERE, т. е. при соединении двух таблиц. Раздел, содержащий условие

```
WHERE column1 = column2
```

не возвращает строки, в которых столбцы содержат NULL. NULL или не-NULL можно также обнаружить со следующим образцом:

```
WHERE column_name IS [NOT] NULL
```

Например:

```
WHERE advance < $5000
OR advance IS NULL
```

Для получения дополнительной информации см. раздел "Значение NULL" (NULL value) на стр. 47 в документе *"Справочник по SQL для ASA" (ASA SQL Reference Manual)*.

Свойства NULL

В следующем списке подробно представлены свойства NULL.

- **Различие между FALSE и UNKNOWN**. Несмотря на то, что ни FALSE, ни UNKNOWN не возвращают значения, существует важное логическое различие между FALSE и UNKNOWN, потому что антипод FALSE - TRUE. Например,

  ```
  1 = 2
  ```

  относится к FALSE и его противоположному значению,

  ```
  1 != 2
  ```

  относится к TRUE. Но "не неизвестное" по-прежнему остается неизвестным. Если нулевые значения включены в сравнение, то инвертировать выражение для получения противоположного набора строк или противоположного истинного значения нельзя.

- **Замена значения на NULL.** Для замены какого-либо значения на NULL пользуйтесь встроенной функцией ISNULL. Замена производится только для целей отображения; на фактические значения столбцов она не влияет. Синтаксис:

  ISNULL *(выражение, значение)*

  Например, используйте следующий оператор для выбора всех строк из теста и отобразите все значения NULL в столбце t1 с неизвестным значением.

  ```
  SELECT ISNULL(t1, 'unknown')
  FROM test
  ```

- **Выражения со значением NULL.** Выражение с арифметической или поразрядной операцией имеет значение NULL, если какой-либо из операндов равен нулю. Например:

  ```
  1 + column1
  ```

  имеет значение NULL, если column1 - NULL.

- **Связывание строк и NULL**. При связывании строки и NULL выражение имеет значение строки. Например:

  ```
  SELECT 'abc' || NULL || 'def'
  ```

  возвращает строку **abcdef**.

Соединение условий логическими операциями

Логические операции AND, OR и NOT используются для соединения условий поиска в разделах WHERE.

Использование AND

Операция AND соединяет два или более условия и возвращает результаты только в случае истинности этих условий. Например, следующий запрос находит только строки, в которых фамилия контактного лица - Purcell, а имя - Beth. Этот запрос не находит строку, относящуюся к Beth Glassmann.

```
SELECT *
FROM contact
WHERE first_name = 'Beth'
   AND last_name = 'Purcell'
```

Использование OR

Операция OR также соединяет два или более условия, но возвращает результаты, только если *какое-либо* из условий истинно. Следующий запрос выполняет поиск строки, содержащей варианты имени Elizabeth в столбце *first_name*.

```
SELECT *
FROM contact
WHERE first_name = 'Beth'
   OR first_name = 'Liz'
```

Использование NOT

Операция NOT инвертирует выражение, которое следует за ней. Следующий запрос перечисляет всех контактных лиц, не проживающих в Калифорнии:

```
SELECT *
FROM contact
WHERE NOT state = 'CA'
```

Если в операторе используется более одной логической операции, тогда операции AND, как правило, вычисляются до операций OR. Порядок выполнения можно изменить при помощи круглых скобок. Например:

```
SELECT *
FROM contact
WHERE ( city = 'Lexington'
        OR city = 'Burlington' )
   AND state = 'MA'
```

ГЛАВА 7

Сведение, группирование и сортировка результатов запроса

Об этой главе

Агрегатные функции отображают сводные значения в указанных столбцах.

Для группирования и сортировки результатов запросов с использованием агрегатных функций можно использовать раздел GROUP BY, раздел HAVING и раздел ORDER BY, а также оператор UNION для комбинирования результатов запросов.

В этой главе описывается процесс группирования и сортировки результатов запроса.

Содержание

Сведение результатов запросов с использованием агрегатных функций

Агрегатные функции можно применить ко всем строкам в таблице, к поднабору таблицы, определенному оператором WHERE, или к одной или более группам строк в таблице. Из каждого набора строк, к которым применена агрегатная функция, Adaptive Server Anywhere генерирует одиночное значение.

Имеются следующие агрегатные функции:

- **avg(выражение)**. Среднее значение указанного выражения по возвращенным строкам.
- **count(выражение)**. Количество строк в указанной группе, для которых выражение имеет значение, отличное от NULL.
- **count(*)**. Количество строк в каждой группе.
- **list(строковое-выражение)**. Строка, содержащая список элементов, разделенных запятыми, составленный из всех значений строкового-выражения для каждой группы строк.
- **max(выражение).** Максимальное значение выражения по возвращенным строкам.
- **min(выражение).** Минимальное значение выражения по возвращенным строкам.
- **sum(выражение).** Сумма выражений по возвращенным строкам. Перед применением агрегатной функции можно удалить дублирующиеся значения, используя дополнительное ключевое слово DISTINCT с функциями AVG, SUM, LIST и COUNT.

Обычно выражение, на которое ссылается оператор синтаксиса, является именем столбца. Это также может быть более общее выражение.

Например, при помощи этого оператора можно найти среднюю цену на все товары, если к каждой цене прибавить один доллар:

```
SELECT AVG (unit_price + 1)
FROM product
```

Пример

Следующий запрос рассчитывает общую платежную ведомость из значений ежегодной заработной платы в таблице служащих:

```
SELECT SUM(salary)
FROM employee
```

Для использования агрегатных функций следует указывать имя функции, а за ним - выражение, значения которого будут обрабатываться. Выражение, представленное в данном примере столбцом salary, является аргументом функции и должно быть указано в круглых скобках.

Область применения агрегатных функций

Агрегатные функции можно использовать в списке выбора, как показано в примерах выше, или в разделе HAVING оператора SELECT, включающего раздел GROUP BY.

Для получения дополнительной информации о разделе HAVING см. раздел "Раздел HAVING: выбор групп данных" на стр. 214.

Агрегатные функции нельзя использовать в разделе WHERE или условии JOIN. Однако оператор SELECT с агрегатными функциями в списке выбора часто содержит раздел WHERE, ограничивающий строки, к которым применена агрегация.

Если оператор SELECT включает раздел WHERE, но не включает раздел GROUP BY, то агрегатная функция возвращает одиночное значение для поднабора строк, указанных разделом WHERE.

Если агрегатная функция используется в операторе SELECT, не включающем раздел GROUP BY, то она вычисляет одиночное значение. Это справедливо и в случае применения ее ко всем строкам в таблице, и к поднабору строк, указанному разделом WHERE.

В одном и том же списке выбора можно использовать несколько агрегатных функций и вычислять несколько скалярных сумм в одном операторе SELECT.

Агрегатные функции и внешние ссылки

Adaptive Server Anywhere версии 8 следует новым стандартам SQL/99 для разъяснения использования агрегатных функций при их появлении в подзапросе. Эти изменения влияют на поведение операторов, написанных для предыдущих версий ПО: изначально правильные запросы теперь могут вызывать сообщения об ошибках, и результаты могут измениться.

Когда агрегатная функция появляется в подзапросе, и столбец, на который ссылается агрегатная функция, является внешней ссылкой, то сама агрегатная функция обрабатывается как внешняя ссылка. Это означает, что агрегатная функция теперь вычисляется во внешнем блоке, а не в подзапросе, и становится константой внутри подзапроса.

К использованию агрегатных функций с внешними ссылками в подзапросах теперь применяются следующие ограничения:

- Агрегатная функция с внешними ссылками может присутствовать только в тех подзапросах, которые находятся в списке SELECT или разделе HAVING, и эти разделы должны находиться в непосредственно следующем за ними внешнем блоке.

- Агрегатные функции с внешними ссылками могут содержать только одну ссылку на внешний столбец.
- Нельзя одновременно использовать в одной агрегатной функции локальные и внешние ссылки на столбцы.

Следует отметить, что некоторых проблем, касающихся новых стандартов, можно избежать, переписав агрегатную функцию так, чтобы она включала только локальные ссылки. Например, подзапрос `(SELECT MAX(S.y + R.y) FROM S)` содержит как локальную ссылку на столбец (S.y), так и внешнюю ссылку на столбец (R.y), что теперь недопустимо. Его можно переписать следующим образом: `(SELECT MAX(S.y) + R.y FROM S)`. В измененной форме агрегатная функция имеет только локальную ссылку на столбец. Такой же принцип исправления может применяться в случае, если агрегатная функция с внешними ссылками появляется в иных разделах, нежели SELECT и HAVING.

Пример

Следующий запрос представил следующие результаты в Adaptive Server Anywhere версии 7.

```
SELECT name, (SELECT SUM(p.quantity) FROM
   sales_order_items)
FROM product p
```

name	sum(p.quantity)
Tee shirt	30,716
Tee shirt	59,238

В версии 8 тот же запрос выдает сообщение об ошибке "ASA Error - 149: Function or column reference to 'name' must also appear in a GROUP BY" (Функция или ссылка столбца на 'name' должна также присутствовать в разделе GROUP BY).

Оператор становится недопустимым, потому что агрегатная функция с внешней ссылкой `sum(p.quantity)` теперь вычисляется во внешнем блоке. В версии 8 запрос семантически эквивалентен следующему (за исключением того, что "Z" не является частью результирующего набора):

```
SELECT name,
       SUM(p.quantity) as Z,
       (SELECT Z FROM sales_order_items)
FROM product p
```

Поскольку вычисление агрегатной функции теперь выполняется во внешнем блоке, то он обрабатывается как сгруппированный запрос, и для того, чтобы появиться в списке SELECT, столбец name должен присутствовать в разделе GROUP BY.

Агрегатные функции и типы данных

Существуют некоторые агрегатные функции, имеющие смысл только для определенных типов данных. Например, функции SUM и AVG можно применять только со столбцами с числовыми значениями.

Однако функцию MIN можно применять для нахождения в столбце символьного типа минимального значения (наиболее близкого к началу алфавита):

```
SELECT MIN(last_lname)
FROM  contact
```

Использование функции count (*)

Функция COUNT(*) не требует указания того или иного выражения в качестве аргумента, поскольку она по определению не использует информацию о каком-либо конкретном столбце. Функция COUNT(*) находит общее число строк в таблице. Этот оператор находит общее число служащих:

```
SELECT COUNT(*)
FROM employee
```

COUNT(*) возвращает количество строк в указанной таблице, не удаляя дубликаты. Эта функция учитывает каждую строку отдельно, включая строки, содержащие NULL.

Как и в других агрегатных функциях, count(*) можно объединить с другими агрегатами в списке выбора, с разделами WHERE и т.д.:

```
SELECT count(*), AVG(unit_price)
FROM product
WHERE  unit_price > 10
```

count(*)	AVG(product.unit_price)
5	18.2

Использование агрегатных функций с ключевым словом DISTINCT

Ключевое слово DISTINCT не является обязательным для функций SUM, AVG и COUNT. При использовании DISTINCT при вычислении SUM, AVERAGE или COUNT дублирующиеся значения удаляются.

Например, для того чтобы найти число разных городов, в которых имеются контакты, следует ввести:

```
SELECT count(DISTINCT city) FROM contact
```

count(distinct contact.city)
16

Агрегатные функции и NULL

Любые значения NULL в столбце, обрабатываемом агрегатной функцией, игнорируются для функций, за исключением COUNT(*), которая их учитывает. Если все значения столбца равны NULL, то COUNT(имя_столбца) возвращает 0.

Если ни одна строка не соответствует условиям, указанным в разделе WHERE, то COUNT возвращает значение 0. Все остальные функции возвращают NULL. Примеры:

```
SELECT COUNT (DISTINCT name)
FROM product
WHERE unit_price > 50
```

count(DISTINCT name)
0

```
SELECT AVG(unit_price)
FROM product
WHERE unit_price > 50
```

AVG(product.unit_price)
(NULL)

Раздел GROUP BY: группирование результатов запроса

Раздел GROUP BY распределяет выводимые результаты таблицы по группам. Раздел GROUP BY можно применить к нескольким именам столбцов, либо к результатам вычисленных столбцов, используя числовые данные в каком-либо выражении.

Использование раздела GROUP BY с агрегатными функциями

Раздел GROUP BY почти всегда появляется в операторах, включающих агрегатные функции, и в этом случае агрегат производит значение для каждой группы. Эти значения называются векторными агрегатами. (Помните, что скалярный агрегат - это одиночное значение, произведенное агрегатной функцией без раздела GROUP BY.)

Пример

Следующий запрос отображает среднюю цену для каждого вида продукта:

```
SELECT name, AVG(unit_price) AS Price
FROM product
GROUP BY name
```

name	Price
Tee Shirt	12.333333333
Baseball Cap	9.5
Visor	7
Sweatshirt	24
...	...

Итоговые значения (векторные агрегаты), произведенные операторами SELECT с агрегатами и разделом GROUP BY, появляются в виде столбцов в каждой строке результатов. Напротив, итоговые значения (скалярные агрегаты), произведенные запросами с агрегатами и без GROUP BY, также появляются в виде столбцов, но с только одной строкой. Например:

```
SELECT AVG(unit_price)
FROM product
```

AVG(product.unit_price)
13.3

Принципы использования раздела GROUP BY

Понимание того, какие запросы являются допустимыми, а какие - нет, может быть затруднено, если в запрос включен раздел GROUP BY. В данном разделе для рассмотрения запросов с разделом GROUP BY используется подход, позволяющий более точно описать понятия результатов и допустимости.

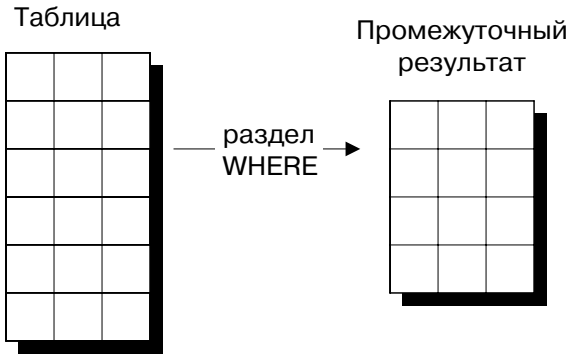
Выполнение запросов с разделом GROUP BY

Рассмотрим запрос для одной таблицы следующей формы:

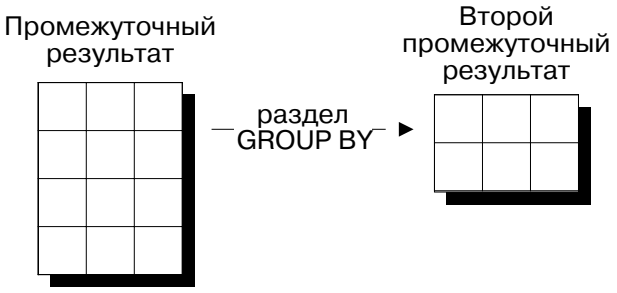
SELECT *список-выбора*
FROM *таблица*
WHERE *условие-поиска-where*
GROUP BY *выражение-group-by*
HAVING *условие-поиска-having*

Может показаться, что этот запрос выполняется следующим образом:

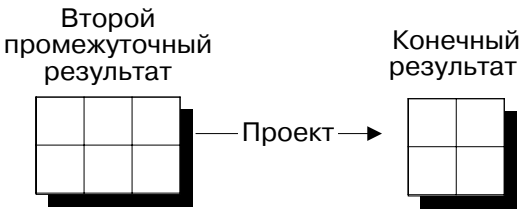
1. **Выполнение раздела WHERE.** Генерируется промежуточный результат, содержащий только некоторые строки из таблицы.



2. **Распределение результатов по группам.** Данное действие генерирует промежуточный результат с одной строкой для каждой группы, как продиктовано разделом GROUP BY. Каждая сгенерированная строка содержит выражение-group-by для каждой группы и вычисленные агрегатные функции в списке-выбора и условии-поиска-having.



3. **Выполнение раздела HAVING.** Любые строки из этого второго промежуточного результата, не соответствующие критериям раздела HAVING, на этом этапе удаляются.
4. **Проектирование отображения результатов.** При этом действии из шага 3 берутся только те столбцы, которые нужно отобразить в результирующем наборе запроса, т. е. только столбцы, соответствующие выражениям из списка-выбора.



Этот процесс накладывает требования на запросы с разделом GROUP BY:

- Сначала вычисляется раздел WHERE. Поэтому любые агрегатные функции вычисляются только по строкам, удовлетворяющим разделу WHERE.
- Конечный результирующий набор построен из второго промежуточного результата, в котором содержатся разделенные строки. Во втором промежуточном результате содержатся строки, соответствующие *выражению-group-by.* Поэтому если в *списке-выбора* появляется какое-либо выражение, не являющееся агрегатной функцией, тогда оно также должно появится в *выражении-group-by.* Во время выполнения шага проектирования вычисление какой-либо функции выполняться не может.
- Выражение может быть включено в *выражение-group-by,* но не в список-выбора. Оно спроектировано в результате.

GROUP BY для нескольких столбцов

В разделе GROUP BY можно перечислить несколько выражений для выполнения вложения групп, т.е. таблицу можно сгруппировать посредством любой комбинации выражений.

Следующий запрос перечисляет средние цены на товары, сгруппированные сначала по наименованию, а потом по размеру:

```
SELECT name, size, AVG(unit_price)
FROM product
GROUP BY name, size
```

name	size	AVG (product.unit_price)
Tee Shirt	Small	9
Tee Shirt	Medium	14
Tee Shirt	One size fits all	14
Baseball Cap	One size fits all	9.5
...	...	...

Столбцы в GROUP BY, не включенные в список выбора

Расширение Sybase для стандарта SQL/92, поддерживаемого как Adaptive Server Enterprise, так и Adaptive Server Anywhere, должно допускать выражения в раздел GROUP BY, не включенные в список выбора. Например, следующий запрос перечисляет контакты в каждом городе:

```
SELECT state, count(id)
FROM contact
GROUP BY state, city
Раздел WHERE и GROUP BY
```

Раздел WHERE можно использовать в операторе с GROUP BY

Раздел WHERE выполняется до раздела GROUP BY. Строки, не соответствующие условиям раздела WHERE, удаляются перед выполнением любого группирования. Например:

```
SELECT  name, AVG(unit_price)
FROM product
WHERE id > 400
GROUP BY name
```

В группы, используемые для получения результатов запроса, включены только строки со значениями id, превышающими 400.

Пример

Следующий запрос иллюстрирует использование разделов WHERE, GROUP BY и HAVING в одном запросе:

```
SELECT name, SUM(quantity)
FROM product
WHERE name LIKE '%shirt%'
GROUP BY name
HAVING SUM(quantity) > 100
```

name	SUM (product.quantity)
Tee Shirt	157

В этом примере:

- Раздел WHERE включает только строки с именем, включающим слово shirt (например, Tee Shirt, Sweatshirt).
- Раздел GROUP BY собирает строки с общим именем.
- Агрегат SUM рассчитывает общее количество товаров, доступных для каждой группы.
- Раздел HAVING исключает из конечных результатов группы, в которых общие запасы товаров на складе не превышают 100.

Раздел HAVING: выбор групп данных

Раздел HAVING ограничивает строки, возвращенные запросом. Он задает условия для раздела GROUP BY способом, похожим на тот, каким раздел WHERE устанавливает условия для раздела SELECT.

Условия поиска раздела HAVING идентичны условиям поиска раздела WHERE за исключением того, что условия поиска WHERE не могут включать агрегаты, тогда как для условий поиска HAVING это возможно. Приведенный ниже пример является допустимым:

```
HAVING AVG(unit_price) > 20
```

Следующий же пример является недопустимым:

```
WHERE AVG(unit_price) > 20
```

Использование раздела HAVING с агрегатными функциями

Следующий оператор является примером простого использования раздела HAVING с агрегатной функцией.

Для перечисления товаров, представленных в нескольких цветах или размерах, необходимо сделать запрос группе строк в таблице product по имени, но удалить группы, включающие только один особый товар:

```
SELECT name
FROM product
GROUP BY name
HAVING COUNT(*) > 1
```

name
Tee Shirt Baseball Cap Visor Sweatshirt

☞ Для получения информации об использовании агрегатных функций в разделах HAVING см. раздел "Область применения агрегатных функций" на стр. 205.

Использование раздела HAVING без агрегатных функций

Раздел HAVING можно использовать без агрегатов.

Следующий запрос группирует товары, а затем ограничивает результирующий набор только теми группами, в которых имя начинается с "B".

```
SELECT name
FROM product
GROUP BY name
HAVING name LIKE 'B%'
```

name
Baseball Cap

Раздел HAVING с несколькими условиями

В раздел HAVING можно включить несколько условий. Они объединены операциями AND, OR или NOT, как показано в примере ниже.

Для перечисления товаров, представленных в нескольких цветах или размерах, для которых цена одного варианта составляет более 10 долларов, необходимо сделать запрос группе строк в таблице product по имени, но удалить группы, включающие только один особый товар, а также удалить группы, максимальная цена на единицу товара в которых составляет меньше 10 долларов.

```
SELECT name
FROM product
GROUP BY name
HAVING COUNT(*) > 1
AND MAX(unit_price) > 10
```

name
Tee Shirt
Sweatshirt

Раздел ORDER BY: сортировка результатов запроса

Раздел ORDER BY позволяет сортировать результаты запросов по одному или более столбцам. Каждая сортировка может быть по возрастанию (ASC) или по убыванию (DESC). Если нет специального указания, применяется сортировка по возрастанию.

Простой пример

Следующий запрос возвращает результаты, отсортированные по имени:

```
SELECT id, name
FROM product
ORDER BY name
```

id	name
400	Baseball Cap
401	Baseball Cap
700	Shorts
600	Sweatshirt
...	...

Сортировка по нескольким столбцам

Если в разделе ORDER BY указывается более одного столбца, то сортировка является вложенной.

Следующий оператор сортирует рубашки в таблице product по наименованию в восходящем порядке, затем по количеству (по убыванию) каждого наименования.

```
SELECT id, name, quantity
FROM product
WHERE name like '%shirt%'
ORDER BY name, quantity DESC
```

id	name	quantity
600	Sweatshirt	39
601	Sweatshirt	32
302	Tee Shirt	75
301	Tee Shirt	54
...	...	...

Использование позиции столбца

Вместо имени столбца в списке выбора можно использовать номер позиции столбца. Имена столбцов и числа в списке выбора могут смешиваться. Следующие операторы оба производят те же результаты, что и предыдущий.

```
SELECT id, name, quantity
FROM product
```

```
WHERE name like '%shirt%'
ORDER BY 2, 3 DESC

SELECT id, name, quantity
FROM product
WHERE name like '%shirt%'
ORDER BY 2, quantity DESC
```

Большинство версий SQL требует, чтобы элементы ORDER BY появлялись в списке выбора, но в Adaptive Server Anywhere такого ограничения нет. Следующий запрос упорядочивает результаты по количеству, хотя этот столбец не появляется в списке выбора:

```
SELECT id, name
FROM product
WHERE name like '%shirt%'
ORDER BY 2, quantity DESC
```

ORDER BY и NULL

При использовании раздела ORDER BY значение NULL появляется перед другими значениями независимо от того, проводится ли сортировка по возрастанию или по убыванию.

ORDER BY и учет регистра

Действия раздела ORDER BY для данных, введенных со смешанным регистром, зависят от порядка сортировки базы данных и от чувствительности к регистру, заданной при создании базы данных.

Извлечение первых нескольких строк запроса

Результаты запроса можно ограничить первыми несколькими строками, возвращенными с использованием ключевых слов FIRST или TOP. Они могут применяться в любом запросе, но наиболее полезны в запросах, использующих раздел ORDER BY.

Примеры

Следующий запрос возвращает информацию о первом служащем при сортировке по фамилии:

```
SELECT FIRST *
FROM employee
ORDER BY emp_lname
```

Следующий запрос возвращает записи первых пяти служащих по фамилиям, отсортированным в алфавитном порядке:

```
SELECT TOP 5 *
FROM employee
ORDER BY emp_lname
```

Ограничения на использование FIRST и TOP

FIRST и TOP поддерживаются в самом внешнем блоке SELECT запроса. Для обеспечения согласованности результатов они должны использоваться только вместе с разделом ORDER BY. FIRST также поддерживается в подзапросах, находящихся либо в списке SELECT запроса, либо участвующих в предикате сравнения, - подзапрос не является частью кванторного предиката, включающего IN, ANY, SOME или ALL. Например, вложенный запрос

```
SELECT *
FROM sales_order_items
WHERE prod_id = (SELECT FIRST from product)
```

поддерживается, тогда как запрос

```
SELECT *
FROM sales_order_items
WHERE prod_id = ANY(SELECT FIRST from product)
```

не поддерживается. Неподдерживаемые экземпляры FIRST или TOP не могут вызвать появление ошибки синтаксиса, но скорее всего выдадут неожиданные или непредсказуемые результаты. По этой причине необходимо воздерживаться от использования FIRST или TOP в других конструкциях SQL, кроме двух, упомянутых выше. Особо не рекомендуется указывать FIRST или TOP в производной таблице, представлении или кванторном подзапросе.

ORDER BY и GROUP BY

Раздел ORDER BY можно использовать для расположения результатов GROUP BY определенным способом.

Пример

Следующий запрос находит среднюю цену каждого товара и упорядочивает результаты по средней цене:

```
SELECT name, AVG(unit_price)
FROM product
GROUP BY name
ORDER BY AVG(unit_price)
```

name	AVG(product.unit_price)
Visor	7
Baseball Cap	9.5
Tee Shirt	12.333333333
Shorts	15
...	...

Операция UNION: объединение запросов

Оператор UNION объединяет результаты двух или больше запросов в одном результирующем наборе.

По умолчанию оператор UNION удаляет дублирующиеся строки из результирующего набора. При использовании параметра ALL дубликаты не удаляются. Столбцы в результирующем наборе имеют те же имена, что и столбцы в первой таблице, на которую сделана ссылка. Можно использовать любое число операций объединения. Например:

```
x UNION y UNION z
```

По умолчанию оператор, содержащий несколько операций UNION, выполняется слева направо. Круглые скобки могут использоваться для указания порядка вычисления.

Например, следующие два выражения не эквивалентны из-за способа, с помощью которого дублирующиеся строки удалены из результирующих наборов:

```
x UNION ALL (y UNION z)
(x UNION ALL y) UNION z
```

В первом выражении дубликаты удалены в UNION между y и z. В UNION между этим набором и x дубликаты не удалены. Во втором выражении дубликаты включены в объединение между x и y, но удалены в последующем объединении с z.

Рекомендации для запросов UNION

Следующие рекомендации относятся к применению операторов объединения:

- **Одинаковое количество элементов в списках выбора.** Все списки выбора в операторе объединения должны иметь одинаковое количество выражений (таких как имена столбца, арифметические выражения и агрегатные функции). Следующий оператор недопустим, потому что первый список выбора длиннее второго:

```
-- Пример недопустимого оператора
SELECT stor_id, city, state
FROM stores
UNION
SELECT stor_id, city
FROM stores_east
```

- **Совпадающие типы данных.** Соответствующие выражения в списках SELECT должны иметь одинаковый тип данных, либо между двумя типами данных должно быть возможно неявное преобразование, или должно обеспечиваться явное преобразование.

Например, UNION не возможно между столбцом с типом данных CHAR и столбцом с типом данных INT, если не обеспечивается явное преобразование. Однако возможно объединение между столбцом с типом данных MONEY и столбцом с типом данных INT.

- **Порядок столбцов.** Соответствующие выражения в отдельных запросах оператора UNION нужно расположить в том же порядке, поскольку UNION сравнивает выражения одно с другим в порядке, установленном в отдельных запросах разделов SELECT.
- **Множественные объединения.** Несколько операций UNION можно объединить в строку, как показано в следующем примере:

```
SELECT city AS Cities
FROM contact
UNION
SELECT city
FROM customer
UNION
SELECT city
FROM employee
```

  В конце оператора допустим только один раздел ORDER BY. Это означает, что никакой отдельный оператор SELECT в запросе UNION не может содержать раздел ORDER BY.
- **Заголовки столбцов.** Имена столбцов в таблице, являющейся результатом UNION, берутся из первого отдельного запроса в операторе. Если для результирующего набора необходимо определить новый заголовок столбца, то это нужно сделать в первом запросе, как показано в примере ниже:

```
SELECT city AS Cities
FROM contact
UNION
SELECT city
FROM customer
```

  В следующем запросе заголовок столбца остается city, как определено в первом запросе оператора UNION.

```
SELECT city
FROM contact
UNION
SELECT city AS Cities
FROM customer
```

- В конце списка запросов можно использовать одиночный раздел ORDER BY, но использовать нужно целые числа, а не имена столбцов, как показано в примере ниже:

```
SELECT Cities = city
FROM contact
UNION
SELECT city
FROM customer
ORDER BY 1
```

Стандарты и совместимость

В этом разделе рассматриваются некоторые аспекты поддержки разделом GROUP BY в Adaptive Server Anywhere стандартов и совместимости.

GROUP BY и стандарт SQL/92

Стандарт SQL/92 для GROUP BY требует следующего:

- Столбец, используемый в выражении раздела SELECT, должен находиться в разделе GROUP BY. В противном случае, выражение, использующее этот столбец, является агрегатной функцией.
- Выражение GROUP BY может содержать только имена столбцов из списка выбора, но не используемые только в качестве аргументов для векторных агрегатов.

Результаты стандартного раздела GROUP BY с векторными агрегатными функциями выдают одну строку с одним значением на группу.

Adaptive Server Anywhere и Adaptive Server Enterprise поддерживают расширения к разделу HAVING, не допускающему наличие агрегатных функций ни в списке выбора, ни в разделе GROUP BY.

Совместимость с Adaptive Server Enterprise

Adaptive Server Enterprise поддерживает несколько расширений к разделу GROUP BY, не поддерживаемых в Adaptive Server Anywhere. Сюда входит следующее:

- **Несгруппированные столбцы в списке выбора.** Adaptive Server Enterprise допускает имена столбцов в списке выбора, которые не появляются в разделе GROUP BY. Например, следующее допустимо в Adaptive Server Enterprise:

  ```
  SELECT name, unit_price
  FROM product
  GROUP BY name
  ```

  Этот синтаксис не поддерживается в Adaptive Server Anywhere.

- **Вложенные агрегатные функции.** Следующий запрос, вкладывающий векторный агрегат в скалярный агрегат, является допустимым в Adaptive Server Enterprise, но не в Adaptive Server Anywhere:

  ```
  SELECT MAX(AVG(unit_price))
  FROM product
  GROUP BY name
  ```

- **GROUP BY и ALL.** Adaptive Server Anywhere не поддерживает использование ALL в разделе GROUP BY.
- **HAVING без GROUP BY.** Adaptive Server Anywhere не поддерживает использование HAVING без раздела GROUP BY, если выражения в разделах HAVING и SELECT не являются агрегатными функциями. Например, следующий запрос допустим в Adaptive Server Enterprise, но не поддерживается в Adaptive Server Anywhere:

```
--неподдерживаемый синтаксис
SELECT unit_price
FROM product
HAVING COUNT(*) > 8
```

  Однако следующий оператор допустим в Adaptive Server Anywhere, потому что функции MAX и COUNT являются агрегатными:

```
SELECT MAX(unit_price)
FROM product
HAVING COUNT(*) > 8
```

- **Условия HAVING.** Adaptive Server Enterprise поддерживает расширения к HAVING, не допускающему наличие неагрегатных функций ни в списке выбора, ни в разделе GROUP BY. В Adaptive Server Anywhere допустимы только агрегатные функции этого типа.
- **DISTINCT с ORDER BY или GROUP BY.** Adaptive Server Enterprise разрешает использование столбцов в разделе ORDER BY или GROUP BY, не появляющихся в списке выбора и даже в запросах SELECT DISTINCT. Это может привести к возникновению повторяющихся значений в результирующем наборе SELECT DISTINCT. Adaptive Server Anywhere не поддерживает это поведение.
- **Имена столбцов в UNION.** Adaptive Server Enterprise разрешает использование столбцов в разделе ORDER BY в объединениях запросов. В Adaptive Server Anywhere раздел ORDER BY должен использовать целое число, чтобы отметить столбец, по которому сортируются результаты.

ГЛАВА 8

Соединения: извлечение данных из нескольких таблиц

Об этой главе

При создании базы данных данные нормализуются путем размещения информации для разных объектов в разных таблицах, а не в одной таблице с множеством избыточных элементов.

Операция соединения обновляет большую таблицу, используя информацию из двух или более таблиц (или представлений). При помощи различных соединений можно создать много виртуальных таблиц подобного рода, каждая из которых будет применима для решения определенной задачи.

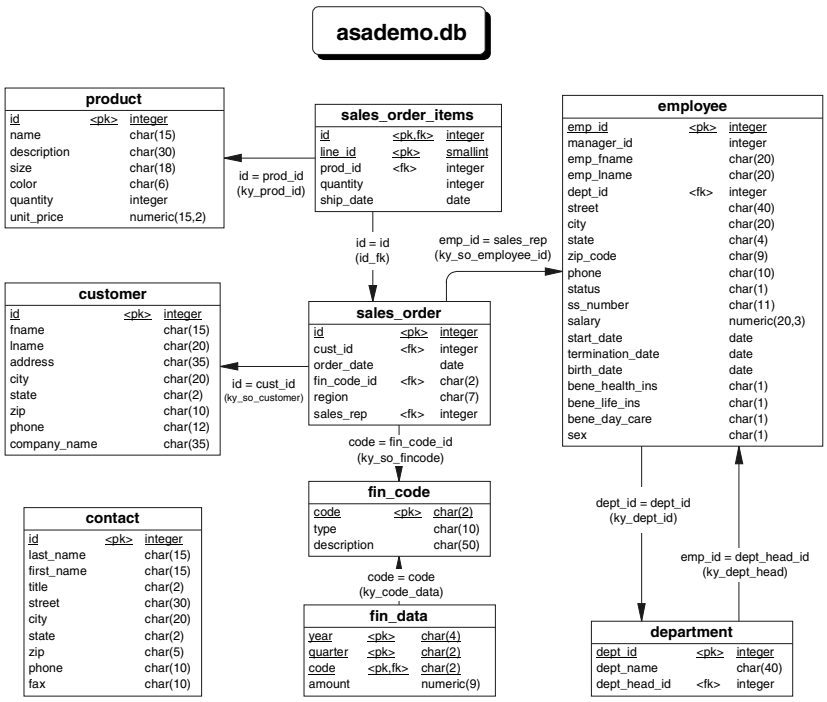
Перед началом работы

Эта глава предполагает наличие определенных знаний о запросах и синтаксисе оператора Select. Для получения информации о запросах см. раздел "Запросы: выбор данных в таблице" на стр. 179.

Содержание

Схема демонстрационной базы данных

В данной главе часто встречаются ссылки на демонстрационную базу данных. В следующей диаграмме демонстрационная база данных представлена с именами внешних ключей, связывающих таблицы. Демонстрационная база данных хранится в файле с именем *asademo.db*, размещенном в папке установки.



Примечания:

address - адрес
amount - сумма
bene_day_care - номер медицинского полиса
bene_health_ins - номер сертификата о страховании здоровья
bene_life_ins - номер сертификата о страховании жизни
birth_date - дата рождения
city - город
city - город
code - код
color - цвет
company_name - название организации
contact - контакты
cust_id - код клиента
customer - клиенты
departament - отделы
dept_head_id - код начальника отдела
dept_id - код отдела
dept_id - код отдела
dept_name - название отдела
description - описание
description - описание
emp_fname - имя служащего
emp_id - код служащего
emp_lname - фамилия служащего
employee - служащие
fax - номер факса
fin_code_id - код счета
fin_data - финансовые данные
fname - имя
id - код
line id - код строки товара
line id - код строки товара
lname - фамилия
manager_id - код менеджера
name - наименование
order_date - дата заказа
phone - телефон
prod_id - код товара
product - товары
quantity - количество
quantity - количество
quarter - квартал
region - регион
salary - заработная плата
sales_order - заказы на покупку
sales_order_items - заказы на покупку
sales_rep - торговый представитель
sex - пол
ship_date - дата отгрузки
size - размер
ss_number - код социального обеспечения
start_date - дата принятия на работу
state - страна
status - семейное положение
street - улица
termination_date - дата увольнения
title - заголовок
type - тип
unit_price - цена за экземпляр
year - год
zip - почтовый индекс
zip_code - почтовый индекс

Принципы работы соединений

В реляционной базе данных хранится информация о разных типах объектов в разных таблицах. Например, информация, относящаяся к служащим, появляется в одной таблице, а информация об отделах - в другой. В таблице служащих содержится такая информация, как имена и адреса служащих. В таблице отдела содержится информация об одном отделе, например, название отдела и имя его начальника.

Ответы на большинство вопросов можно получить только при использовании комбинации данных из разных таблиц. Например, может потребоваться ответ на вопрос: "Кто является начальником отдела сбыта?" Для нахождения имени этого человека его необходимо правильно идентифицировать, используя информацию таблицы отдела, а затем узнать его имя в таблице служащих.

Соединения являются средством, используемым для ответа на подобные вопросы. Они формируют новую виртуальную таблицу, включающую информацию из многих таблиц. Например, можно создать список начальников отделов путем объединения информации, содержащейся в таблице служащих и в таблице отделов. При помощи раздела FROM задаются таблицы, содержащие нужную информацию.

Для эффективного использования соединений необходимо правильно выбирать объединяемые столбцы каждой таблицы. Для составления списка начальников отделов каждая строка объединенной таблицы должна содержать название отдела и имя возглавляющего его служащего. Необходимо контролировать соответствие столбцов в составной таблице либо заданием конкретного типа операции соединения, либо использованием фразы ON.

Обзор соединений

Соединение представляет собой операцию, объединяющую строки в таблицах путем сравнения значений в заданных столбцах. Данный раздел содержит обзор синтаксиса соединений Adaptive Server Anywhere. Более подробно все упоминаемые понятия рассматриваются в последующих разделах документа.

Раздел FROM

Раздел FROM используется для указания базовых таблиц, временных таблиц, представлений или производных таблиц, подлежащих соединению. Раздел FROM может использоваться в операторах UPDATE или SELECT.

FROM *выражение_таблицы, ...*

где:

выражение_таблицы:
таблица
| *представление*
| *производная таблица*
| *соединенная таблица*
| (*выражение_таб*лицы, ...)

таблица или представление:
[*код пользователя.*] *имя-таблицы-или-представления*
[[**AS**] *корреляционное-имя*]

производная таблица:
(*оператор-Select*) [**AS**] *корреляционное-имя* [(*имя-столбца, ...*)]

соединенная таблица:
выражение_таблицы операция_соединения выражение_таблицы
[**ON** *условие_соединения*]

операция_соединения: [**KEY** | **NATURAL**] [*тип_соединения*] **JOIN**
| **CROSS JOIN**

тип_соединения:
INNER
| **FULL** [**OUTER**]
| **LEFT** [**OUTER**]
| **RIGHT** [**OUTER**]

Примечания

Фразу ON нельзя использовать с CROSS JOIN (перекрестным соединением).

Для получения информации о синтаксисе фразы ON см. раздел "Условия поиска" (Search conditions) на стр. 23 в документе "Справочник по SQL для ASA" (ASA SQL Reference Manual).

Условия соединения

Таблицы можно соединить посредством **условий соединения**. Условие соединения является обычным условием поиска. Оно выбирает поднабор строк из таблиц, соединенных по взаимосвязи между значениями в столбцах. Например, следующий запрос получает данные из таблиц *product* и *sales_order_items.*

```
SELECT *
FROM product JOIN sales_order_items
   ON product.id = sales_order_items.prod_id
```

Условие соединения в этом запросе следующее:

```
product.id = sales_order_items.prod_id
```

Это условие соединения означает, что строки могут быть объединены в результирующем наборе только при наличии в обеих таблицах одного и того же кода товара.

Условия соединения могут быть явными или сгенерированными. **Явное условие соединения** - условие соединения, вставленное во фразу ON или раздел WHERE. Следующий запрос использует фразу ON. Он предоставляет список перекрестных товаров этих двух таблиц (все комбинации строк), но исключает строки, если номера кодов не совпадают. В результате появляется список клиентов с подробным описанием их заказов.

```
SELECT *
FROM customer JOIN sales_order
  ON sales_order.cust_id = customer.id
```

Сгенерированное условие соединения - условие соединения, создающееся автоматически при указании KEY JOIN (ключевого соединения) или NATURAL JOIN (естественного соединения). В случае ключевого соединения сгенерированное условие соединения основано на связях по внешнему ключу между таблицами. В случае естественного соединения, сгенерированное условие соединения основано на столбцах с тем же самым именем.

> *Совет*
> Синтаксис как ключевого, так и естественного соединения представляет собой "быструю клавишу": при использовании ключевого слова JOIN без KEY или NATURAL с последующим явным указанием того же условия соединения во фразе ON результат будет одинаковым.

При использовании фразы ON при ключевом или естественном соединении используемое условие соединения является **конъюнкцией** явно указанного условия соединения со сгенерированным условием соединения. Это означает, что условия соединения объединены с помощью ключевого слова AND.

Соединенные таблицы

Adaptive Server Anywhere поддерживает следующие категории соединенных таблиц.

- **CROSS JOIN** (Перекрестное соединение). Перекрестное соединение двух таблиц представляет все возможные комбинации строк из этих двух таблиц. Размер результирующего набора - это количество строк в первой таблице, умноженное на количество строк во второй таблице. Перекрестное соединение также называют векторным (или декартовым) произведением. Использовать фразу ON с перекрестным соединением нельзя.
- **KEY JOIN** (Ключевое соединение, **по умолчанию**). Условие соединения генерируется автоматически на основании встроенных в базу данных связей по внешнему ключу. Ключевое соединение является значением по умолчанию при использовании ключевого слова JOIN без указания типа соединения и отсутствии фразы ON.
- **NATURAL JOIN** (Естественное соединение). Условие соединения генерируется автоматически на основании столбцов с одинаковым именем.
- **Соединение с использованием фразы ON.** Задается явное условие соединения. При использовании с ключевым или естественным соединением условие соединения содержит как генерированное условие соединения, так и явное условие соединения. При использовании с ключевым словом JOIN без KEY или NATURAL генерированное условие соединения отсутствует. Использовать фразу ON с перекрестным соединением нельзя.

Внутренние и внешние соединения

Ключевые соединения, естественные соединения и соединения с разделом ON могут быть указаны как внутренние (INNER), внешние слева (LEFT OUTER), внешние справа (RIGHT OUTER) или полностью внешние (FULL OUTER). INNER является значением по умолчанию. При использовании ключевых слов LEFT, RIGHT или FULL ключевое слово OUTER является необязательным.

В полученной в результате внутреннего соединения таблице каждая строка удовлетворяет условию соединения.

Во внешнем слева или справа соединении все строки сохраняются для одной из таблиц, а для строк других таблиц, не удовлетворяющих условию соединения, возвращаются нули. Например, во внешнем справа соединении правая сторона сохраняется, а левая сторона возвращает нуль.

В полностью внешнем соединении все строки сохраняются для обеих таблиц, а нули возвращаются для строк, не удовлетворяющих условию соединения.

Соединение двух таблиц

Для понимания процесса создания простого внутреннего соединения рассмотрим следующий запрос. Он отвечает на вопрос: "Какие размеры товара были заказаны в количестве, имеющемся на складе?".

```
SELECT DISTINCT name, size,
    sales_order_items.quantity
FROM product JOIN sales_order_items
ON product.id = sales_order_items.prod_id
  AND product.quantity = sales_order_items.quantity
```

name	size	quantity
Baseball Cap	One size fits all	12
Visor	One size fits all	36

Данный запрос можно интерпретировать следующим образом. Учтите, что это - отвлеченное пояснение обработки данного запроса, иллюстрирующее семантику запроса, использующего соединение. В нем не представлен способ, с помощью которого Adaptive Server Anywhere рассчитывает результирующий набор фактически.

- Создание перекрестного товара таблицы *product* и таблицы *sales_order_items.* Перекрестный товар содержит каждую комбинацию строк из этих двух таблиц.
- Исключение всех строк, где коды товаров не идентичны (из-за условия соединения `product.id = sales_order_items.prod_id)`.
- Исключение всех строк, где не идентично количество (из-за условия соединения `product.quantity = sales_order_items.quantity)`.
- Создание результирующей таблицы с тремя столбцами: *product.name, product.size* и *sales_order_items.quantity.*
- Исключение всех дублирующихся строк (из-за ключевого слова DISTINCT).

Для получения информации об процедуре вычисления внешних соединений см. раздел "Внешние соединения" на стр. 236.

Соединение более двух таблиц

В Adaptive Server Anywhere нет установленного предела для числа таблиц, которые можно соединить.

При соединении более двух таблиц круглые скобки необязательны. Если круглые скобки не используются, то Adaptive Server Anywhere выполняет оператор слева направо. Таким образом, `A JOIN B JOIN C` эквивалентно `(A JOIN B) JOIN C`.
Другой пример:

```
SELECT *
FROM A JOIN B JOIN C JOIN D
```

является эквивалентным

```
SELECT *
FROM ((A JOIN B) JOIN C) JOIN D
```

Всякий раз при соединении более двух таблиц соединение использует выражения таблицы. В примере `A JOIN B JOIN C` выражение таблицы `A JOIN B` соединяется с C. В принципе, это означает, что соединяются A и B, а

затем результат соединяется с C.

Порядок соединений важен, если выражение таблицы содержит внешние соединения. Например, `A JOIN B LEFT OUTER JOIN C` интерпретируется как `(A JOIN B) LEFT OUTER JOIN C`. Это означает, что выражение таблицы `A JOIN B` соединено с C. Выражение таблицы `A JOIN B` сохраняется, и таблица C возвращает нуль.

Для получения дополнительной информации о внешних соединениях см. раздел "Внешние соединения" на стр. 236.

Для получения дополнительной информации о выполнении Adaptive Server Anywhere ключевого соединения выражений таблицы см. раздел "Ключевые соединения выражений таблицы" на стр. 257.

Для получения дополнительной информации о выполнении Adaptive Server Anywhere естественного соединения выражений таблицы см. раздел "Естественные соединения выражений таблицы" на стр. 251.

Совместимые типы данных для соединения

При соединении двух таблиц сравниваемые столбцы должны содержать одинаковые или совместимые типы данных.

Для получения дополнительной информации о преобразовании типа данных в соединениях см. раздел "Преобразование при использовании операций сравнения" (Conversion when using comparison operators) на стр. 81 в документе *"Справочник по SQL для ASA" (ASA SQL Reference Manual)*.

Использование соединений в операторах DELETE, UPDATE и INSERT

Соединения можно использовать в операторах DELETE, UPDATE и INSERT, а также в операторах SELECT. Если параметр ANSI_UPDATE_CONSTRAINTS установлен в OFF, то некоторые курсоры, содержащие соединения, можно обновить. Это является значением по умолчанию для баз данных, созданных до выпуска версии 7 Adaptive Server Anywhere. Для баз данных, созданных в версии 7 или более поздних версиях, значение по умолчанию - ON.

Для получения дополнительной информации см. раздел "Параметр ANSI_UPDATE_CONSTRAINTS" на стр. 538 в документе *"Руководство по администрированию баз данных ASA" (ASA Database Administration Guide)*.

Явные условия соединения (фраза ON)

Вместо ключевого или естественного соединения (или наряду с ними) можно задать соединение, используя явное условие соединения. Условие соединения задается вставкой фразы ON сразу после соединения. Условие соединения всегда ссылается на непосредственно предшествующее ему соединение.

Пример

В следующем запросе первая фраза ON используется для соединения *sales_order* с *customer*. Вторая фраза ON используется для соединения выражения таблицы *(sales_order JOIN customer)* к базовой таблице *sales_order_item*.

```
SELECT *
FROM sales_order JOIN customer
    ON sales_order.cust_id = customer.id
  JOIN sales_order_items
    ON sales_order_items.id = sales_order.id
```

Ссылочные таблицы

Таблицы, ссылки на которые присутствуют во фразе ON, должны быть частью соединения, изменяемого фразой ON. Например, следующее недопустимо:

```
FROM (A KEY JOIN B) JOIN (C JOIN D ON A.x = C.x)
```

Проблема в том, что условие соединения `A.x = C.x` включает ссылку на таблицу A, не являющуюся частью соединения, которое она изменяет (в данном случае `C JOIN D`).

Однако начиная со стандарта ANSI/ISO SQL99 и Adaptive Server

Anywhere 7.0, из этого правила существует исключение: если между выражениями таблицы используются запятые, то условие ON может содержать ссылку на таблицу, предшествующую ему синтаксически в разделе FROM. Поэтому допустимо следующее:

```
FROM (A KEY JOIN B) , (C JOIN D ON A.x = C.x)
```

Для получения дополнительной информации о запятых см. раздел "Запятые" на стр. 234.

Сгенерированные соединения и фраза ON

Ключевые соединения являются значением по умолчанию, если используется ключевое слово JOIN, и не указано типа соединения, кроме случаев использования фразы ON. Если используется фраза ON с неуказанным JOIN, то ключевое соединение не является значением по умолчанию, и сгенерированные условия соединения не применяются.

Например, следующее является ключевым соединением, поскольку ключевое соединение - значение по умолчанию при использовании ключевого слова JOIN и отсутствии фразы ON:

```
SELECT *
FROM A JOIN B
```

Далее показано соединение между таблицей A и таблицей B с условием соединения `A.x = B.y`. Это не ключевое соединение.

```
SELECT *
FROM A JOIN B ON A.x = B.y
```

Если задается KEY JOIN или NATURAL JOIN, и используется фраза ON, то окончательное условие соединения является конъюнкцией сгенерированного условия соединения и явного условия (условий) соединения. Например, следующий оператор имеет два условия соединения: одно - сгенерированное из-за ключевого соединения, и другое - явно заявленное во фразе ON.

```
SELECT *
FROM A KEY JOIN B ON A.x = B.y
```

Если условие соединения, сгенерированное ключевым соединением, - `A.w = B.z`, то следующий оператор является эквивалентом:

```
SELECT *
FROM A JOIN B
  ON A.x = B.y
  AND A.w = B.z
```

☞ Для получения дополнительной информации о ключевых соединениях см. "Ключевые соединения" на стр. 254.

Типы явных условий соединения

Большинство условий соединения основано на равенстве и поэтому называются **соединениями по эквивалентности**. Пример:

```
SELECT *
FROM department JOIN employee
ON department.dept_id = employee.dept_id
```

Однако в условии соединения не нужно использовать знак равенства (=). Можно использовать любое условие поиска, например, условия, содержащие LIKE, SOUNDEX, BETWEEN, > (знак "больше") и != ("не равно").

Пример

Следующий пример дает ответ на вопрос: "Каких товаров заказано больше, чем имеется на складе?"

```
SELECT DISTINCT product.name
FROM product JOIN sales_order_items
ON product.id = sales_order_items.prod_id
AND product.quantity > sales_order_items.quantity
```

☞ Для получения дополнительной информации об условиях поиска см. "Условия поиска" (Search conditions) на стр. 23 в документе *"Справочник по SQL для ASA" (ASA SQL Reference Manual)*.

Использование раздела WHERE для условий соединения

Вместо фразы ON условия соединения можно задавать в разделе WHERE, кроме случаев использования внешних соединений. Однако следует помнить о возможности возникновения семантических различий между ними, если запрос содержит внешние соединения.

Фраза ON является частью раздела FROM и поэтому обрабатывается до раздела WHERE. На результаты это не влияет, за исключением случая внешнего соединения, где использование раздела WHERE может привести к преобразованию соединения во внутреннее.

При принятии решения о размещении условий соединения во фразе ON или в разделе WHERE не забывайте соблюдать следующие правила:

- При задании внешнего соединения вставка условия соединения в раздел WHERE может преобразовать внешнее соединение во внутреннее соединение.

Для получения дополнительной информации о разделе WHERE и внешних соединениях см. раздел "Внешние соединения и условия соединения" на стр. 238.

- Условия во фразе ON могут содержать ссылки только на таблицы, находящихся в выражениях таблицы, соединенных соответствующим JOIN. Однако условия в разделе WHERE могут содержать ссылки на любые таблицы, даже если они не являются частью соединения.
- Фразу ON нельзя использовать с ключевыми словами CROSS JOIN, однако всегда можно использовать раздел WHERE.
- Если условия соединения находятся во фразе ON, ключевое соединение не является значением по умолчанию. Однако ключевое соединение может быть значением по умолчанию, если условия соединения помещены в раздел WHERE.

Для получения дополнительной информации об условиях, при которых ключевое соединение является значением по умолчанию, см. раздел "Случаи использования ключевого соединения как значения по умолчанию" на стр. 254.

В примерах этой документации условия соединения помещены во фразу ON. Это необходимо в примерах, использующих внешние соединения. В других случаях это делается для того, чтобы подчеркнуть, что они являются условиями соединения, а не общими условиями поиска.

Перекрестные соединения

Перекрестное соединение двух таблиц содержит все возможные комбинации строк из этих двух таблиц. Перекрестное соединение также называют векторным (или декартовым) произведением.

Каждая строка первой таблицы появляется только один раз с каждой строкой второй таблицы. Следовательно, количество строк в результирующем наборе является произведением количества строк в первой таблице и количества строк во второй таблице минус любые строки, опущенные из-за ограничений в разделе WHERE.

Использовать фразу ON с перекрестными соединениями нельзя. Однако ограничения можно поместить в раздел WHERE.

Внутренние и внешние модификаторы не применяются к перекрестным соединениям

Все строки обеих таблиц всегда появляются в результирующем наборе перекрестных соединений, за исключением случаев наличия дополнительных ограничений в разделе WHERE. Таким образом, ключевые слова INNER, LEFT OUTER и RIGHT OUTER не применяются к перекрестным соединениям.

Например, следующий оператор соединяет две таблицы.

```
SELECT *
FROM A CROSS JOIN B
```

В результирующий набор из этого запроса входят все столбцы в A и все столбцы в B. В результирующем наборе есть одна строка для каждой комбинации строки в A и строки в B. Если в A - *n* строк, а B - *m* строк, то запрос возвращает *n x m* строк.

Запятые

Запятые выполняют функцию подобно функции операции соединения, но не являются ими. Запятая создает векторное произведение точно так же, как ключевое слово CROSS JOIN. Однако ключевые слова создают выражения таблицы, а запятые создают списки выражений таблицы.

В следующем простом внутреннем соединении двух таблиц запятая и ключевые слова CROSS JOIN эквивалентны:

```
Select *
FROM A CROSS JOIN B CROSS JOIN C
WHERE A.x = B.y
```

и

```
Select *
FROM A, B, C
WHERE A.x = B.y
```

Обычно запятую можно использовать вместо ключевых слов CROSS JOIN. Синтаксис запятых эквивалентен синтаксису перекрестного соединения, кроме случая, когда сгенерированные условия соединения в выражениях таблицы используют запятые.

☞ Для получения информации о функции запятых при сгенерированных условиях соединения см. раздел "Ключевые соединения выражений таблицы" на стр. 257.

☞ В синтаксисе звездообразных соединений запятые используются особым образом. Для получения дополнительной информации см. раздел "Повторяющиеся корреляционные имена в соединениях (звездообразные соединения)" на стр. 245.

Внутренние и внешние соединения

Ключевые слова INNER, LEFT OUTER, RIGHT OUTER и FULL OUTER могут использоваться для изменения ключевых соединений, естественных соединений и соединений с фразой ON. INNER является значением по умолчанию. Ключевое слово OUTER является дополнительным. Эти модификаторы не применяются к перекрестным соединениям.

Внутренние соединения

По умолчанию соединения являются **внутренними.** Это означает, что строки включены в результирующий набор, только если они удовлетворяют условию соединения.

Пример

Например, каждая строка результирующего набор следующего запроса содержит информацию из одной строки *customer* и одной строки *sales_order*, удовлетворяя условию ключевого соединения. Если определенный клиент не разместил заказов, то условие не удовлетворено, и результирующий набор не содержит строки, соответствующей этому клиенту.

```
SELECT fname, lname, order_date
FROM customer KEY INNER JOIN sales_order
ORDER BY order_date
```

fname	lname	order_date
Hardy	Mums	1/2/00
Aram	Najarian	1/3/00
Tommie	Wooten	1/3/00
Alfredo	Margolis	1/6/00
...	...	...

Поскольку внутренние соединения и ключевые соединения являются значениями по умолчанию, тот же самый результат получается при использовании следующего раздела FROM.

```
FROM customer JOIN sales_order
```

Внешние соединения

Левое или правое **внешнее соединение** двух таблиц сохраняет все строки в одной таблице и возвращает нули для другой таблицы, если она не соответствует условию соединения. **Внешнее слева соединение** сохраняет каждую строку в левой таблице, а **внешнее справа соединение** сохраняет каждую строку в правой таблице. В **полностью внешнем соединении** сохраняются все строки из обеих таблиц.

Выражения таблицы на каждой стороне внешнего слева или справа соединения называются **сохраняемыми** и **возвращающими нуль.** Во внешнем слева соединении, левое выражение таблицы сохраняется, а правое выражение таблицы возвращает нуль.

Для получения информации о создании внешних соединений с синтаксисом Transact-SQL см. раздел "Внешние соединения в Transact-SQL (* = или = *)" на стр. 240.

Пример

Например, следующий оператор включает всех клиентов, независимо от того, разместили они заказ или нет. Если какой-либо конкретный клиент не разместил заказов, то каждый столбец в результате, соответствующий информации о заказе, содержит значение NULL.

```
SELECT lname, order_date, city
FROM customer LEFT OUTER JOIN sales_order
   ON customer.id = sales_order.cust_id
WHERE customer.state = 'NY'
ORDER BY order_date
```

lname	order_date	city
Thompson	(NULL)	Bancroft
Reiser	2000-01-22	Rockwood
Clarke	2000-01-27	Rockwood
Mentary	2000-01-30	Rockland
...	...	...

Интерпретировать внешнее соединение в этом операторе можно следующим образом. Учтите, что это - концептуальное объяснение, не описывающее фактических вычислений результирующего набора Adaptive Server Anywhere.

- Возвращение одной строки для каждого заказа на покупку, размещенного клиентом. Если клиент разместил два или больше заказа на покупку, то возвращается более одной строки, потому что для каждого заказа на покупку возвращается одна строка. Результат - тот же, что и при внутреннем соединении. Условие ON используется для приведения в соответствие клиента и строк заказа на покупку. Для этого шага раздел WHERE не используется.
- Включение одной строки для каждого клиента, не разместившего заказов. Это гарантирует включение каждой строки в таблицу клиентов. Для всех этих строк столбцы из sales_order заполнены нулями. Эти строки добавлены, потому что используется ключевое слово OUTER, которое не появилось бы во внутреннем соединении. Для этого шага не используется ни условие ON, ни раздел WHERE.
- Исключение каждой строки с клиентом, не проживающим в Нью-Йорке, используя раздел WHERE.

Внешние соединения и условия соединения

Распространенной ошибкой при работе с внешними соединениями является размещение условия соединения. В большинстве случаев, если размещаются ограничения на возвращающую нуль таблицу в разделе WHERE, то соединение эквивалентно внутреннему соединению.

Причиной этого является то, что большинство условий поиска не может быть TRUE, если какое-либо введенное значение - NULL. Ограничение раздела WHERE на возвращающую нуль таблицу сравнивает значения с нулем, что приводит к удалению строки из результирующего набора. Строки в сохраняемой таблице не сохраняются, поэтому соединение является внутренним.

Исключением из этого являются сравнения, которые могут быть истинными, когда какое-либо введенное значение - NULL. Сюда входят: IS NULL, IS UNKNOWN, IS FALSE, IS NOT TRUE, IS NULL и выражения, включающие ISNULL или COALESCE.

Пример

Например, следующий оператор выполняет внешнее слева соединение.

```
SELECT *
FROM customer KEY LEFT OUTER JOIN sales_order
   ON sales_order.order_date < '2000-01-03'
```

Напротив, следующий оператор создает внутреннее соединение.

```
SELECT lname, order_date
FROM customer KEY LEFT OUTER JOIN sales_order
   WHERE sales_order.order_date < '2000-01-03'
```

Первый из этих двух операторов можно интерпретировать следующим образом: сначала создается внешнее слева соединение таблицы customer к таблице sales_order. Результирующий набор включает каждую строку таблицы клиентов. Для клиентов, не размещавших заказы до 3 января 2000 года, поля заказа на покупку заполняются нулями.

Во втором операторе создается внешнее слева соединение таблиц customer и sales_order. Результирующий набор включает каждую строку таблицы клиентов. Для клиентов, не размещавших заказы, поля заказа на покупку заполняются нулями. Затем применяется условие WHERE путем выбора только тех строк, в которых клиент разместил заказ после 3 января 2000 года. Для клиентов, не размещавших заказы, эти значения равны NULL. Сравнение любого значения с NULL приравнивается к UNKNOWN. Следовательно, эти строки удаляются, и оператор сокращается до внутреннего соединения.

Для получения дополнительной информации об условиях поиска см. раздел "Условия поиска" (Search conditions) на стр. 23 в документе *"Справочник по SQL для ASA" (ASA SQL Reference Manual)*.

Пояснения к сложным внешним соединениям

Порядок соединений важен, когда запрос включает в себя выражения таблиц, использующие внешние соединения. Например, `A JOIN B LEFT OUTER JOIN C` интерпретируется как `(A JOIN B) LEFT OUTER JOIN C`. Это означает, что выражение таблицы `(A JOIN B)` присоединяется к C. Выражение таблицы `(A JOIN B)` сохраняется, а таблица C возвращает нуль.

Рассмотрим следующий оператор, в котором A, B и C являются таблицами:

```
SELECT *
FROM A LEFT OUTER JOIN B RIGHT OUTER JOIN C
```

Для понимания схемы работы этого оператора учтите, что Adaptive Server Anywhere выполняет операторы слева направо, добавляя круглые скобки. Результатом является следующее:

```
SELECT *
FROM (A LEFT OUTER JOIN B) RIGHT OUTER JOIN C
```

Затем может возникнуть необходимость преобразования внешнего справа соединения во внешнее слева соединение так, чтобы оба соединения были одного типа. Для этого просто поменяйте местами положение таблиц во внешнем справа соединении, результатом чего будет следующее:

```
SELECT *
FROM C LEFT OUTER JOIN (A LEFT OUTER JOIN B)
```

A - сохраняемая таблица, а B - таблица, возвращающая нуль для вложенного внешнего соединения. C - сохраняемая таблица для первого внешнего соединения.

Это соединение можно интерпретировать следующим образом:

- Присоединение A к B с сохранением всех строк в A.
- Затем следует соединение C с результатами соединения A и B, сохраняя все строки в C.

Соединение не имеет фразы ON, и поэтому по умолчанию является ключевым соединением. Для получения информации о способе, при помощи которого Adaptive Server Anywhere генерирует условия соединения для этого типа соединений, см. раздел "Ключевые соединения выражений таблиц без запятых" на стр. 258.

Кроме этого, условие соединения для внешнего соединения должно включать только те таблицы, на которые предварительно были включены ссылки в разделе FROM. Это ограничение соответствует стандарту ANSI/ISO и применяется во избежание двусмысленности. Например, следующие два оператора синтаксически некорректны, потому что в условии соединения на C содержится ссылка, предшествующая ссылке на саму таблицу.

```
SELECT *
FROM (A LEFT OUTER JOIN B ON B.x = C.x) JOIN C
```

и

```
SELECT *
FROM A LEFT OUTER JOIN B ON A.x = C.x, C
```

Внешние соединения представлений и производных таблиц

Внешние соединения также могут быть определены для представлений и производных таблиц.

Оператор

```
SELECT *
FROM V LEFT OUTER JOIN A ON (V.x = A.x)
```

может интерпретироваться следующим образом:

- Вычисление представления V.
- Соединение всех строки из вычисленного представления V с A путем сохранения всех строк из V, используя условие соединения `V.x = A.x`

Пример

Следующий пример определяет представление с именем V, которое возвращает коды служащих и названия отделов, относящихся к женщинам, которые зарабатывают больше 60 000 долл.

```
CREATE VIEW V AS
SELECT employee.emp_id, dept_name
  FROM employee JOIN department
    ON employee.dept_id = department.dept_id
  WHERE sex = 'F' and salary > 60000
```

Затем воспользуйтесь этим представлением для добавления списка отделов, в которых работают женщины, и регионов их продаж. Представление V сохраняется, и *sales_order* возвращает нуль.

```
SELECT DISTINCT V.emp_id, region, V.dept_name
  FROM V LEFT OUTER JOIN sales_order
    ON V.emp_id = sales_order.sales_rep
```

emp_id	region	dept_name
243	(NULL)	R & D
316	(NULL)	R & D
529	(NULL)	R & D
902	Eastern	Sales
...	...	...

Внешние соединения Transact-SQL (*= или =*)

В соответствии со стандартами SQL ANSI/ISO, Adaptive Server Anywhere поддерживает внешние соединения посредством ключевых слов LEFT OUTER и RIGHT OUTER. Для совместимости с Adaptive Server Enterprise в Adaptive Server Anywhere также поддерживаются аналоги Transact-SQL этих ключевых слов - *= и =*. Однако существуют некоторые ограничения, которые описаны ниже.

В диалекте Transact-SQL внешние соединения создаются предоставлением списка таблиц с разделителями-запятыми в разделе FROM и использованием специальных операций *= или =* в разделе WHERE. В Adaptive Server Enterprise до версии 12 условие соединения должно появляться в разделе WHERE (ON не поддерживалась).

> *Предупреждение*
> При создании внешних соединений не смешивайте синтаксис *= с синтаксисом фразы ON. Это также применяется к представлениям, на которые в запросе содержатся ссылки.

Пример

Например, следующее внешнее слева соединение перечисляет всех клиентов и находит даты их заказов (если они есть):

```
SELECT fname, lname, order_date
FROM customer, sales_order
WHERE customer.id *= sales_order.cust_id
ORDER BY order_date
```

Этот оператор эквивалентен следующему оператору, в котором используется синтаксис ANSI/ISO:

```
SELECT fname, lname, order_date
FROM customer LEFT OUTER JOIN sales_order
ON customer.id = sales_order.cust_id
ORDER BY order_date
```

Ограничения внешних соединений Transact-SQL

Для внешних соединений Transact-SQL существует ряд ограничений:

- Если внешнее соединение и уточнение определяются по столбцу таблицы внешнего соединения, возвращающей нуль, то результаты могут не соответствовать ожидаемым. Уточнение в запросе исключает строки из результирующего набора, но влияет скорее на значения, появляющиеся в строках результирующего набора. Для строк, не соответствующих уточнению, в таблице, возвращающей нуль, появляется нулевое значение.
- Синтаксис ANSI/ISO SQL и синтаксис внешнего соединения Transact-SQL нельзя смешивать в одном запросе. Если представление определено с использованием одного диалекта для внешнего соединения, то необходимо использовать тот же диалект для любых запросов внешнего соединения в этом представлении.
- Таблица, возвращающая нуль, не может участвовать одновременно во внешнем соединении Transact-SQL и в обычном соединении, либо в двух внешних соединениях. Например, следующий раздел WHERE недопустим, потому что таблица S нарушает это ограничение.

```
WHERE R.x *= S.x
AND S.y = T.y
```

Если нельзя переписать запрос, избежав использования таблицы как в разделе внешнего соединения, так и в разделе обычного соединения, то необходимо разделить оператор на два отдельных запроса, либо использовать только синтаксис ANSI/ISO SQL.

- Нельзя использовать подзапрос, содержащий условие соединения, задействующее возвращающую ноль таблицу внешнего соединения. Например, следующий раздел WHERE недопустим:

```
WHERE R.x *= S.y
AND EXISTS ( SELECT *
             FROM T
             WHERE T.x = S.x )
```

Использование представлений с внешними соединениями Transact-SQL

Если определяется представление с внешним соединением, и затем выполняется запрос представления с квалификацией по столбцу из таблицы внешнего соединения, возвращающей ноль, то результаты могут отличаться от ожидаемых. Запрос возвращает все строки из таблицы, возвращающей ноль. Не соответствующие квалификации строки имеют значение NULL в соответствующих столбцах этих строк.

Следующие правила определяют допустимые типы обновлений столбцов через представления, содержащие внешние соединения:

- Операторы INSERT и DELETE недопустимы в представлениях внешних соединений.
- Операторы UPDATE допустимы в представлениях внешних соединений. Если представление определено параметром WITH CHECK, обновление не выполняется, если любой из задействуемых столбцов появляется в разделе WHERE в выражении, включающее столбцы более чем из одной таблицы.

NULL в соединениях Transact-SQL

Значения NULL в соединяемых таблицах или представлениях никогда не соответствуют друг другу во внешнем соединении Transact-SQL. Результатом сравнения значения NULL с любым другим значением NULL является FALSE.

Специализированные соединения

В этом разделе описывается создание некоторых специализированных соединений, таких, как самосоединения, звездообразные соединения и соединения, использующие производные таблицы.

Самосоединения

В самосоединении таблица присоединяется сама к себе путем ссылки на ту же таблицу с использованием другого корреляционного имени.

Пример 1

Следующее самосоединение представляет список пар служащих. Имя каждого служащего появляется в комбинации с именами каждого служащего.

```
SELECT a.emp_fname, a.emp_lname,
       b.emp_fname, b.emp_lname
FROM employee AS a CROSS JOIN employee AS b
```

emp_fname	emp_lname	emp_fname	emp_lname
Fran	Whitney	Fran	Whitney
Fran	Whitney	Matthew	Cobb
Fran	Whitney	Philip	Chin
Fran	Whitney	Julie	Jordan
...	...	...	...

Поскольку в таблице employee содержится 75 строк, это соединение содержит 75 x 75 = 5 625 строк. Сюда также входят строки, в которых каждый сотрудник перечислен сам с собой. Например, это соединение содержит строку

emp_fname	emp_lname	emp_fname	emp_lname
Fran	Whitney	Fran	Whitney

Если необходимо исключить строки, содержащие одно и то же имя дважды, добавьте условие соединения, запрещающее идентичность кодов служащих.

```
SELECT a.emp_fname, a.emp_lname,
       b.emp_fname, b.emp_lname
FROM employee AS a CROSS JOIN employee AS b
WHERE a.emp_id != b.emp_id
```

Без этих повторяющихся строк соединение содержит 75 x 74 = 5 550 строк.

В этом новом соединении содержатся строки, располагающие каждого служащего в паре с каждым другим служащего, но каждая пара появляется дважды, поскольку каждая пара имен может появляться в двух возможных заказах. Например, результат вышеуказанного соединения содержит следующие две строки:

.

emp_fname	emp_lname	emp_fname	emp_lname
Matthew	Cobb	Fran	Whitney
Fran	Whitney	Matthew	Cobb

Если порядок имен неважен, то можно составить список из (75 x 74)/2 = 2 775 уникальных пар.

```
SELECT a.emp_fname, a.emp_lname,
       b.emp_fname, b.emp_lname
FROM employee AS a CROSS JOIN employee AS b
WHERE a.emp_id < b.emp_id
```

Этот оператор устраняет повторяющиеся строки, выбирая только строки, в которых *emp_id* служащего a меньше, чем emp_id служащего b.

Пример 2

Следующее самосоединение использует корреляционные имена *report* и *manager*, чтобы различить два экземпляра таблицы *employee*, и создает список служащих и их менеджеров.

```
SELECT report.emp_fname, report.emp_lname,
       manager.emp_fname, manager.emp_lname
FROM employee AS report JOIN employee AS manager
       ON (report.manager_id = manager.emp_id)
ORDER BY report.emp_lname, report.emp_fname
```

Результат действия этого оператора частично показан ниже. Имена служащих появляются в двух столбцах слева, а имена их менеджеров - справа.

emp_fname	emp_lname	emp_fname	emp_lname
Alex	Ahmed	Scott	Evans
Joseph	Barker	Jose	Martinez
Irene	Barletta	Scott	Evans
Jeannette	Bertrand	Jose	Martinez
...	...	...	...

Повторяющиеся корреляционные имена в соединениях (звездообразные соединения)

Повторяющиеся имена таблиц используются для создания **звездообразного соединения**. В звездообразном соединении к одной таблице или представлению присоединяются несколько других.

Для создания звездообразного соединения то же имя таблицы, имя представления или корреляционное имя используются из раздела FROM более одного раза. Это является расширением к стандарту ANSI/ISO SQL. Способность использовать повторяющиеся имена не добавляет дополнительных функциональных возможностей, но формулировать некоторые запросы становится гораздо проще.

Для того чтобы синтаксис имел смысл, повторяющиеся имена должны находиться в разных соединениях. Когда имя таблицы или имя представления используется дважды в одном соединении, второй экземпляр игнорируется. Например, и `FROM A,A`, и `FROM A CROSS JOIN A` интерпретируются как `FROM A`.

Следующий пример, в котором A, B и C являются таблицами, допустим в Adaptive Server Anywhere. В этом примере тот же экземпляр таблицы A присоединяется к B и C. Помните, что для разделения соединений в звездообразном соединении требуется запятая. Использование запятой в звездообразных соединениях является отличительной чертой синтаксиса звездообразных соединений.

```
SELECT *
FROM A LEFT OUTER JOIN B ON A.x = B.x,
     A LEFT OUTER JOIN C ON A.y = C.y
```

Следующий пример является эквивалентом.

```
SELECT *
FROM A LEFT OUTER JOIN B ON A.x = B.x,
     C RIGHT OUTER JOIN A ON A.y = C.y
```

Оба из них эквивалентны следующему стандартному синтаксису ANSI/ISO (круглые скобки необязательны).

```
SELECT *
FROM (A LEFT OUTER JOIN B ON A.x = B.x)
LEFT OUTER JOIN C ON A.y = C.y
```

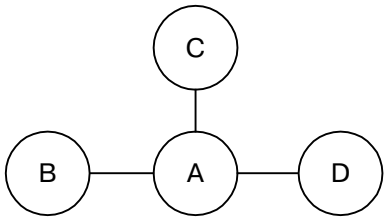
В следующем примере таблица A присоединяется к трем таблицам: B, C и D.

```
SELECT *
FROM A JOIN B ON A.x = B.x,
     A JOIN C ON A.y = C.y,
     A JOIN D ON A.w = D.w
```

Это эквивалентно следующему стандартному синтаксису ANSI/ISO (круглые скобки необязательны).

```
SELECT *
FROM ((A JOIN B ON A.x = B.x)
JOIN C ON A.y = C.y)
```

В сложных соединениях это помогает построить диаграмму. Предыдущий пример можно описать следующей диаграммой, иллюстрирующей соединение таблиц B, C и D через таблицу A.



Примечание

Повторяющиеся имена таблиц можно использовать только тогда, когда параметр EXTENDED_JOIN_SYNTAX установлен в ON (значение по умолчанию).

Для получения дополнительной информации см. раздел "Параметр EXTENDED_JOIN_SYNTAX" (EXTENDED_JOIN_SYNTAX option) на стр. 554 в документе *"Руководство по администрированию баз данных ASA" (ASA Database Administration Guide)*.

Пример 1

Создайте список имен клиентов, разместивших заказы с Роллином Оверби (Rollin Overbey). Обратите внимание, что одна из таблиц в разделе FROM - *employee* - не вносит никаких столбцов в результаты. В результатах также не появляется ни один из присоединенных столбцов, например, *customer.id* или *employee.id.*

Тем не менее, это соединение возможно только с использованием таблицы *employee* в разделе FROM.

```
SELECT customer.fname, customer.lname,
   sales_order.order_date
FROM sales_order KEY JOIN customer,
     sales_order KEY JOIN employee
WHERE employee.emp_fname = 'Rollin'
  AND employee.emp_lname = 'Overbey'
ORDER BY sales_order.order_date
```

fname	lname	order_date
Tommie	Wooten	1/3/00
Michael	Agliori	1/8/00
Salton	Pepper	1/17/00
Tommie	Wooten	1/23/00
...	...	...

Следующее является эквивалентным оператором в стандартном синтаксисе ANSI/ISO:

```
SELECT customer.fname, customer.lname,
  sales_order.order_date
FROM sales_order JOIN customer
  ON sales_order.cust_id = customer.id
JOIN employee
  ON sales_order.sales_rep = employee.emp_id
WHERE employee.emp_fname = 'Rollin'
  AND employee.emp_lname = 'Overbey'
ORDER BY sales_order.order_date
```

Пример 2

Этот пример дает ответ на вопрос: "Сколько каждого товара заказал каждый клиент, и кто является менеджером продавца, принявшего заказ?"

Для получения ответа на этот вопрос начните с составления списка информации, которую необходимо получить. В данном случае это товар, количество, имя клиента и имя менеджера. Затем перечислите таблицы, в которых содержится эта информация. Это таблицы *product*, *sales_order_items*, *customer* и *employee*. Если посмотреть на структуру демонстрационной базы данных (см. раздел "Схема демонстрационной базы данных" на стр. 224), то можно заметить, что все эти таблицы взаимосвязаны через таблицу *sales_order*. Можно создать звездообразное соединение для таблицы *sales_order* для получения информации из других таблиц.

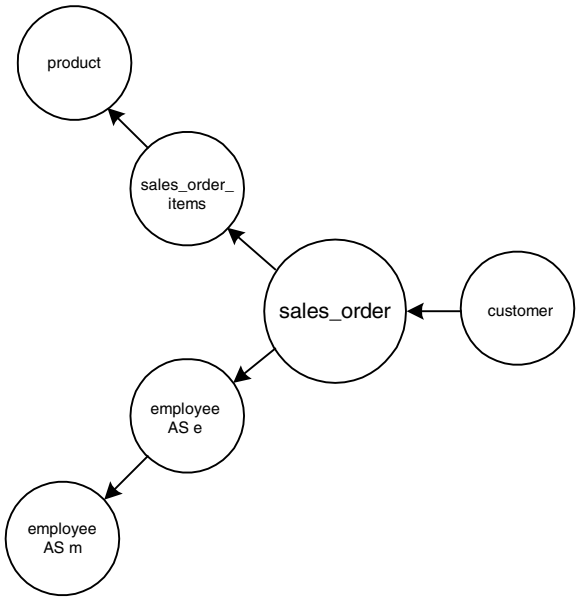
Кроме того, необходимо создать самосоединение для получения имени менеджера, потому что в таблице *employee* содержатся коды менеджеров и имена всех служащих, но не содержится столбца с именами только менеджеров. Для получения дополнительной информации см. раздел "Самосоединения" на стр. 243.

Следующий оператор создает звездообразное соединение вокруг таблицы *sales_order*. Все соединения являются внешними для того, чтобы результирующий набор включал всех клиентов. Некоторые клиенты не размещали заказов, поэтому другие значения для этих клиентов будут NULL. Результирующий набор содержит следующие столбцы: клиент, товар, заказанное количество и имя менеджера продавца.

```
SELECT customer.fname, product.name,
  SUM(sales_order_items.quantity), m.emp_fname
FROM sales_order
        KEY RIGHT OUTER JOIN customer,
     sales_order
        KEY LEFT OUTER JOIN sales_order_items
        KEY LEFT OUTER JOIN product,
     sales_order
        KEY LEFT OUTER JOIN employee AS e
        LEFT OUTER JOIN employee AS m
        ON (e.manager_id = m.emp_id)
WHERE customer.state = 'CA'
GROUP BY customer.fname, product.name, m.emp_fname
ORDER BY SUM(sales_order_items.quantity) DESC,
customer.fname
```

fname	name	SUM(sales_order_items.quantity	emp_fname
Sheng	Baseball Cap	240	Moira
Laura	Tee Shirt	192	Moira
Moe	Tee Shirt	192	Moira
Leilani	Sweatshirt	132	Moira
...	...	...	...

Ниже приведена диаграмма таблиц в этом звездообразном соединении. Стрелки указывают направленность (влево или вправо) внешних соединений. Как видно, полный список клиентов поддерживается во всех соединениях.



Следующий стандартный синтаксис ANSI/ISO эквивалентен звездообразному соединению, приведенному в примере 2.

```
SELECT customer.fname, product.name,
  SUM(sales_order_items.quantity), m.emp_fname
FROM sales_order LEFT OUTER JOIN sales_order_items
    ON sales_order.id = sales_order_items.id
  LEFT OUTER JOIN product
    ON sales_order_items.prod_id = product.id
  LEFT OUTER JOIN employee as e
    ON sales_order.sales_rep = e.emp_id
  LEFT OUTER JOIN employee as m
    ON e.manager_id = m.emp_id
  RIGHT OUTER JOIN customer
    ON sales_order.cust_id = customer.id
WHERE customer.state = 'CA'
GROUP BY customer.fname, product.name, m.emp_fname
ORDER BY SUM(sales_order_items.quantity) DESC,
customer.fname
```

Соединения, включающие производные таблицы

Производные таблицы позволяют вкладывать запросы в пределах раздела FROM. При помощи производных таблиц можно выполнить группировку групп или соединение с группой без необходимости создания представления.

В следующем примере внутренний оператор SELECT (заключенный в круглые скобки) создает производную таблицу, сгруппированную по значениям кодов клиентов. Внешний оператор SELECT назначает этой таблице корреляционное имя *sales_order_counts* и присоединяет его к таблице *customer*, используя условие соединения.

```
SELECT lname, fname, number_of_orders
FROM customer JOIN
   ( SELECT cust_id, count(*)
     FROM sales_order
     GROUP BY cust_id )
   AS sales_order_counts (cust_id, number_of_orders)
   ON (customer.id = sales_order_counts.cust_id)
WHERE number_of_orders > 3
```

Результатом является таблица имен клиентов, разместивших более трех заказов, включая количество заказов, размещенных каждым клиентом.

Для получения информации о ключевых соединениях производных таблиц см. раздел "Ключевые соединения представлений и производных таблиц" на стр. 263.

Для получения информации о естественных соединениях производных таблиц см. раздел "Естественные соединения представлений и производных таблиц" на стр. 252.

Для получения информации о внешних соединениях производных таблиц см. раздел "Внешние соединения представлений и производных таблиц" на стр. 240.

Естественные соединения

При определении естественного соединения Adaptive Server Anywhere генерирует условие соединения, основанное на столбцах с тем же именем. Для того чтобы задействовать это в естественном соединении базовых таблиц, необходимо иметь, по крайней мере, одну пару столбцов с тем же именем, по одному столбцу из каждой таблицы. Если столбца с одинаковым имени нет, то выдается ошибка.

Если таблица A и таблица B имеют одно общее имя столбца, и столбец называется *x*, то

```
SELECT *
FROM A NATURAL JOIN B
```

является эквивалентным следующему:

```
SELECT *
 FROM A JOIN B
 ON A.x = B.x
```

Если таблица A и таблица B имеют два общих имени столбцов, названных a и b, то `A NATURAL JOIN B` эквивалентно следующему:

```
A JOIN B
 ON A.a = B.a
 AND A.b = B.b
```

Пример

Например, можно соединить таблицы employee и department, используя естественное соединение, потому что у них есть общее имя столбца - столбец *dept_id*.

```
SELECT emp_fname, emp_lname, dept_name
FROM employee NATURAL JOIN department
ORDER BY dept_name, emp_lname, emp_fname
```

emp_fname	emp_lname	dept_name
Janet	Bigelow	Finance
Kristen	Coe	Finance
James	Coleman	Finance
Jo Ann	Davidson	Finance
...	...	...

Следующий оператор эквивалентен. Он явно определяет условие соединения, сгенерированное в предыдущем примере.

```
SELECT emp_fname, emp_lname, dept_name
FROM employee JOIN department
  ON (employee.dept_id = department.dept_id)
ORDER BY dept_name, emp_lname, emp_fname
```

Естественные соединения с фразой ON

При определении NATURAL JOIN *и* размещении условия соединения во фразе ON результатом является конъюнкция этих двух условий соединения.

Например, следующие два запроса эквивалентны. В первом запросе Adaptive Server Anywhere генерирует условие соединения `employee.dept_id = department.dept_id`. Этот запрос также содержит явное условие соединения.

```
SELECT emp_fname, emp_lname, dept_name
FROM employee NATURAL JOIN department
  ON employee.manager_id = department.dept_head_id
```

Следующий запрос эквивалентен. Естественное условие соединения, сгенерированное в предыдущем запросе, определено в нем во фразе ON.

```
SELECT emp_fname, emp_lname, dept_name
FROM employee JOIN department
  ON employee.manager_id = department.dept_head_id
   AND employee.dept_id = department.dept_id
```

Естественные соединения выражений таблиц

При наличии множественного выражения таблиц как минимум на одной стороне естественного соединения Adaptive Server Anywhere генерирует условие соединения, сравнивая набор столбцов для каждой стороны операции соединения и выполняя поиск столбцов с тем же именем.

Например, в операторе

```
SELECT *
FROM (A JOIN B) NATURAL JOIN (C JOIN D)
```

есть два выражения таблиц. Имена столбцов в выражении таблиц `A JOIN B` сравниваются с именами столбцов в выражении таблиц `C JOIN D`, и условие соединения генерируется для каждой однозначной пары соответствия именам столбцов. *Однозначная пара соответствия столбцам означает*, что имя столбца появляется в обоих выражениях таблиц, но не появляется дважды в том же выражении таблиц.

Если существует пара неоднозначных имен столбцов, то выдается ошибка. Однако имя столбца может появиться дважды в одном выражении таблиц, если оно не соответствует имени столбца в другом выражении таблиц.

Естественные соединения списков

Когда список выражений таблиц находится, по крайней мере, на одной стороне естественного соединения, тогда для каждого выражения таблиц в списке генерируется отдельное условие соединения.

Рассмотрите следующие таблицы:

- таблица A состоит из столбцов, названных *a, b* и *c*;
- таблица B состоит из столбцов, названных *a* и *d*;
- таблица C состоит из столбцов, названных *d* и *c*.

В этом случае соединение `(A,B) NATURAL JOIN C` заставляет Adaptive Server Anywhere генерировать два условия соединения:

```
ON A.c = C.c
 AND B.d = C.d
```

Если для A-C или B-C нет общего имени столбца, то выдается ошибка.

Если таблица C состоит из столбцов *a, d* и *c*, то соединение `(A,B) NATURAL JOIN C` недопустимо. Причиной является то, что столбец *a* появляется во всех трех таблицах, и данное соединение становится неоднозначным.

Пример

В следующем примере для каждой продажи предоставляется информация о том, что и кем было продано.

```
SELECT *
FROM (employee KEY JOIN sales_order)
  NATURAL JOIN (sales_order_items KEY JOIN product)
```

Это эквивалентно следующему:

```
SELECT *
FROM (employee KEY JOIN sales_order)
  JOIN (sales_order_items KEY JOIN product)
    ON sales_order.id = sales_order_items.id
```

Естественные соединения представлений и производных таблиц

Расширением к стандарту SQL ANSI/ISO является то, что можно определить представления или производные таблицы с обеих сторон естественного соединения. В следующем операторе

```
SELECT *
FROM View1 NATURAL JOIN View2
```

столбцы в View1 сравниваются со столбцами в *View2*. Если, например, обнаруживается, что столбец с именем *emp_id*, появлялся в обоих представлениях, а других столбцов с идентичными именами нет, тогда сгенерированное условие соединения следующее:

```
(View1.emp_id = View2.emp_id).
```

Пример

Следующий пример иллюстрирует тот факт, что представление, использованное в естественном соединении, может включать в себя выражения, а не только столбцы, и эти выражения рассматриваются в естественном соединении таким же образом. Для начала создайте следующим образом представление *V* со столбцом, названным *x*:

```
CREATE VIEW V(x) AS
SELECT R.y + 1
FROM R
```

Затем создайте естественное соединение представления с производной таблицей. Производная таблица имеет корреляционное имя T со столбцом, названным *x*.

```
SELECT *
FROM V NATURAL JOIN (SELECT P.y FROM P) as T(x)
```

Это соединение эквивалентно следующему:

```
SELECT *
FROM V JOIN (SELECT P.y FROM P) as T(x) ON (V.x = T.x)
```

Ключевые соединения

При определении ключевого соединения Adaptive Server Anywhere генерирует условие соединения, основанное на связях по внешнему ключу в базе данных. Для использования ключевого соединения необходимо наличие связи по внешнему ключу между таблицами, в противном случае выдается ошибка.

Ключевое соединение - это расширение Sybase к стандарту SQL ANSI/ISO. Оно не обеспечивает повышенной функциональности, но формулировать некоторые запросы становится проще.

Случаи использования ключевого соединения как значения по умолчанию

Ключевое соединение является в Adaptive Server Anywhere значением по умолчанию в следующих случаях:

- используется ключевое слово JOIN;
- ключевые слова CROLL, NATURAL или KEY не определены;
- отсутствует фраза ON.

Пример

Например, следующий запрос является простым ключевым соединением, связывающим таблицы product и *sales_order_items* на основе связи по внешнему ключу в базе данных.

```
SELECT *
FROM product KEY JOIN sales_order_items
```

Следующий запрос эквивалентен. Он не учитывает слово KEY, но по умолчанию JOIN без фразы ON является KEY JOIN.

```
SELECT *
FROM product JOIN sales_order_items
```

Следующий запрос также эквивалентен, потому что условие соединения, указанное во фразе ON, оказывается таким же, как условие соединения, генерируемое Adaptive Server Anywhere для этих таблиц, основанных на их связи по внешнему ключу в демонстрационной базе данных.

```
SELECT *
FROM product JOIN sales_order_items
ON sales_order_items.prod_id = product.id
```

Ключевые соединения с фразой ON

При определении KEY JOIN и размещении условия соединения во фразе ON результатом является конъюнкция этих двух условий соединения. Например:

```
SELECT *
FROM A KEY JOIN B
ON A.x = B.y
```

Если условие соединения, сгенерированное ключевым соединением A и B, является `A.w = B.z`, то этот запрос эквивалентен следующему:

```
SELECT *
FROM A JOIN B
ON A.x = B.y AND A.w = B.z
```

Ключевые соединения при наличии нескольких связей по внешнему ключу

Когда Adaptive Server Anywhere пытается сгенерировать условие соединения, основанное на связи по внешнему ключу, то иногда обнаруживается наличие более одной такой связи. В этих случаях Adaptive Server Anywhere определяет, какую связь по внешнему ключу использовать, путем сопоставления функционального имени внешнего ключа с корреляционным именем таблицы первичного ключа, на которую ссылается внешний ключ.

☞ В последующих разделах описывается генерирование Adaptive Server Anywhere условий соединения для ключевых соединений. Итоговая информация представлена в разделе "Правила построения ключевых соединений" на стр. 265.

Корреляционное имя и функциональное имя

Корреляционное имя - это имя таблицы или представления, используемое в разделе FROM запроса, либо их обычное имя или псевдоним, определенные в разделе FROM.

Функциональное имя - имя внешнего ключа. Оно должно быть уникальным для данной внешней (дочерней) таблицы.

Если для внешнего ключа не определено функциональное имя, то это имя присваивается следующим образом:

- В отсутствие внешнего ключа с тем же именем, что и первичное имя таблицы, первичное имя таблицы назначается как функциональное имя.
- Если первичное имя таблицы уже используется другим внешним ключом, тогда функциональное имя является первичным именем таблицы, объединенным заполняемым нулями числом из трех цифр, уникальным для внешней таблицы.

Если функциональное имя внешнего ключа неизвестно, его можно найти в Sybase Central, развернув папку базы данных в левой области окна. Каждая таблица в базе данных имеет подпапку с именем *Foreign Keys (Внешние ключи)*. Нажмите на нее, и внешние ключи появятся в правой области окна.

☞ Диаграмму, включающую функциональные имена всех внешних ключей в демонстрационной базе данных, см. в разделе "Схема демонстрационной базы данных" на стр. 224.

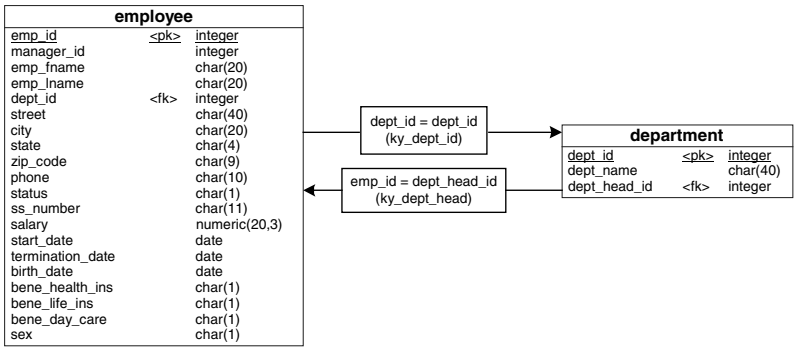
Генерация условий соединения

Adaptive Server Anywhere выполняет поиск внешнего ключа, имеющего то же функциональное имя, что и корреляционное имя таблицы первичного ключа:

- Если точно есть один внешний ключ с тем же именем, что и таблица в соединении, Adaptive Server Anywhere использует его для генерирования условия соединения.
- Если имеется более одного внешнего ключа с таким же именем, как у таблицы, тогда соединение неоднозначно, и выдается ошибка.
- Если внешнего ключа с именем как у таблицы нет, тогда Adaptive Server Anywhere выполняет поиск любой связи по внешнему ключу, даже если имена не совпадают. Если существует более одной связи по внешнему ключу, тогда соединение неоднозначно, и выдается ошибка.

Пример 1

В демонстрационной базе данных две связи по внешнему ключу определены между таблицами *employee* и *department*: внешний ключ *ky_dept_id* в таблице *employee* ссылается на таблицу *department*, и внешний ключ *ky_dept_head* в таблице *department* ссылается на таблицу *employee*.



Следующий запрос неоднозначен, потому что существуют две взаимосвязи по внешнему ключу, и ни у одной нет того же функционального имени, что и у таблицы первичного ключа. Поэтому попытка выполнить этот запрос приводит к синтаксической ошибке SQLE_AMBIGUOUS_JOIN (-147).

```
SELECT employee.emp_lname, department.dept_name
FROM employee KEY JOIN department
```

Пример 2

Определением корреляционного имени *ky_dept_id* для таблицы department этот запрос изменяет запрос в примере 1. Теперь внешний ключ *ky_dept_id* имеет то же имя, что и таблица, на которую он ссылается, и поэтому он используется для определения условия соединения. Результат включает все фамилии служащих и отделы, где они работают.

```
SELECT employee.emp_lname, ky_dept_id.dept_name
FROM employee KEY JOIN department AS ky_dept_id
```

Следующий запрос эквивалентен. В этом примере нет необходимости создания псевдонима для таблицы *department*. То же сгенерированное выше условие соединения определено во фразе ON в данном запросе:

```
SELECT employee.emp_lname, department.dept_name
FROM employee JOIN department
   ON department.dept_id = employee.dept_id
```

Пример 3

Если требуется перечислить всех служащих, возглавляющих отделы, тогда должен использоваться внешний ключ *ky_dept_head*, и пример 1 необходимо переписать следующим образом. Этот запрос делает необходимым использование внешнего ключа *ky_dept_head* путем определения корреляционного имени *ky_dept_head* для таблицы *employee* первичного ключа.

```
SELECT ky_dept_head.emp_lname, department.dept_name
FROM employee AS ky_dept_head KEY JOIN department
```

Следующий запрос эквивалентен. Условие соединения, сгенерированное выше, определено в этом запросе во фразе ON:

```
SELECT employee.emp_lname, department.dept_name
FROM employee JOIN department
   ON department.dept_head_id = employee.emp_id
```

Пример 4

В корреляционном имени нет необходимости, если функциональное имя внешнего ключа идентично имени таблицы первичного ключа. Например, можно определить department внешнего ключа для таблицы *employee*:

```
ALTER TABLE employee ADD FOREIGN KEY department
(dept_id) REFERENCES department (dept_id)
```

Теперь эта связь по внешнему ключу является условием соединения по умолчанию, если между этими двумя таблицами определено соединение KEY JOIN. Если определен *department* внешнего ключа, то следующий запрос эквивалентен примеру 3.

```
SELECT employee.emp_lname, department.dept_name
FROM employee KEY JOIN department
```

Примечание

Если попробовать применить этот пример в Interactive SQL, то изменения в демонстрационной базе данных придется отменить следующим оператором:

```
ALTER TABLE employee DROP FOREIGN KEY department
```

Ключевые соединения выражений таблиц

Adaptive Server Anywhere генерирует условия соединения для ключевого соединения выражений таблиц, проверяя связь по внешнему ключу каждой пары таблиц в операторе.

В следующем примере соединены четыре пары таблиц.

```
SELECT *
FROM (A NATURAL JOIN B) KEY JOIN (C NATURAL JOIN D)
```

Пары таблиц следующие: A-C, A-D, B-C и B-D. Adaptive Server Anywhere проверяет связь в пределах каждой пары и затем создает сгенерированное условие соединения для выражения таблиц в целом. Способ выполнения этого действия в Adaptive Server Anywhere зависит от того, используются ли в выражениях таблиц запятые. Поэтому сгенерированные условия соединения в следующих двух примерах различны. `A JOIN B` - *выражение таблиц, не содержащее запятые*, а (A, B) - *список выражений таблиц.*

```
SELECT *
FROM (A JOIN B) KEY JOIN C
```

семантически отличается от

```
SELECT *
FROM (A,B) KEY JOIN C
```

Два типа поведения соединения рассмотрены в следующих разделах:

- "Ключевые соединения выражений таблиц без запятых" на стр. 258;
- "Ключевые соединения списков выражений таблиц" на стр. 259.

Ключевые соединения выражений таблиц без запятых

Когда оба соединяемые выражения таблиц не содержат запятые, Adaptive Server Anywhere проверяет связи по внешнему ключу в парах таблиц в операторе и генерирует *единственное* условие соединения.

Например, следующее соединение имеет две пары таблиц: A-C и B-C.

```
(A NATURAL JOIN B) KEY JOIN C
```

Adaptive Server Anywhere генерирует единственное условие соединения для присоединения C к `(A NATURAL JOIN B)` путем просмотра связей по внешнему ключу в пределах пар таблиц A-C и B-C. При наличии нескольких связей по внешнему ключу он генерирует одно условие соединения для этих двух пар согласно правилам определения ключевых соединений:

- Сначала как в A-C, так и в B-C осуществляется поиск единственного внешнего ключа с функциональным именем, идентичным корреляционному имени одной из таблиц первичного ключа, на которую содержится ссылка. Если есть только один внешний ключ, соответствующий этому критерию, то этот внешний ключ и используется. Если существует более одного внешнего ключа с функциональным именем, идентичным корреляционному имени таблицы, тогда соединение считается неоднозначным, и выдается ошибка.
- Если внешнего ключа с именем, идентичным корреляционному имени таблицы нет, тогда Adaptive Server Anywhere выполняет поиск любой связи по внешнему ключу между таблицами. Если такой внешний ключ есть, он используется. Если существует более одного таких внешних ключей, тогда соединение считается неоднозначным, и выдается ошибка.

- Если связи по внешнему ключу нет, то выдается ошибка.

Для получения дополнительной информации см. раздел "Ключевые соединения с несколькими связями по внешнему ключу" на стр. 255.

Пример

Следующий запрос находит всех служащих, являющихся торговыми представителями, и их отделы.

```
SELECT employee.emp_lname, ky_dept_id.dept_name
FROM (employee KEY JOIN department as ky_dept_id)
   KEY JOIN sales_order
```

Этот запрос можно интерпретировать следующим образом.

- Adaptive Server Anywhere рассматривает выражение таблиц `(employee KEY JOIN department as ky_dept_id)` и генерирует условие соединения `employee.dept_id = ky_dept_id.dept_id`, основанное на внешнем ключе *ky_dept_id*.
- Затем Adaptive Server Anywhere рассматривает пары таблиц *employee-sales_order* и *ky_dept_id-sales_order*. Помните, что между таблицами *sales_order* и *employee* и между *sales_order* и *ky_dept_id* может существовать только один внешний ключ, иначе соединение неоднозначно. Получилось так, что существует только одна связь по внешнему ключу между таблицами *sales_order* и *employee (ky_so_employee_id)*, и не существует связи по внешнему ключу между *sales_order* и *ky_dept_id*. Следовательно, сгенерированное условие соединения - `sales_order.emp_id = employee.sales_rep`.

Поэтому следующий запрос эквивалентен предыдущему запросу:

```
SELECT employee.emp_lname, department.dept_name
FROM (employee JOIN department
   ON ( employee.dept_id = department.dept_id ) )
JOIN sales_order
   ON (employee.emp_id = sales_order.sales_rep)
```

Ключевые соединения списков выражений таблицы

Для того чтобы сгенерировать условие соединения для ключевого соединения двух списков выражений таблицы, Adaptive Server Anywhere изучает пары таблиц в операторе и генерирует условие соединения для каждой пары. Заключительное условие соединения является конъюнкцией условий соединения для *каждой пары*. Между каждой парой должна быть связь по внешнему ключу.

В следующем примере соединены две пары таблиц: A-C и B-C.

```
SELECT *
FROM (A,B) KEY JOIN C
```

Adaptive Server Anywhere генерирует условие соединения для соединения C с (A,B), создавая условие соединения для каждой из этих двух пар A-C и B-C. Он выполняет это согласно правилам определения ключевых соединений при наличии нескольких связей по внешнему ключу:

- Для каждой пары Adaptive Server Anywhere выполняет поиск внешнего ключа, имеющего функциональное имя, идентичное корреляционному имени таблицы первичного ключа. Если есть только один внешний ключ, соответствующий этому критерию, то этот внешний ключ и используется. Если существует более одного внешнего ключа с функциональным именем, идентичным корреляционному имени таблицы, тогда соединение считается неоднозначным, и выдается ошибка.
- Если для каждой пары нет внешнего ключа с именем, идентичным корреляционному имени таблицы нет, тогда Adaptive Server Anywhere выполняет поиск любой связи по внешнему ключу между таблицами. Если такой внешний ключ есть, то он используется. Если существует более одного таких внешних ключей, тогда соединение считается неоднозначным, и выдается ошибка.
- Если для каждой пары нет связи по внешнему ключу, то выдается ошибка.
- Если Adaptive Server Anywhere способен точно определить одно условие соединения для каждой пары, он объединяет эти условия соединения, используя AND.

Для получения дополнительной информации см. раздел "Ключевые соединения при наличии нескольких связей по внешнему ключу" на стр. 255.

Пример

Следующий запрос возвращает имена всех продавцов, которые осуществили продажу минимум по одному заказу в том или ином регионе.

```
SELECT DISTINCT employee.emp_lname,
  ky_dept_id.dept_name, sales_order.region
FROM (sales_order, department AS ky_dept_id)
   KEY JOIN employee
```

emp_lname	dept_name	region
Chin	Sales	Eastern
Chin	Sales	Western
Chin	Sales	Central
...	...	...

Этот запрос имеет дело с двумя парами таблиц: *sales_order* и *employee*; и *department AS ky_dept_id* и *employee*.

Для пары sales_order и employee не существует внешнего ключа с тем же функциональным именем, что и у одной из таблиц. Однако существует внешний ключ *(ky_so_employee_id),* объединяющий эти две таблицы. Это единственный внешний ключ, объединяющий эти две таблицы, поэтому он используется, и результатом является сгенерированное условие соединения `(employee.emp_id = sales_order.sales_rep)`.

Для пары *department AS ky_dept_id* и *employee* существует один внешний ключ, имеющий то же функциональное имя, что и таблица первичного ключа. Это - *ky_dept_id*, и он соответствует корреляционному имени, данному таблице *department* в запросе. Других внешних ключей с именем, идентичным корреляционному имени таблицы первичного ключа, нет, поэтому *ky_dept_id* используется для формирования условия соединения для пары таблиц. Сгенерированное условие соединения следующее: `(employee.dept_id = ky_dept_id.dept_id)`. Помните, что существует еще один внешний ключ, объединяющий эти две таблицы, но он не имеет значения, поскольку его имя отличается от имен любой из таблиц.

Заключительное условие соединения объединяет условие соединения, сгенерированное для каждой пары таблиц. Поэтому следующий запрос эквивалентен:

```
SELECT DISTINCT employee.emp_lname, depart-
ment.dept_name, sales_order.region
FROM sales_order, department
   JOIN employee
   ON employee.emp_id = sales_order.sales_rep
   AND employee.dept_id = department.dept_id
```

Ключевые соединения списков и выражений таблиц без запятых

Когда списки выражения таблиц объединяются через ключевое соединение с выражениями таблиц, не содержащих запятые, Adaptive Server Anywhere генерирует условие соединения для каждой таблицы в списке выражений таблиц.

Например, следующий оператор является ключевым соединением списка выражений таблиц с выражением таблиц, не содержащим запятые. Этот пример генерирует условие соединения для таблицы A с выражением таблицы `C NATURAL JOIN D` и для таблицы B с выражением таблицы `C NATURAL JOIN D`.

```
SELECT *
FROM (A,B) KEY JOIN (C NATURAL JOIN D)
```

`(A, B)` - список выражений таблиц, а `C NATURAL JOIN D` - выражение таблиц. Таким образом, Adaptive Server Anywhere должен генерировать два условия соединения: одно для пар A-C и A-D, а второе - для пар B-C и B-D. Это выполняется согласно правилам определения ключевых соединений при наличии нескольких связей по внешнему ключу:

- Для каждого набора пар таблиц Adaptive Server Anywhere выполняет поиск внешнего ключа, имеющего функциональное имя, идентичное корреляционному имени одной из таблиц первичного ключа. Если есть только один внешний ключ, соответствующий этому критерию, то этот внешний ключ и используется. Если существует более одного внешнего ключа с функциональным именем, идентичным корреляционному имени таблицы, тогда соединение считается неоднозначным, и выдается ошибка.

- Если для каждого набора пар таблиц внешнего ключа с именем, идентичным корреляционному имени таблицы нет, тогда Adaptive Server Anywhere выполняет поиск любой связи по внешнему ключу между таблицами. Если одна такая связь существует, то она используется. Если существует более одной таких связей, тогда соединение считается неоднозначным, и выдается ошибка.
- Если для каждого набора пар связей по внешнему ключу нет, то выдается ошибка.
- Если Adaptive Server Anywhere способен точно определить одно условие соединения для каждой пары, он объединяет эти условия соединения, используя AND.

Пример 1

Рассмотрим следующее соединение пяти таблиц:

```
((A,B) JOIN (C NATURAL JOIN D) ON A.x = D.y) KEY JOIN E
```

В этом случае Adaptive Server Anywhere генерирует условие соединения для ключевого соединения к E путем генерирования условия *либо* между `(A, B)` и E, *либо* между `C NATURAL JOIN D` и E. Это соответствует описанному в разделе "Ключевые соединения выражений таблиц без запятых " на стр. 258.

Если Adaptive Server Anywhere генерирует условие соединения между `(A, B)` и E, то ему необходимо создать два условия соединения, одно для A-E, а другое - для B-E. Потребуется найти допустимую связь по внешнему ключу в пределах каждой пары таблиц. Это соответствует описанному в разделе "Ключевые соединения списков выражений таблиц" на стр. 259.

Если Adaptive Server Anywhere создает условие соединения между `C NATURAL JOIN D` и E, то он создает только одно условие соединения, и поэтому должен найти только одну связь по внешнему ключу в парах C-E и D-E. Это соответствует описанному в разделе "Ключевые соединения выражений таблиц без запятых" на стр. 258.

Пример 2

Ниже представлен пример ключевого соединения выражения таблицы и списка выражений таблиц. В примере представлены имена и отделы служащих, являющихся тогровыми представителями, а также менеджерами.

```
SELECT DISTINCT employee.emp_lname,
ky_dept_id.dept_name
FROM (sales_order, department AS ky_dept_id)
  KEY JOIN (employee JOIN department AS d
    ON employee.emp_id = d.dept_head_id)
```

Adaptive Server Anywhere генерирует два условия соединения:

- Необходимо наличие только одной связи по внешнему ключу между парами таблиц *sales_order-employee* и *sales_order-d*. Она существует:
  `sales_order.sales_rep = employee.emp_id.`
- Необходимо наличие только одной связи по внешнему ключу между парами таблиц *ky_dept_id-employee* и *ky_dept_id-d.* Она существует:
  `ky_dept_id.dept_id = employee.dept_id.`

Этот пример эквивалентен следующему. В следующей версии необходимости создавать корреляционное имя `department AS ky_dept_id` нет, потому что оно требовалось только для прояснения того, который из двух внешних ключей должен быть применен для соединения *employee* и *department*.

```
SELECT DISTINCT employee.emp_lname, depart-
ment.dept_name
FROM (sales_order, department)
  JOIN (employee JOIN department AS d
      ON employee.emp_id = d.dept_head_id)
    ON sales_order.sales_rep = employee.emp_id
      AND department.dept_id = employee.dept_id
```

Ключевые соединения представлений и производных таблиц

При включении в ключевое соединение представления или производной таблицы Adaptive Server Anywhere следует той же основной процедуре, что и при работе с таблицами, но со следующими различиями:

- Для каждого ключевого соединения Adaptive Server Anywhere рассматривает пары таблиц в разделе FROM запроса и представления и генерирует одно условие соединения для набора всех пар, независимо от того, содержит раздел FROM в представлении запятые или ключевые слова соединения.
- Adaptive Server Anywhere соединяет таблицы по внешнему ключу, имеющему функциональное имя, идентичное корреляционному имени представления или производной таблицы.
- При включении в ключевое соединение представления или производной таблицы представление не может содержать UNION, ORDER BY, DISTINCT, GROUP BY или агрегатной функции. Если в представлении содержится какой-либо из вышеуказанных элементов, то выдается ошибка.

Производная таблица работает идентично представлению. Единственным различием является то, что вместо ссылки на предварительно определенное представление в оператор включено определение для таблицы.

Пример 1

Например, в следующем операторе View1 является представлением.

```
SELECT *
FROM View1 KEY JOIN B
```

Определением *View1* может быть любое из следующих, и результатом будет то же условие соединения с B. (Результирующий набор будет другим, но условия соединения остаются теми же).

```
SELECT *
FROM C CROSS JOIN D
```

или

```
SELECT *
FROM C,D
```

или

```
SELECT *
FROM C JOIN D ON (C.x = D.y)
```

В любом случае для того, чтобы сгенерировать условие соединения для ключевого соединения *View1* и B, Adaptive Server Anywhere рассматривает пары таблиц C-B и D-B, и генерирует единственное условие соединения. Он генерирует условие соединения, основанное на правилах для нескольких связей по внешнему ключу, описанных в разделе "Ключевые соединения выражений таблиц" на стр. 257, за исключением того, что осуществляется поиск внешнего ключа с именем, идентичным корреляционному имени представления (а не таблицы, на которую содержится ссылка в представлении).

Используя любое из определений представлений выше, обработку

`View1 KEY JOIN B` можно интерпретировать следующим образом:

Adaptive Server Anywhere генерирует единственное условие соединения, рассматривая пары таблиц C-B и D-B. Условие соединения генерируется согласно правилам определения ключевых соединений при наличии нескольких взаимосвязей по внешнему ключу:

- Сначала как в C-B, так и в D-B осуществляется поиск единственного внешнего ключа с функциональным именем, идентичным корреляционному имени представления. Если есть только один внешний ключ, соответствующий этому критерию, то этот внешний ключ и используется. Если существует более одного внешнего ключа с функциональным именем, идентичным корреляционному имени представления, тогда соединение считается неоднозначным, и выдается ошибка.
- Если внешнего ключа с именем, идентичным корреляционному имени представления, нет, тогда Adaptive Server Anywhere выполняет поиск любой связи по внешнему ключу между таблицами. Если такой внешний ключ есть, то он используется. Если существует более одного таких внешних ключей, тогда соединение считается неоднозначным, и выдается ошибка.
- Если связи по внешнему ключу нет, то выдается ошибка.

Предположим, что это сгенерированное условие соединения - `B.y = D.z`. Теперь можно развернуть первоначальное соединение.

```
SELECT *
FROM View1 KEY JOIN B
```

является эквивалентным

```
SELECT *
FROM View1 JOIN B ON B.y = View1.z
```

Для получения дополнительной информации см. раздел "Ключевые соединения при наличии нескольких взаимосвязей по внешнему ключу" на стр. 255.

Пример 2

Следующее представление содержит всю информацию о служащем для менеджеров каждого отдела.

```
CREATE VIEW V AS
SELECT department.dept_name, employee.*
FROM employee JOIN department
  ON employee.emp_id = department.dept_head_id
```

Следующий запрос соединяет представление с выражением таблиц.

```
SELECT *
FROM V KEY JOIN (sales_order, department ky_dept_id)
```

Это эквивалентно следующему:

```
SELECT *
FROM V JOIN (sales_order, department ky_dept_id)
ON (V.emp_id = sales_order.sales_rep
AND V.dept_id = ky_dept_id.dept_id)
```

Правила построения ключевых соединений

Следующие правила подводят итог вышеизложенной информации.

Правило 1: ключевое соединение двух таблиц

Это правило применяется к `A KEY JOIN B`, где A и B - базовая или временная таблица.

1. Поиск всех внешних ключей из A, содержащих ссылки на B.

   Если существует внешний ключ с функциональным именем, являющимся корреляционным именем таблицы B, то он отмечается как основной внешний ключ.

2. Поиск всех внешних ключей из B, содержащих ссылки на A.

   Если существует внешний ключ с функциональным именем, являющимся корреляционным именем таблицы A, то он отмечается как основной внешний ключ.

3. Если существует более одного основного ключа, то соединение неоднозначно. Выдается синтаксическая ошибка SQLE_AMBIGUOUS_JOIN (-147).

4. Если существует только один основной ключ, тогда этот внешний ключ выбирается для определения сгенерированного условия соединения для этого выражения KEY JOIN.

5. Если основного ключа нет, то используются другие внешние ключи между A и B:

   - Если между A и B существует более одного внешнего ключа, то соединение является неоднозначным. Выдается синтаксическая ошибка SQLE_AMBIGUOUS_JOIN (-147).
   - Если существует только один внешний ключ, тогда этот внешний ключ выбирается для определения сгенерированного условия соединения для этого выражения KEY JOIN.
   - Если внешнего ключа нет, то соединение недопустимо, и выдается ошибка.

Правило 2: ключевое соединение запятых выражений таблицы

Это правило применяется к `A KEY JOIN B`, где A и B - выражения таблиц, не содержащие запятые.

1. Для каждой пары таблиц: одной из выражения A, и другой - из выражения B, перечисляются все внешние ключи и отмечаются все основные внешние ключи между таблицами. Алгоритм определения основного внешнего ключа представлено в Правиле 1 выше.
2. Если существует более одного основного ключа, тогда соединение считается неоднозначным. Выдается синтаксическая ошибка SQLE_AMBIGUOUS_JOIN (-147).
3. Если существует только один основной ключ, тогда этот внешний ключ выбирается для определения сгенерированного условия соединения для этого выражения KEY JOIN.
4. Если основного ключа нет, то используются другие внешние ключи между парами таблиц:
   - Если существует более одного внешнего ключа, тогда соединение считается неоднозначным. Выдается синтаксическая ошибка SQLE_AMBIGUOUS_JOIN (-147).
   - Если существует только один внешний ключ, тогда этот внешний ключ выбирается для определения сгенерированного условия соединения для этого выражения KEY JOIN.
   - Если внешнего ключа нет, то соединение недопустимо, и выдается ошибка.

Правило 3: ключевое соединение списков выражений таблиц

Это правило применяется к `(A1, A2, …) KEY JOIN ( B1, B2, …)`, где A1, B1, и т. д. - выражения таблиц, не содержащие запятые.

1. Для каждой пары выражений таблиц Ai и Bj выполняется поиск уникального сгенерированного условия соединения для выражения таблиц `(Ai KEY JOIN Bj)` с применением Правила 1 или 2. Если какое-либо соединение KEY JOIN для пары выражений таблиц неоднозначно по Правилу 1 или 2, то выдается синтаксическая ошибка.
2. Сгенерированное условие соединения для этого выражения KEY JOIN является конъюнкцией условий соединения, найденных в шаге 1.

Правило 4: ключевое соединение списков и выражений таблиц, не содержащих запятые

Это правило применяется к `(A1, A2, …) KEY JOIN ( B1, B2, …)`, где A1, B1, и т. д. - выражения таблиц, которые могут содержать запятые.

1. Для каждой пары выражений таблиц Ai и Bj выполняется поиск уникального сгенерированного условия соединения для выражения таблиц `(Ai KEY JOIN Bj)` с применением Правила 1 или 2. Если какое-либо соединение KEY JOIN для пары выражений таблиц неоднозначно по Правилу 1, 2 или 3, то выдается синтаксическая ошибка.
2. Сгенерированное условие соединения для этого выражения KEY JOIN является конъюнкцией условий соединения, найденных в шаге 1.

ГЛАВА 9

Использование подзапросов

Об этой главе

При создании запроса разделы WHERE и HAVING используются для ограничения строк, возвращаемых запросом.

Иногда выбранные строки зависят от информации, хранящейся более чем в одной таблице. Подзапрос в разделе WHERE или HAVING позволяет выбирать строки из одной таблицы согласно описаниям, полученным из другой таблицы. Дополнительные способы выполнения этой задачи описаны в разделе "Соединения: извлечение данных из нескольких таблиц" на стр. 223.

Перед началом работы

Эта глава предполагает наличие определенных знаний о запросах и синтаксисе оператора Select. Для получения информации о запросах см. раздел "Запросы: выбор данных в таблице" на стр. 179.

Содержание

Введение в подзапросы

В реляционной базе данных хранится информация о разных типах объектов в разных таблицах. Например, информацию, относящуюся к товарам, необходимо хранить в одной таблице, а относящуюся к заказам на покупку - в другой. В таблице товаров содержится информация о разных товарах. В таблице заказов на покупку содержится информация о заказах клиентов.

Как правило, при использовании только одной таблицы ответы можно получить только на элементарные вопросы. Например, если компания повторно заказывает товары, которых на складе осталось менее 50 штук, тогда в результате этого запроса можно получить ответ на вопрос: "Какие товары почти отсутствуют на складе?":

```
SELECT id, name, description, quantity
FROM product
WHERE quantity < 50
```

Однако если "почти отсутствуют на складе" зависит от того, сколько наименований каждого типа заказывает типичный клиент, тогда число "50" придется заменить значением, взятым из таблицы *sales_order_items*.

Структура подзапроса

Подзапрос структурирован подобно обычному запросу и появляется в разделе SELECT, FROM, WHERE или HAVING основного запроса. Если продолжить предыдущий пример, то подзапрос можно использовать для выбора среднего числа товаров, заказываемых клиентом, после чего использовать эту цифру в основном запросе для нахождения товаров, запас которых заканчивается на складе. Следующий запрос находит наименования и описания товаров числом меньшим в два раза среднего числа товаров каждого типа, заказываемых клиентом.

```
SELECT name, description
FROM product
WHERE quantity <  2 * (
   SELECT avg(quantity)
   FROM sales_order_items
   )
```

В разделе WHERE использование подзапросов облегчает выбор строк из таблиц, перечисленных в разделе FROM, которые появляются в результатах запроса. В разделе HAVING подзапросы используются для выбора групп строк, которые появляются в результатах запроса, как определено разделом GROUP BY основного запроса.

Использование подзапросов в разделе WHERE

Подзапросы в разделе WHERE функционируют как часть процесса выбора строк. Подзапрос в разделе WHERE используется тогда, когда критерии выбора строк зависят от результатов другой таблицы.

Пример

Найдите товары, запасы которых на складе меньше, чем удвоенное среднее заказываемое количество.

```
SELECT name, description
FROM product
WHERE quantity <  2 *(
   SELECT avg(quantity)
   FROM sales_order_items)
```

Это двухступенчатый запрос: сначала нужно найти среднее количество товаров, затребованных в заказах, затем - наименования товаров, количество которых на складе меньше найденного среднего количества, умноженного на два.

Двухступенчатый запрос

В столбце quantity таблицы *sales_order_items* указано количество наименований, затребованных по типу товара, клиенту и заказу. Подзапрос следующий:

```
SELECT avg(quantity)
FROM sales_order_items
```

Он возвращает среднее количество наименований товаров в таблице *sales_order_items*, составляющее 25.851413.

Следующий запрос возвращает наименования и описания товаров, запасы которых на складе меньше полученного ранее значения, умноженного на два.

```
SELECT name, description
FROM product
WHERE quantity < 2*25.851413
```

Использование подзапроса объединяет два шага в одну операцию.

Функция подзапроса в разделе WHERE

Подзапрос в разделе WHERE является частью условия поиска. Простые условия поиска, которые можно использовать в разделе WHERE, описаны в главе "Запросы: выбор данных в таблице" на стр. 179.

Подзапросы в разделе HAVING

Несмотря на то, что обычно подзапросы используются в качестве условий поиска в разделе WHERE, иногда их можно использовать в разделе HAVING запроса. Когда подзапрос появляется в разделе HAVING, то, подобно любому выражению в разделе HAVING, он используется как часть выбора группы строк.

Вот запрос, являющийся запросом с подзапросом в разделе HAVING: "Среднее количество каких товаров на складе больше, чем среднее удвоенное количество каждого товара, заказанного клиентами?".

Пример

```
SELECT name, avg(quantity)
FROM product
GROUP BY name
HAVING avg(quantity) > 2* (
   SELECT avg(quantity)
   FROM sales_order_items
   )
```

name	avg(product.quantity)
Tee Shirt	52.333333
Baseball Cap	62
Shorts	80

Запрос выполняется следующим образом:

- Подзапрос вычисляет среднее количество наименований в таблице *sales_order_items*.
- Затем основной запрос просматривает таблицу product, вычисляя среднее количество товара и выполняя группирование по наименованию.
- После этого раздел HAVING проверяет, превышает ли среднее количество товара удвоенное количество, найденное подзапросом. Если превышает, то основной запрос возвращает эту группу строк; в противном случае эта группа строк не возвращается.
- Раздел SELECT получает одну итоговую строку для каждой группы с указанием наименования каждого товара и его среднего количества на складе.

В разделе HAVING можно также использовать внешние ссылки, как это показано в следующем примере, являющимся разновидностью предыдущего.

Пример

"Найдите коды товаров и коды строк товаров, среднее заказанное количество которых превышает половину количества этих товаров, имеющегося на складе".

```
SELECT prod_id, line_id
FROM sales_order_items
GROUP BY prod_id, line_id
HAVING 2* avg(quantity) > (
   SELECT quantity
   FROM product
   WHERE product.id = sales_order_items.prod_id)
```

prod_id	line_id
401	2
401	1
401	4
501	3
...	...

В этом примере подзапрос должен предоставить количество товара на складе, соответствующее группе строк, проверяемой разделом HAVING. Подзапрос выбирает записи для этого конкретного товара, используя внешнюю ссылку *sales_order_items.prod_id.*

Подзапрос со сравнением возвращает одиночное значение

Этот запрос использует сравнение ">", предполагая, что подзапрос должен возвратить только одно значение. В этом случае именно это и происходит. Поскольку поле id таблицы product является первичным ключом, то в таблице product, существует только одна запись, соответствующая любому определенному коду товара.

Проверка подзапросов

Простые условия поиска, которые можно использовать в разделе HAVING, описаны в главе "Запросы: выбор данных в таблице" на стр. 179. Поскольку подзапрос - всего лишь выражение, появляющееся в разделах WHERE и HAVING, то условия поиска в подзапросах могут выглядеть похожими.

Они включают в себя следующее:

- **Сравнительная проверка подзапросов.** Сравнение значения выражения с одиночным значением, полученным подзапросом для каждой записи в таблице (таблицах) основного запроса.
- **Кванторная сравнительная проверка.** Сравнение значения выражения с каждым значением из набора значений, полученного подзапросом.
- **Проверка на членство в наборе подзапроса.** Проверка соответствия значения выражения одному значению из набора значений, полученного подзапросом.
- **Проверка существования.** Проверка факта получения подзапросом каких-либо строк.

Сравнительная проверка подзапросов

Сравнительная проверка подзапросов (=, <>, < . <=,>,> =) является измененной версией простой сравнительной проверки. Единственное различие между ними - то, что в первой выражение, следующее за операцией, является подзапросом. Эта проверка применяется для сравнения значения из строки основного запроса с одиночным значением, полученным подзапросом.

Пример

Этот запрос содержит пример сравнительной проверки подзапросов:

```
SELECT name, description, quantity
FROM product
WHERE quantity <  2 *(
   SELECT avg(quantity)
   FROM sales_order_items)
```

name	description	quantity
Tee Shirt	Tank Top	28
Baseball Cap	Wool cap	12
Visor	Cloth Visor	36
Visor	Plastic Visor	28
...	...	...

Следующий подзапрос получает одиночное значение - среднее количество наименований каждого типа согласно заказу клиента - из таблицы *sales_order_items*.

```
SELECT avg(quantity)
FROM sales_order_items
```

Затем основной запрос сравнивает количество каждого наименования товара на складе с этим значением.

Подзапрос в сравнительной проверке возвращает одно значение
Подзапрос в сравнительной проверке должен возвратить только одно значение. Рассмотрим запрос, подзапрос которого выделяет два столбца из таблицы *sales_order_items*:

```
SELECT name, description, quantity
FROM product
WHERE quantity <  2 * (
   SELECT avg(quantity), max (quantity)
   FROM sales_order_items)
```

Он возвращает ошибку: "Subquery allowed only one select list item" (Подзапросом допускается только один элемент списка выбора).

Кванторная сравнительная проверка с ANY и ALL

Кванторная сравнительная проверка имеет две категории: проверка ALL и проверка ANY:

Проверка ANY

Проверка ANY , используемая вместе с одной из операций сравнения SQL (=, <>, < , <=,>,> =), сравнивает одиночное значение со столбцом значений данных, полученным подзапросом. Для выполнения проверки в SQL используется указанная операция сравнения для сравнения проверяемого значения с каждым значением данных в столбце. Если какое-либо из сравнений приводит к результату TRUE, то проверка ANY возвращает TRUE.

Подзапрос, используемый с ANY, должен возвращать одиночный столбец.

Пример

Найдите коды заказов и коды клиентов для заказов, размещенных после отгрузки первого товара согласно заказу № 2005.

```
SELECT id, cust_id
FROM sales_order
WHERE order_date > ANY (
   SELECT ship_date
   FROM sales_order_items
   WHERE id=2005)
```

id	cust_id
2006	105
2007	106
2008	107
2009	108
...	...

При выполнении этого запроса основной запрос сверяет дату каждого заказа с датами отгрузки *каждого* товара в заказе № 2005. Если дата заказа более поздняя, нежели дата одной отгрузки заказа № 2005, тогда этот код и код клиента из таблицы *sales_order* входят в результирующий набор. Таким образом, проверка ANY является аналогом операции OR, поскольку вышеупомянутый запрос можно прочитать так: "Этот заказ был размещен после отгрузки первого товара согласно заказу № 2005 или после отгрузки второго товара согласно заказу № 2005 или..."

Пояснения к операции ANY

Операция ANY может показаться немного запутанной. Возникает непреодолимое желание прочитать запрос так: "Возвратить заказы, размещенные после отгрузки любых товаров по заказу № 2005". Но тогда получается, что запрос возвратит коды заказов и клиентов для заказов, размещенных после отгрузки *всех* товаров по заказу № 2005, - на самом деле это не то, что выполняется в запросе.

Попробуйте прочитать запрос следующим образом: "Возвратить коды заказов и клиентов для заказов, размещенных после того, как был отгружен *хотя бы один* товар согласно заказу № 2005". При использовании ключевого слова SOME можно предложить более понятный способ формулировки запроса. Следующий запрос эквивалентен предыдущему запросу.

```
SELECT id, cust_id
FROM sales_order
WHERE order_date > SOME (
   SELECT ship_date
   FROM sales_order_items
   WHERE id=2005)
```

Ключевое слово SOME эквивалентно ключевому слову ANY.

Примечания к операции ANY

Проверка ANY обладает двумя важными дополнительными характеристиками:

- **Пустой результирующий набор подзапроса.** Если подзапрос производит пустой результирующий набор, то проверка ANY возвращает FALSE. Это логично, поскольку, если результатов нет, то утверждение, что как минимум один результат удовлетворяет сравнительной проверке, является ложным.
- **Значения NULL в результирующем наборе подзапроса.** Предположим, что в результирующем наборе подзапроса имеется хотя бы одно значение NULL. Если сравнительная проверка ложна для всех значений данных результирующего набора, отличных от NULL, тогда поиск ANY возвращает FALSE. Это происходит потому, что в этой ситуации нельзя с уверенностью говорить о наличии значения подзапроса, для которого проводится сравнительная проверка. Значение может присутствовать, а может и нет, в зависимости от "правильных" значений для данных NULL в результирующем наборе.

Проверка ALL

Как и проверка ANY, проверка ALL применяется вместе с одной из шести операций сравнения SQL (=, <>, < , <=,>,> =) для сравнения одиночного значения со значениями данных, произведенных подзапросом. Для выполнения проверки SQL использует указанную операцию сравнения для сравнения проверяемого значения с каждым значением данных в результирующем наборе. Если *все* сравнения приводят к результатам TRUE, тогда проверка ALL возвращает TRUE.

Пример

Ниже приведен запрос, обработанный проверкой ALL: "Найдите коды клиентов и коды заказов, размещенных после отгрузки всех товаров по заказу № 2001".

```
SELECT id, cust_id
FROM sales_order
WHERE order_date > ALL (
   SELECT ship_date
   FROM sales_order_items
   WHERE id=2001)
```

код	cust_id
2002	102
2003	103
2004	104
2005	101
...	...

При выполнении этого запроса основной запрос сверяет дату каждого заказа с датами отгрузки каждого товара в заказе № 2001. Если дата заказа более поздняя, нежели дата *каждой* отгрузки заказа № 2001, тогда этот код и код клиента из таблицы *sales_order* входят в результирующий набор. Таким образом, проверка ANY является аналогом операции AND, потому что вышеупомянутый запрос можно прочитать так: "Этот заказ был размещен после отгрузки первого товара согласно заказу № 2001 или после отгрузки второго товара заказа № 2001 или..."

Примечания к операции ALL

Проверка ALL обладает тремя важными дополнительными характеристиками:

- **Пустой результирующий набор подзапроса.** Если подзапрос производит пустой результирующий набор, то проверка ALL возвращает TRUE. Это логично, поскольку, если результатов нет, то утверждение, что сравнительная проверка сохраняет каждое значение результирующего набора, является истинным.
- **Значения NULL в результирующем наборе подзапроса.** Если сравнительная проверка ложна для каких-либо значений результирующего набора, тогда ANY возвращает FALSE. ANY возвращает TRUE, если все значения истинны. В противном случае возвращается UNKNOWN. Это, например, может произойти, если в результирующем наборе подзапроса имеется значение NULL, но условие поиска - TRUE для всех значений, отличных от NULL.
- **Инвертирование проверки ALL.** Следующие выражения не эквивалентны.

```
NOT a = ALL (subquery)
a <> ALL (subquery)
```

Для получения дополнительной информации об этой проверке см. раздел "Кванторная сравнительная проверка" на стр. 287.

Проверка на членство в наборе с условиями IN

Проверку на членство в наборе подзапроса можно использовать для сравнения значения из основного запроса с более чем одним значением в подзапросе.

При проверке на членство в наборе подзапроса одиночное значение данных для каждой строки в основном запросе сравнивается с одиночным столбцом значений данных, полученных подзапросом. Если значение данных из основного запроса соответствует *одному* из значений данных в столбце, то подзапрос возвращает TRUE.

Пример

Выберите имена сотрудников, возглавляющих отделы отгрузки и финансов:

```
SELECT emp_fname, emp_lname
FROM employee
WHERE emp_id IN (
   SELECT dept_head_id
   FROM department
WHERE (dept_name='Finance' or dept_name = 'Ship-
ping'))
```

emp_fname	emp_lname
Jose	Martinez
Mary Anne	Shea

Подзапрос в этом примере

```
SELECT dept_head_id
FROM department
WHERE (dept_name='Finance' OR dept_name = 'Shipping')
```

выделяет из таблицы *department* коды, соответствующие начальникам отделов отгрузки и финансового. Затем основной запрос возвращает имена служащих, коды которых соответствуют одному из двух, найденных подзапросом.

Проверка на членство в наборе эквивалентна проверке ANY

Проверка на членство в наборе подзапроса эквивалентна проверке ANY. Следующий запрос эквивалентен запросу из вышеупомянутого примера.

```
SELECT emp_fname, emp_lname
FROM employee
WHERE emp_id =ANY (
   SELECT dept_head_id
   FROM department
WHERE (dept_name='Finance' or dept_name = 'Ship-
ping'))
```

Инвертирование проверки на членство в наборе

Проверку на членство в наборе подзапроса можно также использовать для получения строк, значения столбцов которых не равны ни одному из значений, полученных подзапросом.

Для инвертирования проверки на членство в наборе введите слово NOT перед ключевым словом IN.

Пример

Подзапрос в этом запросе возвращает имена и фамилии служащих, не являющихся начальниками ни финансового отдела, ни отдела отгрузок.

```
SELECT emp_fname, emp_lname
FROM employee
WHERE emp_id NOT IN (
   SELECT dept_head_id
   FROM department
WHERE (dept_name='Finance' OR dept_name = 'Shipping'))
```

Проверка существования

Подзапросы, используемые при сравнительной проверке подзапросов и проверке на членство в наборе, возвращают значения данных из таблицы подзапроса. Иногда, впрочем, вопрос "Возвращает ли подзапрос *какие-либо* результаты?" может оказаться важнее вопроса "*Какие* результаты возвращаются?". С помощью проверки существования (EXISTS) выясняется, находит ли подзапрос какие-либо строки результатов запроса. Если подзапрос находит одну или более строк результатов, тогда проверка EXISTS возвращает TRUE. В противном случае возвращается FALSE.

Пример

Ниже приведен пример запроса, выраженного с использованием подзапроса: "Какие клиенты разместили заказы после 13 июля 2001 г.?"

```
SELECT fname, lname
FROM customer
WHERE EXISTS (
   SELECT *
   FROM sales_order
WHERE (order_date > '2001-07-13') AND
   (customer.id = sales_order.cust_id))
```

fname	lname
Almen	de Joie
Grover	Pendelton
Ling Ling	Andrews
Bubba	Murphy

Пояснение проверки существования

Здесь для каждой строки в таблице *customer* подзапрос проверяет соответствие кода этого клиента коду клиента, разместившего заказ после 13 июля 2001 г.

Если коды соответствуют, то запрос находит имя и фамилию этого клиента в основной таблице.

Проверка EXISTS не использует результаты подзапроса; она проверяет только факт нахождения подзапросом каких-либо строк. Поэтому проверка существования, примененная к двум следующим подзапросам, возвращает одинаковые результаты. Это подзапросы, и они не могут быть обработаны сами по себе, потому что относятся к таблице клиентов, являющейся частью основного запроса, но не частью подзапроса.

Для получения дополнительной информации см. раздел "Коррелированные подзапросы" на стр. 285.

```
SELECT *
FROM sales_order
WHERE (order_date > '2001-07-13') AND (customer.id =
sales_order.cust_id)
SELECT ship_date
FROM sales_order
WHERE (order_date > '2001-07-13') AND (customer.id =
sales_order.cust_id)
```

Какой из столбцов таблицы *sales_order* появляется в операторе SELECT, не имеет значения, хотя традиционно добавляется уточняющая запись "SELECT *".

Инвертирование проверки существования

Используя форму NOT EXISTS, можно полностью изменить логику проверки EXISTS. В этом случае проверка возвращает TRUE, если подзапрос не находит никаких строк, и FALSE, если находит.

Коррелированые подзапросы

Как можно было заметить ранее, подзапрос содержит ссылку на столбец *id* в таблице *customers*. Ссылка на столбцы или выражения в основной таблице (таблицах) называется **внешней ссылкой**, а подзапрос - **коррелированым**. В принципе, в SQL вышеуказанный запрос обрабатывается путем просмотра таблицы customer и выполнения подзапроса для каждого клиента. Если дата заказа в таблице *sales_order* является более поздней, чем 13 июля 2001 г., и код клиента в таблицах customer и *sales_order* совпадает, тогда из таблицы клиентов извлекаются имя и фамилия. Поскольку подзапрос ссылается на основной запрос, то, в отличие от подзапросов из других разделов, подзапрос в этом разделе возвратит ошибку, если будет произведена попытка запустить его сам по себе.

Внешние ссылки

В пределах тела подзапроса часто бывает необходимо поместить ссылку на значение столбца в активной строке основного запроса. Рассмотрим следующий запрос:

```
SELECT name, description
FROM product
WHERE quantity < 2 * (
   SELECT avg(quantity)
   FROM sales_order_items
WHERE product.id = sales_order_items.prod_id)
```

Этот запрос получает наименования и описания товаров, количество которых на складе меньше удвоенного среднего заказанного количества этих товаров, в частности, проверенных разделом WHERE в основном запросе. Подзапрос выполняет эту процедуру путем сканирования таблицы *sales_order_items*. Но столбец *product.id* в разделе WHERE подзапроса содержит ссылку на столбец в таблице, названной в разделе FROM основного запроса, а не подзапроса. По мере просмотра SQL каждой строки таблицы *product* при выполнении раздела WHERE подзапроса применяется значение id текущей строки.

Описание внешней ссылки

Столбец *product.id* в этом подзапросе является примером внешней ссылки. Подзапрос, использующий внешнюю ссылку, является коррелированным. Внешняя ссылка - это имя столбца, не содержащего ссылок на какие-либо столбцы тех или иных таблиц в разделе FROM подзапроса. Вместо этого имя столбца содержит ссылку на столбец таблицы, указанной в разделе FROM основного запроса.

Как показано в примере выше, значение столбца во внешней ссылке исходит из строки, проверяемой в данный момент основным запросом.

Подзапросы и соединения

Оптимизатор подзапросов автоматически перезаписывает в виде соединений многие из запросов, которые используют подзапросы.

Пример

Рассмотрим следующий запрос: "Когда г-жа Кларк и г-жа Суреш разместили свои заказы, и через каких коммерческих представителей?" Ответ можно получить при помощи следующего запроса:

```
SELECT order_date, sales_rep
FROM sales_order
WHERE cust_id IN (
   SELECT id
   FROM customer
WHERE lname = 'Clarke' OR fname = 'Suresh')
```

Order_date	sales_rep
2001-01-05	1596
2000-01-27	667
2000-11-11	467
2001-02-04	195
...	...

Подзапрос предоставляет список кодов клиентов, соответствующий двум клиентам, имена которых приведены в разделе WHERE, и основной запрос находит даты заказов и имена торговых представителей, имеющих отношение к заказам этих двух дам.

Замена подзапроса соединением

На этот же вопрос можно ответить, используя соединение. Ниже приведена альтернативная форма запроса, в которой использовано соединение двух таблиц:

```
SELECT order_date, sales_rep
FROM sales_order, customer
WHERE cust_id=customer.id AND
  (lname = 'Clarke' OR fname = 'Suresh')
```

Эта форма запроса соединяет таблицу sales_order с таблицей customer для нахождения заказов каждого клиента, после чего возвращает только записи, относящиеся к Кларк и Суреш.

Некоторые соединения нельзя записать в виде подзапросов

Оба эти запроса находят правильные даты заказов и нужных торговых представителей, и эти запросы верны в одинаковой степени. Форма подзапроса может показаться более удобной, потому что в запросе не требуется информация о кодах клиентов, а также потому, что соединение таблиц *sales_order* и customers для получения ответа на этот вопрос может показаться странным.

Однако если запрос изменяется с включением какой-либо информации из таблицы customer, то форма подзапроса перестает функционировать. Например, запрос: "Когда госпожа Кларк и госпожа Суреш разместили свои заказы, через каких коммерческих представителей, и каковы полные имена последних?", то тогда таблицу customer необходимо включить в основной раздел WHERE:

```
SELECT fname, lname, order_date, sales_rep FROM
sales_order, customer
WHERE cust_id=customer.id AND (lname = 'Clarke' OR
fname = 'Suresh')
```

fname	lname	order_date	sales_rep
Belinda	Clarke	1/5/01	1596
Belinda	Clarke	1/27/00	667
Belinda	Clarke	11/11/00	467
Belinda	Clarke	2/4/01	195
...	...	...	...

Некоторые подзапросы нельзя записать в виде соединений

Точно так же бывают случаи, когда подзапрос функционирует, а соединение - нет. Например:

```
SELECT name, description, quantity
FROM product
WHERE quantity <  2 * (
   SELECT avg(quantity)
   FROM sales_order_items)
```

name	description	quantity
Tee Shirt	Tank Top	28
Baseball Cap	Wool cap	12
Visor	Cloth Visor	36
...	...	...

В этом случае внутренний запрос является итоговым, а внешний запрос - нет, поэтому объединить эти два запроса простым соединением нельзя.

Для получения дополнительной информации о соединениях см. раздел "Соединения: извлечение данных из нескольких таблиц" на стр. 223.

Вложенные подзапросы

Как было указано ранее, подзапросы всегда появляются в разделе HAVING или разделе WHERE запроса. Подзапрос может сам содержать раздел WHERE и/или раздел HAVING и, следовательно, один подзапрос может появиться в другом. Подзапросы в пределах других подзапросов называются вложенными подзапросами.

Примеры

Перечислите коды заказов и коды строк заказов, отгруженных в тот же день, когда в отделе оплаты был заказан любой товар.

```
SELECT id, line_id
FROM sales_order_items
WHERE ship_date = ANY (
   SELECT order_date
   FROM sales_order
   WHERE fin_code_id IN (
      SELECT code
      FROM fin_code
WHERE (description = 'Fees')))
```

id	line_id
2001	1
2001	2
2001	3
2002	1
...	...

Пояснение вложенных подзапросов

- В этом примере самый внутренний подзапрос получает столбец финансовых кодов с описанием "Fees":

```
SELECT code
FROM fin_code
WHERE (description = 'Fees')
```

- Следующий подзапрос находит даты заказов товаров, коды которых совпадают с одним из кодов, выбранных в самом внутреннем подзапросе:

```
SELECT order_date
FROM sales_order
WHERE fin_code_id IN (subquery)
```

- Наконец, самый внешний запрос находит коды заказов и коды строк заказов, отгруженных в одну из дат, найденных в подзапросе.

```
SELECT id, line_id
FROM sales_order_items
WHERE ship_date = ANY (subquery)
```

Вложенные подзапросы могут иметь более трех уровней. Хотя максимальное число уровней не ограничено, запросы с тремя или более уровнями требуют значительно больше времени на выполнение, нежели небольшие запросы.

Принципы работы подзапросов

Понимание того, какие запросы являются допустимыми, а какие - нет, может оказаться сложным в случае наличия в запросе подзапроса. Также в большой степени потребуется ознакомление с принципами работы многоуровневого запроса, что поможет понять, как Adaptive Server Anywhere обрабатывает подзапросы. Для получения общей информации об обработке запросов см. раздел "Сведение, группирование и сортировка результатов запроса" на стр. 203.

Коррелированные подзапросы

В простом запросе сервер базы данных оценивает и обрабатывает раздел WHERE один раз для каждой строки. Впрочем, иногда подзапрос возвращает только один результат, вынуждая сервер базы данных вычислять его более одного раза для всего результирующего набора.

Некоррелированные подзапросы

Рассмотрим следующий запрос:

```
SELECT name, description
FROM product
WHERE quantity <  2 * (
   SELECT avg(quantity)
   FROM sales_order_items)
```

В этом примере подзапрос вычисляет только одно значение: среднее количество из таблицы *sales_order_items*. При вычислении запроса сервер базы данных один раз высчитывает это значение и сравнивает с ним каждое значение в поле *quantity* (количество) таблицы *product*, чтобы решить, выбрать ли соответствующую строку.

Коррелированные подзапросы

Если подзапрос содержит внешнюю ссылку, то использовать такое сокращение нельзя. Например, подзапрос в запросе

```
SELECT name, description
FROM product
WHERE quantity < 2 * (
   SELECT avg(quantity)
   FROM sales_order_items
WHERE product.id=sales_order_items.prod_id)
```

возвращает значение, зависящее от активной строки в таблице *product*. Такой подзапрос называется коррелированным. В этих случаях подзапрос мог возвратить различные значения для каждой строки внешнего запроса, заставляя сервер базы данных выполнить более одного вычисления.

Преобразование подзапросов в разделе WHERE в соединения

Как правило, запрос, использующий соединения, выполняется быстрее многоуровневого запроса. По этой причине при первой возможности оптимизатор запросов Adaptive Server Anywhere преобразует многоуровневый запрос в запрос, использующий соединения. Преобразование выполняется без каких-либо действий со стороны пользователя. В этом разделе описываются подзапросы, которые можно преобразовать в соединения с тем, чтобы пояснить принципы функционирования подзапросов в базе данных.

Пример

Вопрос: "Когда г-жа Кларк и г-жа Суреш разместили свои заказы, и с помощью каких торговых представителей?" можно записать в виде двухуровневого запроса:

```
SELECT order_date, sales_rep
FROM sales_order
WHERE cust_id IN (
   SELECT id
   FROM customer
WHERE lname = 'Clarke' OR fname = 'Suresh')
```

Альтернативный и не менее правильный способ заключается в записи запроса с соединениями:

```
SELECT fname, lname, order_date, sales_rep
FROM sales_order, customer
WHERE cust_id=customer.id AND
  (lname = 'Clarke' OR fname = 'Suresh')
```

Для различных типов операций применяются разные критерии, которые необходимо соблюдать для того, чтобы переписать многоуровневый запрос с соединениями. Помните, что при появлении подзапроса в разделе WHERE запроса, он имеет следующую форму:

SELECT *список-выбора*
FROM *таблица*
WHERE
[**NOT**] *выражение операция-сравнения* **(** *подзапрос* **)**
| [**NOT**] *выражение операция-сравнения*{ **ANY** | **SOME** } **(** *подзапрос* **)**
| [**NOT**] *выражение операция-сравнения* **ALL (** *подзапрос* **)**
| [**NOT**] *выражение* **IN (** *подзапрос* **)**
| [**NOT**] **EXISTS (** *подзапрос* **)**
GROUP BY *группирование-по-выражению*
HAVING *условие-поиска*

Преобразование подзапроса зависит от нескольких факторов, таких как тип операции, а также структур запроса и подзапроса.

Операции сравнения

Если подзапрос, следующий за операцией сравнения (=, <>, < , <=,>,> =), необходимо преобразовать в соединение, то он должен удовлетворять определенным условиям. Подзапросы, следующие за операциями сравнения, допустимы только в случае возвращения ими единственного значения для каждой строки основного запроса. В дополнение к этому критерию подзапрос преобразуется в соединение только в случае, если он

- не содержит раздел GROUP BY;
- не содержит ключевое слово DISTINCT;
- не является запросом UNION;
- не является агрегатным запросом.

Пример

Предположим, что запрос "Когда были заказаны товары г-жи Суреш, и с помощью каких торговых представителей?" перефразирован в следующий подзапрос:

```
SELECT order_date, sales_rep
FROM sales_order
WHERE cust_id = (
   SELECT id
   FROM customer
   WHERE fname = 'Suresh')
```

Этот запрос удовлетворяет вышеуказанным критериям, и поэтому он будет преобразован в запрос с использованием соединения:

```
SELECT order_date, sales_rep
FROM sales_order, customer
WHERE cust_id=customer.id AND (lname = 'Clarke' OR
fname
= 'Suresh')
```

Однако, запрос "Найдите товары, запас которых на складе меньше удвоенного среднего заказываемого количества" не может быть преобразован в соединение, поскольку в подзапросе содержится агрегатная функция avg:

```
SELECT name, description
FROM product
WHERE quantity <  2 * (
   SELECT avg(quantity)
   FROM sales_order_items)
```

Кванторная сравнительная проверка

Подзапрос, следующий за одним из ключевых слов ALL, ANY и SOME, преобразуется в соединение только при условии соответствия определенным критериям.

- Основной запрос не содержит раздел GROUP BY, этот запрос - не агрегатный, и подзапрос возвращает только одно значение.

- Подзапрос не содержит раздел GROUP BY.
- Подзапрос не содержит ключевое слово DISTINCT.
- Подзапрос не является запросом типа UNION.
- Подзапрос не является агрегатным запросом.
- Конъюнкт ‘выражение операция-сравнения ANY/SOME (подзапрос)’ должен быть инвертирован.
- Конъюнкт ‘выражение операция-сравнения ALL (подзапрос)’ не должен быть инвертирован.

Первые из четырех этих условий являются относительно очевидными.

Пример

Запрос "Когда г-жа Кларк и г-жа Суреш разместили свои заказы, и с помощью каких торговых представителей?" может быть преобразован в форму подзапроса:

```
SELECT order_date, sales_rep
FROM sales_order
WHERE cust_id = ANY (
   SELECT id
   FROM customer
WHERE lname = 'Clarke' OR fname = 'Suresh')
```

С другой стороны, его можно сформулировать в форме соединения:

```
SELECT fname, lname, order_date, sales_rep
FROM sales_order, customer
WHERE cust_id=customer.id AND (lname = 'Clarke' OR
fname
= 'Suresh')
```

Однако запрос "Когда г-жа Кларк, г-жа Суреш и любой служащий, также являющийся клиентом, разместили свои заказы?" будет сформулирован как запрос объединения (union), и поэтому его нельзя преобразовать в соединение:

```
SELECT order_date, sales_rep
FROM sales_order
WHERE cust_id = ANY (
   SELECT id
   FROM customer
WHERE lname = 'Clarke' OR fname = 'Suresh'
UNION
SELECT id
FROM employee)
```

Точно так же запрос "Найти коды клиентов и заказов, которые не были размещены после отгрузки всех товаров по заказу № 2001" естественным образом выражается подзапросом:

```
SELECT id, cust_id
FROM sales_order
WHERE NOT order_date > ALL (
   SELECT ship_date
   FROM sales_order_items
   WHERE id=2001)
```

Он будет преобразован в соединение:

```
SELECT sales_order.id, cust_id
FROM sales_order, sales_order_items
WHERE (sales_order_items.id=2001) and (order_date <=
ship_date)
```

Однако запрос "Найти коды клиентов и заказов, отгрузка по которым не была осуществлена по истечении первых дат поставки всех товаров" будет сформулирован как агрегатный запрос:

```
SELECT id, cust_id
FROM sales_order
WHERE NOT order_date > ALL (
   SELECT first (ship_date)
   FROM sales_order_items )
```

Поэтому он не будет преобразован в соединение.

Инвертирование подзапросов с ANY и ALL

Пятый критерий является немного более озадачивающим: запросы в форме

SELECT *список-выбора*
Операции **FROM** *таблица*
WHERE NOT *выражение операция-сравнения* **ALL(** *подзапрос* **)**

преобразованы в соединения, как и запросы в форме

SELECT *список-выбора*
FROM *таблица*
WHERE *выражение операция-сравнения* **ANY(** *подзапрос* **)**

но запросы

SELECT *список-выбора*
FROM *таблица*
WHERE *выражение операция-сравнения* **ALL(** *подзапрос* **)**

и

SELECT *список-выбора*
FROM *таблица*
WHERE NOT *выражение операция-сравнения* **ANY(** *подзапрос* **)**

не преобразованы в соединения.

Логическая эквивалентность выражений ANY и ALL

Это происходит из-за того, что фактически первые два запроса эквивалентны, как и два последние. Помните, что операция ANY аналогична операции OR, но с переменным числом аргументов, и что операция ALL аналогична операции AND. Точно так же, как выражение

```
NOT ((X > A) AND (X > B))
```

эквивалентно выражению

```
(X <= A) OR (X <= B)
```

выражение

```
NOT order_date > ALL (
   SELECT first (ship_date)
   FROM sales_order_items )
```

эквивалентно выражению

```
order_date <= ANY (
SELECT first (ship_date)
FROM sales_order_items )
```

Инвертирование выражений ANY и ALL

В принципе, выражение

NOT *имя-столбца операция* **ANY(** *подзапрос* **)**

является эквивалентным выражению

имя-столбца обратная-операция **ALL(** *подзапрос* **)**

и выражению

NOT *имя-столбца операция* **ALL(** *подзапрос* **)**

эквивалентно выражению

имя-столбца обратная-операция **ANY(** *подзапрос* **)**

где обратная-операция получена инвертированием операции, как показано в следующей таблице:

Таблица операций и их инверсии

В таблице ниже приведена инверсия каждой операции.

Операция	Обратная операция
=	<>
<	=>
>	=<
=<	>
=>	<
<>	=

Проверка членства в наборе

Запрос, содержащий подзапрос, следующий за ключевым слово IN, преобразуется в соединение, только если:

- Основной запрос не содержит раздел GROUP BY, не является агрегатным запросом, и подзапрос возвращает только одно значение.
- Подзапрос не содержит раздел GROUP BY.
- Подзапрос не содержит ключевое слово DISTINCT.
- Подзапрос не является запросом UNION.
- Подзапрос не является агрегатным запросом.
- Конъюнкт ‘выражение IN (подзапрос)’ не должен инвертироваться.

Пример

Так, запрос "Найти имена служащих, являющихся начальниками отделов", выраженный запросом:

```
SELECT emp_fname, emp_lname
FROM employee
WHERE emp_id IN (
   SELECT dept_head_id
   FROM department
WHERE (dept_name='Finance' or dept_name = 'Shipping'))
```

будет преобразован в соединенный запрос, поскольку он удовлетворяет указанным выше условиям. Однако запрос "Найти имена служащих, являющихся либо начальниками отделов, либо клиентами" не будет преобразован в соединение, если он выражен запросом UNION.

Запрос UNION, следующий за операцией IN, не может быть преобразован

```
SELECT emp_fname, emp_lname
FROM employee
WHERE emp_id IN (
SELECT dept_head_id
FROM department
WHERE (dept_name='Finance' or dept_name = 'Shipping')
UNION
SELECT cust_id
FROM sales_order)
```

Точно так же запрос "Найти имена служащих, не являющихся начальниками отделов" сформулирован как инвертированный подзапрос:

```
SELECT emp_fname, emp_lname
FROM employee
  WHERE NOT emp_id IN (
   SELECT dept_head_id
   FROM department
WHERE (dept_name='Finance' OR dept_name = 'Shipping'))
```

и преобразован не будет.

Условия, подлежащие преобразованию в соединение, которые должны быть выполнены для подзапроса, следующего за ключевым словом IN и ключевым словом ANY, идентичны. Это не совпадение, поскольку выражение

Запрос с операцией IN может быть преобразован в запрос с операцией ANY

WHERE *имя-столбца* **IN(** *подзапрос* **)**

является логически эквивалентным выражению

WHERE *имя-столбца* = **ANY(** *подзапрос* **)**

Поэтому запрос

```
SELECT emp_fname, emp_lname
FROM employee
WHERE emp_id IN (
   SELECT dept_head_id
   FROM department
WHERE (dept_name='Finance' or dept_name = 'Ship-
ping'))
```

является эквивалентным запросу

```
SELECT emp_fname, emp_lname
FROM employee
WHERE emp_id = ANY (
   SELECT dept_head_id
   FROM department
WHERE (dept_name='Finance' or dept_name = 'Ship-
ping'))
```

В принципе, Adaptive Server Anywhere преобразует запрос с операцией IN в запрос с операцией ANY, и соответственным образом решает, преобразовывать ли подзапрос в соединение.

Проверка существования

Подзапрос, следующий за ключевым словом EXISTS, преобразовывается в соединение только в случае его удовлетворения двум нижеприведенным условиям:

- Основной запрос не содержит раздел GROUP BY, не является агрегатным запросом, и подзапрос возвращает только одно значение.
- Конъюнкт "EXISTS (подзапрос)" не инвертирован.
- Подзапрос является коррелированным, т. е. он содержит внешнюю ссылку.

Пример

Таким образом, вопрос "Какие клиенты разместили заказы после 13 июля 2001 г.?", который можно сформулировать в виде запроса, и неинвертированный подзапрос которого содержит внешнюю ссылку **customer.id = sales_order.cust_id**, может быть преобразован в соединение.

```
SELECT fname, lname
FROM customer
WHERE EXISTS (
   SELECT *
   FROM sales_order
WHERE (order_date > '2001-07-13') AND (customer.id =
sales_order.cust_id))
```

В сущности, ключевое слово EXISTS сообщает серверу базы данных о необходимости выполнения проверки на пустые результирующие наборы. При использовании внутренних соединений сервер базы данных автоматически отображает только строки, содержащие данные из всех таблиц в разделе FROM. Поэтому этот запрос возвращает те же строки, что и запрос с подзапросом:

```
SELECT fname, lname
FROM customer, sales_order
WHERE (sales_order.order_date > '2001-07-13') AND
(customer.id = sales_order.cust_id)
```

ГЛАВА 10

Добавление, изменение и удаление данных

Об этой главе

В этой главе описывается процесс изменения данных в базе данных.

Большая часть главы посвящена операторам INSERT, UPDATE и DELETE, а также операторам для массовой загрузки и выгрузки.

Содержание

Операторы изменения данных

Операторы, используемые для добавления, изменения или удаления данных, называются операторами **изменения данных**. Наиболее общие такие операторы включают:

- **Insert** (Вставка). Добавляет новые строки в таблицу.
- **Update** (Обновление). Изменяет существующие строки в таблице.
- **Delete** (Удаление). Удаляет указанные строки из таблицы.

Один оператор INSERT, UPDATE или DELETE изменяет данные только в одной таблице или представлении.

Помимо общих операторов, операторы LOAD TABLE и TRUNCATE TABLE особенно полезны для массовой загрузки и удаления данных.

Иногда все операторы изменения данных собирательно называются **языком изменения данных** (DML), который является частью SQL.

Полномочия на изменение данных

Выполнение операторов изменения данных возможно только при наличии соответствующих полномочий на изменение требуемых таблиц. Администратор базы данных и владельцы объектов дазы данных предоставляют определенные права изменения объектов определенным пользователям с помощью операторов GRANT и REVOKE.

☞ Полномочия могут быть предоставлены отдельным пользователям, группам пользователей или группам общего пользования. Для получения дополнительной информации см. раздел "Управление кодами пользователей и полномочиями" (Managing User IDs and Permissions) на стр. 347 в документе *"Руководство по администрированию баз данных ASA" (ASA Database Administration Guide)*.

Транзакции и изменение данных

При изменении данных в журнале транзакций сохраняется копия старого и нового состояния каждой строки, на которую воздействует оператор изменения. Это означает, что если после начала транзакции обнаруживается ошибка, и выполняется откат транзакции, то база данных возвращается в первоначальное состояние.

☞ Для получения дополнительной информации о транзакциях см. раздел "Использование транзакций и уровней изоляции" на стр. 89.

Добавление данных с использованием оператора INSERT

Строки в базу данных добавляются при помощи оператора INSERT. Оператор INSERT имеет две формы: можно использовать ключевое слово VALUES или оператор SELECT:

Оператор INSERT, использующий значения

Ключевое слово VALUES определяет значения для некоторых или всех столбцов в новой строке. Упрощенная версия синтаксиса для оператора INSERT, использующего ключевое слово VALUES, следующая:

INSERT [INTO] *имя-таблицы* [(*имя-столбца, …*)]
VALUES (*выражение , …*)

Список имен столбцов можно опустить, если предоставляется значение для каждого столбца таблицы в порядке их появления при выполнении запроса с использованием SELECT *.

INSERT из SELECT

В операторе INSERT можно использовать оператор SELECT для перемещения значений из одной или более таблиц. Упрощенная версия синтаксиса для оператора INSERT, использующего оператор SELECT, следующая:

INSERT [**INTO**] *имя-таблицы* (*имя-столбца, …*) *оператор-Select*

Вставка значений во все столбцы строки

Следующий оператор INSERT добавляет новую строку в таблицу *department*, присваивая значение каждому столбцу в таблице:

```
INSERT INTO department
VALUES ( 702, 'Eastern Sales', 902 )
```

Примечания

- Введите значения в том же порядке, что и имена столбцов в первоначальном операторе CREATE TABLE, т. е. укажите сначала номер кода, затем имя, затем код начальника отдела.
- Заключите значения в круглые скобки.
- Заключите все символьные данные в одиночные кавычки.
- Используйте отдельный оператор INSERT для каждой добавляемой строки.

Вставка значений в определенные столбцы

Можно добавить данные в несколько столбцов в строке путем указания только этих столбцов и их значений. Определите все столбцы, не включенные в список столбцов, которые должны содержать NULL или значения по умолчанию. При пропуске столбца, имеющего значение по умолчанию, это значение по умолчанию появляется в этом столбце.

Добавление данных только в два столбца, например, *dept_id* и *dept_name*, требует оператора, подобного следующему:

```
INSERT INTO department (dept_id, dept_name)
VALUES ( 703, 'Western Sales' )
```

Столбец *dept_head_id* не содержит значения по умолчанию, но может предоставить NULL. Этому столбцу назначено значение NULL.

Порядок, в котором перечисляются имена столбцов, должен соответствовать порядку, в котором перечисляются значения. Результаты следующего примера аналогичны результатам предыдущего примера:

```
INSERT INTO department (dept_name, dept_id )
VALUES ('Western Sales', 703)
```

Значения для неопределенных столбцов

Когда значения определяются только для некоторых столбцов в строке, то для столбцов, для которых значения не указаны, может быть применен один из четырех вариантов:

- **Ввод NULL.** NULL появляется, если в столбце может быть указан NULL, и для этого столбца не существует значения по умолчанию.
- **Ввод значения по умолчанию.** Значение по умолчанию появляется, если для столбца существует такое значение.
- **Ввод уникального последовательного значения.** Уникальное последовательное значение появляется, если столбец имеет значение по умолчанию AUTOINCREMENT или свойство IDENTITY.
- **Отклонение INSERT и вывод сообщения об ошибке.** Сообщение об ошибке появляется, если столбец не содержит NULL, и для него не существует значения по умолчанию.

По умолчанию столбцы содержат NULL, если при создании таблиц в определении столбца явно не указано NOT NULL. Используя параметр ALLOW_NULLS_BY_DEFAULT, можно изменить значение по умолчанию.

Ограничение данных столбца с использованием ограничений

Для столбца или домена можно создать ограничения. Ограничения управляют видом данных, которые можно или нельзя добавить.

Для получения дополнительной информации об ограничениях см. раздел "Использование ограничений таблиц и столбцов" на стр. 75.

Явная вставка NULL

Вводом NULL в столбец можно явно вставить NULL. Не заключайте его в кавычки, иначе NULL будет принят за строку.

Например, следующий оператор явно вставляет NULL в столбец *dept_head_id*:

```
INSERT INTO department
VALUES (703, 'Western Sales', NULL )
```

Использование значений по умолчанию для предоставления значений

Столбец можно определить так, что, даже если он не получает никакого значения, значение по умолчанию автоматически появляется всякий раз при вставке строки. Это выполняется посредством назначением значения по умолчанию для столбца.

Для получения дополнительной информации о значениях по умолчанию см. раздел "Использование значений столбца по умолчанию" на стр. 70.

Добавление новых строк с SELECT

Для перемещения значений из одной таблицы в одну или более таблицы в операторе INSERT можно воспользоваться разделом SELECT. Раздел SELECT может вставлять значения в некоторые или во все столбцы строки.

Вставка значений только в некоторые столбцы может пригодиться, когда необходимо получить какие-либо значения из существующей таблицы. Затем можно использовать оператор UPDATE для добавления значений в другие столбцы.

Перед вставкой значений в некоторые, но не все столбцы таблицы убедитесь, что существует значение по умолчанию, либо что для столбцов, для которых значения не вставляются, вставляется NULL. В противном случае появляется сообщение об ошибке.

При вставке строк из одной таблицы в другую эти две таблицы должны иметь совместимую структуру, т.е. совпадающие столбцы должны содержать либо одинаковые типы данных, либо типы данных, которые Adaptive Server автоматически преобразовывает между собой.

Пример

Если столбцы находятся в своих операторах CREATE TABLES в одинаковом порядке, то указывать имена столбцов в той или иной таблице необходимости нет. Предположим наличие таблицы *newproduct*, содержащей некоторые строки с информацией о товаре в том же формате, что и в таблице *product*. Добавление всех строк в таблице *newproduct* в таблицу *product*:

```
INSERT product
SELECT *
FROM newproduct
```

Можно использовать выражения в операторе SELECT в рамках оператора INSERT.

Вставка данных в некоторые столбцы

Оператор SELECT можно использовать для добавления данных в некоторые, но не во все столбцы в строке, точно так же, как при работе с разделом VALUES. Просто определите столбцы, к которым необходимо добавить данные в разделе INSERT.

Вставка данных из той же таблицы

Данные в таблицу можно вставить на основании других данных, хранящихся в этой же таблицы. По существу это означает копирование всей строки или ее части.

Например, в таблицу *product* можно вставить новые товары на базе существующих в ней. Следующий оператор добавляет новые футболки самого большого размера (разновидности: с бретелями, с V-образным вырезом и вырезом "лодочкой") в таблицу *product*. Номер кода - на десять больше, чем код рубашек существующих размеров:

```
INSERT INTO product
SELECT id+ 10, name, description,
    'Extra large', color, 50, unit_price
```

```
FROM product
WHERE name = 'Tee Shirt'
```

Вставка документов и изображений

Если в столбцах LONG BINARY базы данных необходимо хранить документы или изображения, то можно написать приложение, которое будет считывать содержимое файла в переменную и поставлять эту переменную в качестве значения в оператор INSERT.

☞ Для получения дополнительной информации о добавлении операторов INSERT к приложениям см. раздел "Использование готовых операторов" (How to use prepared statements) на стр. 13 в документе *"Руководство по программированию для ASA" (ASA Programming Guide)* и раздел "Оператор SET" (SET statement) на стр. 495 в документе *"Справочник по SQL для ASA" (ASA SQL Reference Manual)*.

Для вставки содержимого файла в таблицу можно использовать системную функцию *xp_read_file*. Эта функция полезна, если необходимо вставлять содержимое файла из Interactive SQL или некоторых других сред, не предоставляющих полный язык программирования.

Для использования этой внешней функции необходимы полномочия администратора БД.

Пример

В этом примере создается таблица, и в ее столбец вставляется изображение. Эти шаги можно выполнить из Interactive SQL.

1. Создайте таблицу для хранения изображений.

```
CREATE TABLE pictures
( c1 INT DEFAULT AUTOINCREMENT PRIMARY KEY,
   filename VARCHAR(254),
   picture LONG BINARY )
```

2. Вставьте в таблицу содержимое файла *portrait.gif* , находящегося в текущем рабочей папке сервера базы данных.

```
INSERT INTO pictures (filename, picture)
VALUES ( 'portrait.gif',
   xp_read_file( 'portrait.gif' ) )
```

☞ Для получения дополнительной информации см. раздел "Системная процедура xp_read_file" (xp_read_file system procedure) на стр. 680 в документе *"Справочник по SQL для ASA" (ASA SQL Reference Manual)*.

Изменение данных с использованием оператора UPDATE

Для изменения отдельных строк, групп строк или всех строк в таблице можно воспользоваться оператором UPDATE с указанием после него названия таблицы или представления. Как и во всех операторах изменения данных, данные можно изменить одновременно только в одной таблице или представлении.

Оператор UPDATE определяет строку или строки, которые необходимо изменить, и новые данные. Новыми данными может быть константа или определенное выражение, либо данные, перенесенные из других таблиц.

Если оператор UPDATE нарушает ограничения, наложенные для обеспечения целостности, обновления не происходит, и появляется сообщение об ошибке. Например, если у одного из добавляемых значений неправильный тип данных, либо оно нарушает ограничение, определенное для одного из столбцов или задействованных типов данных, то обновления не происходит.

Синтаксис UPDATE

Упрощенная версия синтаксиса UPDATE такова:

UPDATE *имя-таблицы*
SET *имя_столбца = выражение*
WHERE *условие-поиска*

Если управление компанией Newton Ent. (в таблице клиентов демонстрационной базы данных) передано компании Einstein, Inc., то название компании можно обновить, используя следующий оператор:

```
UPDATE customer
SET company_name = 'Einstein, Inc.'
WHERE company_name = 'Newton Ent.'
```

В разделе WHERE можно использовать любое выражение. Если нет уверенности относительно правильного написания названия компании по буквам, то можно попробовать обновить любую компанию с именем Newton при помощи следующего оператора:

```
UPDATE customer
SET company_name = 'Einstein, Inc.'
WHERE company_name LIKE 'Newton%'
```

Условию поиска нет необходимости ссылаться на обновляемый столбец. Код компании Newton Entertainments - 109. Поскольку значение кода является первичным ключом для таблицы, то можно убедиться в обновлении нужной строки, используя следующий оператор:

```
UPDATE customer
SET company_name = 'Einstein, Inc.' WHERE id = 109
```

Раздел SET

Раздел SET определяет столбцы для обновления и их новые значения.

Раздел WHERE определяет строку или строки, которые будут обновлены. Если раздела WHERE нет, то определенные столбцы всех строк обновляются значениями, указанными в разделе SET.

В разделе SET можно указать любое выражение корректного типа данных.

Раздел WHERE

Раздел WHERE определяет строки, которые будут обновлены. Например, следующий оператор заменяет безразмерную футболку (one size fits all) на футболку очень большого размера (extra large):

```
UPDATE product
SET size  = 'Extra Large'
WHERE name = 'Tee Shirt'
   AND size = 'One Size Fits All'
```

Раздел FROM

Для переноса данных из одной или более таблиц в обновляемую таблицу можно воспользоваться разделом FROM.

Удаление данных с использованием оператора DELETE

Простые операторы DELETE имеют следующую форму:

DELETE [**FROM**] *имя-таблицы*
WHERE *имя-столбца = выражение*

Также можно использовать усложненную форму:

DELETE [**FROM**] *имя-таблицы*
FROM *список-таблиц*
WHERE *условие-поиска*

Раздел WHERE

Используйте раздел WHERE для определения строк для удаления. Если раздел WHERE не появляется, то оператор DELETE удаляет все строки в таблице.

Раздел FROM

Раздел FROM во второй позиции оператора DELETE является специальным средством, позволяющее выбирать данные из таблицы или таблиц и удалять соответствующие данные из первой именованной таблицы. Строки, выбираемые в разделе FROM, определяют условия для DELETE.

Пример

В этом примере используется демонстрационная база данных. Для выполнения операторов в примере параметр WAIT_FOR_COMMIT необходимо установить в OFF. Следующий оператор выполняет это только для текущего подключения:

```
SET TEMPORARY OPTION WAIT_FOR_COMMIT = 'OFF'
```

Это позволяет удалять строки, даже если они содержат первичные ключи, на которые ссылается внешний ключ, но не позволяет выполнить COMMIT, если удален соответствующий внешний ключ.

Следующее представление отображает товары и сумму для проданного товара:

```
CREATE VIEW ProductPopularity as
SELECT  product.id,
SUM(product.unit_price * sales_order_items.quantity)
as "Value Sold"
FROM  product JOIN sales_order_items
ON product.id = sales_order_items.prod_id
GROUP BY product.id
```

Используя это представление, можно удалить из таблицы product товары, проданные меньше чем на 20 000 долл..

```
DELETE
FROM product
FROM product NATURAL JOIN ProductPopularity
WHERE "Value Sold" < 20000
```

По завершении данного примера необходимо выполнить откат всех внесенных изменений:

```
ROLLBACK
```

Удаление всех строк из таблицы

Оператор TRUNCATE TABLE (усечение таблицы) можно использовать в качестве быстрого способа удаления всех строк из таблицы. Он работает быстрее, чем оператор DELETE без условий, потому что изменяется каждый журнал удалений, в то время как журнал транзакций не записывает отдельно операции усечения таблицы.

Определение для таблицы, освобождаемой оператором TRUNCATE TABLE, остается в базе данных вместе с индексами и другими связанными объектами, если не выполнен оператор DROP TABLE.

Оператор TRUNCATE TABLE выполнить нельзя, если в другой таблице содержатся строки, которые ссылаются на них через средства контроля за ссылочной целостностью. Удалите строки из внешней таблицы или усеките внешнюю таблицу, после чего сократите первичную таблицу.

Синтаксис TRUNCATE TABLE

Синтаксис TRUNCATE TABLE следующий:

TRUNCATE TABLE *имя-таблицы*

Например, для удаления всех данных из таблицы sales_order введите следующее:

```
TRUNCATE TABLE sales_order
```

Оператор TRUNCATE TABLE не запускает триггеры, определенные для таблицы.

ГЛАВА 11

Оптимизация и выполнение запросов

Об этой главе

После получения запроса оптимизатор анализирует его и выбирает план доступа, использующий наименьшее количество ресурсов. В этой главе описаны действия оптимизатора при оптимизации запроса, принципы, которые лежат в основе построения оптимизатора. Также рассматривается оценка выборочности, оценка затрат и другие шаги оптимизации.

Хотя операторы обновления, вставки и удаления также должны быть оптимизированы, в этой главе рассмотрены только запросы выбора. Оптимизация других команд осуществляется по этим же принципам.

Содержание

Задачи оптимизатора

Главная задача оптимизатора состоит в определении эффективного способа выполнения оператора SQL. Оптимизатор выдает выбранный метод в виде **плана доступа**. План доступа описывает таблицы, которые следует просматривать, индексы для каждой таблицы, которые следует использовать, если они существуют, алгоритм соединения и порядок чтения таблиц. Обычно существует большое число планов доступа, которые удовлетворяют заданным условиям. Другие переменные могут еще увеличить количество возможных планов доступа.

Основанность на оценке затрат ресурсов

Оптимизатор выбирает доступные планы доступа, используя эффективные и в некоторых случаях особые алгоритмы. Выбор основывается на предсказании количества ресурсов, которые требует каждый запрос. Оптимизатор учитывает затраты на доступ к диску и загрузку CPU при каждой операции.

Независимость от синтаксиса

Большинство команд может быть выражено различными способами с использованием языка SQL. Эти выражения семантически эквивалентны, так как они выполняют одну и ту же задачу, но могут существенно различаться по синтаксису. С небольшими исключениями оптимизатор Adaptive Server Anywhere создает подходящий план доступа, основанный только на семантике каждого оператора.

Синтаксические различия, хотя они могут казаться существенными, обычно на результат не влияют. Например, различный порядок предикатов, таблиц и атрибутов в синтаксисе запроса не влияет на выбор плана доступа. На действие оптимизатора также не влияет наличие или отсутствие представления в запросе.

Хороший план - не обязательно лучший план

Цель оптимизатора состоит в нахождении “лучшего” плана доступа. В идеальном случае оптимизатор находит самый эффективный из возможных планов доступа, но обычно постановка такой цели непрактична. При обработке сложного запроса может существовать большое количество вариантов.

Однако независимо от степени эффективности оптимизатора при анализе каждого варианта требуется время и ресурсы. Оптимизатор сравнивает затраты на дальнейшую оптимизацию с затратами на выполнение лучшего уже найденного плана. Если найденный план требует меньшего количества ресурсов, оптимизатор останавливается и запускает его выполнение. Дальнейшая оптимизация потребует больше ресурсов, чем выполнение уже найденного плана доступа.

В случае ресурсоемких и сложных запросов оптимизатор работает дольше. В случае очень ресурсоемких запросов он может вызвать заметную задержку в работе.

Принципы работы оптимизатора

Оптимизатор Adaptive Server Anywhere определяет порядок обращения к таблицам в запросе и необходимость использования индекса для каждой таблицы. Оптимизатор пытается выбрать “лучшую” стратегию.

Лучшей стратегией выполнения запроса является та, при которой результат может быть получен за самый короткий отрезок времени с наименьшими затратами ресурсов. Оптимизатор определяет затраты ресурсов для каждой стратегии, оценивая требуемый объем чтения и записи на диск, и выбирает стратегию с самыми низкими затратами.

Оптимизатор использует универсальную модель оценки затрат на обращение к диску, чтобы дифференцировать различия в относительной производительности между случайным и последовательным поиском в файле базы данных. Калибровка базы данных под определенную аппаратную конфигурацию возможна с помощью оператора ALTER DATABASE. Особенности конкретной модели затрат могут быть установлены с помощью хранимой процедуры sa_get_dtt().

По умолчанию обработка запроса оптимизируется для быстрого возвращения первой строки. С помощью параметра OPTIMIZATION_GOAL можно изменить это для минимизации затрат на возвращение всего результата.

В Interactive SQL план выполнения запроса на закладке Plan окна Results показывает порядок чтения таблиц для текущего запроса и индекс (в круглых скобках), используемый для таблицы.

Для получения дополнительной информации о целях оптимизации см. раздел "Параметр OPTIMIZATION_GOAL" (OPTIMIZATION_GOAL option) на стр. 572 в документе *"Руководство по администрированию баз данных ASA" (ASA Database Administration Guide)*.

Для получения дополнительной информации о чтении планов доступа см. раздел "Чтение планов доступа" на стр. 353.

Оценки оптимизатора

Оптимизатор выбирает стратегию обработки оператора, основываясь на гистограммах, которые размещены в базе данных, и эвристике (обучаемых предположениях).

Гистограммы, называемые также статистикой столбца, содержат информацию о распределении значений в столбце. В Adaptive Server Anywhere гистограмма представляет распределение данных для столбца, разделяя домен столбца на набор последовательных областей значений и сохраняя для каждой области количество строк в таблице, для которой значение столбца уменьшается.

В Adaptive Server Anywhere особое внимание уделяется отдельным значениям столбцов, которые повторяются в большом количестве строк в таблице. Фактически любое значение, которое повторяется больше чем в 1 % строк таблицы, сохраняется как одноэлементная область значений в гистограмме.

Используя гистограмму столбца, Adaptive Server Anywhere пытается оценить число строк, удовлетворяющих данному запросу по столбцу, суммируя число строк во всех областях значений, которые перекрывают значение указанного предиката. Области значений (или области памяти) в гистограммах, которые частично содержатся в результате запроса, Adaptive Server Anywhere интерполирует в пределах одной области значений.

Реализация гистограмм Adaptive Server Anywhere позволяет им совершенствоваться при выполнении запроса. Во время выполнения запроса Adaptive Server Anywhere сравнивает ожидаемое по гистограммам количество строк для данного предиката с количеством фактически найденных и затем корректирует значения в гистограмме для уменьшения погрешности при последующей оптимизации.

Для каждой таблицы в возможном плане выполнения оптимизатор оценивает число строк, которые составят часть результата. Количество строк зависит от размера таблицы и ограничений в разделах WHERE и ON запроса.

Во многих случаях оптимизатор использует более сложную эвристику. Например, оптимизатор использует оценки по умолчанию только в тех случаях, когда другие статистические данные недоступны. Оптимизатор также применяет индексы и ключи для повышения точности предсказания количества строк. Далее приведено несколько примеров с одним столбцом:

- Приравнивание столбца к значению: ожидание одной строки, если столбец имеет уникальный индекс или первичный ключ.
- Сравнение индексированного столбца с константой: проверка индекса для оценки процента строк, которые удовлетворяют сравнению.
- Приравнивание внешнего ключа к первичному ключу (ключевое соединение): используются относительные размеры таблицы для вычисления оценки. Например, если таблица из 5000 строк имеет внешний ключ к таблице из 1000 строк, оптимизатор предположит, что на каждую строку первичного ключа приходится пять строк внешнего ключа.

☞ Для получения дополнительной информации о статистике столбца см. раздел "Системная таблица SYSCOLSTAT" (SYSCOLSTAT system table) на стр. 568 в документе *"Справочник по SQL для ASA" (ASA SQL Reference Manual)*.

☞ Для получения дополнительной информации об определении выборочности предикатов и распределения значений столбцов см. следующие разделы:

- "Системная процедура sa_get_histogram" (sa_get_histogram system procedure) на стр. 653 в документе *"Справочник по SQL для ASA" (ASA SQL Reference Manual);*

- "Утилита гистограмм" (The Histogram utility) на стр. 454 в документе *"Руководство по администрированию баз данных ASA" (ASA Database Administration Guide);*
- "Функция ESTIMATE" (ESTIMATE function) на стр. 127 в документе *"Справочник по SQL для ASA" (ASA SQL Reference Manual);*
- "Функция ESTIMATE_SOURCE" (ESTIMATE_SOURCE function) на стр. 127 в документе *"Справочник по SQL для ASA" (ASA SQL Reference Manual).*

Обновление статистики столбца

Статистические данные столбца постоянно хранятся в базе данных в системной таблице SYSCOLSTAT.

Иногда эти статистические данные могут неточно отражать текущие значения столбцов. Обычно такая ситуация возникает после действий, значительно изменяющих содержание таблицы; например, удаления или вставки большого количества строк.

В таких ситуациях можно выполнить оператор DROP STATISTICS или CREATE STATISTICS. CREATE STATISTICS удаляет старые статистические данные и создает новые, а DROP STATISTICS только удаляет старые данные.

При использовании более точной статистики оценки оптимизатора становятся точнее, повышая эффективность следующих запросов. Однако неправильные оценки становятся проблемой только в случае, если это приводит к недостаточно оптимизированным запросам.

Статистические данные для таблицы создаются при выполнении LOAD TABLE.

Однако при добавлении, удалении или обновлении строк таблицы статистические данные не обновляются.

При работе с маленькими таблицами гистограмма не влияет существенно на выбор оптимизатором эффективного плана. Можно определить минимальный размер таблицы, для которой будут созданы гистограммы. По умолчанию это 1000 строк. При выполнении оператора CREATE STATISTICS гистограммы создаются для всех таблиц независимо от количества строк.

Для получения дополнительной информации см. раздел "Параметр MIN_TABLE_SIZE_FOR_HISTOGRAM" (MIN_TABLE_SIZE_FOR_HISTOGRAM option) на стр. 568 в документе *"Руководство по администрированию баз данных ASA" (ASA Database Administration Guide).*

Для получения дополнительной информации о статистике столбца см. следующие разделы:

- "Системная таблица SYSCOLSTAT" (SYSCOLSTAT system table) на стр. 568 в документе *"Справочник по SQL для ASA" (ASA SQL Reference Manual);*
- "Оператор DROP STATISTICS" (DROP STATISTICS statement) на стр. 375 в документе *"Справочник по SQL для ASA" (ASA SQL Reference Manual);*

- "Оператор CREATE STATISTICS" (CREATE STATISTICS statement) на стр. 302 в документе *"Справочник по SQL для ASA" (ASA SQL Reference Manual)*.

Автоматическая настройка производительности

Одним из самых распространенных ограничений в запросе является равенство со значением столбца. В следующем примере выполняется проверка на равенство столбца *sex*.

```
SELECT *
FROM employee
WHERE sex = 'f'
```

Запросы часто оптимизируются по-разному при повторном выполнении. При ограничении вышеуказанного типа Adaptive Server Anywhere использует предыдущий опыт, автоматически учитывая столбцы, которые имеют необычное распределение значений. Эту информацию база данных содержит постоянно до выполнения команды DROP STATISTICS. Следует отметить, что последующие запросы с предикатами по этому столбцу могут заставить сервер обновить гистограмму на столбце.

Используемые принципы

В основе философии оптимизатора запроса Adaptive Server Anywhere лежит множество принципов. Понимание того, на чем основаны принимаемые оптимизатором решения, позволяет повысить качество или производительность пользовательских приложений. Описание принципов работы оптимизатора содержится в следующих разделах этого документа.

Минимальная работа по администрированию

Обычно высокоэффективные серверы баз данных рассчитаны на то, что администрирование будут выполнять опытные и хорошо осведомленные администраторы. Чтобы обеспечить высокую производительность базы данных, требуется много времени на настройку запоминающего устройства и элементов управления производительностью всех видов. Эти элементы управления часто требуется настраивать при изменении информации в базе данных.

По мере роста и изменения базы данных Adaptive Server Anywhere использует накопленные знания и автоматически корректирует свою работу. Количество доступных ему сведений о распределения данных в базе данных повышается с каждым запросом. Adaptive Server Anywhere автоматически сохраняет и использует эту информацию для оптимизации дальнейших запросов.

При выполнении каждого запроса увеличивается доля накопленных сведений, и делаются соответствующие выводы о дальнейшей работе. Каждый пользователь может воспользоваться знанием, приобретенным Adaptive Server Anywhere при выполнении запросов других пользователей.

Механизмы сбора статистики составляют неотъемлемую часть сервера баз данных и не требуют внешнего обеспечения. При необходимости можно предоставить серверу баз данных оценки распределений данных для использования во время оптимизации. Если поместить их, например, в триггер или процедуру, тогда ответственность за их поддержку и обновление ляжет на пользователя.

Оптимизация для первой строки или всего результата

Параметр OPTIMIZATION_GOAL определяет оптимизацию для быстрого возвращения первой строки или уменьшения затрат ресурсов при возвращении всего результата. По умолчанию оптимизируется время возвращения первой строки.

Для получения дополнительной информации см. раздел "Параметр OPTIMIZATION_GOAL" (OPTIMIZATION_GOAL option) на стр. 572 в документе *"Руководство по администрированию баз данных ASA" (ASA Database Administration Guide)*.

Статистические данные правильны и актуальны

Оптимизатор производить самонастройку, сохраняя всю необходимую информацию внутри. Системная таблица SYSCOLSTAT содержит постоянный архив распределений данных и оценок выборочности предикатов. После завершения каждого запроса Adaptive Server Anywhere использует статистику, собранную в течение выполнения запроса, для обновления таблицы SYSCOLSTAT. Все последующие запросы получают доступ к более точным оценкам.

Оптимизатор полагается на эти статистические данные, поэтому качество планов доступа тоже зависит от них. При добавлении большого количества новых строк эти статистические данные могут не описывать точно информацию таблицы. Может казаться, что последующие запросы выполняются очень медленно.

При значительном изменении данных и при заметном замедлении выполнения запросов можно выполнить DROP STATISTICS и/или CREATE STATISTICS.

Для каждого предиката обычно может быть найден соответствующий индекс

Обычно Adaptive Server Anywhere может оценить предикаты при помощи индекса. Используя индекс, оптимизатор ускоряет доступ к данным и сокращает количество читаемой информации. Например, если параметр OPTIMIZATION_GOAL настроен на оптимизацию времени возвращения первой строки, оптимизатор Adaptive Server Anywhere попытается использовать индексы, соответствующие разделам ORDER BY и GROUP BY.

Если оптимизатор не находит подходящий индекс, то он производит последовательную проверку таблицы, которая может потребовать много ресурсов. Присоединение индекса к таблице может значительно повысить производительность. Добавьте индексы к таблицам или перезапишите запросы везде, где это повысит эффективность обработки запросов.

Виртуальная память как дефицитный ресурс

Операционная система и приложения часто совместно используют оперативную память компьютера. Adaptive Server Anywhere рассматривает память как дефицитный ресурс. Так как память расходуется экономно, Adaptive Server Anywhere может работать на относительно небольших компьютерах. Эта важно при использовании базы данных на портативных или на устаревших компьютерах.

Резервирование дополнительной памяти, например, для хранения курсора, может потребовать много дополнительных ресурсов. Если буферный кэш занят, происходит запись одной или более страниц на диск для освобождения места под новые страницы. Некоторые страницы, возможно, будут заново прочитаны при выполнении следующей операции.

Для учета этого Adaptive Server Anywhere связывает более высокие затраты с планами выполнения, которые требуют дополнительных расходов места в буферном кэше. Эти затраты препятствуют оптимизатору при выборе планов, которые используют рабочие таблицы.

С другой стороны, он позволяет использовать память там, где это повышает производительность. Например, оптимизатор кэширует результаты подзапросов, если они будут необходимы в течение обработки запроса.

Перезапись подзапросов как предикатов EXISTS

В основе Adaptive Server Anywhere лежит требование сохранения памяти, и, по умолчанию, он возвращает первые строки результата курсора настолько быстро, насколько это возможно. В соответствии с этими целями Adaptive Server Anywhere перезаписывает все предикаты IN, ANY или SOME как предикаты EXISTS. Таким образом, Adaptive Server Anywhere не создает ненужные рабочие таблицы и может быстрее найти подходящий индекс для обращения к таблице.

Некоррелированные подзапросы - это подзапросы, в которых не содержится явной ссылки на таблицу или таблицы остальной части запроса более высокого уровня.

В следующем примере содержится запрос с некоррелированным подзапросом. Он извлекает информацию обо всех клиентах, которые не размещали заказов 1 января 2001.

Некоррелированный подзапрос

```
SELECT *
FROM customer c
WHERE c.id NOT IN
   ( SELECT o.cust_id
 FROM sales_order o
 WHERE o.order_date = '2001-01-01' )
```

Возможный способ обработки этого запроса состоит в том, чтобы сначала прочитать таблицу *sales_order* и создать таблицу всех клиентов, которые поместили заказы 1 января 1998, а потом читать таблицу *customer* и извлекать по одной строке с клиентами, которых нет в рабочей таблице.

Adaptive Server Anywhere избегает помещения результатов в рабочие таблицы. Предпочтение отдается планам, которые возвращают первые строки результата наиболее быстро. Оптимизатор перезаписывает такие запросы, используя предикаты EXISTS. В такой форме подзапрос становится коррелированным: подзапрос содержит явную ссылку на столбец *id* таблицы *customers*.

Коррелированный подзапрос

```
SELECT *
FROM customer c
WHERE NOT EXISTS
   (  SELECT *
   FROM sales_order o
   WHERE o.order_date = '2000-01-01'
      AND o.cust_id = c.id )
```

c<seq> : o<key_so_customer>

Этот запрос семантически эквивалентен приведенному выше, но при выражении в этом новом синтаксисе становятся очевидными два преимущества.

1. Оптимизатор может использовать либо индекс *cust_id*, либо атрибут *order_date* таблицы *sales_order*. (Однако в демонстрационной базе данных индексированы только столбцы *id* и *cust_id*.)
2. Оптимизатор имеет возможность выбрать обработку подзапроса без создания промежуточной рабочей таблицы.

Adaptive Server Anywhere может кэшировать результаты этого коррелированного подзапроса во время обработки. Эта стратегия позволяет Adaptive Server Anywhere многократно использовать предварительно вычисленные результаты. В случае вышеописанного запроса кэширование не помогает, потому что идентификационные номера клиентов в таблице клиентов уникальны.

Для получения дополнительной информации о кэшировании подзапросов см. раздел "Кэширование подзапросов и функций" на стр. 352.

Кэширование плана доступа

Обычно оптимизатор выбирает план доступа при выполнении каждого запроса. Оптимизация во время выполнения позволяет выбрать план в соответствии с текущим состоянием системы, а также значениями текущей оценки выборочности и оценками, основанными на значениях хост-переменных. Затраты на оптимизацию часто повторяющихся запросов могут превзойти преимущества от оптимизации во время выполнения запроса. Оптимизатор кэширует планы выполнения для запросов INSERT, UPDATE и DELETE, находящихся в хранимых процедурах, хранимых функциях и триггерах, между выполнениями запроса.

После выполнения оператора в хранимой процедуре, хранимой функции или триггере несколько раз при одном подключении оптимизатор создает для него план многократного использования. План многократного использования не содержит значений хост-переменных для оценки выборочности и не производит перезаписи данных оптимизации. Из-за этого затраты на выполнение плана многократного использования могут быть больше. Если затраты на выполнение плана многократного использования близки к лучшим наблюдаемым затратам для оператора, то оптимизатор добавит план многократного использования в кэш. В противном случае преимущества оптимизации при каждом запросе перевесят экономию затрат при игнорировании оптимизации и отсутствии кэширования плана доступа.

Кэш плана доступа - это кэш данных, используемый для выполнения плана доступа во время соединения. При многократном использовании кэшируемого плана включается поиск плана в кэше и его сброс в первоначальное состояние. Обычно это существенно быстрее, чем оптимизация оператора. Если кэшируемые планы используются редко, они могут быть сохранены на диск, в этом случае кэш не используется. Оптимизатор периодически заново оптимизирует запросы, чтобы проверить относительную эффективность кэшируемого плана.

Свойство QueryCachePages базы данных/подключения служит для определения количества страниц кэша, используемых под планы выполнения. Они занимают место во временном файле, но не всегда располагаются резидентно в памяти.

Параметр MAX_PLANS_CACHED устанавливает максимальное количество кэшируемых планов. По умолчанию установлено значение 20. Для выключения кэширования планов установите значение 0.

☞ Для получения дополнительной информации см. раздел "Параметр MAX_PLANS_CACHED" (MAX_PLANS_CACHED option) на стр. 566 в документе *"Руководство по администрированию баз данных ASA" (ASA Database Administration Guide)*.

Этапы оптимизации

При генерации подходящего плана оптимизатор Adaptive Server Anywhere выполняет следующие шаги.

1. Синтаксический анализатор конвертирует SQL-запрос во внутреннее представление. Запрос может быть преобразован синтаксически, оставаясь семантическим эквивалентом. Например, подзапрос может

быть преобразован в соединение. Эти преобразования производится для упрощения анализа оператора.

2. Оптимизация начинается перед самым выполнением. Если в приложении используются курсоры, оптимизация начинается при открытии курсора. В отличие от многих других систем коммерческих баз данных, Adaptive Server Anywhere оптимизирует каждый оператор перед самым выполнением.
3. Оптимизатор выполняет семантическую оптимизацию оператора. Операторы SQL перезаписываются, если это позволяет получить лучшие, более эффективные планы доступа.
4. Оптимизатор устанавливает количество соединений для каждого подзапроса.
5. Оптимизатор выполняет оптимизацию порядка доступа.

Поскольку Adaptive Server Anywhere выполняет оптимизацию каждого оператора во время запроса, оптимизатору доступны значения хост-переменных и переменные хранимой процедуры. Следовательно, он делает лучший выбор, потому что использует лучший анализ выборочности.

Adaptive Server Anywhere оптимизирует каждый выполняемый запрос независимо от того, сколько раз он выполнялся прежде, за исключением запросов, которые содержатся в хранимых процедурах или функциях, определяемых пользователем. Оптимизатор может кэшировать планы доступа для многократного использования запросов, содержащихся в хранимых процедурах или функциях, определяемых пользователем.

☞ Для получения дополнительной информации см. раздел "Кэширование плана доступа" на стр. 314.

Adaptive Server Anywhere сохраняет статистику при выполнении каждого запроса, поэтому оптимизатор может использовать опыт выполнения предыдущих планов доступа и корректировать их выбор.

Простые запросы

Если запрос распознан как простой запрос, используется эвристический метод, а не оптимизация на основе анализа требуемых ресурсов. Оптимизатор решает, использовать ли просмотр индекса или последовательный просмотр таблицы и немедленно создает и выполняет план доступа.

Шаги 4 и 5 не выполняются.

Простым запросом, например, является курсор DYNAMIC SCROLL или NO SCROLL, который не содержит подзапросов, более одной таблицы, таблицы прокси, функций определяемых пользователем, NUMBER(*), UNION, агрегирования, DISTINCT, GROUP BY или более одного предиката для отдельного столбца. Простые запросы могут содержать ORDER BY, только если раздел WHERE содержит условия для каждого первичного ключа столбца.

Алгоритмы выполнения запроса

Далее следует пояснение использования алгоритмов для выполнения запросов в Adaptive Server Anywhere.

Обращение к таблицам

Просмотр индекса и последовательный просмотр таблицы являются основными способами обращения к отдельной таблице.

Сканирование с использованием индекса

При этом способе индекс сканируется для определения строк, которые удовлетворяют условию поиска. Читаются только те страницы, которые удовлетворяют заданному условию. Индексы могут возвращать отсортированные строки.

Индексы отображаются в коротком и длинном плане как *имя_корреляции* <*имя_индекса*>, где *имя_корреляции* - это имя корреляции, указанное в разделе FROM, или имя таблицы, если ничего не указано; *имя_индекса* - имя используемого индекса.

Индексы предоставляют эффективный механизм для чтения нескольких строк из большой таблицы. Однако при использовании индексов страницы из базы данных могут читаться в случайном порядке, что требует больше ресурсов, чем последовательное чтение. Индекс может сослаться на одну страницу таблицы несколько раз, если на странице есть несколько строк, удовлетворяющих условию поиска. Если только несколько страниц вызываются при просмотре индекса, эти страницы, возможно, останутся в кэше, и повторный доступ не потребует дополнительных операций чтения и записи. Однако если много страниц удовлетворяют условию поиска, они не поместятся в кэше. Это может привести к многократному чтению одной страницы с диска.

Оптимизатор предпочтет использование просмотра индекса последовательному просмотру таблицы, если параметр OPTIMIZATION_GOAL установлен на первую строку (по умолчанию). Это происходит из-за того, что при использовании индексов возвращение первых запрошенных строк происходит обычно быстрее, чем при последовательном просмотре таблицы.

Индексы также используются для удовлетворения условиям сортировки, либо явно определенным в разделе оператора ORDER BY, либо требуемым неявно в разделах оператора GROUP BY или DISTINCT. Методы упорядоченного группирования и упорядочивания с исключением строк могут возвращать первые строки быстрее, чем при группировании на основе хэша, но могут быть медленнее при возвращении всего результата.

Оптимизатор использует индекс для удовлетворения условиям поиска, если условие поиска позволяет использовать аргументы поиска, и оценка оптимизатора затрат ресурсов достаточно низка для просмотра индекса по сравнению с затратами на последовательный просмотр таблицы.

Для получения дополнительной информации об использовании индексов Adaptive Server Anywhere см. раздел "Анализ предиката" на стр. 339.

Для получения дополнительной информации о целях оптимизации см. раздел "Параметр OPTIMIZATION_GOAL" (OPTIMIZATION_GOAL option) на стр. 572 в документе *"Руководство по администрированию баз данных ASA" (ASA Database Administration Guide)*.

Последовательный просмотр таблицы

При последовательном просмотре таблицы происходит чтение всех строк на всех страницах таблицы в том порядке, в котором они хранятся в базе данных.

Последовательный просмотр таблицы отображается в коротком и длинном плане как *имя_корреляции <seq>*, где *имя_корреляции* - это имя корреляции, указанное в разделе FROM, или имя таблицы, если ничего не указано.

Этот тип просмотра используется, когда большинство страниц таблицы содержат строки, которые удовлетворяют условию поиска запроса, или подходящий индекс не определен.

Хотя при последовательном просмотре таблицы происходит чтение большего количества страниц, чем при использовании индекса, затраты на операции с обращением к диску могут быть существенно ниже, потому что страницы читаются непрерывными блоками с диска (повышение производительности особенно заметно, если файл базы данных не фрагментирован на диске). При последовательном вводе/выводе уменьшаются задержки на перемещение головки диска и задержки из-за вращения диска. При последовательном просмотре больших таблиц происходит одновременное чтение нескольких страниц. Это снижает затраты на последовательный просмотр таблицы по сравнению со сканированием с использованием индекса.

Хотя при выборке большого количества строк из таблицы последовательный просмотр требует меньше времени, чем просмотр по индексу, эффективность использования кэша в случае выполнения просмотра несколько раз выше при использовании индекса. Так как при использовании индекса, скорее всего, будет прочитано меньшее количество страниц таблицы, более вероятно, что эти страницы останутся в кэше. Это приведет к более быстрому обращению к ним. Поэтому для неоднократного обращения к таблице, например, на правой стороне вложенного соединения, предпочтительным является сканирование с использованием индексов.

При уровне изоляции 3 Adaptive Server Anywhere устанавливает блокировку на каждой вызываемой строке, даже если она не удовлетворяет условию поиска. На этом уровне при последовательном просмотре таблицы блокировка требуется на всех строках таблицы, а при использовании индекса только на строках, которые соответствуют условию поиска. Таким образом, последовательный просмотр таблицы может существенно уменьшить пропускную способность в многопользовательских средах. По этой причине на уровне изоляции 3 оптимизатор предпочитает индексированный доступ последовательному чтению.

Список IN

Алгоритм списка IN используется в случаях, когда предикат IN можно удовлетворить с использованием индекса. Например, в следующем запросе оптимизатор распознает, что он может обратиться к таблице служащего, используя ее первичный ключ.

```
SELECT *
FROM employee
WHERE emp_id in (102, 105, 129)
```

Для этого создается соединение со специальной таблицей списка IN слева. Строки выбираются из списка IN и используются для просмотра таблицы служащего. Для множественных списков IN используется этот же индекс. Если оптимизатор предпочитает не использовать индекс для удовлетворения условий предиката IN (например, потому что использование другого индекса приводит к более высокой производительности), то список IN появляется как предикат в фильтре.

Алгоритмы соединения

Алгоритмы соединения требуются, когда в разделе FROM содержится больше одной таблицы. Оптимизатор может самостоятельно выбрать алгоритм соединения, действия пользователя для этого не требуются.

Соединение с вложенными циклами

При соединении с вложенными циклами происходит полное чтение правой стороны для каждой строки левой стороны. (Синтаксический порядок таблиц в запросе не имеет значения, так как оптимизатор сам выбирает подходящий порядок соединения для каждого блока в запросе.)

Оптимизатор выбирает этот алгоритм, если условие соединения не содержит условия равенства, или при оптимизации для быстрого возвращения первой строки.

Так как при соединении с вложенными циклами происходит многократное чтение правой стороны, этот алгоритм очень чувствителен к затратам на ее чтение. Если правая сторона представляет собой результаты сканирования индекса или небольшую таблицу, то она, вероятно, может быть быстро вычислена с использованием кэшируемых страниц предыдущих итераций. С другой стороны, если правая сторона представляет собой результат последовательного просмотра таблицы или индекса, который соответствует многим строкам, то правая сторона читается с диска много раз. Как правило, соединение с вложенными циклами менее эффективно, чем другие типы соединения. Однако этот метод позволяет быстрее вычислить первую строку, по сравнению с другими типами соединения, которые вычисляют весь результат перед возвращением строки.

Соединение с вложенными циклами - это единственный алгоритм соединения, в котором предоставляется чувствительная семантика для запросов, содержащих соединения. Это означает, что чувствительные курсоры на соединениях могут выполняться только при соединении с вложенными циклами.

Соединение Exists выбирает только первую совпавшую строку справа. Это более эффективная версия соединения с вложенными циклами, но она может использоваться только при наличии ключевого слова EXISTS или иногда DISTINCT.

Соединение с вложенными блоками и сортированный блок

Соединение с вложенными блоками (также называемое соединением блоков с вложенными циклами) читает блок строк слева и сортирует строки по атрибутам соединения (столбцы, используемые в условиях соединения). Левую часть соединения с вложенными блоками называют узлом сортированных блоков. Для каждого блока строк с одинаковыми атрибутами соединения правая сторона сканируется только один раз. Этот алгоритм улучшает соединение с вложенными циклами при наличии нескольких строк слева, которые соединяются с каждой строкой справа.

Оптимизатор выберет соединение с вложенными блоками, если левая сторона содержит много строк с одинаковыми значениями атрибутов соединения, а правая сторона имеет индекс, удовлетворяющий условию поиска.

Каждое соединение с вложенными блоками содержит левую часть, которая является узлом сортированных блоков. Затраты для этого узла являются затратами на чтение и сортировку строк с левой стороны.

Данные с левой стороны сохраняются в памяти блоками. Изменения таблиц данных с левой стороны могут не отражаться в результатах. Из-за этого соединение с вложенными блоками не может предоставить чувствительную семантику.

Соединения с вложенными блоками блокируют строки слева прежде, чем они копируются в память.

Соединение хэша

Алгоритм соединения хэша создает в памяти таблицу хэша меньших данных с одной из сторон, затем читает большую таблицу и проверяет в памяти таблицу хэша для того, чтобы найти соответствия, которые будут записаны в рабочую таблицу. Если меньшие данные не помещаются в память, операция соединения хэша разделяет оба набора данных и помещает их в меньшие рабочие таблицы. Эти меньшие рабочие таблицы обрабатываются рекурсивно до тех пор, пока меньший набор данных не сможет быть сохранен в памяти.

Алгоритм соединения хэша предоставляет наивысшую производительность, если меньшие наборы данных помещаются в память, независимо от размера всего набора данных. Оптимизатор выбирает соединение хэша, если один из наборов данных предположительно будет существенно меньшим, чем другой.

Если алгоритм соединения хэша выполняется в среде с недостаточным количеством кэш-памяти для хранения всех строк, которые имеют различные значения атрибутов соединения, то его выполнение невозможно. В этом случае при соединении хэша не сохраняются промежуточные результаты, и вместо этого используется соединение с вложенными циклами, основанное на индексе. Все строки меньшей таблицы читаются и используются для проверки рабочей таблицы с целью поиска соответствий. Стратегия, основанная на индексировании, значительно медленнее других методов соединения. Оптимизатор старается избежать использования планов доступа, использующих соединение хэша, если обнаруживает, что во время выполнения запроса может возникнуть ситуация нехватки памяти. Когда из-за малого количества памяти используется соединение с вложенными циклами, счетчик производительности увеличивается приращением. Посмотреть этот монитор можно в свойстве QueryLowMemoryStrategy базы данных/подключения или в счетчике "Query: Low Memory Strategies" системного монитора NT.

Для получения дополнительной информации см. QueryLowMemoryStrategy в разделе "Свойства на уровне подключения" (Connection-level properties) на стр. 601 в документе *"Руководство по администрированию баз данных ASA" (ASA Database Administration Guide)*.

Алгоритм соединения хэша вычисляет все строки результата перед возвращением первой строки.

Алгоритм соединения хэша использует рабочую таблицу с нечувствительной семантикой, за исключением случаев, когда требуется чувствительный к значениям курсор.

Соединение хэша блокирует строки в его наборах данных до тех пор, пока они не будут скопированы в память.

Соединение со слиянием

Соединение со слиянием читает два набора данных, отсортированные по атрибутам соединения. Для каждой строки левого набора данных алгоритм читает все совпавшие строки правого набора данных, обращаясь к отсортированным строкам.

Если наборы данных не отсортированы по атрибутам соединения (возможно, из-за более раннего соединения со слиянием или из-за использования индекса для удовлетворения условиям поиска), то оптимизатор выполняет сортировку с целью расположения строк в правильном порядке. Эта сортировка влечет дополнительные затраты при соединении со слиянием.

Преимущество соединения со слиянием по сравнению с соединением хэша состоит в том, что затраты на сортировку могут амортизироваться несколькими соединениями, если эти соединения происходят с теми же атрибутами. Оптимизатор предпочтет соединение со слиянием соединению хэша, если размеры наборов данных будут подобными, или возможно будет амортизировать затраты сортировки для нескольких операций.

Устранение дубликатов

Операция устранения дубликатов выводит набор данных, в котором не содержится дублирующихся строк. Узлы устранения дубликатов вводятся оптимизатором, например, при преобразовании вложенного запроса в соединение.

Для получения дополнительной информации см. раздел "Исключение хэшированных строк" на стр. 321 и раздел "Исключение индексированных строк" на стр. 322.

Исключение хэшированных строк

Алгоритм исключения хэшированных строк читает набор данных и создает в оперативной памяти таблицу хэша. Если строка набора данных найдена в таблице хэша, она игнорируется; в противном случае она записывается в рабочую таблицу. Если набор данных полностью не помещается в таблицу хэша, находящуюся в оперативной памяти, то он делится на меньшие рабочие таблицы и обрабатывается рекурсивно.

Алгоритм исключения хэшированных строк работает очень хорошо, если различные строки помещаются в таблице в оперативной памяти независимо от общего количества строк в наборе данных.

Метод исключения хэшированных строк использует рабочую таблицу и может иметь нечувствительную или чувствительную к параметрам семантику.

Если алгоритм исключения хэшированных строк выполняется в среде с малым количеством доступной кэш-памяти, то он не может выполниться. В этом случае промежуточные результаты хэша с исключением строк сбрасываются, и вместо них используется алгоритм исключения индексированных строк. Оптимизатор старается избежать использования планов доступа, использующих метод исключения хэшированных строк, если обнаруживает, что во время выполнения запроса может возникнуть ситуация нехватки памяти.

Метод исключения хэшированных строк вычисляет весь результат до возвращения первой строки.

Строки набора данных во время работы этого алгоритма блокируются.

Исключение упорядоченных строк

Если набор данных отсортирован по всем столбцам, то может использоваться метод исключения упорядоченных строк. Алгоритм читает каждую строку и сравнивает ее с предыдущей. Если строки совпадают, то новая строка игнорируется; в противном случае она выдается. Исключение упорядоченных строк эффективно, если строки уже отсортированы (например, из-за индекса или соединения со слиянием). Если набор данных не отсортирован, то оптимизатор добавляет сортировку. Сам алгоритм не использует рабочие таблицы, но при добавлении сортировки рабочая таблица будет создана.

Исключение индексированных строк

Алгоритм исключения индексированных строк создает рабочую таблицу с уникальными строками из набора данных. Индекс рабочей таблицы выполняет поиск предварительно введенного дубликата строки набора данных. Если дубликат найден, то строка набора данных игнорируется. В противном случае строка набора данных вставляется в рабочую таблицу. Индекс рабочей таблицы создается для всех столбцов списка SELECT. Для повышения производительности индекса выражение хэша указывается как первое выражение. Это выражение хэша является вычисленным значением, включающим значения всех столбцов в списке SELECT.

Метод исключения индексированных строк возвращает различные строки в порядке чтения. Этот позволяет возвращать первые строки быстрее по сравнению с другими методами исключения строк. Алгоритм исключения индексированных строк во время работы хранит в памяти только строки и может работать на компьютерах с очень малым количеством доступной памяти. Однако если количество различных строк велико, затраты на этот алгоритм больше, чем на алгоритм исключения хэшированных строк. Если рабочая таблица с различными строками не помещается в кэш-памяти, это приведет к многократному считыванию таблицы с диска.

Так как метод исключения индексированных строк использует рабочую таблицу, он не может предоставить полностью чувствительную семантику; однако он также не предоставляет и полностью нечувствительную семантику. Для нечувствительных курсоров требуется другая рабочая таблица.

При использовании алгоритма исключения индексированных строк строки в наборе данных блокируются.

Группирование

Алгоритмы группирования вычисляют резюме на основе набора данных. Они применяются, если запрос содержит раздел GROUP BY, или если в запросе присутствуют агрегатные функции (например, SELECT COUNT(*) FROM T).

Для получения дополнительной информации см. раздел "Группирование хэша" на стр. 323, раздел "Группирование упорядоченных строк" на стр. 323, раздел "Группирование индексированных строк" на стр. 323 и раздел "Группирование по одной строке" на стр. 323.

Группирование хэша

Алгоритм группирования хэша создает в оперативной памяти таблицу хэша строк для группирования. При чтении строк из ввода обновляются строки группы. Если таблица хэша не помещается в памяти, набор данных разделяется на меньшие рабочие таблицы рекурсивно, пока они не поместятся в память. Если памяти не хватает для меньших сегментов, оптимизатор отказывается от сохранения промежуточных результатов группирования хэша и использует алгоритм группирования индексированных строк. Оптимизатор избегает генерации планов доступа с использованием группирования хэша, если возможна ситуация нехватки памяти во время запроса.

Алгоритм группирования хэша работает эффективно, если группы помещаются в память, независимо от размера набора данных.

Алгоритм группирования хэша вычисляет все строки результата до возвращения первой строки и может использоваться с полностью чувствительным или чувствительным к значениям курсором.

Группирование упорядоченных строк

При группировании упорядоченных строк читается отсортированный по столбцам группирования набор данных. Каждая читаемая строка сравнивается с предыдущей. При соответствии столбцов текущая группа обновляется; в противном случае, текущая группа выводится, и начинается чтение новой группы.

Группирование индексированных строк

Алгоритм группирования индексированных строк подобен алгоритму исключения индексированных строк. Он создает рабочую таблицу, содержащую по строке на группу. При чтении строк набора данных производится поиск связанной группы в рабочей таблице при помощи индекса. Агрегатные функции обновляются, и строка группы перезаписывается в рабочей таблице. Если строка группы не найдена, то создается новая группа, и она добавляется в рабочую таблицу.

Алгоритм группирования индексированных строк выбирается, когда размер набора данных очень мал.

Метод группирования индексированных строк возвращает первую строку результата только после вычисления всех строк и завершения ввода. Этот алгоритм может использоваться при требовании полной нечувствительности (FULLY INSENSITIVE).

Группирование по одной строке

Если раздел GROUP BY не определен, то происходит простое объединение по одной строке.

Сортировка и объединения

Алгоритмы сортировки применяются, когда запрос включает раздел группирования, а алгоритмы объединения используются при запросах объединения (union).

Для получения дополнительной информации см. раздел "Сортировка со слиянием" на стр. 324 и раздел "Полное объединение" на стр. 324.

Сортировка со слиянием

Операция сортировки сохраняет набор данных в памяти, сортирует его в памяти и затем выводит результат в рабочую таблицу. Если набор данных полностью не помещается в памяти, то выполняется несколько сортировок, а потом происходит их слияние. Сортировка использует рабочую таблицу и блокирует строки набора данных.

При выполнении алгоритма сортировки со слиянием в среде с малым количеством доступной кэш-памяти возможно, что этот алгоритм не будет завершен. В этом случае полученные промежуточные результаты не сохраняются, и используется метод сортировки на основе индекса. Все строки набора данных читаются и вставляются в рабочую таблицу. Создается индекс столбца, по которому осуществляется сортировка. В этом случае строки читаются из рабочей таблицы с использованием комплексного сканирования индекса. Стратегия на основе индекса значительно медленнее других методов соединения. Оптимизатор старается избежать использования планов доступа, использующих алгоритм сортировки со слиянием, если обнаруживает, что во время выполнения запроса может возникнуть ситуация нехватки памяти. Если из-за малого количества памяти необходимо использоваться алгоритм соединения с вложенными циклами, счетчик производительности увеличивается приращением. Просмотреть этот монитор можно в свойстве QueryLowMemoryStrategy или в счетчике "Query: Low Memory Strategies" системного монитора NT.

На производительность сортировки влияют: размер ключа сортировки, размера строки и полный размер набора данных. Для большинства строк затраты при использовании чувствительного к значениям (VALUES SENSITIVE) курсора будут меньше. В этом случае столбцы в списке SELECT не копируются в рабочие таблицы, используемые при сортировке.

Полное объединение

Алгоритм полного объединения читает строки из наборов данных и выводит их независимо от наличия дубликатов. Этот алгоритм используется для реализации разделов UNION и UNION ALL. В случае UNION алгоритм устранения дубликатов необходим для удаления дубликатов, создаваемых в результате полного объединения.

Другие способы

Далее описаны дополнительные методы, которые могут использоваться в плане доступа.

Фильтр и предфильтр

Условия поиска применяются посредством фильтров. Условия поиска расположены в разделах WHERE или HAVING оператора или, в случае внешних соединений, в разделе ON.

Предфильтр выполняет ту же функцию, что и обычный фильтр, за исключением того, что он не зависит от ввода. Например, в случае `WHERE 1 = 2` применяется предфильтр.

Для получения дополнительной информации см. раздел "Раздел WHERE: определение строк" на стр. 191.

Блокировка

“Блокировка” указывает на существование блокировки при определенном уровне изоляции. Например, при уровне изоляции 1 блокировка поддерживается только для одной строки. На уровне изоляции 0 блокировки не устанавливаются, но узел все равно будет называться Lock (блокировка). В этом случае узел блокировки проверяет существование строки.

Для получения дополнительной информации см. раздел "Принципы блокировки" на стр. 121.

Ограничение количества строк

Ограничения количества строк устанавливаются разделами TOP n или FIRST оператора SELECT.

Для получения дополнительной информации см. раздел "Оператор SELECT" (SELECT statement) на стр. 490 в документе *"Справочник по SQL для ASA" (ASA SQL Reference Manual)*.

Физическая организация данных и доступ к ним

Место размещения в памяти каждой таблицы или элемента влияет на эффективность запросов. Следующие пункты имеют особое значение, потому что каждый из них влияет на время выполнения запросов.

Размещение на диске вставленных строк

Adaptive Server Anywhere размещает строки непрерывно, если это возможно

Новая строка, если она меньше, чем размер страницы файла базы данных, всегда будет размещена на одной странице. Если ни одна из существующих страниц не содержит достаточного пространства для размещения новой строки, Adaptive Server Anywhere создаст строку на новой странице. Например, если новая строка занимает 600 байтов, а на частично заполненной странице доступно только 500 байтов, то Adaptive Server Anywhere поместит строку на новую страницу.

Чтобы размещение на диске страниц таблиц было в большей степени непрерывным, Adaptive Server Anywhere хранит их в блоках из восьми страниц. Например, при создании новой страницы происходит выделение пространства для восьми страниц, страница вставляется в этот блок, и затем блок заполняется следующими семью страницами. Adaptive Server Anywhere использует битовую карту свободных страниц для поиска непрерывного пространства на диске в пределах базы данных (dbspace) и выполняет последовательное сканирование, читая сегменты по 64 КБ с использованием битовой карты для поиска релевантных страниц. Это повышает эффективность последовательного поиска.

Adaptive Server Anywhere может расположить строки в любом порядке

Adaptive Server Anywhere находит свободное пространство на страницах и вставляет строки в том порядке, в котором они были получены. Каждая строка помещается на страницу, но положение в таблице может не соответствовать порядку, в котором строки были вставлены. Например, процессор базы данных создаст новую страницу для размещения длинной строки. Если следующая полученная строка короче, она может быть размещена в свободном пространстве предыдущей страницы.

Строки всех таблиц содержатся неупорядоченно. Если при получении или обработке важен порядок строк, можно использовать раздел ORDER BY оператора SELECT для упорядочивания результата. Приложения, рассчитывающие на упорядоченное расположение строк в таблице, могут выдавать ошибки без предупреждения.

Если часто требуется определенный порядок строк таблицы, следует создать индекс для столбцов, указываемых в разделе ORDER BY запроса.

Пространство не резервируется для столбцов NULL

При добавлении строки Adaptive Server Anywhere резервирует только пространство, необходимое для ее размещения со значениями, которые она содержит при создании. Место для размещения значений NULL не резервируется. Также не резервируется дополнительное пространство для размещения полей, которые могут увеличиться, например, текстовых строк.

После добавления идентификаторы строк не изменяются

После присвоения той или иной позиции на странице строка не может быть перемещена с этой страницы Если обновление строки изменяет ее так, что она больше не помещается на назначенной странице, то происходит разбиение строки, и добавляемая информация располагается на другой странице.

Это заслуживает особого внимания, тем более что Adaptive Server Anywhere не резервирует дополнительного пространства при добавлении строки. Например, предположим, что вставляется большое количество пустых строк в таблицу, а затем эти строки заполняются значениями по одному столбцу за раз с использованием операторов обновления. Почти каждое значения строки будет размещено на отдельной странице. Для получения всех значений одной строки процессору базы данных придется прочитать несколько страниц. Эта простая операция будет проходить чрезвычайно медленно.

Необходимо учитывать заполнение новых строк данными во время вставки. После вставки они должны иметь достаточное количество места для данных, которые требуется сохранить.

Файл базы данных не сжимается

При добавлении и удалении строк базы данных Adaptive Server Anywhere многократно использует занимаемое пространство. Таким образом, Adaptive Server Anywhere может поместить новую строку на место, прежде занятое другой строкой.

Adaptive Server Anywhere ведет учет количества пустого места на каждой странице. При вставке новой строки сначала происходит поиск требуемого пространства на существующих страницах. Если Adaptive Server Anywhere находит достаточное количество свободного места на существующей странице, то на этой странице размещается новая строка, изменяя при необходимости содержание страницы. В противном случае создается новая страница.

Если удаляется большое количество строк и не добавляются новые, использующие свободное пространство строки, информация в базе данных может стать разреженной. Можно перезагрузить таблицу или использовать оператор REORGANIZE TABLE для дефрагментации таблицы.

☞ Для получения дополнительной информации см. раздел "Оператор REORGANIZE TABLE" (REORGANIZE TABLE statement) на стр. 472 в документе *"Справочник по SQL для ASA" (ASA SQL Reference Manual)*.

Размер таблицы и страницы

Размер страницы, выбранный для базы данных, может повлиять на ее производительность. При меньшем размере страницы быстрее будут выполняться операции, которые получают относительно небольшое количество строк из разных мест. При большем размере страницы запросы, выполняющие последовательное сканирование таблицы, работают быстрее, особенно когда строки размещены на страницах в порядке индекса. В этом случае, когда страница помещается в память для получения значения одной строки, могут быть загружены данные из следующих строк. Обычно диски физически устроены так, что получение меньшего количество больших блоков эффективнее, чем большого числа маленьких.

Adaptive Server Anywhere создает битовую карту для достаточно больших таблиц баз данных с размером страницы более 2 КБ. Каждая битовая карта таблицы показывает размещение всех страниц таблицы в файле базы данных. Для баз данных с размером страниц 2 КБ, 4 КБ или 8 КБ сервер использует битовую карту для чтения сразу больших блоков (64 КБ) таблицы вместо отдельных страниц, сокращая таким образом количество дисковых операций ввода/вывода и улучшая производительность. Пользователи не могут управлять критериями, используемыми сервером при создании или использовании битовой карты.

Заметьте, что битовые карты (также называемые картами страницы) доступны только для баз данных, созданных в версии 8.0 и выше. Если база данных модернизирована из старой версии, сервер не будет создавать битовые карты для таблиц базы данных даже при соответствии критериям создания. Битовые карты не создаются для рабочих и системных таблиц.

При выборе большего размера страницы, например, 4 КБ, следует увеличить размер кэша. В то же пространство может поместиться меньшее число больших страниц. Например, 1 МБ памяти хватит для размещения 1000 страниц размером в 1 КБ или 250 страниц размером 4 КБ. Необходимое количество кэшируемых страниц зависит только от базы данных и характера выполняемых запросов. Можно выполнить проверку производительности с различным размером кэша. Если в кэш не помещается достаточное количество страниц, производительность падает, поскольку Adaptive Server Anywhere применяет свопинг часто используемых страниц на диск.

Размер страницы влияет на индексы. По умолчанию страницы индексов имеют размер хэша 10 байтов. В них размещается приблизительно первые 10 байтов данных каждой записи индекса. Этого достаточно примерно для 200 разветвлений при использовании 4 КБ страниц, то есть каждая страница индексов хранит 200 строк или 40 000 строк при индексе второго уровня. Каждый новый уровень индекса увеличивает это число в 200 раз. Размер страницы может значительно повлиять на разветвление, требуя большую глубину индекса. Размер страницы больших баз данных должен составлять 4 КБ.

Adaptive Server Anywhere стремится к полному заполнению страниц. Свободное пространство увеличивается, только когда новые объекты не помещаются в незанятое пространство существующих страниц. Следовательно, настройка размера страниц не повлияет на полный размер базы данных.

Индексы

Индексы могут значительно повысить производительность поиска по индексированным столбцам, но они занимают место в базе данных и замедляют операции вставки, обновления и удаления. В этом разделе описано, когда следует создавать индекс, и как достичь максимальной эффективности индекса.

Существует много ситуаций, когда создание индекса повышает производительность базы данных. Индекс упорядочивает строки таблицы по значениям в нескольких или во всех столбцах. С помощью индекса Adaptive Server Anywhere быстро находит строки. Больший параллелизм достигается при ограничении числа открытых страниц базы данных. Индекс Adaptive Server Anywhere является удобным средством установки контроля за уникальностью строк в таблице.

Вопросы необходимости создания индекса

Простой формулы для определения необходимости создания индекса для того или иного столбца не существует. Следует рассмотреть преимущества индексированного чтения и затраты по обслуживанию этого индекса. Следующие факторы помогут определить необходимость создания индекса.

- **Ключи и уникальные столбцы.** Adaptive Server Anywhere автоматически создает индексы для первичных ключей, внешних ключей и уникальных столбцов. Дополнительные индексы этих столбцов создавать не следует. Исключением являются составные ключи, которые иногда расширяются с помощью дополнительных индексов.

Для получения дополнительной информации см. раздел "Составные индексы" на стр. 331.

- **Частота выполнения поиска.** Если поиск по определенному столбцу выполняется часто, то при создании индекса этого столбца производительность запросов возрастет. Создание индекса столбца, по которому поиск происходит редко, приведет к лишним затратам ресурсов.
- **Размер таблицы.** Индексы в относительно больших таблицах дают большие преимущества, чем индексы в относительно маленьких таблицах. Например, использование индекса в таблице с 20 строками не будет эффективнее последовательного сканирования.

- **Количество обновлений**. Индекс обновляется при вставке или удалении строки таблицы и обновлении индексированного столбца. Индекс столбца увеличивает время выполнения операций вставки, обновления и удаления. Часто обновляемая база данных должна иметь меньше индексов, чем база, доступная только для чтения.
- **Анализ пространства.** Индексы занимают много места в базе данных. Если размер базы данных является важным критерием, индексы следует создавать экономно.
- **Распределение данных.** Если поиск с помощью индекса возвращает слишком много значений, это может потребовать больше ресурсов, чем последовательное сканирование. В таких случаях Adaptive Server Anywhere не использует индекс. Например, Adaptive Server Anywhere не будет использовать индекс столбца, который содержит только два значения, например, столбец employee.sex в демонстрационной базе данных. Поэтому создавать индекс столбца, который имеет только несколько различных значений, не нужно.

Временные таблицы

Индексы могут быть созданы и в локальных, и в глобальных временных таблицах. Следует создавать индекс временной таблицы, если предполагается, что эта таблица будет большого размера, и обращение к ней будет многократным в отсортированном порядке или в соединении. В противном случае, возможно, повышение производительности запросов не будет достигнуто вследствие слишком высоких затрат на создание и удаление индекса.

Для получения дополнительной информации см. раздел "Работа с индексами" на стр. 59

Повышение эффективности индексов

Если эффективность индексов не соответствует ожиданиям, можно выполнить следующие действия.

- Реорганизуйте составные индексы.
- Увеличьте размер страницы.

Эти меры направлены на увеличение выборочности индекса и разветвление индекса, как указано ниже.

Выборочность индекса

Выборочность индекса определяет способность индекса определить местонахождение требуемой записи без чтения дополнительных данных.

При низкой выборочности необходимо извлечение дополнительной информации со страницы таблицы, на которую указывает индекс. Такое извлечение называется **полным сравнением**. Оно негативно отражается на эффективности индекса.

Свойство *FullCompare* отслеживает количество выполненных полных сравнений. Эти статистические данные можно просмотреть, используя системный монитор Sybase Central или системный монитор Windows NT.

Количество полных сравнений также представлено в графическом плане со статистикой. Для получения дополнительной информации см. раздел "Общая статистика, используемая в плане" на стр. 354

Для получения дополнительной информации о функции FullCompare см. раздел "Свойства на уровне базы данных" (Database-level properties) на стр. 612 в документе "Руководство по администрированию баз данных ASA" (ASA Database Administration Guide).

Структура индекса и коэффициент разветвления индекса

Подобно дереву, индексы делятся на множество уровней. Первая страница индекса, называемая корневой страницей, содержит ссылки на одну или более страниц следующего подуровня, и каждая из этих страницу - переходы на страницы более низкого уровня, пока не будет достигнут самый низкий уровень индекса. Страницы индексов самого низкого уровня называются базовыми страницами. Для определения местонахождения определенной строки индекс с количеством уровней n требует чтения n страниц индексов и одной страницы данных, которая содержит требуемую строку. С диска требуется прочитать меньше чем n страниц, так как страницы индексов обычно размещаются в кэше.

Коэффициент разветвления индекса - это число записей индекса, размещенных на странице. Индекс с большим разветвлением имеет меньше уровней, чем индекс с меньшим разветвлением. Индекс с более высоким коэффициентом разветвления обычно имеет большую эффективность.

Можно посмотреть количество уровней индекса, используя системную процедуру *sa_index_levels*.

Для получения дополнительной информации см. раздел "Системная процедура sa_index_levels" на стр. 655 в документе *"Справочник по SQL для ASA" (ASA SQL Reference Manual)*.

Составные индексы

Индекс может содержать один, два или большее количество полей. Индекс по полям двух и более полям называют составным индексом. Следующий оператор создает составной индекс по двум полям:

```
CREATE INDEX name
ON employee (emp_lname, emp_fname)
```

Составной индекс используется, когда первый столбец не предоставляет высокую выборочность. Например, составной индекс по *emp_lname* (имя служащего) и *emp_fname* (фамилия служащего) применяется, когда несколько служащих имеют одну фамилию. Составной индекс по *emp_id* и *emp_lname* может оказаться ненужным, потому что у каждого служащего есть уникальный код. Поэтому столбец *emp_lname* не предоставляет дополнительной выборочности.

Дополнительные столбцы индекса позволяют сузить поиск, но индекс по двум столбцам - это не то же самое, что два отдельные индекса. Структура составного индекса подобна телефонной книге, в которой записи отсортированы по фамилиям, а затем люди с одной фамилией - по именам. Телефонная книга полезна, если известна фамилия, еще более полезна, если известны фамилия и имя, но совершенно бесполезна, если известно только имя.

Сжатый метод индексации в виде B-дерева, появившийся в 8 версии Adaptive Server Anywhere, существенно улучшает производительность составных индексов.

Порядок полей

При создании составных индексов следует тщательно выбирать порядок столбцов. Составные индексы используются для поиска по всем полям индекса или только по первым полям; но невозможно выполнить поиск только по последним полям составного индекса.

Если по одному столбцу поиск выполняется многократно, то этот столбец логично сделать первым в составном индексе. Если обрабатывается много запросов по двум столбцам отдельно, следует создать второй индекс только по второму столбцу.

Первичные ключи, которые содержат более одного столбца, автоматически индексируются как составные индексы с таким порядком столбцов, с каким они определены в таблице, а не в том порядке, в котором они определены в первичном ключе. Следует определить, какие запросы будут выполняться с привлечением первичного ключа для выбора первого столбца. Добавьте дополнительные индексы по столбцам первичного ключа, часто используемым для поиска.

Например, создан составной индекс по двум столбцам. Один столбец содержит имена служащих, другой - их фамилии. Можно создать индекс, в котором указывается сначала имя, а потом фамилия. В качестве альтернативы, можно создать индекс, в котором указывается сначала фамилия, а потом имя. Хотя эти два индекса организуют информацию в обоих столбцах, они имеют различные функции.

```
CREATE INDEX fname_lname
ON employee emp_fname, emp_lname;
CREATE INDEX lname_fname
ON employee emp_lname, emp_fname;
```

Предположим, требуется найти имя Джон. Можно использовать только индекс, содержащий имя в первом столбце индекса. Индекс, содержащий сначала фамилию, потом имя, бесполезен, потому что служащий с именем Джон может быть в любом месте индекса.

Если требуется поиск людей только по имени или по фамилии, то следует создать два индекса.

В качестве альтернативы, можно создать два индекса, каждый из которых содержит только один из столбцов. Однако Adaptive Server Anywhere использует только один индекс для обращения к любой таблице при обработке одного запроса. Даже при указании имени и фамилии, возможно, что Adaptive Server Anywhere будет читать дополнительные строки, пытаясь найти строку с правильной фамилией.

При создании индекса с помощью команды CREATE INDEX, как в примере выше, столбцы появляются в порядке, указанном в команде.

Индексы первичного ключа и порядок столбцов

Порядок столбцов в индексе первичного ключа совпадает с порядком, в котором столбцы расположены при определении таблицы, независимо от расположения столбцов в порядке, указанном в ограничении PRIMARY KEY. Более того, Adaptive Server Anywhere создает дополнительное ограничение. Столбцы первичного ключа таблицы должны находиться в начале каждой строки. Таким образом, при добавлении первичного ключа к существующей таблице сервер может перезаписать всю таблицу, чтобы расположить ключевые столбцы в начале каждой строки.

В случаях, когда первичный ключ содержит более одного столбца, следует учитывать необходимые типы поиска. Следует установить порядок столбцов в определении таблицы так, чтобы часто используемые для поиска столбцы оказались первыми, или создать отдельные индексы, как это требуется для других столбцов.

Составные индексы и ORDER BY

По умолчанию столбцы индекса отсортированы по возрастанию, но при необходимости они могут быть отсортированы в порядке убывания, что определяется указанием DESC в операторе CREATE INDEX.

Adaptive Server Anywhere может использовать индекс для оптимизации запроса ORDER BY, если раздел ORDER BY содержит только включенные в этот индекс столбцы. Кроме того, столбцы индекса должны быть упорядочены точно так же или точно противоположным способом, как указано в разделе ORDER BY. Для индексов по одному столбцу упорядочение всегда такое, что его можно оптимизировать, но оптимизация составных индексов гораздо сложнее. Таблица, приведенная ниже, показывает возможности индекса с двумя столбцами.

Столбцы индекса	Оптимизируемые запросы	Неоптимизируемые запросы
	ORDER BY	ORDER BY
ASC, ASC	ASC, ASC или DESC, DESC	ASC, DESC или DESC, ASC
ASC, DESC	ASC, DESC или DESC, ASC	ASC, ASC или DESC, DESC
DESC, ASC	DESC, ASC или ASC, DESC	ASC, ASC или DESC, DESC
DESC, DESC	DESC, DESC или ASC, ASC	ASC, DESC или DESC, ASC

Индексы с большим количеством столбцов подчиняются этому же правилу. Например, предположим, что существует следующий индекс:

```
CREATE INDEX idx_example
ON table1 (col1 ASC, col2 DESC, col3 ASC)
```

В этом случае могут быть оптимизированы следующие запросы:

```
SELECT col1, col2, col3 from table1 ORDER BY col1 ASC,
col2 DESC, col3 ASC
```

и

```
SELECT col1, col2, col3 from example
ORDER BY col1 DESC, col2 ASC, col3 DESC
```

Индекс не используется при оптимизации запросов с другими наборами ASC и DESC в разделе ORDER BY. Например:

```
SELECT col1, col2, col3 from table1
ORDER BY col1 ASC, col2 ASC, col3 ASC
```

не оптимизируется.

Другие области применения индексов

Adaptive Server Anywhere использует индексы для повышения производительности другими способами. Индекс позволяет Adaptive Server Anywhere обеспечивать уникальность столбца, уменьшать количество блокированных строк и страниц и точнее оценивать выборочность предиката.

- **Обеспечение уникальности столбца.** Без использования индекса для обеспечения уникальности столбца Adaptive Server Anywhere должен просмотреть всю таблицу при вставке каждого значения. Поэтому Adaptive Server Anywhere автоматически создает индекс по каждому столбцу с ограничением для обеспечения уникальности.
- **Уменьшение количества блокировок.** Индексы уменьшают количество строк и страниц, которые блокируются при операциях вставки, обновления и удаления. Это является результатом упорядочения, которое индексы налагают на таблицу.

  Для получения дополнительной информации об индексах и блокировках см. раздел "Принципы блокировки" на стр. 121.
- **Оценка выборочности.** Поскольку индекс упорядочен, оптимизатор оценивает процент значений, удовлетворяющих данному запросу, сканируя верхние уровни индекса. Это называется частичным сканированием индекса.

Типы индексов

Adaptive Server Anywhere поддерживает два типа индекса и автоматически выбирает один из них в зависимости от установленной ширины индексируемых столбцов. При полной ширине столбца меньше 10 байтов Adaptive Server Anywhere используют индекс типа Б-дерева, который содержит кодирование с сохранением порядка или значение хэша, которое представляет индексированные данные. Индексы хэша типа Б-дерева используются при длине ключа индекса, превышающей либо одну восьмую размера страницы для базы данных, либо 512 байтов. Для данных с общей длинной, находящейся между этими двумя пределами, Adaptive Server Anywhere использует сжатый индекс типа Б-дерева, который содержит каждый ключ в сжатой форме.

Индексы хэша типа Б-дерева

Индекс хэша типа Б-дерева не содержит фактического значения(ий) строк таблицы. Вместо этого индекс хэша Б-дерева хранит кодирование с порядком оригинальных данных. Количество байтов каждой записи индекса, используемых для размещения этого значения хэша, называется размером хэша и автоматически выбирается сервером на основании заданной ширины всех индексированных столбцов. Сервер сравнивает эти хэшированные значения при поиске по индексу определенной строки.

При индексировании типа данных, занимающего малый объем памяти, например, целых чисел, значение хэша, создаваемое Adaptive Server Anywhere, занимает столько же памяти, как и первоначальное значение. Например, значение хэша для целого числа занимает 4 байта, и такой же объем памяти требуется для размещения самого целого числа. Поскольку значение хэша имеет тот же размер, Adaptive Server Anywhere может использовать значение, идентичное с фактическим. Adaptive Server Anywhere всегда может определить, равны ли два значения, или какое из них больше, сравнивая их значения хэша. Однако фактическое значение может быть получено только при чтении строки из соответствующей таблицы.

При индексировании столбца, содержащего тип данных большого размера, значение хэша будет меньше размера этого типа. Например, при индексировании столбца со строками максимальный объем значения хэша составляет 9 байтов.

Следовательно, Adaptive Server Anywhere не всегда может сравнить две строки, используя только значения хэша. Если значения хэша равны, Adaptive Server Anywhere сравнивает фактические значения из соответствующих таблиц.

Значения хэша

Adaptive Server Anywhere должен представить значения в индексе для определения порядка их расположения. Например, при индексировании столбца имен он должен знать, что имя Amos должно находиться перед Smith.

Для каждого значения индекса Adaptive Server Anywhere создает соответствующее значение хэша. В индексе сохраняется значение хэша, а не фактическое значение. Adaptive Server Anywhere может выполнять операции со значением хэша. Например, может определить равенство двух значений или большее из двух значений.

При индексировании типа данных, занимающего малый объем памяти, например, целых чисел, значение хэша, создаваемое Adaptive Server Anywhere, занимает столько же памяти, как и первоначальное значение. Например, значение хэша для целого числа занимает 4 байта, такой же объем памяти требуется для размещения самого целого числа. Поскольку значение хэша имеет такой же размер, Adaptive Server Anywhere в качестве значения хэша может использовать фактическое значение. Adaptive Server Anywhere всегда может определить равенство двух значений или большее из них, сравнивая их значения хэша. Однако фактическое значение может быть получено только при чтении строки из соответствующей таблицы.

При индексировании столбца с типом данных, занимающим большое количество памяти, значение хэша занимает меньше места. Например, при индексировании столбца со строками максимальный объем значение хэша составляет 9 байтов.

Следовательно, Adaptive Server Anywhere не всегда может сравнить две строки, используя только значение хэша. При равенстве значений хэша Adaptive Server Anywhere читает и сравнивает фактические значения из таблицы.

Например, индексируются заголовки документов, многие из которых подобны. При поиске определенного заголовка индекс идентифицирует только набор возможных строк. В этом случае Adaptive Server Anywhere должен прочитать все найденные строки и сравнить их с полным заголовком.

Составные индексы

Упорядоченную последовательность столбцов называют составным индексом. Однако каждый ключ индекса в этих индексах состоит максимум из 9-байтового значения хэша. Следовательно, значение хэша не обязательно идентифицирует только нужную строку. При равенстве двух значений хэша Adaptive Server Anywhere должен прочитать и сравнить фактические значения.

Сжатые индексы типа Б-дерева

Сжатые индексы типа Б-дерева содержат сжатую форму каждого индексированного значения во внутренних узлах индекса. Сжатые индексы типа Б-дерева хранят значения, используя деревья цифрового поиска - оптимизированную форму структуры данных в виде дерева, дополненную счетчиком пропуска для сжатия. В результате при больших общих объемах данных сжатые индексы типа Б-дерева имеют значительное преимущество по сравнению с индексами хэша. Более существенно, что алгоритм уплотнения эффективно обрабатывает идентичные (или почти идентичные) значения индекса, так что общие подстроки значений влияют на требования к памяти и производительности незначительно. Сжатые индексы типа Б-дерева выбираются автоматически, если сумма объявленной ширины индексированных столбцов находится между 10 байтами и одной восьмой размера страницы базы данных и максимумом 512 байтами.

Рекомендованные размеры страницы

Размер страницы базы данных существенно влияет на степень разветвленности индекса. Коэффициент разветвления индекса примерно удваивается при увеличении размера страницы в два раза.

При поиске по индексу каждый раз происходит чтение одной страницы для каждого уровня индекса и одной страницы из таблицы. Для выполнения одного запроса может потребоваться просмотреть индекс нескольких тысяч раз. При высоком коэффициенте разветвления требуется прочитать меньшее количество уровней индекса, что может значительно увеличить скорость поиск. Поэтому в целях повышения эффективности индексов рекомендуется использовать страницы большого размера (например, 4 КБ) для улучшения. Использование страниц большего размера требуется при необходимости индексации длинных строчных столбцов с использованием сжатых индексов типа Б-дерева, а ограничение длины на страницах меньшего размера не позволяет их создать.

Семантические преобразования запроса

Для повышения эффективности Adaptive Server Anywhere обычно перезаписывает запрос в новую форму, выполняя это иногда в несколько этапов. Таким образом обеспечивается то, что новая версия запроса получает тот же результат. Другими словами, Adaptive Server Anywhere перезаписывает запросы в другой синтаксической форме с сохранением семантической эквивалентности.

Adaptive Server Anywhere выполняет множество различных операций перезаписи. При обращении к планам доступа может выясниться, что они не соответствуют буквальной интерпретации первоначального оператора. Например, оптимизатор стремится перезаписывать подзапросы так, чтобы они содержали соединения (насколько это возможно). Эта способность оптимизатора переписывать вводимые пользователем операторы SQL, а также методы такой перезаписи, очень важны.

Пример

В отличие от определений языка SQL, некоторые языки устанавливают строгое выполнение операций AND и OR. Некоторые обеспечивают первоначальное выполнение условия с левой стороны. Если при этом может быть определен результат всего выражения, то условие с правой стороны компилятором выполняться не будет.

Это соглашение позволяет объединять условия, которые в противном случае потребовали бы наличия двух вложенных операторов IF. Например, на языке C можно проверить, является ли указатель NULL (неопределенным), до использования его следующим образом. Можно заменить вложенные условия

```
if ( X != NULL ) {
   if ( X->var != 0 ) {
... операторы ...
}
}
```

более компактным выражением

```
if ( X != NULL && X->var != 0 ) {
      ... операторы ...
}
```

В отличие от C, в SQL нет таких правил порядка выполнения. Adaptive Server Anywhere выбирает наиболее подходящий порядок выполнения таких условий. Форма с измененным порядком остается семантически эквивалентной, так как в языке SQL не предусмотрено это различение. В частности, оптимизаторы запроса могут любым образом переупорядочить предикаты в разделах WHERE, HAVING и ON.

Анализ предиката

Предикат - это условное выражение, объединенное логическими операторами AND и OR, которое представляет собой набор условий в разделах WHERE, HAVING или ON. В SQL предикат, определенный как UNKNOWN, интерпретируется как FALSE.

О предикате, использующем индекс для получения строк из таблицы, говорят, что он **позволяет использовать аргументы поиска** (sargable). Название "*sargable*" происходит от фразы "*search argument able*", означающей возможность осуществления поиска аргумента.

Предикаты, включающие сравнение значений столбца с константами, другими столбцами или выражениями, могут позволить использовать аргументы поиска.

В следующем операторе предикат позволяет использовать аргументы поиска. Adaptive Server Anywhere эффективно выполняет этот предикат, используя первичный индекс таблицы служащих.

```
SELECT *
FROM employee
WHERE employee.emp_id = 123
employee<employee>
```

Напротив, тип следующего предиката не позволяет ему использовать аргументы поиска. Хотя столбец *emp_id* имеет первичный индекс, использование этого индекса не ускорит вычисление, потому что в результате содержатся все, или все за исключением одной, строки.

```
SELECT *
FROM employee
where employee.emp_id <> 123
employee<seq>
```

Использование индекса также не принесет результата при поиске всех служащих, имя которых *заканчивается* на "k". Единственный способ вычисления результата заключается в просмотре всех строк отдельно.

Функции

Если предикат содержит функцию для имени столбца, то он не разрешает использование аргументов поиска. Например, индекс не будет использоваться при следующем запросе:

```
SELECT * from sales_order
WHERE year(order_date)='2000'
```

Иногда можно перестроить запрос так, чтобы он не содержал функций, что позволит запросу использовать аргументы поиска. Например, вышеуказанный запрос можно изменить:

```
SELECT * from sales_order
WHERE order_date > '1999-12-31'
AND order_date < '2001-01-01'
```

Запрос, использующий функцию, может использовать аргументы поиска, если поместить значения, вычисляемые функцией, в столбец и создать индекс этого столбца. **Вычисляемый столбец** - это столбец, значения которого получены из других столбцов таблицы. Например, если есть столбец *order_date*, который хранит дату заказа, можно создать вычисляемый столбец *order_year*, который будет содержать значения года, извлеченные из столбца *order_date*.

```
ALTER TABLE sales_order
ADD order_year INTEGER
COMPUTE year(order_date)
```

Можно добавить индекс столбца *order_year* обычным способом:

```
CREATE INDEX idx_year
ON sales_order (order_year)
```

При выполнении следующего оператора

```
SELECT * from sales_order
WHERE year(order_date) = '2000'
```

сервер определяет наличие индексированного столбца, содержащего требуемую информацию, и использует индекс для обработки запроса.

Домен вычисляемого столбца должен быть эквивалентен домену выражения COMPUTE для осуществления подстановки в столбец. Если бы функция year(order_date) в вышеуказанном примере возвращала строку вместо целого числа, оптимизатор не подставил бы в вычисляемый столбец выражение, и, следовательно, индекс *idx_year* не мог бы использоваться для получения требуемых строк.

Для получения дополнительной информации о вычисляемых столбцах см. раздел "Использование вычисляемых столбцов с Java-классами" на стр. 124 в документе *"Руководство по программированию ASA" (ASA Programming Guide)*.

Примеры

В этих примерах атрибуты *x* и *y* - столбцы отдельной таблицы. Атрибут z содержится в отдельной таблице. Предположим, что индекс существует для каждого из этих атрибутов.

Может использовать аргументы поиска	Не может использовать аргументы поиска
x = 10	x <> 10
x IS NULL	x IS NOT NULL
x> 25	x = 4 OR y = 5
x = z	x = y
x IN (4, 5, 6)	x NOT IN (4, 5, 6)
x LIKE 'pat%	'x LIKE '%tern'
x = 20 - 2	x + 2 = 20

Иногда неочевидно, может ли предикат использовать аргументы поиска. В этих случаях можно переписать предикат так, чтобы получить возможность использования аргументов поиска. Во всех примерах можно изменить предикат x LIKE 'pat%' учитывая, что "u" - следующий символом в алфавите после "t": *x*> = 'pat' и *x* < 'pau'. В этой форме индекс атрибута x помогает найти значения в ограниченной области. К счастью, Adaptive Server Anywhere выполняет это преобразование автоматически.

Предикат, способный использовать аргументы поиска и используемый для индексированного поиска в таблице, является **совпадающим** предикатом. Раздел WHERE может содержать множество совпадающих предикатов. Какой из них наиболее подходящий, зависит от стратегии соединения. Оптимизатор пересматривает выбор совпадающего предиката при рассмотрении разных стратегий соединения.

Типы семантических преобразований

С целью нахождения наиболее эффективных и удобных представлений запроса оптимизатор может выполнять множество преобразований. После выполнения этих преобразований план доступа может сильно отличаться от буквальной интерпретации первоначального запроса. Общие изменения включают следующие операции:

- устранение ненужных условий DISTINCT;
- расформирование вложенных подзапросов;
- включение предиката в представления с UNION или GROUP;
- исключение соединений;
- оптимизация функций поиска минимума или максимума;
- оптимизация OR и списка IN;
- оптимизация LIKE;
- преобразование внешних соединений во внутренние;
- обнаружение применимых условий на основе заключений из предиката;
- удаление ненужного перевода регистра;

Все эти операции рассматриваются в следующих подразделах.

Устранение ненужных условий DISTINCT

Иногда DISTINCT условие не является необходимым. Например, свойства одного или более столбцов результата могут явно или неявно содержать условие UNIQUE, потому что этот столбец фактически является первичным ключом.

Примеры

Ключевое слово *distinct* в следующем операторе не нужно, потому что таблица продуктов содержит первичный ключ *p.id* в результирующем наборе.

```
SELECT DISTINCT p.id, p.quantity
FROM product p
p <seq>
```

На самом деле сервер базы данных выполняет семантически эквивалентный запрос:

```
SELECT p.id, p.quantity
FROM product p
```

Точно так же результат следующего запроса содержит первичные ключи обеих таблиц, поэтому все строки результата будут различными.

```
SELECT DISTINCT *
FROM sales_order o JOIN customer c
   ON o.cust_id = c.id
WHERE c.state = 'NY'
```

c<seq> JNL o<ix_sales_cust>

Расформирование вложенных подзапросов

Можно создавать операторы с вложенными запросами, используя удобный синтаксис языка SQL. Однако замена вложенных запросов соединениями часто ведет к более эффективному выполнению и лучшей оптимизации, так как Adaptive Server Anywhere может использовать условия с высокой выборочностью в разделе подзапроса WHERE.

Примеры

В следующем примере подзапрос может выбрать только по одной строке для каждой строки внешнего блока. Так как может быть найдено только одно соответствие строк, Adaptive Server Anywhere преобразовывает этот запрос во внутреннее соединение.

```
SELECT s.*
FROM sales_order_items s
WHERE EXISTS
 ( SELECT *
   FROM product p
WHERE s.prod_id = p.id
   AND p.id = 300 AND p.quantity> 300)
```

При преобразовании тот же оператор выражен внутренне с использованием соединения:

```
SELECT s.*
FROM product p JOIN sales_order_items s
   ON p.id = s.prod_id
WHERE p.id = 300 AND p.quantity > 20
```

p<seq> JNL s<ky_prod_id>

Аналогично, следующий запрос содержит связывающий предикат EXISTS в подзапросе. Этот подзапрос выбирает более одной строки.

```
SELECT p.*
FROM product p
WHERE EXISTS
    ( SELECT *
      FROM sales_order_items s
      WHERE s.prod_id = p.id
         AND s.id = 2001)
```

Adaptive Server Anywhere преобразовывает этот запрос во внутреннее соединение с условием DISTINCT в списке SELECT.

```
SELECT DISTINCT p.*
FROM product p JOIN sales_order_items s
   ON p.id = s.prod_id
WHERE s.id = 2001
```

```
Distl[ s<id_fk> JNL p<product> ]
```

Adaptive Server Anywhere может удалять подзапросы в сравнениях, если подзапрос может выбрать не более одной строки для каждой строки внешнего блока. В следующем примере показан такой запрос.

```
SELECT *
FROM product p
WHERE p.id =
    ( SELECT s.prod_id
      FROM sales_order_items s
      WHERE s.id = 2001
         AND s.line_id = 1)
```

Adaptive Server Anywhere перестраивает этот запрос следующим образом.

```
SELECT p.*
FROM product p, sales_order_items s
WHERE p.id = s.prod_id
   AND s.id = 2001
   AND s.line_id = 1
```

```
p<seq> JNL s<sales_order_items>
```

Включение предиката в представления с GROUP или UNION

Обычно запросы ограничивают результаты представления, чтобы получить только несколько определенные записи. Наличие GROUP BY или UNION в представлении предпочтительнее для сервера, так как при этом необходимо вычислять результат только для требуемых строк.

Пример

Предположим, представление product_summary определено следующим образом:

```
CREATE VIEW product_summary( product_id, num_orders,
total_qty) as
SELECT prod_id, count(*), sum( quantity )
FROM sales_order_items
GROUP BY prod_id
```

Оно возвращает для каждого продукта число заказов, в которых присутствует этот продукт, а также общее количество заказанных продуктов. Теперь рассмотрим запрос по этому представлению:

```
SELECT *
FROM product_summary
WHERE product_id = 300
```

Этот запрос ограничивает вывод только для продукта с кодом 300. Запрос и запрос представления можно объединить в один семантически эквивалентный оператор SELECT, а именно:

```
SELECT prod_id, count(*), sum( quantity )
FROM sales_order_items
GROUP BY prod_id
HAVING prod_id = 300.
```

Простое выполнение плана этого запроса потребовало бы вычисления агрегатов для всех продуктов, а затем выбор только единственной строки с кодом продукта 300. Однако предикат HAVING в столбце *product_id* может быть помещен в раздел запроса WHERE, так как это столбец группирования:

```
SELECT prod_id, count(*), sum( quantity )
FROM sales_order_items
WHERE prod_id = 300
GROUP BY prod_id
```

Это значительно уменьшит объем требуемых вычислений. Если этот предикат обладает высокой выборочностью, оптимизатор может использовать индекс по *prod_id* для быстрого получения строк с продуктом 300 вместо последовательного сканирования таблицы *sales_order_items*.

Такая же оптимизация используется для представлений, использующих UNION или UNION ALL.

Исключение соединений

Оптимизация посредством перезаписи с исключением соединений уменьшает количество соединений в запросе путем удаления таблиц из запроса там, где это не приведет к потере данных. Этот тип оптимизации используется, если запрос содержит соединение по первичному и внешнему ключу, и в запросе есть ссылки только на столбцы первичного ключа в исходной таблице.

Пример

Например:

```
SELECT s.id, s.line_id, p.id
FROM sales_order_items s KEY JOIN product p
был бы перестроен как
SELECT s.id, s.line_id, s.prod_id
FROM sales_order_items s
WHERE s.prod_id IS NOT NULL
```

Второй запрос семантически эквивалентен первому, потому что любая строка из таблицы *sales_order_items* с внешним ключом NULL для продуктов не появится в результате.

Оптимизация посредством исключения соединений применяется к таблицам, участвующим во внешних соединениях, но условия, при которых этот тип оптимизации допустим, намного сложнее. При соответствующих условиях таблицы, участвующие в соединениях первичного ключа, также могут быть устранены.

Следует помнить, что при использовании этого типа оптимизации результат операции DESCRIBE может отличаться от ожидаемого из-за замещения столбцов. Кроме того, может возникнуть ошибка операторов UPDATE или DELETE WHERE CURRENT, если оператор обновления ссылается на один или более исключенных базовых таблиц. Для решения этой проблемы убедитесь, что дополнительные столбцы из исключенной таблицы присутствуют в списке SELECT запроса (во избежание немедленной оптимизации), или для обновления строк используется отдельный оператор.

Оптимизация функций поиска минимума или максимума

При оптимизации функций минимума/максимума производится попытка использовать существующий индекс для повышения эффективности вычисления результата агрегатного запроса, использующего функции MAX() или MIN(). Цель этой оптимизации состоит в вычислении результата при поиске по индексу отдельной строки. Для применения этого типа оптимизации запрос:

- не должен содержать раздел GROUP BY;
- должен быть для одной таблице;
- не может содержать ничего, кроме связывающих условий равенства в разделе WHERE;
- должен содержать только одну агрегатную функцию (MAX или MIN) в списке SELECT.

Пример

Для иллюстрации этой оптимизации, предположим, что индекс *prod_qty* по (*prod_id ASC, quantity ASC*) существует в таблице *sales_order_items*. Тогда запрос

```
SELECT MIN( quantity )
FROM sales_order_items
Where prod_id = 300
```

будет перестроен как

```
SELECT MIN( quantity )
FROM ( SELECT FIRST quantity
       FROM sales_order_items
       WHERE prod_id = 300 and quantity IS NOT NULL
       ORDER BY prod_id ASC, quantity ASC ) as
s(quantity)
```

Этот запрос формально недопустим из-за присутствия раздела ORDER BY в выражении производной таблицы. Предупреждение NULL_VALUE_ELIMINATED не генерируется для составных запросов при применении оптимизации.

План доступа (в краткой форме) для перестроенного запроса выглядит следующим образом:

```
GrByS[ RL[sales_order_items<prod_qty> ] ]
```

Оптимизация списка IN

Оптимизатор Adaptive Server Anywhere поддерживает специальную оптимизацию для использования предикатов IN в индексированных столбцах. Эта оптимизация применяется одинаково к нескольким предикатам одного индексированного столбца, которые соединяются с помощью OR, так как это семантически эквивалентно. Оптимизация применяется, если список IN содержит только константы.

Когда оптимизатор встречает предикат списка IN, и этот предикат обладает достаточной выборочностью для использования индексированного поиска, оптимизатор преобразовывает предикат списка IN в соединение с вложенными циклами. Следующий пример иллюстрирует применение оптимизации.

Предположим, существует запрос

```
SELECT *
FROM sales_order
WHERE sales_rep = 142 or sales_rep = 1596
```

перечисляющий все заказы двух торговых представителей. Этот запрос семантически эквивалентен следующему:

```
SELECT *
FROM sales_order
WHERE sales_rep IN (142, 1596)
```

Оптимизатор оценивает, достаточно ли высока объединенная выборочность предиката списка IN для выполнения индексированного поиска. Следовательно, оптимизатор обрабатывает список IN как виртуальную таблицу и присоединяет ее к таблице *sales_order* по атрибуту *sales_rep*. В то время как результат оптимизации должен включить дополнительное "соединение" в план доступа, степень присутствия соединений в запросе не увеличивается, и время оптимизации не должно измениться.

У этого типа оптимизации есть два основных преимущества. Во-первых, предикат списка IN обрабатывается как предикат, позволяющий использовать аргументы поиска и применимый для индексированного поиска. Во-вторых, оптимизатор может отсортировать список IN для более эффективного поиска в отсортированном индексе.

Краткое представление плана доступа вышеуказанного запроса:

```
IN JNL sales_order<ky_so_employee_id>
```

Оптимизация LIKE

Предикаты LIKE часто используются с образцами, являющимися либо литерными константами, либо хост-переменными. В зависимости от образца оптимизатор перезаписывает предикат LIKE полностью или добавляет к нему дополнительные условия, чтобы в соответствующей таблице был возможен индексированный поиск.

Примеры

В следующих примерах предполагается, что образец в предикате LIKE - литерная константа или хост-переменная, а X - столбец базовой таблицы.

- `X LIKE '%'` перестраивается как `X IS NOT NULL`;
- `X LIKE 'abc'` перестраивается как X = 'abc';
- `X LIKE 'abc%'` дополняется предикатами
  `X < 'abcZ' and X > = 'abc_'`;

где Z и _ - соответственно наибольшее и наименьшее значения порядка сортировки этой базы данных. Если база данных может содержать строки с пробелами, то вторая операция сравнения >, а не > = - для обеспечения правильной семантики.

Преобразование внешних соединений во внутренние

В большинстве случаев оптимизатор создает планы доступа с глубоким левосторонним деревом обработки. Единственное исключение из этого правила - глубокое правостороннее вложенное внешнее соединение. Алгоритмы выполнения запросов LEFT или RIGHT OUTER JOIN процессом базы данных требуют, чтобы сохраняемые таблицы предшествовали таблицам, не представляющим результата, при любой стратегии соединения. Поэтому оптимизатор преобразовывает внешние соединения LEFT или RIGHT OUTER JOIN во внутренние соединения, если это возможно, так как внутренние соединения заменимы и дают оптимизатору больше возможностей при выполнении соединений.

LEFT или RIGHT OUTER JOIN могут преобразовываться во внутренние соединения, если существует предикат по таблице, не представляющей результата, в разделе запроса WHERE. Так как этот предикат является не пустым, любая строка со всеми значениями NULL, полученная при внешнем соединении, будет удалена из результата, что делает запрос семантически эквивалентным внутреннему соединению.

Пример

Например, запрос

```
SELECT *
FROM product p KEY LEFT OUTER JOIN sales_order_items
s
WHERE s.quantity > 15
```

перечисляет все продукты и их заказы на большее количество; LEFT OUTER JOIN обеспечивает то, что все продукты будут перечислены, даже если их не заказывали. Проблема с этим запросом состоит в том, что предикат в разделе WHERE устранит из результата продукты без заказов, потому что предикат `s.quantity>15` интерпретируется как FALSE, если `s.quantity` - NULL. Поэтому этот запрос семантически эквивалентен следующему:

```
SELECT *
FROM product p KEY JOIN sales_order_items s
WHERE s.quantity > 15
```

и именно эта форма является запросом, оптимизируемым сервером.

В этом примере запрос, скорее всего, написан неправильно; он должен быть прочитан как

```
SELECT *
FROM product p
    KEY LEFT OUTER JOIN sales_order_items s
          ON s.quantity > 15
```

для того, чтобы проверка количества была частью условия внешнего соединения.

Хотя оптимизация редко применяется к прямым запросам с внешними соединениями, она может применяться, если запрос ссылается на одно или более представлений, которые написаны с использованием внешних соединений. Раздел WHERE запроса может содержать условия, ограничивающие вывод представления так, что все строки, не представляющие результата, из одного или более выражений таблицы будут удалены, что делает эту оптимизацию применимой.

Обнаружение применимых условий

Эффективная стратегия доступа для практически любого запроса основана на присутствии условий, позволяющих использовать аргументы поиска, в разделах WHERE/ON/HAVING. Индексированный поиск возможен, только если условия, позволяющие использовать аргументы поиска, являются совпадающими предикатами. Соединения хэша, соединения со слиянием и соединения с вложенными циклами могут использоваться, только если присутствует условие соединения по эквивалентности. Поэтому Adaptive Server Anywhere детально анализирует условия поиска в первоначальном тексте запроса, чтобы обнаружить упрощенные или подразумеваемые условия, которые могут применяться оптимизатором.

При предварительной обработке осуществляются некоторые упрощения предикатов оператора, если происходит расширение или слияние представления. Например:

- `X = X` перезаписывается как `X IS NOT NULL`, если X может принимать значение NULL; в противном случае предикат удаляется.
- `ISNULL (X, X)` перезаписывается как X.
- `X+0` перезаписывается как X, если X - числовой столбец.
- `AND 1=1` устраняется.
- `OR 1=0` устраняется.

После этого шага предварительной обработки Adaptive Server Anywhere пытается нормализовать первоначальное условие поиска в конъюнктивную нормальную форму (КНФ). В выражении, которое будет включено в КНФ, все условия должны соединятся логической функцией AND. Каждое условие состоит из отдельного атомарного условия или набора условий, соединенных OR.

Преобразование произвольного условия в форму КНФ может привести к выражению подобной сложности, но с большим количеством условий. В таких случаях Adaptive Server Anywhere воздерживается от простого преобразования в форму КНФ. Вместо этого Adaptive Server Anywhere анализирует первоначальное выражение на наличие применимых предикатов, которые содержатся в первоначальном условии поиска, и присоединяет их функцией AND к запросу. Полная нормализация также исключается, если это требует дублирования важных предикатов (например, предиката, определяющего количество строк в подзапросе).

Однако алгоритм объединяет предикаты списка IN, если это возможно.

После полной нормализации условий поиска или нахождения применимых условий оптимизатор выполняет анализ транзитивности с целью обнаружения транзитивных условий равенства, первичных транзитивных условий соединения и условий с константой. Этим оптимизатор увеличивает свободу действий при установке количества соединений во время оптимизации затрат, так как эти транзитивные условия позволяют использовать альтернативный порядок соединения.

Пример

Предположим, первоначальный запрос был следующим:

```
SELECT e.emp_lname, s.id, s.order_date
FROM sales_order s, employee e
WHERE (e.emp_id = s.sales_rep and
          (s.sales_rep = 142 or s.sales_rep = 1596)
       )
          OR
       ( e.emp_id = s.sales_rep and s.cust_id = 667)
```

Этот запрос не содержит связывающего условия соединения по эквивалентности; следовательно, без детального анализа предиката оптимизатор не сможет найти эффективный план доступа. Однако Adaptive Server Anywhere может преобразовать выражение в КНФ, сохраняя эквивалентность запроса:

```
SELECT e.emp_lname, s.id, s.order_date
FROM sales_order as s, employee as e
WHERE e.emp_id = s.sales_rep AND
   (s.sales_rep = 142 or s.sales_rep = 1596 or
s.cust_id
= 667)'
```

Этот запрос можно эффективно оптимизировать как запрос с внутренним соединением.

Удаление ненужного перевода регистра

Базы данных Adaptive Server Anywhere по умолчанию поддерживают строковые сравнения без учета регистра. Иногда оптимизатором обрабатываются запросы, в которые пользователь явно включил текстовые преобразования с помощью встроенных функций UPPER() или LOWER(), когда как эти преобразования не являются необходимыми.

Adaptive Server Anywhere автоматически удаляет такие необязательные преобразования, если это не нарушает порядка сортировки базы данных. Невыполнение перевода регистра может привести к появлению возможности использовать предикат сравнения для индексированного поиска в соответствующей таблице.

Пример

В базе данных без учета регистра запрос

```
SELECT * FROM customer
WHERE UPPER(lname) = 'SMITH'
```

будет внутренне перезаписан как

```
SELECT * FROM customer
WHERE lname = 'SMITH'
```

и оптимизатор сможет использовать индекс *customer.lname*.

Кэширование подзапросов и функций

При обработке подзапроса Adaptive Server Anywhere полученный результат кэшируется. Кэширование выполняется путем последовательной обработки запросов; кэшируемые результаты никогда не используются совместно параллельными запросами или подключениями. Если Adaptive Server Anywhere должен снова обработать подзапрос для того же самого набора корреляционных значений, результат этого запроса просто считывается из кэша. В этом случае Adaptive Server Anywhere избегает выполнения повторных и избыточных вычислений. После завершения запроса (когда курсор запроса закрыт) Adaptive Server Anywhere удаляет значения из кэша.

Во время обработки запроса Adaptive Server Anywhere контролирует степень многократного использования кэшируемых значений подзапроса. Если значения коррелированной переменной повторяются редко, то Adaptive Server Anywhere вычисляет большинство значений только один раз. В этой ситуации Adaptive Server Anywhere распознает, что повторное вычисление случайных дублирующихся значений более эффективно, чем кэширование большого количества значений, которые вычисляются только один раз. Сервер приостанавливает кэширование этого подзапроса для остальных действий оператора и продолжает обработку подзапроса для каждой строки во внешнем блоке запроса.

Adaptive Server Anywhere не выполняет кэширование, если размер связанного столбца превышает 255 байтов. В таких случаях можно перестроить запрос или добавить другой столбец к таблице, чтобы сделать эти операции более эффективными.

Кэширование определяемых пользователем функций

Функции, определяемые пользователем, кэшируются так же, как результаты подзапроса. Это приводит к существенному увеличению производительности функций, требующих больших затрат, которые вызываются несколько раз в течение обработки запроса с одинаковыми параметрами. Это значит, что функция вызывается меньше раз, чем ожидалось. Оптимизатор предполагает, что функции, используемые при обработке запросе, всегда возвращают одинаковый результат для одного набора параметров и не имеют побочных эффектов.

☞ Для получения дополнительной информации об определяемых пользователем функциях см. раздел "Оператор CREATE FUNCTION" (CREATE FUNCTION statement) на стр. 277 в документе *"Справочник по SQL для ASA" (ASA SQL Reference Manual)*.

ЧАСТЬ 3

Диалекты SQL и совместимость

В этой части описываются Transact SQL и те средства Adaptive Server Anywhere, которые, как правило, отсутствуют в других реализациях SQL.

ГЛАВА 12

Совместимость с Transact-SQL

Об этой главе

Transact-SQL – это диалект SQL, который поддерживается Sybase Adaptive Server Enterprise.

Эта глава представляет собой руководство по созданию приложений, совместимых и с Adaptive Server Anywhere, и с Adaptive Server Enterprise. В ней описывается поддержка Adaptive Server Anywhere элементов и операторов языка Transact-SQL, системных таблиц, представлений и процедур Adaptive Server Enterprise.

Содержание

Обзор поддержки Transact-SQL

Adaptive Server Anywhere поддерживает большой поднабор элементов **Transact-SQL**, который является диалектом SQL, поддерживаемым Sybase Adaptive Server Enterprise. В этой главе рассматриваются вопросы совместимости SQL для Adaptive Server Anywhere и для Adaptive Server Enterprise.

Цели

Поддержка Transact-SQL в Adaptive Server Anywhere имеет следующие цели:

- **Переносимость приложений.** Многие приложения, хранимые процедуры и пакетные файлы могут быть написаны для использования в базах данных и Adaptive Server Anywhere, и Adaptive Server Enterprise.
- **Переносимость данных.** Базы данных Adaptive Server Anywhere и Adaptive Server Enterprise могут легко обмениваться данными и реплицировать данные между собой.

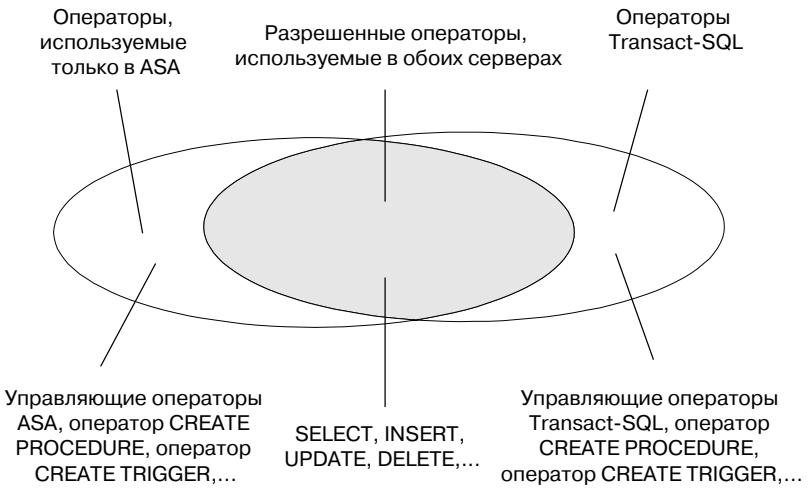
Главная цель состоит в возможности создания приложений для работы и с Adaptive Server Enterprise, и с Adaptive Server Anywhere. Существующие приложения Adaptive Server Enterprise обычно требуют некоторых изменений для их выполнения в базе данных Adaptive Server Anywhere.

Реализация поддержки Transact-SQL

Поддержка Transact-SQL в Adaptive Server Anywhere обеспечивается за счет следующего:

- Многие операторы SQL совместимы одновременно с Adaptive Server Anywhere и Adaptive Server Enterprise.
- Для некоторых операторов, особенно языка процедур, используемого при написании процедур, триггеров и пакетов, поддерживается отдельный оператор Transact-SQL, а также синтаксис, который использовался в предыдущих версиях Adaptive Server Anywhere. Для таких операторов Adaptive Server Anywhere поддерживает два **диалекта** SQL. В этой главе эти диалекты называются Transact-SQL и Watcom-SQL.
- Процедура, триггер или пакет выполняются либо в диалекте Transact-SQL, либо в диалекте Watcom-SQL. В пределах одного пакета или процедуры необходимо использовать управляющие операторы только одного диалекта. Например, в диалектах используются различные операторы управления потоком.

Следующая диаграмма иллюстрирует области пересечения двух диалектов.



Сходство и различия

Adaptive Server Anywhere поддерживает очень большое количество элементов, функций и операторов языка Transact-SQL для работы с существующими данными.

Например, в Adaptive Server Anywhere поддерживаются все числовые функции, все кроме одной функции для работы со строками, все агрегатные функции и все функции для обработки даты и времени. Другой пример: Adaptive Server Anywhere поддерживает внешние соединения Transact-SQL (используя операции = * и * =) и расширенные операторы DELETE и UPDATE с использованием соединений.

Кроме того, в Adaptive Server Anywhere поддерживается большое количество хранимых процедур языка Transact-SQL (синтаксис CREATE PROCEDURE и CREATE TRIGGER, управляющие операторы и др.) и многие, но не все, аспекты операторов языка определения данных Transact-SQL.

Существуют различия в организации архитектурных средств и средств конфигурирования, поддерживаемых каждым их этих продуктом. Управление устройствами, управление пользователями и задачи по обслуживанию, например, резервное копирование, обычно зависят от конкретной системы. Однако даже в этом случае Adaptive Server Anywhere предоставляет системные таблицы Transact-SQL в виде представлений, в которых те таблицы, которые оказываются незначимыми в Adaptive Server Anywhere, не содержат строк. Adaptive Server Anywhere предоставляет также набор системных процедур для решения некоторых общих административных задач.

В этой главе сначала рассматриваются некоторые вопросы системного уровня, где различия наиболее велики, после чего описывается работа с данными и аспекты языка определения данных этих диалектов, обеспечивающие высокую совместимость.

Только Transact-SQL

Некоторые операторы SQL, поддерживаемые Adaptive Server Anywhere, содержатся только в одном из двух диалектов. Смешивать эти два диалекта в пределах процедуры, триггера или пакета нельзя. Например, Adaptive Server

Anywhere поддерживает следующие операторы, но только как часть диалекта Transact-SQL:

- управляющие операторы IF и WHILE (Transact-SQL);
- оператор EXECUTE (Transact-SQL);
- операторы CREATE PROCEDURE и CREATE TRIGGER (Transact-SQL);
- оператор BEGIN TRANSACTION (Transact-SQL);
- операторы SQL, *не* отделенные точками с запятой, являются частью процедуры или пакета Transact-SQL.

Только Adaptive Server Anywhere

Adaptive Server Enterprise не поддерживает следующие операторы:

- управляющие операторы CASE, LOOP и FOR;
- версии Adaptive Server Anywhere операторов IF и WHILE;
- оператор CALL;
- версии Adaptive Server Anywhere операторов CREATE PROCEDURE, CREATE FUNCTION и CREATE TRIGGER;
- операторы SQL, отделенные точками с запятой.

Примечания

Эти два диалекта не могут быть использоваться одновременно в пределах процедуры, триггера или пакета. Это означает следующее:

- В пределах пакета, процедуры или триггера можно использовать операторы диалекта Transact-SQL вместе с операторами, которые входят в оба диалекта.
- В пакет, процедуру или триггер можно включать операторы, не поддерживаемые Adaptive Server Enterprise, вместе с операторами, которые поддерживаются обоими серверами.
- В пределах пакета, процедуры или триггера нельзя использовать операторы диалекта Transact-SQL вместе с операторами, применимыми только в Adaptive Server Anywhere.

Архитектура Adaptive Server

Adaptive Server Enterprise и Adaptive Server Anywhere являются взаимно дополняющими друг друга продуктами с архитектурой, предназначенной для выполнения различных целей. Adaptive Server Anywhere используется как сервер рабочей группы или отдела, не требующий выполнения большого объема работ по администрированию, а также как персональная база данных. Adaptive Server Enterprise используется как сервер уровня предприятия для баз данных наибольшего размера.

В этом разделе описываются архитектурные различия Adaptive Server Enterprise и Adaptive Server Anywhere. Также описываются инструментальные средства Adaptive Server Anywhere для совместимого управления базой данных, подобные Adaptive Server Enterprise.

Серверы и базы данных

Взаимосвязь между серверами и базами данных различна в Adaptive Server Enterprise и Adaptive Server Anywhere.

В Adaptive Server Enterprise каждая база данных существует на сервере, и каждый сервер может содержать несколько баз данных. Пользователи имеют права входа на сервер и могут подключиться к серверу. Они могут использовать любую базу данных на этом сервере, на доступ к которой они обладают полномочиями. Системные таблицы всей системы, хранящиеся в главной базе данных, содержат общую для всех баз данных сервера информацию.

В Adaptive Server Anywhere нет главной базы данных

В Adaptive Server Anywhere нет уровня, соответствующего главной базе данных Adaptive Server Enterprise. Вместо этого, каждая база данных представляет собой независимый объект, содержащий все необходимые системные таблицы. Пользователям даются права подключения к базе данных, но не к серверу. Пользователи могут подключаться только к конкретной базе данных. Не существует набора системных таблиц всей системы, поддерживаемого на уровне главной базы данных. Каждый сервер базы данных Adaptive Server Anywhere динамически загружает и выгружает несколько баз данных, и пользователи могут независимо подключаться к каждой из них.

Adaptive Server Anywhere предоставляет инструментальные средства для поддержки Transact-SQL и Open Server для того, чтобы определенные задачи могли выполняться так же, как и в случае Adaptive Server Enterprise. Например, в Adaptive Server Anywhere реализация системной процедуры Adaptive Server Enterprise **sp_addlogin** выполняет наиболее эквивалентное действие: добавление пользователя к базе данных.

☞ Для получения информации о поддержке Open Server см. раздел "Adaptive Server Anywhere как Open Server" (Adaptive Server Anywhere as an Open Server) на стр. 113 в документе *"Руководство по администрированию баз данных ASA" (ASA Database Administration Guide)*.

Операторы для работы с файлами

Adaptive Server Anywhere не поддерживает операторы DUMP DATABASE и LOAD DATABASE диалекта Transact-SQL. В Adaptive Server Anywhere существуют собственные операторы CREATE DATABASE и DROP DATABASE с другим синтаксисом.

Управление устройствами

Adaptive Server Anywhere и Adaptive Server Enterprise используют различные модели для управления устройствами и дисковым пространством в соответствии с разными способами применения этих двух продуктов. В Adaptive Server Enterprise применяется комплексная схема управления ресурсами с использованием разнообразных операторов Transact-SQL. Adaptive Server Anywhere управляет собственными ресурсами автоматически, а его базы данных являются обычными файлами операционной системы.

Adaptive Server Anywhere не поддерживает следующие операторы Transact-SQL: DISK INIT, DISK MIRROR, DISK REFIT, DISK REINIT, DISK REMIRROR и DISK UNMIRROR.

☞ Для получения информации об управлении дисковым пространством см. раздел "Работа с файлами базы данных" (Working with Database Files) на стр. 215 в документе *"Руководство по администрированию баз данных ASA" (ASA Database Administration Guide)*.

Значения по умолчанию и правила

Adaptive Server Anywhere не поддерживает операторы CREATE DEFAULT и CREATE RULE диалекта Transact-SQL. Оператор CREATE DOMAIN позволяет установить значение по умолчанию и правило (называемое условием CHECK) в определении домена и таким образом предоставляет функциональные возможности, подобные операторам CREATE DEFAULT и CREATE RULE диалекта Transact-SQL.

В Adaptive Server Enterprise оператор CREATE DEFAULT создает поименованное **значение по умолчанию**. Это значение можно использовать как значение по умолчанию для столбцов путем связывания с определенным столбцом, или как значение по умолчанию для всех столбцов домена путем связывания с определенным типом данных с помощью системной процедуры **sp_bindefault**.

Оператор CREATE RULE создает поименованное **правило**, которое может использоваться для определения домена для столбцов путем связывания этого правила с определенным столбцом или являться правилом для всех столбцов домена путем связывания этого правила с определенным типом данных. Правило привязывается к типу данных или столбцу с помощью системной процедуры **sp_bindrule**.

В Adaptive Server Anywhere домен может иметь значение по умолчанию и связанное с ним условие CHECK, которое применяется ко всем столбцам этого типа данных. Домен создается с помощью оператора CREATE DOMAIN.

Значения по умолчанию и правила, а также условия CHECK для отдельных столбцов можно установить с помощью операторов CREATE TABLE или ALTER TABLE.

☞ Для получения информации о синтаксисе этих операторов в Adaptive Server Anywhere см. раздел "Операторы SQL" (SQL Statements) на стр. 189 в документе *"Справочник по SQL для ASA" (ASA SQL Reference Manual)*.

Системные таблицы

В дополнение к собственным системным таблицам, Adaptive Server Anywhere предоставляет набор системных представлений, которые имитируют соответствующие части системных таблиц Adaptive Server Enterprise. Для получения списка таблиц и их описания см. раздел "Представления для совместимости с Transact-SQL" (Views for Transact-SQL Compatibility), в котором описываются системные каталоги двух серверов, на стр. 639 в документе *"Справочник по SQL для ASA" (ASA SQL Reference Manual)*. В данном разделе содержится краткий обзор различий.

Системные таблицы Adaptive Server Anywhere расположены в пределах каждой базы данных полностью, а системные таблицы Adaptive Server Enterprise находятся частично в каждой базе данных и частично в главной базе данных. Архитектура Adaptive Server Anywhere не предусматривает наличия главной базы данных.

В Adaptive Server Enterprise владелец базы данных (код пользователя **dbo**) является владельцем системных таблиц. В Adaptive Server Anywhere владелец системы (код пользователя **SYS**) является владельцем системных таблиц. Пользователь с кодом **dbo** владеет совместимыми с Adaptive Server Enterprise системными представлениями, предоставляемыми Adaptive Server Anywhere.

Административные роли

Adaptive Server Enterprise имеет более широкий набор административных ролей, чем Adaptive Server Anywhere. В Adaptive Server Enterprise существует определенное количество различных ролей. Любой роли предоставляется более одной учетной записи, и одной учетной записи предоставляется более одной роли.

Роли Adaptive Server Enterprise

В Adaptive Server Enterprise определены следующие роли:

- **Системный администратор.** Ответственный за общие административные задачи, не связанные с определенными приложениями; имеет доступ ко всем объектам базы данных.

- **Ответственный за безопасность системы.** Выполняет задачи по обеспечению безопасности Adaptive Server Enterprise; не имеет специальных полномочий по отношению к объектам базы данных.
- **Владелец базы данных.** Обладает все полномочия на объекты базы данных, владельцем которой он является, может добавлять пользователей и предоставлять другим пользователям полномочия на создание объектов и выполнение команд в базе данных.
- **Операторы определения данных.** Полномочия, получаемые пользователями и позволяющие с помощью определенных операторов, например, CREATE TABLE или CREATE VIEW, создавать объекты базы данных.
- **Владелец объекта.** Каждый объект базы данных имеет владельца, который может выдавать полномочия для обращения к объекту другим пользователям. Владелец объекта автоматически получает все полномочия на данный объект.

В Adaptive Server Anywhere следующие полномочия для всей базы данных имеют административные роли:

- Администратор базы данных (полномочия администратора БД) имеет, подобно владельцу базы данных Adaptive Server Enterprise, полные права на все объекты базы данных (объекты отличаются от объектов с владельцем SYS) и может предоставлять другим пользователям полномочия на создание объектов и выполнение команд в пределах базы данных. По умолчанию код администратора базы данных – **DBA.**
- Полномочия RESOURCE позволяют пользователю создавать любые объекты в пределах базы данных. В Adaptive Server Enterprise для этого производится выдача полномочий отдельному пользователю на собственный оператор CREATE.
- Владельцы объектов Adaptive Server Anywhere реализованы так же, как они реализованы в Adaptive Server Enterprise. Владелец объекта автоматически получает все полномочия на свой объект, включая право предоставления полномочий другим пользователям.

Для обеспечения "бесшовного" доступа к данным, содержащимся в Adaptive Server Enterprise и в Adaptive Server Anywhere, следует создать коды пользователей с соответствующими полномочиями в базе данных (RESOURCE в Adaptive Server Anywhere или право на собственный оператор CREATE в Adaptive Server Enterprise) и объекты, используя этот код пользователя. При использовании одного кода пользователя в обеих средах имена объектов и спецификаторы могут быть идентичными в этих базах данных, что обеспечивает возможность совместимого доступа.

Пользователи и группы

Существуют различия между моделями пользователей и групп в Adaptive Server Enterprise и Adaptive Server Anywhere.

В Adaptive Server Enterprise при подключении пользователя к серверу каждый пользователь имеет имя и пароль для доступа к серверу, а также код пользователя для каждой базы данных, к которой он обращается. Каждый пользователь базы данных может быть членом только одной группы.

В Adaptive Server Anywhere пользователи подключаются непосредственно к базе данных. Отдельного имени для сервера базы данных не требуется. Вместо этого каждый пользователь получает код и пароль пользователя для работы с базой данных. Пользователи могут быть членами многих групп, также допускается иерархия групп.

Оба сервера поддерживают группы пользователей, что позволяет выдавать полномочия одновременно нескольким пользователям. Однако есть различия в характерных особенностях групп этих двух серверов. Например, в Adaptive Server Enterprise каждый пользователь может быть членом только одной группы, а в Adaptive Server Anywhere такого ограничения не существует. Для получения дополнительной информации следует обратиться к документации по пользователям и группам для этих двух серверов.

Для определения полномочий по умолчанию в Adaptive Server Enterprise и Adaptive Server Anywhere определена группа public. Каждый пользователь автоматически становится членом группы public.

Adaptive Server Anywhere поддерживает следующие системные процедуры Adaptive Server Enterprise для управления пользователями и группами.

☞ Для получения информации об аргументах каждой процедуры см. раздел "Системные процедуры и процедуры каталога Adaptive Server Enterprise" (Adaptive Server Enterprise system and catalog procedures) на стр. 683 в документе *"Справочник по SQL для ASA" (ASA SQL Reference Manual)*.

Системная процедура	**Описание**
sp_addlogin	В Adaptive Server Enterprise это процедура добавляет пользователя на сервер. В Adaptive Server Anywhere она добавляет пользователя к базе данных.
sp_adduser	В Adaptive Server Enterprise и Adaptive Server Anywhere эта процедура добавляет пользователя к базе данных. Эта задача отличается от **sp_addlogin** в Adaptive Server Enterprise, в Adaptive Server Anywhere различий нет.
sp_addgroup	Добавляет группу к базе данных.
sp_changegroup	Добавляет пользователя к группе или перемещает пользователя из одной группы в другую.
sp_droplogin	В Adaptive Server Enterprise удаляет пользователя с сервера. В Adaptive Server Anywhere удаляет пользователя из базы данных.
sp_dropuser	Удаляет пользователя из базы данных.
sp_dropgroup	Удаляет группу из базы данных.

В Adaptive Server Enterprise код пользователя используется в пределах всего сервера. В Adaptive Server Anywhere пользователи принадлежат отдельным базам данных.

Полномочия на доступ к объектам базы данных

Операторы GRANT и REVOKE в Adaptive Server Enterprise и Adaptive Server Anywhere для предоставления полномочий на доступ к отдельным объектам базы данных очень похожи. Оба разрешают предоставление полномочий SELECT, INSERT, DELETE, UPDATE и REFERENCES на таблицы и представления базы данных и полномочий UPDATE на выбранные столбцы таблиц базы данных. Оба разрешают предоставление полномочий EXECUTE на хранимые процедуры.

Например, следующий оператор допустим и в Adaptive Server Enterprise, и в Adaptive Server Anywhere:

```
GRANT INSERT, DELETE
ON TITLES
TO MARY, SALES
```

Этот оператор предоставляет полномочия на использование операторов INSERT и DELETE в таблице **TITLES** пользователю **MARY** и группе **SALES**.

И Adaptive Server Anywhere, и Adaptive Server Enterprise поддерживают раздел WITH GRANT OPTION, разрешающий обладателю полномочий предоставлять в свою очередь полномочия другим пользователям, хотя Adaptive Server Anywhere не позволяет использовать WITH GRANT OPTION в операторе GRANT EXECUTE.

Полномочия на всю базу данных

Adaptive Server Enterprise и Adaptive Server Anywhere используют различные модели полномочий пользователей на всю базу данных. Они рассматриваются в разделе "Пользователи и группы" на стр. 378. Adaptive Server Anywhere предоставляет полномочия администратора БД, разрешая пользователю выполнять любые действия в пределах базы данных. В Adaptive Server Enterprise системный администратор обладает такими полномочиями для всех баз данных сервера. Однако полномочия администратора БД в Adaptive Server Anywhere отличаются от полномочий владельца базы данных Adaptive Server Enterprise, который должен использовать оператор **SETUSER** для получения полномочий доступа к объектам, принадлежащим другим пользователям.

Adaptive Server Anywhere предоставляет полномочия RESOURCE, дающие пользователю право создания объектов в базе данных. Соответствующее полномочие Adaptive Server Enterprise GRANT ALL используется владельцем базы данных.

Конфигурирование баз данных для совместимости с Transact-SQL

Некоторые различия в поведении Adaptive Server Anywhere и Adaptive Server Enterprise можно устранить, выбирая соответствующие параметры при создании базы данных или при перестройке существующей базы данных. Другие различия можно изменить при помощи параметров на уровне подключения, используя оператор SET TEMPORARY OPTION в Adaptive Server Anywhere или оператор SET в Adaptive Server Enterprise.

Создание совместимой с Transact-SQL базы данных

В этом разделе описываются возможности создания или перестройки базы данных.

Быстрое начало

Здесь указаны шаги, необходимые для создания совместимой с Transact-SQL базы данных. В этом разделе описаны параметры, которые необходимо установить.

❖ **Создание совместимой с Transact-SQL базы данных (Sybase Central)**

1. Запустите Sybase Central.
2. Выберите Tools➤Adaptive Server Anywhere 8➤Create Database.
3. Выполняйте указания мастера.
4. Когда появится кнопка Emulate Adaptive Server Enterprise, нажмите ее и затем нажмите Next.
5. Выполняйте указания мастера.

❖ **Создание совместимой с Transact-SQL базы данных (командная строка):**

- Введите следующую команду в командной строке:

```
dbinit -b -c -k db-name.db
```

☞ Для получения дополнительной информации об этих параметрах см. раздел "Параметры утилиты инициализации" (Initialization utility options) на стр. 459 в документе *"Руководство по администрированию баз данных ASA" (ASA Database Administration Guide)*.

❖ **Создание совместимой с Transact-SQL базы данных (SQL)**

1. Выполните подключение к базе данных Adaptive Server Anywhere.
2. Введите следующий оператор, например, в Interactive SQL:

```
CREATE DATABASE 'db-name.db'
ASE COMPATIBLE
```

В этом операторе параметр ASE COMPATIBLE указывает на совместимость с Adaptive Server Enterprise.

Создание базы данных с учетом регистра

По умолчанию в базах данных Adaptive Server Enterprise сравнения строк выполняются с учетом регистра, а в Adaptive Server Anywhere регистр не учитывается.

При построении базы данных, совместимой с Adaptive Server Enterprise, с помощью Adaptive Server Anywhere следует установить параметр учета регистра.

- В Sybase Central этот параметр указывается в мастере создания базы данных (Create Database wizard).
- При использовании утилиты dbinit командной строки следует установить параметр -c.

Игнорирование конечных пробелов при сравнениях

При создании совместимой с Adaptive Server Enterprise базы данных с помощью Adaptive Server Anywhere следует выбрать параметр игнорирования конечных пробелов при сравнениях.

- В Sybase Central этот параметр указывается в мастере создания базы данных на странице Choose the Database Attributes.
- При использовании утилиты dbinit командной строки следует установить параметр -b.

Если выбран этот параметр, Adaptive Server Enterprise и Adaptive Server Anywhere считают следующие две строки одинаковыми:

```
'ignore the trailing blanks '
'ignore the trailing blanks'
```

Если этот параметр не установлен, Adaptive Server Anywhere рассматривает эти строки как разные.

Побочный эффект этого выбора заключается в заполнении строк пробелами при их выборке клиентским приложением.

Удаление старых системных представлений

Более старые версии Adaptive Server Anywhere использовали два системных представления, названия которых конфликтуют с системными представлениями Adaptive Server Enterprise, обеспечивающими совместимость. Этими представлениями являются SYSCOLUMNS и SYSINDEXES. При использовании интерфейсов Open Client или JDBC следует создавать базу данных без этих представлений. Это можно сделать, используя параметр *dbinit -k* командной строки.

В противном случае, следующие два оператора возвратят различные результаты:

```
SELECT * FROM SYSCOLUMNS ;
SELECT * FROM dbo.syscolumns ;
```

- **Удаление системных представлений Adaptive Server Anywhere из существующей базы данных**

  1 Выполните подключение к базе данных с использованием полномочий администратора БД.

  2 Выполните следующие операторы:

```
DROP VIEW SYS.SYSCOLUMNS ;
DROP VIEW SYS.SYSINDEXES
```

Предостережение

Убедитесь, что системные представления dbo.syscolumns или dbo.sysindexes не удаляются.

Установка параметров для совместимости с Transact-SQL

Параметры базы данных Adaptive Server Anywhere устанавливаются оператором SET OPTION. Некоторые параметры настройки базы данных существенны для поведения в стиле Transact-SQL.

Установка параметра allow_nulls_by_default

По умолчанию, Adaptive Server Enterprise не позволяет устанавливать значение NULL в новых столбцах, если это явно не разрешено. Adaptive Server Anywhere по умолчанию разрешает присваивать NULL в новых столбцах, что совместимо со стандартом SQL/92 ISO.

Для работы Adaptive Server Enterprise в соответствии со стандартом SQL/92 следует использовать системную процедуру **sp_dboption** и установить параметр **allow_nulls_by_default** в значение TRUE.

Обеспечить совместимость Adaptive Server Anywhere с Transact-SQL можно, установив параметр **allow_nulls_by_default** в значение OFF. Можно использовать оператор SET OPTION следующим образом:

```
SET OPTION PUBLIC.allow_nulls_by_default = 'OFF'
```

Установка параметра quoted_identifier

По умолчанию, Adaptive Server Enterprise обрабатывает идентификаторы и строки отлично от Adaptive Server Anywhere, действия которого совместимы со стандартом SQL/92 ISO.

Параметр **quoted_identifier** существует и в Adaptive Server Enterprise, и в Adaptive Server Anywhere. Следует установить параметр одинаково в обеих базах данных для аналогичной обработки идентификаторов и строк.

Для поведения в стиле SQL/92 присвойте параметру **quoted_identifier** значение ON и в Adaptive Server Enterprise, и в Adaptive Server Anywhere.

Для поведения в стиле Transact-SQL установите значение OFF параметра **quoted_identifier** и в Adaptive Server Enterprise, и в Adaptive Server Anywhere. При таком выборе нельзя использовать идентификаторы, являющиеся ключевыми словами, заключенные в двойные кавычки.

☞ Для получения дополнительной информации о параметре **quoted_identifier** см. раздел "Параметр QUOTED_IDENTIFIER" (QUOTED_IDENTIFIER option) на стр. 578 в документе *"Руководство по администрированию баз данных ASA" (ASA Database Administration Guide)*.

Установка параметра автоматического штампа времени в ON

Transact-SQL определяет столбец **штампа времени** (timestamp) со специальными свойствами. Если параметр **automatic_timestamp** (автоматический штамп времени) установлен в ON, Adaptive Server Anywhere обрабатывает столбцы **штампа времени** подобно Adaptive Server Enterprise.

Если параметр **automatic_timestamp** установлен в ON в Adaptive Server Anywhere (по умолчанию OFF), любые новые столбцы с типом данных TIMESTAMP, в которых не определено явное значение по умолчанию, получают значение по умолчанию **штамп времени.**

☞ Для получения дополнительной информации о столбцах **штампов времени** см. раздел "Специальный столбец и тип данных штампа времени в Transact-SQL" на стр. 386.

Установка параметра string_rtruncation

Adaptive Server Enterprise и Adaptive Server Anywhere поддерживают параметр string_rtruncation, который управляет выдачей сообщения об ошибке при усечении строки INSERT или UPDATE. Этот параметр следует установить одинаково в двух базах данных.

☞ Для получения дополнительной информации о параметре STRING_RTRUNCATION см. раздел "Параметр STRING_RTRUNCATION" (STRING_RTRUNCATION option) на стр. 583 в документе *"Руководство по администрированию баз данных ASA" (ASA Database Administration Guide)*.

☞ Для получения дополнительной информации о параметрах базы данных для совместимости с Transact-SQL см. "Параметры для совместимости с Transact-SQL и SQL/92" (Transact-SQL and SQL/92 compatibility options) на стр. 530 в документе *"Руководство по администрированию баз данных ASA" (ASA Database Administration Guide)*.

Учет регистра

In AdB базах данных учет регистра может относиться к следующему:

- **Данные.** Учет регистра данных отражается в индексах, в результатах запросов и так далее.
- **Идентификаторы.** Идентификаторы включают имена таблиц, имена столбцов и так далее.
- **Коды пользователей и пароли.** При учете регистра коды пользователей и пароли рассматриваются отлично от других идентификаторов.

Учет регистра данных

Установка учета регистра данных в Adaptive Server Anywhere в сравнениях производится при создании базы данных. По умолчанию, базы данных Adaptive Server Anywhere не учитывают регистр при сравнениях, хотя данные всегда хранятся в том регистре, в котором они были введены.

Чувствительность Adaptive Server Enterprise к регистру зависит от установленного в системе порядка сортировки. Учет регистра для однобайтовых символов может быть изменен при переопределении порядка сортировки Adaptive Server Enterprise.

Учет регистра идентификаторов

Adaptive Server Anywhere не поддерживает чувствительность к регистру идентификаторов. В Adaptive Server Enterprise учет регистра идентификаторов является следствием учета регистра данных.

В Adaptive Server Enterprise имена доменов чувствительны к регистру. В Adaptive Server Anywhere для таких имен регистр не учитывается, за исключением типов данных Java.

Коды пользователей и пароли

В Adaptive Server Anywhere в кодах пользователей и паролях регистр данных учитывается. Заданные по умолчанию код пользователя и пароль для чувствительных к регистру баз данных – **DBA** и **SQL** в верхнем регистре, соответственно.

В Adaptive Server Enterprise учет регистра кодов пользователей и паролей соответствует чувствительности к регистру сервера.

Совместимость имен объектов

Каждый объект базы данных должен иметь уникальное имя в пределах некоторого **пространства имени.** Вне этого пространства возможно дублирование имени. Некоторые объекты базы данных занимают различные пространства имен в Adaptive Server Enterprise и Adaptive Server Anywhere.

В Adaptive Server Anywhere индексы и триггеры принадлежат владельцу таблицы, в которой они созданы. Имена индексов и триггеров должны быть уникальны для данного владельца. Например, в таблице t1, принадлежащей пользователю user1, и таблице t2, принадлежащей пользователю user2, могут присутствовать индексы с одним именем, но в двух таблицах, принадлежащих одному пользователю, не может быть индексов с одинаковым именем.

В Adaptive Server Enterprise пространство имен индекса менее ограничено, чем в Adaptive Server Anywhere. Имена индексов должны быть уникальными только в пределах таблицы, но любые две таблицы могут содержать индекс с одинаковым именем. Для совместимости SQL следует использовать ограничение Adaptive Server Anywhere на уникальность имени индекса для владельца таблицы.

В Adaptive Server Enterprise пространство имен триггеров имеет большие ограничения, чем в Adaptive Server Anywhere. В базе данных имена триггеров должны быть уникальными. Для совместимости SQL следует использовать ограничения Adaptive Server Enterprise и использовать в базе данных уникальные имена триггеров.

Специальный столбец и тип данных штампа времени в Transact-SQL

Adaptive Server Anywhere поддерживает специальный столбец штампа времени *timestamp* в Transact-SQL. Столбец *timestamp* и системная функция *tsequal* проверяют, была ли строка обновлена.

> **Два значения штампа времени**
>
> В Adaptive Server Anywhere существует тип данных TIMESTAMP, который содержит точную информацию о дате и времени. Он отличается от специального столбца и типа данных TIMESTAMP в Transact-SQL.

Создание столбца штампа времени Transact-SQL в Adaptive Server Anywhere

Для создания столбца *timestamp* в Transact-SQL следует создать столбец, который имеет тип данных TIMESTAMP (Adaptive Server Anywhere), и установить значение по умолчанию *timestamp*. Столбец может иметь любое имя, хотя обычно это имя *timestamp*.

Например, следующий оператор CREATE TABLE создает в таблице столбец *timestamp* в Transact-SQL:

```
CREATE TABLE table_name (
  column_1 INTEGER ,
  column_2 TIMESTAMP DEFAULT TIMESTAMP
)
```

Следующий оператор ALTER TABLE добавляет столбец *timestamp* в Transact-SQL в таблицу *sales_order:*

```
ALTER TABLE sales_order
ADD timestamp TIMESTAMP DEFAULT TIMESTAMP
```

В Adaptive Server Enterprise столбец с именем *timestamp* и неуказанным типом данных автоматически получает тип данных TIMESTAMP. В Adaptive Server Anywhere следует явно назначить тип данных.

При установке значения ON параметра AUTOMATIC_TIMESTAMP задавать значение по умолчанию не обязательно: любой новый столбец, созданный с типом данных TIMESTAMP без указания значения по умолчанию, получает значение по умолчанию *timestamp*. Следующий оператор устанавливает параметр AUTOMATIC_TIMESTAMP в значение ON:

```
SET OPTION PUBLIC.AUTOMATIC_TIMESTAMP='ON'
```

Тип данных столбца штампа времени

Adaptive Server Enterprise рассматривает столбец timestamp как домен типа VARBINARY(8), позволяя устанавливать NULL, в то время как Adaptive Server Anywhere определяет тип данных столбца timestamp как TIMESTAMP, который состоит из даты и времени с долями секунды в шести десятичных разрядах.

При выборке из таблицы более поздних обновлений переменная, в которую происходит запись значения штампа времени, должна соответствовать описанию столбца.

Добавление столбца штампа времени в существующую таблицу

При добавлении специального столбца *timestamp* в существующую таблицу все имеющиеся строки получат значение NULL в столбце timestamp. Для установки значения штампа времени (текущего времени) в существующих строках следует обновить все строки в таблице, не изменяя данные. Например, следующий оператор обновляет все строки таблицы *sales_order*, не изменяя их значения:

```
UPDATE sales_order
SET region = region
```

В Interactive SQL можно установить параметр TIMESTAMP_FORMAT, чтобы просмотреть различия значений строк. Следующий оператор изменяет параметр TIMESTAMP_FORMAT для отображения шести цифр долей секунды:

```
SET OPTION TIMESTAMP_FORMAT='YYYY-MM-DD HH:NN:ss.SSSSSS'
```

Если отображается менее шести цифр, некоторые значения столбца *timestamp* могут казаться равными, хотя на самом деле это не так.

Использование tsequal для обновлений

Системная функция *tsequal* определяет, был ли обновлен столбец *timestamp* или нет.

Например, приложение может выполнить оператор SELECT для столбца *timestamp* и поместить его в переменную. При выполнении оператора UPDATE одной из выбранных строк можно воспользоваться функцией *tsequal* для проверки факта изменения строки. Функция *tsequal* сравнивает значение штампа времени в таблице со значением, полученным с помощью SELECT. Одинаковые значения штампа времени означают, что изменений нет. Если значения штампа времени различные, строка была изменена после выполнения оператора SELECT.

Оператор UPDATE, использующий функцию *tsequal*, выглядит следующим образом:

```
UPDATE publishers
SET city = 'Springfield'
WHERE pub_id = '0736'
AND TSEQUAL(timestamp, '1995/10/25 11:08:34.173226')
```

Первый аргумент функции *tsequal* - имя специального столбца *timestamp;* второй аргумент – значение штампа времени, полученное в операторе SELECT. В Embedded SQL второй аргумент, вероятно, будет хост-переменной, содержащей значение TIMESTAMP из недавней выборки FETCH столбца.

Специальный столбец IDENTITY

Для создания столбца IDENTITY (столбца идентификации) используется следующий синтаксис оператора CREATE TABLE:

```
CREATE TABLE имя-таблицы (
  ...
  имя-таблицы numeric(n,0) IDENTITY NOT NULL,
  ...
)
```

где *n* достаточно велико для хранения значения максимального числа строк, которые могут быть вставлены в таблицу.

Столбец IDENTITY содержит последовательные числа, например, номера счетов или номера служащих, которые создаются автоматически. Значение столбца IDENTITY уникально идентифицирует каждую строку таблицы.

В Adaptive Server Enterprise каждая таблица базы данных может содержать столбец IDENTITY. Тип данных должен быть числовым, начиная с нуля, и столбец IDENTITY не должен принимать значений NULL.

В Adaptive Server Anywhere столбец IDENTITY является столбцом по умолчанию. Можно явно вставить значения, не являющиеся частью последовательности, в столбец оператором INSERT. Adaptive Server Enterprise не позволяет добавлять значения в столбец идентификации, если параметр *identity_insert* имеет значение *on*. В Adaptive Server Anywhere следует установить свойство NOT NULL самостоятельно и создать только один столбец IDENTITY. В Adaptive Server Anywhere любой столбец с числовым типом данных может быть столбцом IDENTITY.

В Adaptive Server Anywhere столбец *identity* и установка по умолчанию AUTOINCREMENT для столбца идентичны.

Получение значений столбцов IDENTITY с помощью @@identity

При вставке первой строку в таблицу значение столбца IDENTITY для нее - 1. При каждой последующей вставке значение столбца увеличивается на единицу. Значение столбца идентификации последней вставленной строки содержится в глобальной переменной @@*identity*.

Значение @@ *identity* изменяется при каждой попытке оператора вставить строку в таблицу.

- Если оператор затрагивает таблицу без столбца IDENTITY, значение @@*identity* - 0.
- Если оператор вставляет несколько строк, @@*identity* содержит последнее добавленное в столбец IDENTITY значение.

Это изменение является постоянным. @@identity не возвращается к предыдущим значениям, если оператор не достигает успеха или выполняется откат содержащей его транзакции.

Для получения дополнительной информации о @@identity см. раздел "Глобальная переменная @@identity" (@@identity global variable) на стр. 44 в документе *"Справочник по SQL для ASA" (ASA SQL Reference Manual)*.

Написание совместимых операторов SQL

В этом разделе приводятся общие рекомендации по написанию операторов SQL для их использования в нескольких системах управления базами данных. Также рассматриваются вопросы совместимости Adaptive Server Enterprise и Adaptive Server Anywhere на уровне операторов SQL.

Общие рекомендации по написанию переносимых операторов SQL

При написании операторов SQL для их использования в нескольких системах управления базами данных, в операторах SQL в максимально возможной степени следует применять явные формулировки. Даже если какой-либо оператор SQL поддерживается несколькими серверами, было бы ошибкой предполагать, что поведение по умолчанию одинаково в любой системе. Ниже приводятся общие рекомендации по написанию совместимых операторов SQL.

- Не используйте поведение по умолчанию, вместо этого задайте все доступные параметры.
- Используйте круглые скобки для явного указания порядка выполнения в пределах операторов и не полагайтесь на то, что существует некий стандартный порядок старшинства для операций, используемый по умолчанию.
- Для обеспечения переносимости Adaptive Server Enterprise используйте принятое в Transact-SQL соглашение о предшествовании символа @ имени переменной.
- Объявляйте переменные и курсоры в процедурах, триггерах и пакетах сразу же после оператора BEGIN. Это необходимо для Adaptive Server Anywhere, хотя в Adaptive Server Enterprise допускается ввод объявлений в произвольном месте внутри процедуры, триггера или пакета.
- Для идентификаторов в базах данных не используйте зарезервированные слова, предназначенные для внутреннего использования в Adaptive Server Enterprise или Adaptive Server Anywhere.
- Не ограничивайте пространство имен. Например, каждому индексу следует присваивать уникальное имя.

Создание совместимых таблиц

Adaptive Server Anywhere поддерживает домены, которые допускают включение определений ограничений и определений по умолчанию в определение типов данных. Также поддерживаются явные значения по умолчанию и условия CHECK в операторе CREATE TABLE. Однако не поддерживаются поименованные ограничения или поименованные значения по умолчанию.

NULL

Adaptive Server Anywhere и Adaptive Server Enterprise имеют некоторые различия в обработке NULL. В Adaptive Server Enterprise NULL в некоторых случаях обрабатывается не как нуль, а как значение.

Например, в уникальном индексе в Adaptive Server Enterprise не могут содержаться строки с нулем, поскольку они рассматривались бы как идентичные строки. Однако в Adaptive Server Anywhere уникальный индекс может содержать такие строки.

По умолчанию для столбцов в Adaptive Server Enterprise устанавливается NOT NULL, тогда как в Adaptive Server Anywhere установкой по умолчанию является NULL. Управлять этой установкой можно с помощью параметра **allow_nulls_by_default**. Для обеспечения переносимости операторов определения данных следует явно задать NULL или NOT NULL.

☞ Для получения информации об этом параметре см. раздел "Установка параметров для совместимости Transact-SQL" на стр. 383.

Временные таблицы

Создать временную таблицу можно путем ввода символа # перед оператором CREATE TABLE. Временные таблицы, объявленные в Adaptive Server Anywhere, доступны только для текущего подключения. Для получения информации об объявленных временных таблицах в Adaptive Server Anywhere см. раздел "Оператор DECLARE LOCAL TEMPORARY TABLE" (DECLARE LOCAL TEMPORARY TABLE statement) на стр. 356 в документе *"Справочник по SQL для ASA" (ASA SQL Reference Manual)*.

Физическое размещение таблицы в Adaptive Server Enterprise и в Adaptive Server Anywhere выполняется по-разному. Adaptive Server Anywhere поддерживает раздел **ON** *имя-сегмента,* но имя-сегмента относится к dbspace в Adaptive Server Anywhere.

☞ Для получения информации об операторе CREATE TABLE см. раздел "Оператор CREATE TABLE" (CREATE TABLE statement) на стр. 321 в документе *"Справочник по SQL для ASA" (ASA SQL Reference Manual)*.

Написание совместимых запросов

Существует два критерия для написания запросов, выполнение которых возможно как в базе данных Adaptive Server Enterprise, так и в базе данных Adaptive Server Anywhere:

- должны быть совместимыми типы данных, выражения и условия поиска в запросе;
- должен быть совместимым синтаксис оператора SELECT.

В этом разделе рассматриваются вопросы совместимости синтаксиса оператора SELECT, и предлагаются совместимые типы данных, выражения и условия поиска. В приведенном примере предполагается, что для QUOTED_IDENTIFIER установлено **OFF** (это является установкой по умолчанию в Adaptive Server Enterprise, но не является таковой в Adaptive Server Anywhere).

Adaptive Server Anywhere поддерживает следующее подмножество оператора SELECT на языке Transact-SQL.

Синтаксис

SELECT [**ALL** | **DISTINCT**] *список-выбора*
...[**INTO** *#имя-временной-таблицы*]
...[**FROM** *описание-таблицы* [**HOLDLOCK** | **NOHOLDLOCK**],
... *описание-таблицы* [**HOLDLOCK** | **NOHOLDLOCK**], ...]
...[**WHERE** *условие-поиска*] ...[**GROUP BY** *имя-столбца*, ...]
...[**HAVING** *условие-поиска*]
[**ORDER BY** { *выражение* | *целое число* } [**ASC** | **DESC**], ...]

Параметры

список-выбора:
*имя-таблицы.**
| *
| *выражение*
| *имя-псевдонима = выражение*
| *выражение как идентификатор*
| *выражение как T_string*
описание-таблицы:
[*владелец* .]*имя-таблицы*
...[[**AS**] *корреляционное-имя*]
...[(**INDEX** *имя_индекса* [**PREFETCH** *размер*][**LRU** | **MRU**])]
имя-псевдонима:
идентификатор | '*строка*' | "*строка*"

☞ Полное описание оператора SELECT см. в разделе "Оператор SELECT" (SELECT statement) на стр. 490 в документе *"Справочник по SQL для ASA" (ASA SQL Reference Manual)*.

Adaptive Server Anywhere не поддерживает следующие ключевые слова и разделы в синтаксисе оператора SELECT в Transact-SQL:

- ключевое слово SHARED;
- раздел COMPUTE;
- раздел FOR BROWSE;
- раздел GROUP BY ALL.

Примечания

- Раздел INTO имя_таблицы, обеспечивающий создание новой таблицы на основе результирующего набора оператора SELECT, поддерживается только для объявленных временных таблиц, имена которых начинается с #. Объявленные временные таблицы существуют только для одного подключения.
- Adaptive Server Anywhere не поддерживает расширение Transact-SQL для такого применения раздела **GROUP BY**, которое допускает ссылки на столбцы и выражения, не используемые для создания групп. В Adaptive Server Enterprise это расширение создает итоговые отчеты.
- Производится только синтаксическая разборка раздела **FOR READ ONLY** и раздела **FOR UPDATE**, однако эти разделы не действуют.
- Производится только синтаксическая разборка системных параметров описания таблицы, эти параметры не действуют.
- Adaptive Server Anywhere поддерживает ключевое слово HOLDLOCK. Это обеспечивает более жесткую совместную блокировку заданной

таблицы или представления, т. е. удержание таблицы/представления вплоть до завершения транзакции (совместная блокировка не отменяется, даже если используемая страница данных больше не требуется, вне зависимости от того, завершена или не завершена транзакция). По отношению к таблице, для которой задано HOLDLOCK, запрос выполняется на уровне изоляции 3.

- Параметр HOLDLOCK применяется только к той таблице или к тому представлению, для которой/которого он задан, и только в течение выполнения транзакции, определенной оператором, в котором используется этот параметр. Установка уровня изоляции 3 применяется по отношению к удержанию блокировки, установленному для каждого выбранного объекта в пределах транзакции. В запросе не допускается одновременный ввод параметров HOLDLOCK и NOHOLDLOCK.
- Adaptive Server Anywhere распознает ключевое слово NOHOLDLOCK, однако оно не действует.
- В Transact-SQL оператор SELECT используется для присвоения значений локальным переменным:

```
SELECT @localvar = 42
```

Соответствующим оператором в Adaptive Server Anywhere является оператор SET:

```
SET localvar = 42
```

Однако использование **SELECT** в Transact-SQL для присвоения значений переменным поддерживается в пакетах.

- **Adaptive Server Enterprise** не поддерживает следующие разделы в синтаксисе оператора SELECT:
- **INTO** список-хост-переменных;
- **INTO** список-переменных;
- запросы, заключенные в скобки.
- В Adaptive Server Enterprise для соединений используются операции соединений в разделе **WHERE**, но не раздел FROM и условие **ON**.

Совместимость соединений

В Transact-SQL соединения применяются в разделе **WHERE**. При этом используется следующий синтаксис:

начало выбора, обновления, вставки, удаления или подзапросы
FROM { *список-таблиц* | *список-представлений* } **WHERE** [**NOT**]
...[*имя-таблицы*. | *имя-представления*.]*имя-столбца*
операция-соединения
...[*имя-таблицы*. | *имя-представления*.]*имя_столбца*
...[{ **AND** | **OR** } [**NOT**]
...[*имя-таблицы*. | *имя-представления*.]*имя_столбца*
операция-соединения
[*имя-таблицы*.| *имя-представления*.]*имя-столбца*
]...
конец выбора, обновления, вставки, удаления или подзапросы

Операция-соединения в разделе WHERE – это любая из операций сравнения или любая из следующих **операций внешнего соединения:**

- *= операция внешнего соединения слева;
- =* операция внешнего соединения справа.

В Adaptive Server Anywhere поддерживаются операции внешнего соединения, предусмотренные в Transact-SQL (в качестве альтернативы внутреннему синтаксису SQL/92). В пределах одного запроса использование разных диалектов не допускается. Это правило также распространяется на представления, используемые запросом. В запросе внешнего соединения на представлении должен использоваться тот же диалект, который использовался в запросе определения представления.

В Adaptive Server Anywhere также применяется синтаксис SQL/92 для соединений, не являющихся внешними соединениями. Согласно этому синтаксису соединения помещаются в раздел FROM, а не в раздел **WHERE**.

☞ Для получения информации о соединениях в Adaptive Server Anywhere и в стандартах SQL **ANSI/ISO** см. раздел "Соединения: извлечение данных из нескольких таблиц" (Joins: Retrieving Data from Several Tables) на стр. 223 и "Раздел FROM" (**FROM** clause) на стр. 402 в документе *"Справочник по SQL для ASA" (ASA SQL Reference Manual)*.

☞ Для получения дополнительной информации о совместимости соединений для Transact-SQL см. раздел "Внешние соединения в Transact-SQL (*= или =*)" на стр. 240.

Обзор языка процедур Transact-SQL

Язык хранимых процедур - часть SQL, используемая в хранимых процедурах, триггерах и пакетах.

Adaptive Server Anywhere поддерживает значительную часть языка хранимых процедур Transact-SQL в дополнение к диалекту Watcom-SQL, основанному на SQL/92.

Обзор хранимых процедур Transact-SQL

Основанный на проекте стандарта ISO/ANSI язык хранимых процедур в Adaptive Server Anywhere отличается от диалекта Transact-SQL по многим аспектам. Многие из понятий и функций совпадают, но отличается синтаксис. Поддержка Transact-SQL в Adaptive Server Anywhere основывается на использовании схожих понятий для автоматического перевода диалектов. Тем не менее, процедура должна быть написана с использованием только одного из двух диалектов; смешивание двух диалектов не допускается.

Поддержка Adaptive Server Anywhere для хранимых процедур Transact-SQL

В Adaptive Server Anywhere существуют различные возможности по поддержке хранимых процедур Transact-SQL, а именно:

- передаваемые параметры;
- возвращаемые результирующие наборы;
- возвращаемая информация о статусе;
- значения по умолчанию для параметров;
- управляющие операторы;
- обработка ошибок.

Обзор триггеров Transact-SQL

Для совместимости триггеров требуется совместимость их средств и синтаксиса. В этом разделе приводится основная информация о совместимости средств триггеров в Adaptive Server Anywhere и Transact-SQL.

Adaptive Server Enterprise выполняет триггер по завершении оператора с триггером. Эти триггеры являются триггерами **уровня завершения операторов**. Adaptive Server Anywhere поддерживает как триггеры **уровня строки** (которые выполняются до или после модификации каждой строки), так и триггеры уровня операторов (которые выполняются после полного завершения оператора).

Триггеры уровня строки не относятся к средствам совместимости Transact-SQL и рассматриваются в разделе "Использование процедур, триггеров и пакетов" на стр. 495.

Описание неподдерживаемых или отличающихся триггеров Transact-SQL

К средствам триггеров Transact-SQL, которые не поддерживаются или отличаются в Adaptive Server Anywhere, относится следующее:

- **Триггеры, запускающие другие триггеры.** Предположим, что триггер выполняет такое действие, которое, если бы оно было выполнено непосредственно пользователем, приводит к запуску другого триггера.

  Adaptive Server Anywhere и Adaptive Server Enterprise несколько по-разному реагируют на эту ситуацию. По умолчанию в Adaptive Server Enterprise триггеры запускают другие триггеры вплоть до устанавливаемого уровня вложенности (уровень вложенности имеет значение по умолчанию 16). Пользователь может управлять уровнем вложенности в Adaptive Server Enterprise с помощью параметра **nested triggers** (вложенные триггеры). В Adaptive Server Anywhere триггеры могут запускать другие триггеры без ограничений (но только до тех пор, пока достаточно памяти).
- **Самостоятельный запуск триггера.** Предположим, что триггер выполняет такое действие, которое, если бы оно было выполнено непосредственно пользователем, приводит к запуску этого же самого триггера. Adaptive Server Anywhere и Adaptive Server Enterprise несколько по-разному реагируют на эту ситуацию. В Adaptive Server Anywhere самостоятельный запуск триггеров, не являющихся триггерами Transact-SQL, осуществляется рекурсивно, в то время как самостоятельный запуск триггеров диалекта Transact-SQL не является рекурсивным.

  По умолчанию в Adaptive Server Enterprise триггер не вызывает сам себя рекурсивно, но пользователь может установить параметр **self_recursion,** для того чтобы разрешить рекурсивный самостоятельный вызов триггеров.
- **Оператор ROLLBACK в триггерах.** В Adaptive Server Enterprise разрешается использование оператора **ROLLBACK TRANSACTION** внутри триггеров. Этот оператор выполняет откат всей транзакции, в состав которой входит данный триггер. В Adaptive Server Anywhere не разрешается использование операторов ROLLBACK (или ROLLBACK TRANSACTION) в триггерах. Это обусловлено тем, что инициируемое триггером действие и сам триггер вместе составляют атомарный оператор.

  В Adaptive Server Anywhere применяется совместимый с Adaptive Server Enterprise оператор **ROLLBACK TRIGGER**, обеспечивающий отмену действий внутри триггеров. См. раздел "Оператор **ROLLBACK TRIGGER" (ROLLBACK TRIGGER statement)** на стр. 488 документа *"Справочник по SQL для ASA" (ASA SQL Reference Manual)*.

Обзор пакетов Transact-SQL

В Transact-SQL **пакетом** является набор операторов SQL, совместно направляемых в систему и выполняемых последовательно, друг за другом. Пакеты могут храниться в командных файлах. Утилита Interactive SQL в Adaptive Server Anywhere и утилита isql в Adaptive Server Enterprise

предоставляет схожие возможности по выполнению пакетов в интерактивном режиме.

В пакетах также могут использоваться управляющие операторы, используемые в процедурах. В Adaptive Server Anywhere поддерживается использование управляющих операторов в пакетах, а также использование групп операторов без разделителей, которые заканчиваются оператором GO, обозначающим конец пакета (в стиле Transact-SQL).

Применительно к пакетам, сохраненным в командных файлах, Adaptive Server Anywhere поддерживает использование параметров в командных файлах. Adaptive Server Enterprise использование параметров не поддерживает.

☞ Для получения информации о параметрах см. раздел "Оператор PARAMETERS [Interactive SQL]" (PARAMETERS statement [Interactive SQL]) на стр. 457 в документе *"Справочник по SQL для ASA" (ASA SQL Reference Manual)*.

Автоматический перевод хранимых процедур

Помимо поддержки альтернативного синтаксиса Transact-SQL, в Adaptive Server Anywhere предусмотрены средства, обеспечивающие перевод операторов, написанных с использованием диалектов Transact-SQL и Watcom-SQL. Функции, возвращающие информацию об операторах SQL и обеспечивающие автоматический перевод операторов SQL:

- **SQLDialect (оператор).** Возвращает **Watcom-SQL** или **Transact-SQL.**
- **WatcomSQL(оператор).** Возвращает синтаксис Watcom-SQL для оператора.
- **TransactSQL(оператор).** Возвращает синтаксис Transact-SQL для оператора.

Доступ к этим функциям может быть получен с помощью оператора Select в утилите Interactive SQL. Например:

```
select SqlDialect('select * from employee')
```

Этот оператор возвращает значение Watcom-SQL.

Использование Sybase Central для перевода хранимых процедур

В Sybase Central существуют средства для создания, просмотра и изменения процедур и триггеров.

❖ Перевод хранимой процедуры с использованием Sybase Central

1. Выполните подключение к базе данных с использованием Sybase Central (в качестве владельца процедуры, которую необходимо изменить, или в качестве администратора БД).
2. Откройте папку Procedures & Functions.
3. Нажмите правой кнопкой мыши на процедуру, которую необходимо перевести, и во всплывающем меню выберите одну из команд Open As (в зависимости от требуемого диалекта).

   Процедура появляется в Code Editor в выбранном диалекте. Если выбранный диалект отличается от диалекта, в котором сохранена процедура, сервер переводит процедуру в требуемый диалект. Все непереведенные строки выводятся как комментарии.
4. Исправьте все непереведенные строки требуемым образом.
5. По завершении работы выберите File➤Save/Execute in Database для сохранения перевода в базе данных. Кроме того, текст можно экспортировать в файл для последующего редактирования вне Sybase Central.

Возвращение результирующих наборов из процедур Transact-SQL

Для задания результирующих наборов в Adaptive Server Anywhere используется раздел RESULT. В процедурах Transact-SQL в среду вызова возвращаются имена столбцов или имена псевдонимов первого запроса.

Пример процедуры Transact-SQL

Следующая процедура Transact-SQL иллюстрирует то, как хранимые процедуры Transact-SQL возвращают результирующие наборы:

```
CREATE PROCEDURE showdept (@deptname varchar(30))
AS
    SELECT employee.emp_lname, employee.emp_fname
    FROM department, employee
    WHERE department.dept_name = @deptname
    AND department.dept_id = employee.dept_id
```

Пример процедуры Watcom-SQL

Ниже приводится соответствующая процедура Adaptive Server Anywhere:

```
CREATE PROCEDURE showdept(in deptname varchar(30))
RESULT ( lastname char(20), firstname char(20))
BEGIN
    SELECT employee.emp_lname, employee.emp_fname
    FROM department, employee
    WHERE department.dept_name = deptname
    AND department.dept_id = employee.dept_id
END
```

☞ Для получения дополнительной информации о процедурах и результатах см. раздел "Возвращение результатов из процедур" на стр. 527.

Переменные в процедурах Transact-SQL

Для присвоения значений переменным в процедурах в Adaptive Server Anywhere используется оператор SET. В Transact-SQL значения присваиваются с использованием оператора SELECT с пустым списком таблиц. Следующая простая процедура иллюстрирует то, как используется синтаксис Transact-SQL:

```
CREATE PROCEDURE multiply @mult1 int, @mult2 int, @result int
output AS
SELECT @result = @mult1 * @mult2
```

Эту процедуру можно вызвать следующим образом:

```
CREATE VARIABLE @product int go
EXECUTE multiply 5, 6, @product OUTPUT go
```

По завершении выполнения процедуры переменная **@product** получает значение 30.

☞ Для получения дополнительной информации об использовании оператора SELECT для присвоения значений переменным см. раздел "Написание совместимых запросов" на стр. 390. Для получения дополнительной информации об использовании оператора SET для присвоения значений переменным см. раздел "Оператор SET" (SET statement) на стр. 495 в документе *"Справочник по SQL для ASA" (ASA SQL Reference Manual)*.

Обработка ошибок в процедурах Transact-SQL

Обработка ошибок процедуры по умолчанию в диалектах Transact-SQL и Watcom-SQL выполняется по-разному. При возникновении ошибки в процедуре диалекта Watcom-SQL по умолчанию происходит выход с возвращением значений SQLSTATE и SQLCODE в среду вызова.

Явная обработка ошибок может быть встроена в хранимые процедуры Watcom-SQL с помощью оператора EXCEPTION. Другой вариант: с помощью оператора EXCEPTION RESUME в процедуру можно ввести инструкцию на продолжение выполнения со следующего оператора, если в процедуре встречается ошибка.

При возникновении ошибки в процедуре диалекта Transact-SQL выполнение продолжается со следующего оператора. В глобальной переменной **@@error** сохраняется состояние ошибки последнего выполненного оператора. Эту переменную можно проверять вслед за оператором и, в зависимости от результата проверки, осуществлять принудительный выход из процедуры. Например, в случае появления ошибки выход из процедуры может обеспечить следующий оператор:

```
IF @@error != 0 RETURN
```

По завершении выполнения процедуры возвращаемое значение информирует об успешном или неуспешном результате выполнения. Это возвращаемое значение (информация о статусе) является целым числом, которое может быть получено следующим образом:

```
DECLARE @status INT

EXECUTE @status = proc_sample IF @status = 0 PRINT 'procedure
succeeded' ELSE

PRINT 'procedure failed'
```

В следующей таблице приведены возвращаемые значения для процедуры и их смысл:

Значение	Смысл
0	Процедура выполнена без ошибок
-1	Отсутствует объект
-2	Ошибка типа данных
-3	Процесс выбран вследствие взаимоблокировки
-4	Ошибка полномочий
-5	Синтаксическая ошибка
-6	Прочие ошибки пользователя
-7	Ошибка ресурса, например, недостаточно места
-8	Устранимая внутренняя ошибка

Значение	Смысл
-9	Достижение системного предела
-10	Неустранимая внутренняя несогласованность
-11	Неустранимая внутренняя несогласованность
-12	Повреждение таблицы или индекса
-13	Повреждение базы данных
-14	Аппаратная ошибка

Для возвращения других целочисленных значений, смысл которых устанавливается пользователем по своему усмотрению, может применяться оператор RETURN.

Использование оператора RAISERROR в процедурах

Оператор RAISERROR диалекта Transact-SQL предназначен для генерации ошибок, определяемых пользователем. Функции этого оператора схожи с функциями оператора SIGNAL.

Описание оператора RAISERROR см. в разделе "Оператор RAISERROR [T-SQL]" (RAISERROR statement [T-SQL]) на стр. 465 в документе *"Справочник по SQL для ASA" (ASA SQL Reference Manual)*.

Сам по себе оператор RAISERROR не приводит к выходу из процедуры, но он может быть объединен с оператором RETURN или с оператором проверки глобальной переменной **@@error** для управления выполнением процедуры после появления определяемой пользователем ошибки.

Если в параметре базы данных ON_TSQL_ERROR установлено CONTINUE, то оператор RAISERROR не сообщает об ошибке, при появлении которой прекращается выполнение процедуры. В этом случае по завершении процедуры сохраняется код состояния и сообщение RAISERROR, а также возвращается последнее значение RAISERROR. Если процедура, приведшая к инициированию RAISERROR, была вызвана из другой процедуры, то возвращение значения RAISERROR происходит после завершения вызывающей процедуры.

По завершении процедуры информация о промежуточных состояниях и кодах RAISERROR утрачивается. Если в момент возвращения информации происходит другая ошибка (при обработке RAISERROR), то возвращается информация об этой ошибке, а информация RAISERROR утрачивается. Приложение может запрашивать информацию о промежуточных состояниях RAISERROR путем проверки глобальной переменной **@@error** в различные моменты выполнения процедуры.

Обработка ошибок в стиле Transact-SQL при использовании диалекта Watcom-SQL

При использовании Watcom-SQL процедура обработки ошибок может быть составлена в стиле Transact-SQL. Для этого вводится раздел ON EXCEPTION RESUME в операторе CREATE PROCEDURE:

```
CREATE PROCEDURE sample_proc() ON EXCEPTION RESUME
BEGIN
    ...
END
```

Наличие раздела ON EXCEPTION RESUME не допускает явное выполнение обработки по коду обработки исключений, поэтому не следует использовать эти два раздела вместе.

ГЛАВА 13

Отличия от других диалектов SQL

Об этой главе

Adaptive Server Anywhere полностью соответствует требованиям основанного на SQL-92 федерального стандарта по средствам обработки информации (FIPS PUB) 127.

Система Adaptive Server Anywhere в минимальной конфигурации совместима со стандартом ISO/ANSI SQL-92 и, с незначительными исключениями, с базисными спецификациями SQL-99.

В этой главе описываются те средства Adaptive Server Anywhere, которые, как правило, отсутствуют в других реализациях SQL.

Содержание

Средства SQL в Adaptive Server Anywhere

Ниже описываются средства SQL, поддерживаемые в Adaptive Server Anywhere, которые отсутствуют в большинстве других реализаций SQL.

Преобразования типов

Поддерживается полное преобразование типов. Любой тип данных может сравниваться с любым другим типом данных или использоваться в любом выражении совместно с любым другим типом данных.

Дата

В Adaptive Server Anywhere поддерживаются такие типы даты, значений времени и штампов времени, в которых указываются год, месяц, день, часы, минуты, секунды и доли секунды. Для вставки/обновления полей даты или для сравнения с полями даты поддерживается свободный формат даты.

Кроме того, применительно к дате возможно выполнение следующих операций:

- **Дата + целое число:** добавление определенного числа дней к дате;
- **Дата - целое число:** вычитание определенного числа дней из даты;
- **Дата – дата:** вычисление числа дней между двумя датами;
- **Дата + время:** создание штампа времени из значений даты и времени.

Кроме того, предусмотрено множество других функций для обработки значений даты и времени. Описание этих функций см. в разделе "Функции SQL" (SQL Functions) на стр. 91 в документе *"Справочник по SQL для ASA" (ASA SQL Reference Manual)*.

Целостность

В Adaptive Server Anywhere поддерживаются объектная целостность и ссылочная целостность.

Эта поддержка реализована с помощью следующих двух расширений команд CREATE TABLE и ALTER TABLE:

```
PRIMARY KEY ( имя-столбца, ... )
[NOT NULL] FOREIGN KEY [имя-роли]
             [(имя-столбца, ...)]
          REFERENCES имя-столбца [(имя-столбца, ...)]
             [ CHECK ON COMMIT ]
```

Раздел PRIMARY KEY объявляет первичный ключ для отношения. В этом случае Adaptive Server Anywhere устанавливает уникальность первичного ключа и обеспечивает то, что ни один из столбцов в первичном ключе не содержит значение NULL.

Раздел FOREIGN KEY определяет взаимосвязь между данной таблицей и другой таблицей. Эта взаимосвязь представлена столбцом (или столбцами) в данной таблице, который (которые) должен (должны) содержать значения в первичном ключе другой таблицы. В этом случае система обеспечивает ссылочную целостность для этих столбцов, даже если осуществляется изменение этих столбцов или вставка строки в данную таблицу. Эти столбцы проходят проверку того, что один или несколько из них содержат NULL, или что их значения совпадают со значениями соответствующих столбцов для некоторой строки в первичном ключе другой таблицы. Для получения дополнительной информации см. раздел "Оператор CREATE TABLE" (CREATE TABLE statement) на стр. 321 в документе *"Справочник по SQL для ASA" (ASA SQL Reference Manual)*.

Соединения

Adaptive Server Anywhere обеспечивает **автоматические соединения** между таблицами. В дополнение к операторам соединения NATURAL и OUTER, поддерживаемым в других реализациях, Adaptive Server Anywhere обеспечивает создание межтабличных соединений KEY, которые основаны на взаимосвязи по внешнему ключу. За счет этого уменьшается степень сложности раздела WHERE при выполнении соединений.

Обновление

В Adaptive Server Anywhere допускается ссылка на несколько таблиц в команде UPDATE. Также возможно обновление представлений, определенных в нескольких таблицах. В других реализациях SQL, как правило, обновление в соединенных таблицах не допускается.

Изменение таблиц

Реализовано расширение команды ALTER TABLE. В дополнение к изменениям объектной и ссылочной целостности также допускаются следующие типы изменений:

```
ADD тип-данных столбца MODIFY

тип-данных столбца DELETE

столбец RENAME новое-имя-таблицы

RENAME старый-столбец TO новый-столбец
```

MODIFY можно использовать для изменения максимальной длины символьного столбца, а также для преобразования одного типа данных в другой. Для получения дополнительной информации см. раздел "Оператор ALTER TABLE" (ALTER TABLE statement) на стр. 219 в документе *"Справочник по SQL для ASA" (ASA SQL Reference Manual)*.

Подзапросы, в которых можно использовать выражения

В Adaptive Server Anywhere допускается использование подзапросов во всех случаях, когда допускается использование выражений. Во многих реализациях SQL использование подзапросов разрешается только в правой части операции сравнения. Например, следующая команда может использоваться в Adaptive Server Anywhere, но не может использоваться в большинстве других реализаций SQL.

```
SELECT   emp_lname,
         emp_birthdate,
         (   SELECT skill
             FROM department
             WHERE emp_id = employee.emp_ID
             AND dept_id = 200 )
FROM employee
```

Дополнительные функции

В Adaptive Server Anywhere поддерживается ряд функций, отсутствующих в определении SQL ANSI. Полный список доступных функций см. в разделе "Функции SQL" (SQL Functions) на стр. 91 в документе *"Справочник по SQL для ASA" (ASA SQL Reference Manual)*.

Курсоры

При использовании Embedded SQL позиции курсоров могут произвольно изменяться оператором FETCH. Курсоры могут перемещаться в прямом или обратном направлении по отношению к текущей позиции или в соответствии с определенным числом записей от начала или конца курсора.

ЧАСТЬ 4

Доступ к данным и перемещение данных

В этой части описываются способы загрузки и выгрузки базы данных, а также порядок получения доступа к удаленной базе данных.

ГЛАВА 14

Импорт и экспорт данных

Об этой главе

В этой главе описываются инструментальные средства и утилиты Adaptive Server Anywhere (в том числе SQL, Interactive SQL, утилита *dbunload* и мастер Sybase Central), с помощью которых решаются задачи импорта и экспорта данных.

Содержание

Введение в импорт и экспорт данных

Передача больших объемов данных в собственную базу данных или их передача в обратном направлении может потребоваться в разных ситуациях, например:

- импорт начального набора данных в новую базу данных;
- экспорт данных из собственной базы данных для их использования в других приложениях (в частности, в электронных таблицах);
- построение новых копий базы данных, возможно с изменением структуры;
- создание извлечений из базы данных для репликации или синхронизации.

Учет производительности при перемещении данных

Команды INPUT и OUTPUT в Interactive SQL являются внешними (клиентскими) командами по отношению к базе данных. Если ISQL работает на другой машине, т. е. не на сервере базы данных, то пути к считываемым или записываемым файлам указываются по отношению к клиенту. INPUT регистрируется в журнале транзакций как отдельный оператор INSERT при каждом чтении строки. В результате команда INPUT выполняется значительно медленнее, чем LOAD TABLE. Это также означает, что при выполнении INPUT запускаются триггеры ON INSERT. Если значения отсутствуют, то в строки NULLABLE вводится NULL, в числовые столбцы, не являющиеся столбцами NULLABLE, вводится 0 (нуль), а в нечисловые столбцы, не являющиеся столбцами NULLABLE, вводится пустая строка. Оператор OUTPUT рекомендуется использовать в том случае, когда необходимо обеспечить совместимость, поскольку этот оператор может записывать результирующий набор оператора SELECT в любой из имеющихся форматов файлов.

В то же время, операторы LOAD TABLE, UNLOAD TABLE и UNLOAD являются внутренними (серверными) операторами по отношению к базе данных. Пути к записываемым или считываемым файлам указываются по отношению к серверу базы данных. Серверу базы данных передается только команда, и вся дальнейшая обработка выполняется на сервере. Оператор LOAD TABLE регистрируется в журнале транзакций как отдельная команда. Количество столбцов в файле данных должно совпадать с количеством столбцов в загружаемой таблице. При отсутствии значений в столбцы со значением по умолчанию вводится NULL, 0 (нуль) или пустая строка, если в параметре DEFAULTS установлено OFF (по умолчанию), или же вводится значение по умолчанию, если в параметре DEFAULTS установлено ON. При внутреннем импорте/экспорте предоставляется доступ только к файлам в текстовом формате и в формате BCP, однако этот метод обеспечивает большее быстродействие.

Для загрузки большого объема информации в базу данных может потребоваться довольно много времени. Однако для ускорения этого процесса можно воспользоваться приведенными ниже рекомендациями.

- Если используется оператор LOAD TABLE, то режим массовых операций (запуск сервера с ключом -b) не требуется.

- Если используется команда INPUT, запустите Interactive SQL или клиентское приложение на той же машине, на которой работает сервер. Загрузка данных через сеть приводит к дополнительным затратам, зависящим от условий на конкретной сети. В некоторых случаях целесообразно загружать новые данные в часы наименьшей нагрузки на сеть.

- Файлы данных следует поместить отдельно от базы данных на отдельном физическом диске. Это позволит избежать излишних перемещений считывающей головки дисковода во время загрузки.

- Если используется команда INPUT, сервер следует запустить с ключом -b для перехода в режим массовых операций. В этом режиме сервер не сохраняет журнал откатов или журнал транзакций, а также не выполняет автоматически команду COMMIT перед командами определения данных и не блокирует какие-либо записи.

  Если используется ключ -b, сервер разрешает выполнение только одного подключения.

  Без журнала откатов использование точек сохранения невозможно, и прерывание выполнения команды всегда приводит к откату транзакции. Без автоматического выполнения COMMIT по команде ROLLBACK отменяется все, что следует за последней явной командой COMMIT.

  Без журнала транзакций изменения не регистрируются. Перед использованием режима массовых операций и после его использования следует выполнить резервное копирование базы данных, поскольку в этом режиме база данных не имеет защиты при отказе носителя информации. Для получения дополнительной информации см. раздел "Резервное копирование и восстановление данных" (Backup and Data Recovery) на стр. 295 в документе *"Руководство по администрированию баз данных ASA" (ASA Database Administration Guide)*.

  Если используются данные, для которых требуется много операций подтверждения, то применение ключа -b может замедлить работу с базой данных. По каждой команде COMMIT сервер устанавливает контрольную точку; частая установка контрольных точек может замедлить работу сервера.

- Возможно увеличение размера базы данных, как описывается в разделе "Оператор ALTER DBSPACE" (ALTER DBSPACE statement) на стр. 199 в документе *"Справочник по SQL для ASA" (ASA SQL Reference Manual)*. С помощью этой команды можно заблаговременно выполнить масштабное расширение базы данных (в обычной ситуации, когда требуется дополнительное пространство, единовременное расширение составляет 32 страницы). Это расширение позволяет не только увеличить производительность при загрузке большого объема данных, но и повысить степень непрерывности базы данных в файловой системе.

- Для загрузки данных можно использовать временные таблицы. Локальные или глобальные временные таблицы целесообразно использовать тогда, когда требуется неоднократная загрузка набора данных или когда необходимо объединить таблицы с разной структурой.

Импорт и экспорт данных

Возможен импорт отдельных таблиц или их фрагментов из файлов базы данных других форматов или из файлов ASCII. В зависимости от конкретного формата добавляемых данных таблица может быть создана либо до импорта, либо во время импорта. Импорт целесообразно выполнять тогда, когда в базу данных необходимо единовременно добавить большой объем данных.

Отдельные таблицы и результаты запросов можно экспортировать в формат ASCII или в другие форматы, поддерживаемые другими программами баз данных. Экспорт целесообразно использовать в том случае, когда требуется совместное использование крупных фрагментов базы данных или извлечение фрагментов базы данных по определенным критериям.

Несмотря на то, что процедуры импорта и экспорта в Adaptive Server Anywhere одновременно обрабатывают только одну таблицу, можно создать такие сценарии, которые автоматизируют эти процедуры и обеспечивают последовательное выполнение импорта и экспорта данных с использованием множества таблиц.

Данные можно вставлять (добавлять) в таблицы, также можно заменять данные в таблицах. В некоторых случаях одновременно с импортом данных можно создавать новые таблицы. Однако если создается новая база данных, то по соображениям производительности рекомендуется использовать процедуру загрузки данных вместо процедуры импорта.

Возможен экспорт результатов запросов, данных таблиц или схемы таблиц. Однако если экспортируется вся база данных, то по соображениям производительности рекомендуется использовать процедуру выгрузки базы данных вместо процедуры экспорта данных.

☞ Для получения дополнительной информации о загрузке и выгрузке целых баз данных см. раздел "Восстановление баз данных" на стр. 429.

Импорт и экспорт файлов между Adaptive Server Anywhere и Adaptive Server Enterprise выполняется с использованием раздела BCP FORMAT.

☞ Для получения дополнительной информации см. раздел "Совместимость с Adaptive Server Enterprise" на стр. 442.

Форматы данных

Interactive SQL поддерживает следующие форматы импортируемых и экспортируемых файлов:

Формат файла	Описание	Доступность для импорта	Доступность для экспорта
ASCII	Текстовый файл, в котором одна текстовая строка соответствует одной табличной строке; между значениями вводятся разделители. Строчные значения могут заключаться в апострофы (одиночные кавычки). Этот формат используется командами LOAD TABLE и UNLOAD TABLE.	✓	✓
DBASEII	Формат DBASE II	✓	✓
DBASEIII	Формат DBASE III	✓	✓
Excel 2.1	Формат Excel 2.1	✓	✓
FIXED	Фиксированный формат записей данных. Ширина каждого столбца устанавливается либо в определении типа столбца, либо в параметре.	✓	✓
FOXPRO	Формат FoxPro	✓	✓
HTML	Формат HTML (Язык разметки гипертекста)		✓
LOTUS	Формат рабочего пространства Lotus	✓	✓
SQL-Statements	Формат операторов SQL. Этот формат может использоваться в качестве аргумента в операторе READ	С использованием только оператора READ	✓
XML	Сгенерированный XML-файл закодирован с использованием UTF-8 и содержит DTD. Двоичные значения кодируются блоками CDATA; двоичные данные представлены в виде символьных строк, состоящих из двух шестнадцатеричных цифр.	Нет	✓

Структуры таблиц для импорта

Структура данных, загружаемых в таблицу, не всегда соответствует структуре целевой таблицы, что может привести к возникновению проблем при импорте данных. Например, типы данных в столбце могут отличаться или иметь другой порядок, либо в импортируемых данных могут присутствовать дополнительные значения, которые не соответствуют столбцам в целевой таблице.

Изменение структуры таблицы или данных

Если структура импортируемых данных не соответствует структуре целевой таблицы, возможно несколько решений этой проблемы: изменение структуры столбцов в таблице с помощью оператора LOAD TABLE; изменение структуры импортируемых данных для приведения их в соответствие таблице (с помощью видоизмененного оператора INSERT и глобальной временной таблицы); определение конкретного набора или порядка столбцов с помощью оператора INPUT.

Разрешение значений NULL в столбцах

Если в импортируемом файле содержатся данные для некоторого поднабора столбцов в таблице, или если столбцы имеют иной порядок, можно воспользоваться параметром DEFAULTS оператора LOAD TABLE для заполнения пустых табличных структур и для слияния несовпадающих табличных структур.

Если в параметре DEFAULTS установлено OFF, то любому столбцу, не указанному в списке столбцов, присваивается значение NULL. Если в параметре DEFAULTS установлено OFF, и столбец не может содержать значения NULL и отсутствует в списке столбцов, сервер базы данных выполняет преобразование пустой строки для приведения ее в соответствие типу столбца. Если в параметре DEFAULTS установлено ON, и столбец имеет значение по умолчанию, то используется это значение.

Например, для загрузки двух столбцов в таблицу служащих и для установки значений по умолчанию (при наличии таковых) в остальных столбцах оператор LOAD TABLE должен выглядеть следующим образом:

```
LOAD TABLE employee (emp_lname, emp_fname)
FROM 'new_employees.txt'
DEFAULTS ON
```

Слияние различных табличных структур

Изменить структуру импортируемых данных для приведения их в соответствие таблице можно с помощью видоизмененного оператора INSERT и глобальной временной таблицы.

❖ **Загрузка данных, имеющих другую структуру, с использованием глобальной временной таблицы**

1. В области SQL Statements окна Interactive SQL создайте глобальную временную таблицу, структура которой соответствует структуре входного файла.

   Для создания глобальной временной таблицы используется оператор CREATE TABLE.

2. С помощью оператора LOAD TABLE загрузите данные в глобальную временную таблицу.

   После закрытия подключения к базе данных данные в глобальной временной таблице не сохраняются. Однако определение таблицы остается. Это определение можно использовать при следующем подключении к базе данных.

3. С помощью оператора INSERT с разделом FROM SELECT извлеките данные из временной таблицы и занесите полученные сводные данные в одну или несколько постоянных таблиц базы данных.

Ошибки преобразования при импорте

В данных, загружаемых из внешних источников, могут встречаться ошибки. Например, могут встретиться недопустимые даты и недопустимые числа. С

помощью параметра базы данных CONVERSION_ERROR можно игнорировать ошибки преобразования и сводить их в значения NULL.

☞ Для получения дополнительной информации об установке параметров базы данных в Interactive SQL см. раздел "Оператор SET OPTION" (SET OPTION statement) на стр. 503 в документе "Справочник по SQL для ASA" (ASA SQL Reference Manual) или раздел "Параметр CONVERSION_ERROR" (CONVERSION_ERROR option) на стр. 547 в документе *"Руководство по администрированию баз данных ASA" (ASA Database Administration Guide)*.

Вывод значений NULL

Часто требуется извлечь данные для их использования в других программных продуктах. Поскольку другие программы могут не понимать значения NULL, предусмотрены два способа вывода этих значений. Можно использовать либо параметр NULLS в Interactive SQL, либо функцию IFNULL. В обоих случаях вместо значения NULL выводится определенное конкретное значение.

Для установки поведения по умолчанию или изменения выводимого значения для данного сеанса следует использовать параметр NULLS в Interactive SQL. Для использования выводимого значения в определенном экземпляре или запросе следует использовать функцию IFNULL.

Определение способа вывода значений NULL расширяет совместимость с другими программами.

❖ **Определение вывода значений NULL (Interactive SQL)**

1. В окне Interactive SQL выберите Tools➤Options для открытия диалога Options.
2. Выберите закладку Results.
3. В поле Display Null Values As введите значение, которое должно использоваться для замещения значений Null.
4. Нажмите Make Permanent, если внесенные изменения в дальнейшем должны использоваться по умолчанию, или нажмите OK, если внесенные изменения должны использоваться только в данном сеансе.

   ☞ Для получения дополнительной информации об установке параметров в Interactive SQL см. раздел "Оператор SET OPTION" (SET OPTION statement) на стр. 503 в документе *"Справочник по SQL для ASA" (ASA SQL Reference Manual)*.

Импорт

Ниже представлен обзор инструментальных средств импорта, а также приводятся инструкции по импорту баз данных, данных и таблиц.

Инструментальные средства импорта

Для поддержки импорта данных предусмотрен целый ряд инструментальных средств.

Мастер импорта Interactive SQL

Обратиться к мастеру импорта можно путем выбора Data➤Import в меню Interactive SQL. Мастер предоставляет интерфейс, позволяющий выбрать импортируемый файл, формат файла и целевую таблицу для размещения данных. Данные можно импортировать в существующую таблицу, либо с помощью мастера можно создать и сконфигурировать совершенно новую таблицу.

Использовать мастер импорта Interactive SQL следует в том случае, когда при импорте данных (не в текстовом формате) удобнее работать с графическим интерфейсом, или когда во время импорта данных необходимо создать таблицу.

Оператор INPUT

Выполнение оператора INPUT производится из области SQL Statements окна Interactive SQL. Оператор INPUT позволяет импортировать данные из файлов различных форматов в одну или несколько таблиц. Можно выбрать формат ввода по умолчанию или же задать формат файла для каждого оператора INPUT.

Interactive SQL может выполнить командный файл, содержащий несколько операторов INPUT.

Если файл данных записан в формате DBASE, DBASEII, DBASEIII, FOXPRO или LOTUS, и таблица не существует, то эта таблица будет создана. Если с использованием оператора INPUT выполняется импорт большого объема данных, производительность системы снижается, поскольку оператор INPUT обеспечивает запись всех действий в журнале транзакций.

Пользуйтесь оператором INPUT в Interactive SQL в тех случаях, когда необходимо импортировать данные в одну или несколько таблиц, когда необходимо автоматизировать процесс импорта с помощью командного файла, а также когда необходимо импортировать данные в формате, отличном от текстового формата.

☞ Для получения дополнительной информации см. раздел "Оператор INPUT [Interactive SQL]" (INPUT statement [Interactive SQL]) на стр. 427 в документе *"Справочник по SQL для ASA" (ASA SQL Reference Manual)*.

Оператор LOAD TABLE

Оператор LOAD TABLE обеспечивает эффективный импорт данных только в таблицу в форматах text/ASCII/FIXED. Таблица должна существовать, количество столбцов в таблице должно совпадать с количеством полей во входном файле, для таблицы должны быть определены совместимые типы данных. Импорт при использовании оператора LOAD TABLE происходит построчно (одна табличная строка на одну файловую строку), между значениями вводятся разделители.

Оператор LOAD TABLE следует использовать тогда, когда требуется импортировать данные в текстовом формате. При выборе между оператором

INPUT и оператором LOAD TABLE следует учитывать, что использование оператора LOAD TABLE повышает производительность системы.

☞ Для получения дополнительной информации см. раздел "Оператор LOAD TABLE" (LOAD TABLE statement) на стр. 438 в документе *"Справочник по SQL для ASA" (ASA SQL Reference Manual)*.

Оператор INSERT

Данные, которые необходимо внести в таблицу, непосредственно вводятся в операторе INSERT, поэтому этот способ ввода рассматривается как интерактивный ввод. Форматы файлов при этом значения не имеют. Оператор INSERT также можно использовать при удаленном доступе к данным для импорта данных из другой базы данных, а не из файла.

Пользуйтесь оператором INSERT в тех случаях, когда требуется импорт небольшого объема данных в одну таблицу.

☞ Для получения дополнительной информации см. раздел "Оператор INSERT" (INSERT statement) на стр. 431 в документе *"Справочник по SQL для ASA" (ASA SQL Reference Manual)*.

Таблицы прокси

Возможен непосредственный импорт данных из другой базы данных. С помощью средств Adaptive Server Anywhere, обеспечивающих доступ к удаленным данным, можно создать таблицу прокси, которая является представлением таблицы удаленной базы данных, а затем с помощью оператора INSERT с разделом SELECT можно вставить данные из удаленной базы данных в постоянную таблицу в собственной базе данных.

☞ Для получения дополнительной информации о доступе к удаленным данным см. раздел "Доступ к удаленным данным" на стр. 443.

Импорт баз данных

Для создания базы данных посредством единовременного импорта одной таблицы можно использовать либо мастер Interactive SQL, либо оператор INPUT. Также можно создать сценарий, который автоматизирует этот процесс. Однако для повышения эффективности рекомендуется применять процедуру перезагрузки базы данных, когда это возможно.

☞ Для получения дополнительной информации об импорте предварительно выгруженной базы данных см. раздел "Перезагрузка базы данных" на стр. 433.

Импорт данных

❖ **Импорт данных с использованием меню Data (Interactive SQL)**

1. В окне Interactive SQL выберите Data➤Import. Появляется диалог Open.
2. Выберите импортируемый файл и нажмите Open.

   Возможен импорт данных в текстовом формате, а также в форматах DBASEII, Excel 2.1, FOXPRO и Lotus.

Появляется мастер импорта.

3 Нажмите Use an existing table и затем введите имя и местоположение существующей таблицы.

Можно нажать кнопку Browse и выбрать таблицу, в которую импортируются данные.

4 Нажмите Finish для запуска импорта данных.

В этом случае при импорте происходит добавление новых данных к данным в существующей таблице. Если импорт завершен успешно, в окне Messages выводится информация о времени, затраченном на импорт данных. Если импорт завершен неудачно, выводится соответствующее сообщение. На закладке Results в окне Results выводится информация о том, какой план выполнения применялся.

❖ Импорт данных с использованием оператора INSERT

1 Проверьте, что таблица, в которую должны быть записаны данные, действительно существует.

2 Выполните оператор INSERT. Например:

```
INSERT INTO t1
VALUES ( ... )
```

В результате выполнения этого оператора новые данные добавляются к данным в существующей таблице.

❖ Импорт данных с использованием оператора INPUT (Interactive SQL)

1 Проверьте, что таблица, в которую должны быть записаны данные, действительно существует.

2 Введите оператор INPUT в области SQL Statements в окне Interactive SQL. Например:

```
INPUT INTO t1
FROM file1
FORMAT ASCII;
```

Здесь *t1* – имя таблицы, в которую должны быть записаны данные, а *file1* – имя файла, в котором содержатся импортируемые данные.

3 Выполните оператор.

Если импорт завершен успешно, в окне Messages выводится информация о времени, затраченном на импорт данных. Если импорт завершен неудачно, выводится соответствующее сообщение. На закладке Results в окне Results выводится информация о том, какой план выполнения применялся.

☞ Для получения дополнительной информации об использовании оператора INPUT для импорта данных см. раздел "Оператор INPUT [Interactive SQL]" (INPUT statement [Interactive SQL]) на стр. 427 в документе *"Справочник по SQL для ASA" (ASA SQL Reference Manual)*.

Импорт таблицы

❖ **Импорт таблицы с использованием меню Data (Interactive SQL)**

1. Проверьте, что таблица, в которую должны быть записаны данные, действительно существует.
2. В окне Interactive SQL выберите Data➤Import. Появляется диалог Open.
3. Выберите импортируемый файл и нажмите Open.

   Возможен импорт данных в текстовом формате, а также в форматах DBASEII, Excel 2.1, FOXPRO и Lotus.
   Появляется мастер импорта.
4. Нажмите Create a new table with the following name и в соответствующем поле введите имя новой таблицы.
5. Нажмите Finish для запуска импорта данных.

   Если импорт завершен успешно, в окне Messages выводится информация о времени, затраченном на импорт данных. Если импорт завершен неудачно, выводится соответствующее сообщение. На закладке Results в окне Results выводится информация о том, какой план выполнения применялся.

❖ **Импорт таблицы (Interactive SQL)**

1. В области SQL Statements окна Interactive SQL создайте таблицу, в которую должны быть загружены данные.

   Для создания таблицы можно использовать оператор CREATE TABLE.
2. Выполните оператор LOAD TABLE. Например:

```
LOAD TABLE department
FROM 'dept.txt'
```

   Оператор LOAD TABLE добавляет содержимое файла к существующим строкам таблицы; выполнение этого оператора не приводит к замене существующих строк в таблице. Для удаления всех строк из таблицы можно использовать оператор TRUNCATE TABLE.

   Ни оператор TRUNCATE TABLE, ни оператор LOAD TABLE не запускают триггеры, в частности триггеры ссылочной целостности, выполняющие каскадное удаление.

В операторе LOAD TABLE может использоваться дополнительный раздел STRIP. Если используется установка по умолчанию (STRIP ON), то перед вставкой значений в них удаляются конечные пробелы. Для того чтобы сохранить конечные пробелы, в операторе LOAD TABLE следует использовать раздел STRIP OFF.

☞ Для получения дополнительной информации о синтаксисе оператора LOAD TABLE см. раздел "Оператор LOAD TABLE" (LOAD TABLE statement) на стр. 438 в документе *"Справочник по SQL для ASA" (ASA SQL Reference Manual)*.

Экспорт

Ниже представлен обзор инструментальных средств экспорта, а также приводятся инструкции по экспорту результатов запросов, баз данных и таблиц.

Инструментальные средства экспорта

Для поддержки экспорта данных предусмотрен целый ряд инструментальных средств.

Мастер экспорта Interactive SQL

Обратиться к мастеру экспорта можно путем выбора Data➤Export в меню Interactive SQL. Мастер предоставляет интерфейс, обеспечивающий экспорт результатов запросов в формате, отличном от текстового.

Использовать мастера экспорта Interactive SQL следует в том случае, когда требуется экспортировать результаты запросов в формате, отличном от текстового.

Оператор OUTPUT

С помощью оператора OUTPUT в Interactive SQL можно экспортировать результаты запросов, таблицы и представления из базы данных. Оператор OUTPUT в Interactive SQL поддерживает несколько различных форматов файла. Можно выбрать формат вывода по умолчанию или же задать формат файла для каждого оператора OUTPUT. Interactive SQL может выполнить командный файл, содержащий несколько операторов OUTPUT.

Если с помощью оператора OUTPUT экспортируется большой объем данных, производительность системы несколько снижается. Оператор OUTPUT также рекомендуется использовать на той же машине, на которой работает сервер, во избежание передачи большого объема данных через сеть.

Пользуйтесь оператором OUTPUT в Interactive SQL в тех случаях, когда необходимо экспортировать таблицу (полностью или частично) или представление в формате, отличном от текстового, или когда необходимо автоматизировать процесс экспорта с помощью командного файла.

☞ Для получения дополнительной информации см. раздел "Оператор OUTPUT [Interactive SQL]" (OUTPUT statement [Interactive SQL]) на стр. 453 в документе *"Справочник по SQL для ASA" (ASA SQL Reference Manual)*.

Оператор UNLOAD TABLE

Выполнение оператора UNLOAD TABLE осуществляется в области SQL Statements окна Interactive SQL. Этот оператор обеспечивает эффективный экспорт данных в форматах text/ASCII/FIXED. Экспорт при использовании оператора UNLOAD TABLE происходит построчно (одна табличная строка на одну файловую строку), между значениями вводятся разделители (запятые). Данные экспортируются в порядке значений первичных ключей (это позволяет ускорить перезагрузку).

Используйте оператор UNLOAD TABLE тогда, когда требуется экспорт полных таблиц в текстовом формате. При выборе между операторами OUTPUT, UNLOAD и UNLOAD TABLE следует учитывать, что использование оператора UNLOAD TABLE повышает производительность системы.

☞ Для получения дополнительной информации см. раздел "Оператор UNLOAD TABLE" (UNLOAD TABLE statement) на стр. 536 в документе *"Справочник по SQL для ASA" (ASA SQL Reference Manual)*.

Оператор UNLOAD

Оператор UNLOAD подобен оператору OUTPUT в том, что он также служит для экспорта результатов запроса в файл. Однако оператор UNLOAD обеспечивает более эффективный экспорт данных и только в форматах text/ASCII/FIXED. Экспорт при использовании оператора UNLOAD происходит построчно (одна табличная строка на одну файловую строку), между значениями вводятся разделители (запятые).

Применять оператор UNLOAD может пользователь с полномочиями ALTER или SELECT для таблицы. Для получения дополнительной информации об управлении полномочиями на использование оператора UNLOAD см. раздел "Параметр –gl командной строки" (–gl command-line option) на стр. 147 в документе *"Руководство по администрированию баз данных ASA" (ASA Database Administration Guide)*.

Оператор UNLOAD используется тогда, когда при экспорте результатов запроса требуется достаточно высокая производительность, и при этом допускается вывод в текстовом формате. Оператор UNLOAD также рекомендуется использовать в тех случаях, когда в приложение необходимо добавить команду экспорта.

☞ Для получения дополнительной информации см. раздел "Оператор UNLOAD" (UNLOAD statement) на стр. 535 в документе *"Справочник по SQL для ASA" (ASA SQL Reference Manual)*.

Утилита Dbunload

Утилита *dbunload* и Sybase Central отличаются по графике, но их функции одинаковы. Для получения требуемого результата можно использовать любое из этих инструментальных средств. Эти инструментальные средства отличаются от операторов Interactive SQL тем, что они могут одновременно работать с несколькими таблицами. Помимо экспорта табличных данных, оба этих средства также позволяют экспортировать схему таблицы.

Если требуется изменить структуру таблицы в базе данных, необходимые изменения можно внести с помощью соответствующего командного файла, созданного с использованием утилиты *dbunload*. Для выгрузки одной, нескольких или всех таблиц базы данных Sybase Central предоставляет мастеров и графический интерфейс пользователя, а утилита *dbunload* содержит ключи командной строки для выполнения тех же действий. При выгрузке таблиц можно выгружать только структуру, только данные или структуру и данные таблицы. Для выгрузки только некоторых таблиц базы данных, следует предварительно выполнить подключение.

Также возможно извлечение одной или нескольких таблиц как с использованием командных файлов, так и без их использования. Эти файлы служат для создания идентичных таблиц в других базах данных.

Используйте Sybase Central или утилиту dbunload тогда, когда требуется экспорт в текстовом формате, когда необходимо быстро обработать большой объем данных, когда нет жестких требований к формату файлов, или когда требуется перестройка или извлечение базы данных.

☞ Для получения дополнительной информации см. раздел "Утилита dbunload командной строки" (The dbunload command-line utility) на стр. 502 в документе *"Руководство по администрированию баз данных ASA" (ASA Database Administration Guide)*.

Экспорт результатов запросов

Результаты запросов (в том числе запросов в представлениях) могут экспортироваться в файл. Для этого используется меню Data в Interactive SQL или оператор OUTPUT.

Импорт и экспорт файлов между Adaptive Server Anywhere и Adaptive Server Enterprise возможен с использованием раздела BCP FORMAT.

☞ Для получения дополнительной информации см. раздел "Совместимость с Adaptive Server Enterprise" на стр. 442.

❖ Экспорт результатов запроса с использованием меню Data (Interactive SQL)

1. Введите запрос в область SQL Statements окна Interactive SQL.
2. Нажмите Execute SQL statement(s) для вывода результирующего набора.
3. Выберите Data➤Export. Появляется диалог Save As.
4. Задайте имя и местоположение для экспортируемых данных.
5. Задайте формат файла и нажмите Save.

   Если экспорт завершен успешно, в окне Messages выводится информация о времени, затраченном на экспорт результирующего набора запроса, имя файла, путь к экспортированным данным и количество записанных строк.

   Если экспорт завершен неудачно, выводится соответствующее сообщение.

❖ Экспорт результатов запроса с использованием оператора OUTPUT (Interactive SQL)

1. Введите запрос в область SQL Statements окна Interactive SQL.
2. В конце запроса введите **OUTPUT TO** *'c:\имя-файла'*.

   Например, для экспорта всей таблицы служащих (employee) в файл employee.dbf введите следующий запрос:

   ```
   SELECT *
   FROM employee;
   OUTPUT TO 'c:\employee.dbf'
   ```
3. Если необходимо экспортировать результаты запроса и добавить их к другому файлу, дополнительно введите оператор APPEND в конце оператора OUTPUT.

   Например:

```
SELECT *
FROM employee;
OUTPUT TO 'c:\employee.dbf' APPEND
```

4 Если необходимо экспортировать результаты запроса с включением сообщений, дополнительно введите оператор VERBOSE в конце оператора OUTPUT.

Например:

```
SELECT * FROM employee;
OUTPUT TO 'c:\employee.dbf' VERBOSE
```

5 Если необходимо задать формат, отличный от формата ASCII, добавьте раздел FORMAT в конец запроса.

Например:

```
SELECT *
FROM employee;
OUTPUT TO 'c:\employee.dbf'
FORMAT dbaseiii;
```

Здесь *c:\employee.dbf* обозначает путь к новому файлу, его имя и расширение, а *dbaseiii* – формат данного файла. Данную строку можно заключить в одиночные или двойные кавычки, но это требуется только в том случае, если в обозначении пути содержатся пробелы.

Здесь *dbaseiii* является форматом данного файла. Если параметр FORMAT не указан, то по умолчанию выбирается тип файла ASCII.

6 Выполните оператор.

Если экспорт завершен успешно, в окне Messages выводится информация о времени, затраченном на экспорт результирующего набора запроса, имя файла, путь к экспортированным данным и количество записанных строк. Если экспорт завершен неудачно, выводится соответствующее сообщение.

Для получения дополнительной информации об экспорте результатов запроса с использованием оператора OUTPUT см. раздел "Оператор OUTPUT [Interactive SQL]" (OUTPUT statement [Interactive SQL]) на стр. 453 в документе *"Справочник по SQL для ASA" (ASA SQL Reference Manual)*.

Советы

При использовании и оператора APPEND, и оператора VERBOSE к существующему файлу можно добавить и результирующие данные, и сообщения. Например, введите **OUTPUT TO** *'c:\имя-файла.SQL'* **APPEND VERBOSE**. Для получения дополнительной информации об операторах APPEND и VERBOSE см. раздел "Оператор OUTPUT [Interactive SQL]" (OUTPUT statement [Interactive SQL]) на стр. 453 в документе *"Справочник по SQL для ASA" (ASA SQL Reference Manual)*.

Операторы OUTPUT TO, APPEND и VERBOSE эквивалентны операциям >#, >>#, >& и >>& в более ранних версиях Interactive SQL. Эти операции по-прежнему можно использовать для перенаправления данных, однако новые операторы в Interactive SQL обеспечивают более точную настройку вывода и упрощают чтение кода.

❖ **Экспорт результатов запроса с использованием оператора UNLOAD**

1 Выполните оператор UNLOAD. Например:

```
UNLOAD
SELECT *
FROM employee;
TO 'c:\employee.dbf'
```

Если экспорт завершен успешно, в окне Messages выводится информация о времени, затраченном на экспорт результирующего набора запроса, имя файла, путь к экспортированным данным и количество записанных строк. Если экспорт завершен неудачно, выводится соответствующее сообщение.

Экспорт базы данных

❖ **Выгрузка всей базы данных или ее части (Sybase Central)**

1 В левой области окна Sybase Central нажмите Utilities.

Все функции, выполнение которых разрешено при работе с базой данных, отображаются в правой области окна.

2 Дважды щелкните Unload Database в правой области окна. Появляется мастер выгрузки базы данных.

Вызвать мастер выгрузки базы данных также можно, щелкнув правой кнопкой мыши на имени базы данных в левой области окна и выбрав Unload Database во всплывающем меню; либо выберите команду Tools➤Adaptive Server Anywhere 8➤Unload Database.

3 Нажмите Next для продолжения работы с мастером выгрузки базы данных.

4 Выберите параметр Use the following connection.

5 В окне выберите подключение, которое необходимо использовать, и затем нажмите Next.

6 Укажите путь и местоположение для командного файла выгрузки базы данных, и нажмите Next.

Для определения местоположения папки, в которой должен быть сохранен командный файл, можно воспользоваться кнопкой Browse. Командный файл имеет расширение *.SQL*. Этот файл требуется для восстановления базы данных из выгруженных файлов данных.

7 Выберите параметр Do not reload the data и затем нажмите Next.

8 Укажите, какая часть базы данных должна быть выгружена и затем нажмите Next.

Можно выбрать одну из следующих опций:

- Extract structure and data (Извлечение структуры и данных)
- Extract structure only (Извлечение только структуры)
- Extract data only (Извлечение только данных)

Для выгрузки всей базы данных выберите Extract structure and data.

9 Укажите тип выгрузки – Internal unload (Внутренняя выгрузка) или External unload (Внешняя выгрузка), и затем нажмите Next.

10 Укажите количество уровней зависимости представлений.

Такое указание уровней зависимости представлений обеспечивает возможность воссоздания представлений на основе других представлений. Например, если существует одно представление, основанное на существующих таблицах, то в этом поле следует ввести 1. Это представление 1 не зависит от других представлений и может быть воссоздано по таблицам. Если же, например, существует и второе представление, основанное на первом представлении, то в этом поле следует ввести 2. Представление 2 зависит от представления 1 и не может быть создано, пока не создано представление 1.

11 Укажите, требуется ли упорядочение данных.

Экспорт данных в упорядоченном виде означает, что в дальнейшем данные будут перезагружаться в упорядоченном виде. Это рекомендуется в том случае, когда необходимо повысить производительность при работе с базой данных или когда необходимо обойти поврежденный индекс.

12 Укажите путь и местоположение для выгружаемых данных и нажмите Next.

Для определения местоположения папки, в которой должна быть сохранена база данных, можно воспользоваться кнопкой Browse. Если выбранная папка не существует, выводится сообщение о необходимости подтверждения создания папки.

13 Убедитесь в том, что информация для мастера введена правильно, и нажмите Finish для выгрузки базы данных.

Вместо окна мастера выгрузки базы данных появляется окно сообщений Extracting Database, в котором выводятся сообщения, информирующие о том, в каких файлах содержатся те или иные таблицы базы данных.

14 Закройте окно сообщений Extracting Database.

❖ **Выгрузка всей базы данных или ее части (командная строка)**

1 В командной строке введите команду *dbunload* и укажите параметры подключения с помощью ключа –c.

Например, по следующей команде вся база данных выгружается в c:\temp:

```
dbunload -c "dbn=asademo;uid=DBA;pwd=SQL" c:\temp
```

2 Если необходимо экспортировать только данные, добавьте ключ –d.

Например, если необходимо экспортировать только данные, конечная команда может выглядеть следующим образом:

```
dbunload -c "dbn=asademo;uid=DBA;pwd=SQL" -d c:\temp
```

3 Если необходимо экспортировать только схему, добавьте ключ –n.

Например, если необходимо экспортировать только схему, конечная команда может выглядеть следующим образом:

```
dbunload -c "dbn=asademo;uid=DBA;pwd=SQL" -n c:\temp
```

4 Нажмите Enter для выполнения команды.

☞ Для получения дополнительной информации о ключах командной строки, применяемых в утилите dbunload, см. раздел "Утилита dbunload командной строки" (The dbunload command-line utility) на стр. 502 в документе *"Руководство по администрированию баз данных ASA" (ASA Database Administration Guide)*.

Экспорт таблиц

В дополнение к описанным ниже способам, экспортировать таблицу также можно путем выделения всех данных в таблице и экспорта результатов запроса. Для получения дополнительной информации см. раздел "Экспорт результатов запросов" на стр. 423.

Совет

Представления можно экспортировать точно так же, как таблицы.

❖ **Экспорт таблицы (командная строка)**

1 В командной строке введите следующую команду *dbunload* и нажмите Enter:

```
dbunload -c "dbn=asademo;uid=DBA;pwd=SQL" -t
employee c:\temp
```

Здесь в ключе –c указываются параметры подключения к базе данных, а в ключе –t указывается имя таблицы, подлежащей экспорту. По этой команде *dbunload* данные выгружаются из демонстрационной базы данных (при этом предполагается, что эта база данных запущена на сервере базы данных по умолчанию с именем базы данных по умолчанию) в набор файлов в папке *c:\temp*. Командный файл, предназначенный для восстановления базы данных из файлов данных, создается с именем по умолчанию *reload.SQL* в текущей папке.

Также можно выгрузить несколько таблиц; в этом случае между именами таблиц вводятся запятые-разделители (,).

❖ Экспорт таблицы (SQL)

1 Выполните оператор UNLOAD TABLE. Например:

```
UNLOAD TABLE department
TO 'dept.txt'
```

Этот оператор выгружает таблицу department из демонстрационной базы данных в файл dept.txt в текущей рабочей папке сервера. При использовании сетевого сервера ввод этой команды приводит к выгрузке данных в файл на серверной машине, а не на клиентской машине. Кроме того, на сервер передается имя файла в виде текстовой строки. Использование символов наклонной черты влево в имени файла предотвращает неверную интерпретацию в случае, когда название папки в имени файла начинается с n (\n = newline, новая строка) или с любого другого специального символа.

Каждая строка таблицы выводится как одна строка выходного файла; имена столбцов не экспортируются. Разделителем между столбцами является запятая. Символ, используемый в качестве разделителя, может быть заменен с помощью раздела DELIMITED BY. Поля не являются полями с фиксированной шириной. Экспортируются только символы, существующие в каждой записи, но не все символы по ширине столбца.

☞ Для получения дополнительной информации о синтаксисе оператора UNLOAD TABLE см. раздел "Оператор UNLOAD TABLE" (UNLOAD TABLE statement) на стр. 536 в документе *"Справочник по SQL для ASA" (ASA SQL Reference Manual)*.

Перестройка баз данных

Перестройка базы данных – это особый тип импорта и экспорта, включающий выгрузку и перезагрузку всей базы данных. При перестройке базы данных выполняется единообразное извлечение всей информации из базы данных и такое же единообразное обратное занесение информации в базу данных. Таким способом достигается максимальная плотность заполнения пространства и обеспечивается повышение производительности при работе с базой данных (т. е. получаемый результат схож с результатом, получаемым при дефрагментации диска).

Перед перестройкой рекомендуется создать резервные копии базы данных.

Загрузку и выгрузку наиболее целесообразно использовать для повышения производительности, устранения излишней фрагментации пространства, а также для обновления базы данных при переходе к более поздней версии Adaptive Server Anywhere.

Перестройка отличается от экспорта тем, что помимо данных при перестройке также экспортируются и импортируются определения и схема таблиц. В процессе перестройки при выгрузке данных создаются файлы данных в формате *ASCII,* а также файл *'reload.SQL',* содержащий определения таблиц и другие определения. Выполнение сценария *reload.SQL* приводит к воссозданию таблиц и к загрузке данных в эти таблицы.

Этот процесс может быть выполнен с использованием Sybase Central или утилиты *dbunload* командной строки.

Перестройку базы данных рекомендуется проводить в тех случаях, когда необходимо обновить базу данных, устранить излишнюю фрагментацию дискового пространства или повысить производительность. При использовании SQL Remote или MobiLink может потребоваться извлечение базы данных (создание новой базы данных из старой базы данных).

Если требуется дефрагментация базы данных, но полная перестройка при этом не может быть выполнена (поскольку должен обеспечиваться непрерывный доступ к базе данных), то в таком случае вместо перестройки рекомендуется выполнить реорганизацию таблиц.

☞ Для получения дополнительной информации о реорганизации таблиц см. раздел "Оператор REORGANIZE TABLE" (REORGANIZE TABLE statement) на стр. 472 в документе *"Справочник по SQL для ASA" (ASA SQL Reference Manual).*

Перестройка базы данных, задействованной в репликации

Если база данных участвует в репликации, то при перестройке такой базы данных должны быть приняты определенные меры предосторожности.

При репликации используются смещения в журнале транзакций. При перестройке базы данных смещения в старом журнале транзакций отличаются от смещений в новом журнале, и старый журнал становится недоступным. По этой причине особое внимание следует уделять созданию резервных копий, если база данных участвует в репликации.

Существует два способы перестройки базы данных, задействованной в репликации. Первый способ состоит в использовании параметра -ar в утилите *dbunload* для того, чтобы выгрузка и перезагрузка выполнялись, не

оказывая влияния на репликацию. Второй способ заключается в выполнении этой же задачи вручную.

Процедуры перестройки (загрузка/выгрузка) и извлечения используются для реконструкции базы данных и создания новых баз данных на основе части старой базы данных.

При импорте и экспорте данные перемещаются либо в собственную базу данных, либо из нее. При импорте выполняется считывание данных в собственную базу данных. При экспорте производится запись данных во внешний объект. Во многих случаях информация поступает из базы данных, не имеющей отношения к Adaptive Server Anywhere, или передается в такую базу данных.

В процедуре перестройки используются сразу две функции – загрузка и выгрузка. При загрузке и выгрузке происходит извлечение данных и схемы из базы данных Adaptive Server Anywhere, после чего данные снова заносятся в эту базу данных. В результате выполнения процедуры выгрузки создаются файлы данных фиксированного формата, а также файл *reload.SQL,* в котором содержатся определения, обеспечивающие точное воссоздание таблицы. Выполнение сценария *reload.SQL* приводит к обновлению таблиц и к загрузке данных обратно в таблицы.

Для перестройки базы данных может потребоваться значительный период времени и большой объем пространства на диске. Кроме того, при выгрузке и перезагрузке база данных становится недоступной. По этим причинам перестройку базы данных не рекомендуется выполнять в продуктивной среде, если только необходимость перестройки не обусловлена особой конкретной задачей.

Перестройка базы данных, задействованной в репликации

Порядок выполнения перестройки базы данных зависит от того, задействована ли база данных в репликации. Если база данных задействована в репликации, то в процессе перестройки следует обеспечить сохранение информации о смещениях в журнале транзакций, поскольку эта информация требуется для Message Agent и Replication Agent. Если база данных не задействована в репликации, выполнение процедуры упрощается.

Инструментальные средства перестройки

Оператор LOAD/ UNLOAD TABLE

Оператор UNLOAD TABLE обеспечивает эффективный экспорт данных в форматах text/ASCII/FIXED. Экспорт при использовании оператора UNLOAD TABLE происходит построчно (одна табличная строка на одну файловую строку), между значениями вводятся разделители (запятые).

Данные экспортируются в порядке значений первичных ключей, что позволяет ускорить процесс перезагрузки.

Применять оператор UNLOAD TABLE может пользователь с полномочиями ALTER или SELECT для таблицы.

Оператор UNLOAD TABLE используется тогда, когда необходимо экспортировать данные в текстовом формате, или когда требуется высокая производительность.

☞ Для получения дополнительной информации см. раздел "Оператор UNLOAD" (UNLOAD statement) на стр. 535 в документе *"Справочник по SQL для ASA" (ASA SQL Reference Manual)*.

Утилиты dbunload/dbisql и Sybase Central

Утилиты *dbunload/dbisql* отличаются от Sybase Central по графике, но их функции одинаковы. Для получения требуемого результата можно использовать любое из этих инструментальных средств.

Мастер выгрузки базы данных Sybase Central или утилиту *dbunload* можно использовать для выгрузки всей базы данных в формате ASCII с разделителями-запятыми и для создания необходимых командных файлов Interactive SQL, обеспечивающих полное воссоздание базы данных. Эти средства могут применяться для создания извлечений SQL Remote или построения новых копий базы данных с той же или несколько измененной структурой. Утилиту *dbunload* и Sybase Central целесообразно использовать для экспорта файлов Adaptive Server Anywhere, предназначенных для повторного использования в среде Adaptive Server Anywhere.

Используйте Sybase Central или утилиту *dbunload* в тех случаях, когда необходимо перестроить базу данных или выполнить извлечение из нее, произвести экспорт в текстовом формате, быстро обработать большой объем данных, или когда отсутствуют жесткие требования к формату файлов.

☞ Для получения дополнительной информации см. раздел "Перестройка базы данных, не задействованной в репликации" на стр. 433 и раздел "Перестройка базы данных, задействованной в репликации" на стр. 436.

Перестройка форматов файлов

Переход из одной базы данных ASA в другую

При перестройке, как правило, данные извлекаются из одной базы данных Adaptive Server Anywhere и заносятся в другую базу данных Adaptive Server Anywhere. Выгрузка и перезагрузка – это тесно связанные процедуры, которые обычно выполняются только вместе, а не отдельно друг от друга.

Перестройка базы данных

Перестройка базы данных может потребоваться в следующих случаях:

- **Обновление формата файла базы данных.** Для внедрения некоторых новых средств применяется утилита обновления, но для других средств требуется обновление формата файла базы данных, которое выполняется путем выгрузки и перезагрузки базы данных. В документации по новым возможностям указывается, требуются ли выгрузка и перезагрузка для внедрения конкретного нового средства.
- **Освобождение дискового пространства.** При удалении данных объем базы данных не сжимается. Все пустые страницы помечаются как свободные, и в дальнейшем они могут использоваться повторно. Пустые страницы не удаляются из базы данных, пока не будет выполнена перестройка. В том случае, когда в базе данных удален большой объем данных и не предполагается добавление новых данных, с помощью

перестройки базы данных можно освободить дисковое пространство и устранить его фрагментацию.

- **Повышение производительности.** Перестройка базы данных может повысить производительность благодаря следующему:
  - Если имеет место фрагментация данных на страницах в базе данных, то путем выгрузки и перезагрузки данных можно устранить эту фрагментацию.
  - Поскольку данные могут быть выгружены и перезагружены в порядке, определяемом первичными ключами, то после этого затрачивается меньше времени на получение доступа к связанной информации, поскольку связанные строки могут находиться на той же самой или на смежной странице.

Обновление базы данных

Новые версии сервера базы данных Adaptive Server Anywhere могут использоваться без обновления существующей базы данных. Если же требуется использование таких средств новой версии, которым необходим доступ к новым системным таблицам или новым параметрам базы данных, то в таком случае следует обновить базу данных с помощью утилиты обновления. Утилита обновления не производит выгрузку или перезагрузку каких-либо данных.

Если необходимо использовать такие средства новой версии, которые базируются на изменениях в формате файла базы данных, следует выгрузить и перезагрузить существующую базу данных. После перестройки базы данных следует выполнить резервное копирование базы данных.

Для обновления файла базы данных следует использовать новую версию Adaptive Server Anywhere.

Для получения дополнительной информации о процедуре обновления базы данных см. раздел "Обновление Adaptive Server Anywhere" (Upgrading Adaptive Server Anywhere) на стр. 114 в документе *"Новое в SQL Anywhere Studio" (What's New in SQL Anywhere Studio)*.

Экспорт данных таблицы или схемы

❖ **Экспорт данных таблицы или схемы (командная строка)**

1. В командной строке введите команду *dbunload* и укажите параметры соединения с помощью ключа -c.
2. Укажите таблицу (таблицы) для экспорта данных или схемы; используйте ключ –t.

   Например, для экспорта части таблицы служащих введите

   ```
   dbunload -c "dbn=asademo;uid=DBA;pwd=SQL" -t employee
   c:\temp
   ```

   Также можно выгрузить несколько таблиц; в этом случае между именами таблиц вводятся разделители-запятые.
3. Если необходимо экспортировать только данные, добавьте ключ -d.

   Например, если необходимо экспортировать только данные, конечная команда может выглядеть следующим образом:

   ```
   dbunload -c "dbn=asademo;uid=DBA;pwd=SQL" -d -t employee
   c:\temp
   ```

4 Если необходимо экспортировать только схему, добавьте ключ -n.

Например, если необходимо экспортировать только схему, конечная команда может выглядеть следующим образом:

```
dbunload -c "dbn=asademo;uid=DBA;pwd=SQL" -n -t employee
c:\temp
```

5 Нажмите Enter для выполнения команды.

По командам *dbunload,* приведенным в этих примерах, выполняется выгрузка данных или схемы из демонстрационной базы данных (при этом предполагается, что эта база данных запущена на сервере базы данных по умолчанию с именем базы данных по умолчанию) в файл в папке *c:\temp*. Командный файл, предназначенный для восстановления базы данных из файлов данных, создается с именем по умолчанию *reload.SQL* в текущей папке.

Перезагрузка базы данных

❖ **Перезагрузка базы данных (командная строка)**

1 В командной строке выполните сценарий *reload.SQL*.

Например, следующая команда загружает сценарий *reload.SQL* из текущей папки.

```
dbisql -c "dbn=asademo;uid=DBA;pwd=SQL" reload.SQL
```

Перестройка базы данных, не задействованной в репликации

Следующие процедуры можно использовать только в том случае, если база данных не задействована в репликации.

❖ **Перестройка базы данных, не задействованной в репликации (Sybase Central)**

1 В левой области окна Sybase Central нажмите Utilities.

Все функции, выполнение которых разрешено при работе с базой данных, отображаются в правой области окна.

2 Дважды щелкните Unload Database в правой области окна. Появляется мастер выгрузки базы данных.

Вызвать мастер выгрузки базы данных можно также, щелкнув правой кнопкой мыши на имени базы данных в левой области окна и выбрав Unload Database во всплывающем меню; либо выберите команду Tools➤Adaptive Server Anywhere 8➤Unload Database.

3 Нажмите Next для продолжения работы с мастером выгрузки базы данных.

4 Выберите параметр Use the following connection.

5 В окне выберите подключение, которое необходимо использовать, и затем нажмите Next.

6 Укажите путь и местоположение для командного файла выгрузки базы данных и нажмите Next.

Для определения местоположения папки, в которой должен быть сохранен командный файл, можно воспользоваться кнопкой Browse. Командный файл имеет расширение *.SQL*. Этот файл требуется для восстановления базы данных из выгруженных файлов данных.

7 Задайте способ восстановления базы данных и затем нажмите Next.

Можно выбрать одну из следующих опций:

- Reload into a new database (Перезагрузка в новую базу данных). В этом случае необходимо указать имя и местоположение новой базы данных.
- Reload into an existing database (Перезагрузка в существующую базу данных). В этом случае появляется диалог Connect. Для продолжения перестройки следует указать базу данных и задать параметры соединения.
- Replace the original database (Замена исходной базы данных). В этом случае следует указать, куда должен быть помещен старый файл журнала.

Для получения дополнительной информации о параметре "Do not reload the data" см. раздел "Экспорт базы данных" на стр. 425.

8 Укажите, какая часть базы данных подлежит перестройке, и затем нажмите Next.

Можно выбрать одну из следующих опций:

- Extract structure and data (Извлечение структуры и данных);
- Extract structure only (Извлечение только структуры);
- Extract data only (Извлечение только данных).

9 Укажите тип выгрузки – Internal unload (Внутренняя выгрузка) или External unload (Внешняя выгрузка), и затем нажмите Next.

10 Укажите количество уровней зависимости представлений.

Такое указание уровней зависимости представлений обеспечивает возможность воссоздания представлений на основе других представлений. Например, если существует одно представление, основанное на существующих таблицах, то в этом поле следует ввести 1. Это представление 1 не зависит от других представлений и может быть воссоздано по таблицам. Если же, например, существует и второе представление, основанное на первом представлении, то в этом поле следует ввести 2. Представление 2 зависит от представления 1 и не может быть создано, пока не создано представление 1.

11 Укажите, требуется ли упорядочение данных.

Экспорт данных в упорядоченном виде означает, что в дальнейшем данные будут перезагружаться в упорядоченном виде. Это рекомендуется в том случае, когда необходимо повысить производительность при работе с базой данных, или когда необходимо обойти поврежденный индекс.

12 Укажите путь и местоположение для выгружаемых данных и нажмите Next.

Для определения местоположения папки, в которой должна быть сохранена база данных, можно воспользоваться кнопкой Browse. Если выбранное местоположение не существует, выводится сообщение о необходимости подтверждения создания папки.

13 Убедитесь в том, что информация для мастера введена правильно, и нажмите Finish для выгрузки базы данных.

Вместо окна мастера выгрузки базы данных появляется инструментальная консоль Adaptive Server Anywhere, где выводятся сообщения, информирующие о том, в каких файлах содержатся те или

иные таблицы базы данных. По завершении выгрузки появляется сообщение 'Complete'.

14 Закройте окно сообщений Extracting Database.

❖ **Перестройка базы данных, не задействованной в репликации (командная строка)**

1 В командной строке выполните утилиту *dbunload* с одним из следующих ключей:

- Ключ –an используется для воссоздания в новой базе данных.

```
dbunload -c "dbf=asademo.db;uid=DBA;pwd=SQL" -an
asacopy.db
```

- Ключ –ac используется для перезагрузки в существующую базу данных.

```
dbunload -c "dbf=asademo.db;uid=DBA;pwd=SQL" -ac
"uid=DBA;pwd=SQL;dbf=newdemo.db"
```

- Ключ –ar используется для замены существующей базы данных.

```
dbunload -c "dbf=asademo.db;uid=DBA;pwd=SQL" -ar
"uid=DBA;pwd=SQL;dbf=newdemo.db"
```

При использовании любого из этих параметров промежуточная копия данных на диске не создается, поэтому в командной строке не требуется указывать папку для выгрузки. Это повышает степень защиты данных. При использовании ключей -ar и -an быстродействие выше, чем при использовании Sybase Central, однако при использовании ключа -ac быстродействие ниже.

2 Завершите работу базы данных и создайте архивную копию журнала транзакций. После этого можно использовать перезагруженную базу данных.

Примечания

Ключи-an и -ar применяются только при подключении к персональному серверу или при подключении к сетевому серверу с использованием разделяемой памяти.

В утилите *dbunload* могут использоваться дополнительные ключи для настройки выгрузки, а также ключи параметров подключения, с помощью которых можно указать запущенную или не запущенную базу данных и задать параметры базы данных.

Перестройка базы данных, задействованной в репликации

❖ **Перестройка базы данных, задействованной в репликации**

1. Завершите работу базу данных.
2. Выполните полное резервное копирование в режиме off-line. Для этого скопируйте базу данных и файлы журналов транзакций в надежное местоположение.
3. В командной строке выполните утилиту *dbunload* для перестройки базы данных:
   `dbunload -c строка_подключения -ar папка`

   Здесь строка_подключения является подключением с полномочиями администратора БД, папка – это папка, используемая в среде репликации для старых журналов транзакций; другие подключения к базе данных отсутствуют.

   Ключ -ar применяется только при подключении к персональному серверу или при подключении к сетевому серверу с использованием разделяемой памяти.

   Для получения дополнительной информации см. раздел "Параметры утилиты выгрузки" (Unload utility options) на стр. 504 в документе *"Руководство по администрированию баз данных ASA" (ASA Database Administration Guide)*.
4. Завершите работу новой базы данных. Выполните проверку правильности, которая обычно выполняется после восстановления базы данных.
5. Запустите базу данных с использованием любых требуемых продуктивных ключей. Теперь может быть разрешен доступ пользователей к перезагруженной базе данных.

Примечания

В утилите *dbunload* могут использоваться дополнительные ключи для настройки выгрузки, а также ключи параметров подключения, с помощью которых можно указать запущенную или не запущенную базу данных и задать параметры базы данных.

Если приведенная выше процедура не отвечает предъявляемым требованиям, смещения журнала транзакций можно скорректировать вручную. Ниже описывается порядок выполнения этой операции.

❖ **Перестройка базы данных, задействованной в репликации (вручную)**

1. Завершите работу базу данных.
2. Выполните полное резервное копирование в режиме off-line. Для этого скопируйте базу данных и файлы журналов транзакций в надежное местоположение.
3. Выполните утилиту *dbtran* для получения информации о начальном смещении и конечном смещении текущего файла журнала транзакций базы данных. Сохраните информацию о конечном смещении для использования в дальнейшем.
4. Переименуйте текущий файл журнала транзакций, чтобы избежать его изменения в процессе выгрузки. Поместите этот файл в папку неактивных журналов транзакций dbremote.
5. Выполните перестройку базы данных.

Для получения информации о выполнении этой процедуры см. раздел "Перестройка баз данных" на стр. 429.

6 Завершите работу новой базы данных.

7 Удалите текущий файл журнала транзакций для новой базы данных.

8 Выполните *dblog* для новой базы данных. В качестве параметра -z введите конечное смещение, информация о котором была получена на шаге 3; также установите нулевое относительное смещение.

```
dblog -x 0 -z 137829 database-name.db
```

9 При использовании Message Agent этой программе необходимо предоставить информацию о местоположении исходной неактивной папки (эта информация вводится в командной строке программы).

10 Запустите базу данных. Теперь может быть разрешен доступ пользователей к перезагруженной базе данных.

Минимизация времени простоя при перестройке

Приведенная ниже последовательность шагов используется для выполнения перестройки базы данных с минимальным временем простоя. Эта процедура особенно полезна в том случае, когда база данных эксплуатируется круглосуточно.

Прежде чем приступить к резервному копированию, имеет смысл произвести пробное выполнение шагов 1-4 и определить время, затрачиваемое на каждый шаг. Также не исключено то, что в разных точках процесса перестройки может потребоваться сохранение копий файлов.

Проверьте, что отсутствуют какие-либо запланированные процедуры резервного копирования, которые могли бы переименовать журнал продуктивной базы данных. Если же это произойдет по ошибке, то для перестройки базы данных в должном порядке потребуется применить транзакции из этих переименованных журналов.

❖ Перестройка базы данных с минимизацией времени простоя

1 С помощью DBBACKUP -r создайте резервную копию базы данных и журнала и переименуйте журнал.

2 Перестройте резервную копию базы данных на другой машине.

3 Снова выполните DBBACKUP -r на продуктивном сервере для переименования журнала.

4 Выполните DBTRAN для журнала, полученного на шаге 3, и примените транзакции на сервере перестраиваемой базы данных.

Теперь имеется перестроенная база данных, в которой содержатся все транзакции, выполненные вплоть до окончания резервного копирования на шаге 3.

5 Завершите работу продуктивного сервера и создайте копии базы данных и журнала.

6 Скопируйте перестроенную базу данных на продуктивный сервер.

7 Выполните DBTRAN для журнала, полученного на шаге 5. Это должен быть относительно небольшой файл.

8 Запустите сервер на перестроенной базе данных, но не разрешайте подключение пользователей.

9 Примените транзакции, полученные на шаге 8.

10 Разрешите подключение пользователей.

Извлечение данных

Извлечение – это полный перенос удаленной базы данных Adaptive Server Anywhere из консолидированной базы данных Adaptive Server Enterprise или Adaptive Server Anywhere.

Для извлечения баз данных может использоваться мастер извлечения Sybase Central или утилита извлечения. Использование утилиты извлечения рекомендуется для создания и синхронизации удаленных баз данных из консолидированных баз данных.

Для получения дополнительной информации об извлечении баз данных см. разделы:

- "Утилита извлечения базы данных" (Database Extraction utility) на стр. 311 в документе *"Руководство пользователя по SQL Remote" (SQL Remote User's Guide);*
- "Использование утилиты извлечения" (Using the extraction utility) на стр. 193 в документе *"Руководство пользователя по SQL Remote" (SQL Remote User's Guide);*
- "Параметры утилиты извлечения" (Extraction utility options) на стр. 314 в документе *"Руководство пользователя по SQL Remote" (SQL Remote User's Guide);*
- "Извлечение групп" (Extracting groups) на стр. 197 в документе *"Руководство пользователя по SQL Remote" (SQL Remote User's Guide);*
- "Развертывание удаленных баз данных" (Deploying remote databases) на стр. 147 в документе *"Руководство пользователя по системе синхронизации MobiLink" (MobiLink Synchronization User's Guide);*
- "Извлечение удаленной базы данных в Sybase Central" (Extracting a remote database in Sybase Central) на стр. 311 в документе *"Руководство пользователя по SQL Remote" (SQL Remote User's Guide).*

Перемещение баз данных в Adaptive Server Anywhere

В Adaptive Server Anywhere можно импортировать удаленные базы данных Oracle, DB2, Microsoft SQL Server, Sybase Adaptive Server Enterprise, Sybase Adaptive Server Anywhere и Microsoft Access. Для этого используется набор хранимых процедур *sa_migrate*.

Если таблицы изменять не требуется, можно использовать одношаговый метод перемещения. Если же необходимо удалить таблицы или отображение внешних ключей, можно использовать расширенный метод перемещения.

❖ Импорт удаленных баз данных (одношаговый метод)

1. В Interactive SQL выполните подключение к целевой базе данных.
2. В области окна операторов Interactive SQL выполните хранимую процедуру. Например:

```
dbo.sa_migrate(
IN local_table_owner    VARCHAR(128),
IN server_name          VARCHAR(128),
IN table_name           VARCHAR(128) DEFAULT NULL,
IN owner_name           VARCHAR(128) DEFAULT NULL,
IN database_name        VARCHAR(128) DEFAULT NULL,
IN migrate_data         BIT DEFAULT 1,
IN drop_proxy_tables    BIT DEFAULT 1)
```

Выполнение этой процедуры приводит к вызову ряда других процедур и к перемещению всех удаленных таблиц с использованием заданных критериев.

Для получения дополнительной информации см. раздел "Доступ к удаленным данным" на стр. 443.

❖ Импорт удаленных баз данных (с изменениями)

1. Создайте список таблиц, подлежащих перемещению.
2. Выполните следующую хранимую процедуру. Например:

```
dbo.sa_migrate_create_remote_table_list(
IN server_name
IN table_name
IN owner_name
IN database_name )
```

Все четыре аргумента являются переменными VARCHAR (128). Кроме того, *table_name, owner_name* и *database_name* имеют значение по умолчанию NULL. Имя базы данных необходимо указывать для баз данных Adaptive Server Enterprise и Microsoft SQL Server.

В результате происходит заполнение таблицы *dbo.migrate_remote_table_list* списком удаленных таблиц, подлежащих перемещению. Из этой таблицы можно удалить строки, соответствующие тем удаленным таблицам, перемещать которые не требуется.

3. Создайте таблицу прокси и определение базовой таблицы для каждой таблицы, подлежащей перемещению.

Введите следующую хранимую процедуру:

```
dbo.sa_migrate_create_tables( IN local_table_owner )
```

Здесь *local_table_owner* является переменной VARCHAR (128).

При выполнении этой процедуры извлекается список удаленных таблиц из *dbo.migrate_remote_table_list,* и создаются таблица прокси и базовая таблица для каждой удаленной таблицы, указанной в списке. В результате выполнения этой процедуры также создаются все индексы первичных ключей.

4 Переместите данные в базовые таблицы. Введите следующую хранимую процедуру:

```
dbo.sa_migrate_data( IN local_table_owner )
```

Здесь *local_table_owner* является переменной VARCHAR (128).

При выполнении этой процедуры происходит перемещение данных из каждой удаленной таблицы в базовую таблицу, созданную с помощью процедуры *dbo.sa_migrate_create_tables.*

5 Создайте список внешних ключей, подлежащих перемещению. Введите следующую хранимую процедуру:

```
dbo.sa_migrate_create_remote_fks_list(IN server_name)
```

Здесь *server_name* является переменной VARCHAR (128).

При выполнении этой процедуры происходит заполнение таблицы *dbo.migrate_remote_fks_list* списком внешних ключей, связанных с каждой из удаленных таблиц, перечисленных в *dbo.migrate_remote_table_list.*

Можно удалить отображения тех внешних ключей, которые не требуется воссоздавать в локальных базовых таблицах.

6 Создайте внешние ключи.

Введите следующую хранимую процедуру:

```
dbo.sa_migrate_create_fks( IN local_table_owner )
```

Здесь *local_table_owner* является переменной VARCHAR (128).

В результате выполнения этой процедуры создаются отображения внешних ключей, определенные в *dbo.migrate_remote_fks_list* для базовых таблиц.

7 Удалите таблицы прокси.

Введите следующую хранимую процедуру:

```
dbo.sa_migrate_drop_proxy_tables( IN local_table_owner )
```

Здесь *local_table_owner* является переменной VARCHAR (128).

В результате выполнения этой процедуры удаляются все таблицы прокси, созданные для обеспечения процесса перемещения, и на этом процесс перемещения завершается.

Совместимость с Adaptive Server Enterprise

Импорт и экспорт файлов между Adaptive Server Anywhere и Adaptive Server Enterprise возможен с использованием раздела BCP FORMAT. Достаточно обеспечить то, чтобы вывод BCP осуществлялся в формате ASCII с разделителями. Если из Adaptive Server Anywhere в Adaptive Server Enterprise экспортируются BLOB-данные, следует использовать раздел BCP FORMAT с оператором UNLOAD TABLE.

☞ Для получения дополнительной информации о BCP и разделе FORMAT см. раздел "Оператор LOAD TABLE" (LOAD TABLE statement) на стр. 438 в документе "Справочник по SQL для ASA" (ASA SQL Reference Manual) или раздел "Оператор UNLOAD TABLE" (UNLOAD TABLE statement) на стр. 536 в документе *"Справочник по SQL для ASA" (ASA SQL Reference Manual)*.

ГЛАВА 15

Доступ к удаленным данным

Об этой главе

Adaptive Server Anywhere может обращаться к данным, хранящимся на других серверах (как на серверах Sybase, так и на серверах другого типа). При этом обработка данных ведется так же, как если бы эти данные находились на локальном сервере.

В этой главе описывается порядок конфигурирования Adaptive Server Anywhere для обеспечения доступа к удаленным данным.

Содержание

Введение

С помощью Adaptive Server Anywhere возможно следующее:

- обращение к данным в реляционных базах данных, таких как Sybase, Oracle и DB2;
- обращение к данным в настольных приложениях, таких как электронные таблицы Excel, базы данных MS Access, FoxPro, а также к текстовым файлам;
- обращение к любому другому источнику данных, который поддерживает интерфейс ODBC;
- выполнение соединения между локальными и удаленными данными;
- выполнение соединения между таблицами в разных базах данных Adaptive Server Anywhere;
- использование средств Adaptive Server Anywhere в отношении тех источников данных, в которых подобные средства не предусмотрены (например, можно использовать функцию Java в отношении данных, сохраненных в Oracle, или выполнить подзапрос в электронных таблицах; средства Adaptive Server Anywhere, не поддерживаемые удаленным источником данных, применяются путем соответствующей обработки данных после их предварительного извлечения);
- использование Adaptive Server Anywhere для перемещения данных из одного местоположения в другое с использованием операций выбора и вставки;
- непосредственное обращение к удаленным серверам с использованием режима ретрансляции;
- выполнение вызовов удаленных процедур на других серверах.

Adaptive Server Anywhere обеспечивает получение доступа к следующим внешним источникам данных:

- Adaptive Server Anywhere;
- Adaptive Server Enterprise;
- Oracle;
- IBM DB2;
- Microsoft SQL Server;
- другие источники данных ODBC.

Для получения информации о доступности платформ см. раздел "Поддерживаемые в Adaptive Server Anywhere операционные системы" (Adaptive Server Anywhere supported operating systems) на стр. 138 в документе *"Введение в SQL Anywhere Studio" (Introducing SQL Anywhere Studio)*.

Доступ к удаленным данным из PowerBuilder DataWindows

Обращение к удаленным данным из PowerBuilder DataWindow осуществляется путем установки значения 1 в параметре DBParm Block при подключении.

- В среде проектирования для установки параметра Block следует открыть закладку Transaction в диалоге Database Profile Setup и установить значение 1 для Retrieve Blocking Factor.
- В строке подключения используйте следующую фразу:

```
DBParm="Block=1"
```

Основные понятия

В этом разделе описываются основные понятия, относящиеся к получению доступа к удаленным данным.

Отображение удаленных таблиц

Adaptive Server Anywhere обеспечивает такое представление таблиц для клиентского приложения, которое соответствует тому случаю, когда все данные в таблицах хранятся в базе данных, подключенной к этому приложению. При выполнении запроса, для которого требуются данные из удаленных таблиц, внутри системы определяется местоположение в памяти, после чего обеспечивается доступ к удаленному местоположению для получения данных.

Для того чтобы удаленные таблицы были представлены клиенту как локальные таблицы, следует создать локальные таблицы прокси, обеспечивающие отображение удаленных данных.

❖ **Создание таблицы прокси**

1. Определите сервер, на котором находятся удаленные данные. Это определение заключается в указании типа и местоположения удаленного сервера.

   Для получения дополнительной информации см. раздел "Работа с удаленными серверами" на стр. 448.

2. Отобразите регистрационную информацию локального пользователя в регистрационной информации пользователя удаленного сервера (если эти данные для двух серверов отличаются друг от друга).

   Для получения дополнительной информации см. раздел "Работа с внешними регистрационными данными" на стр. 453.

3. Создайте определение таблицы прокси. Этим определением устанавливается отображение локальной таблицы прокси в удаленной таблице. В определении указываются сервер, на котором находится удаленная таблица, имя базы данных, имя владельца, имя таблицы, имена столбцов удаленной таблицы.

Для получения дополнительной информации см. раздел "Работа с таблицами прокси" на стр. 455.

Управление отображением удаленных таблиц

Для управления отображением удаленных таблиц и определениями удаленных серверов можно использовать Sybase Central или иное инструментальное средство, например, Interactive SQL, в котором возможно непосредственное выполнение операторов SQL.

Классы серверов

Класс сервера назначается каждому удаленному серверу. Классом сервера определяется метод доступа, используемый для взаимодействия с сервером. Применяемый метод доступа зависит от типа удаленного сервера. Классы серверов используются в Adaptive Server Anywhere для получения подробной информации о возможностях серверов. Adaptive Server Anywhere корректирует свое взаимодействие с удаленным сервером в соответствии с возможностями данного сервера.

В настоящее время существует две группы классов серверов. В первой группе представлены серверы, основанные на JDBC, вторая группа включает серверы, основанные на ODBC.

Классы серверов, основанных на JDBC:

- **asajdbc** – для Adaptive Server Anywhere (версия 6 и выше);
- **asejdbc** – для Adaptive Server Enterprise и SQL Server (версия 10 и выше).

Классы серверов, основанных на ODBC:

- **asaodbc** – для Adaptive Server Anywhere (версия 5.5 и выше);
- **aseodbc** – для Adaptive Server Enterprise и SQL Server (версия 10 и выше);
- **db2odbc** – для IBM DB2;
- **mssodbc** – для Microsoft SQL Server;
- **oraodbc** – для серверов Oracle (версия 8.0 и выше);
- **odbc** – для всех других источников данных ODBC.

Для получения полного описания классов удаленных серверов см. раздел "Классы серверов для доступа к удаленным данным" на стр. 475.

Работа с удаленными серверами

Отображение удаленных объектов в локальной таблице прокси становится возможным только после определения удаленного сервера, на котором хранится удаленный объект. При определении удаленного сервера происходит добавление соответствующей записи о сервере в таблицу *sysservers*. В этом разделе описывается порядок создания, изменения и удаления определения удаленного сервера.

Создание удаленных серверов

Для создания определения удаленного сервера используется оператор CREATE SERVER. Допускается непосредственное выполнение операторов, но можно использовать и Sybase Central.

В случае ODBC-подключений каждый удаленный сервер соответствует источнику данных ODBC. В некоторых системах, в том числе и в Adaptive Server Anywhere, каждый источник данных описывает некоторую базу данных, поэтому для каждой базы данных требуется отдельное определение удаленного сервера.

Для создания сервера необходимо иметь полномочия RESOURCE.

Для получения полного описания оператора CREATE SERVER см. раздел "Оператор CREATE SERVER" (CREATE SERVER statement) на стр. 300 в документе *"Справочник по SQL для ASA" (ASA SQL Reference Manual)*.

Пример 1

С помощью следующего оператора в таблице *sysservers* создается запись для сервера Adaptive Server Enterprise с именем *ASEserver*:

```
CREATE SERVER ASEserver
CLASS 'ASEJDBC'
USING 'rimu:6666'
```

Где:

- **ASEserver** – имя удаленного сервера;
- **ASEJDBC** – ключевое слово, указывающее то, что данный сервер является сервером Adaptive Server Enterprise и подключение к нему основано на JDBC;
- **rimu:6666** - имя машины и номер порта TCP/IP удаленного сервера.

Пример 2

С помощью следующего оператора в таблице sysservers создается запись для ODBC-сервера Adaptive Server Anywhere с именем *testasa*:

```
CREATE SERVER testasa
CLASS 'ASAODBC'
USING 'test4'
```

Где:

- **testasa** – имя, под которым удаленный сервер фигурирует в этой базе данных;

- **ASAODBC** – ключевое слово, указывающее то, что данный сервер является сервером Adaptive Server Anywhere, и подключение к нему основано на ODBC;
- **test4** – имя источника данных ODBC.

Создание удаленных серверов с использованием Sybase Central

Создать удаленный сервер можно с помощью соответствующего мастера в Sybase Central. Для получения дополнительной информации см. раздел "Создание удаленных серверов" на стр. 448.

❖ Создание удаленного сервера (Sybase Central)

1. Выполните подключение к базе данных хоста в Sybase Central.
2. Откройте папку Remote Servers для этой базы данных.
3. Дважды щелкните Add Remote Server.
4. На первой странице мастера введите имя, которое должно использоваться для удаленного сервера. Это имя является обычной ссылкой на удаленный сервер в локальной базе данных; это имя не должно обязательно соответствовать имени, которое сообщает сервер. Нажмите Next.
5. Выберите соответствующий тип сервера и нажмите Next.
6. Выберите метод доступа к данным (JDBC или ODBC) и введите информацию о подключении.
   - Для ODBC: укажите имя источника данных;
   - Для JDBC: укажите URL в виде *имя-машины:номер-порта*.

   Метод доступа к данным (JDBC или ODBC) – это метод, используемый в Adaptive Server Anywhere для получения доступа к удаленной базе данных. Он не связан с методом, используемым в Sybase Central для подключения к собственной базе данных.
7. Нажмите Next. Укажите, должен ли удаленный сервер быть доступен только для чтения.
8. Нажмите Finish для создания определения удаленного сервера.

Удаление удаленных серверов

Для удаления удаленного сервера из системных таблиц Adaptive Server Anywhere можно использовать Sybase Central или оператор DROP SERVER. Выполнение данной операции возможно только после сброса всех удаленных таблиц, определенных в данном сервере.

Для удаления удаленного сервера необходимо обладать полномочиями администратора БД.

❖ **Удаление удаленного сервера (Sybase Central)**

1 Выполните подключение к базе данных хоста в Sybase Central.

2 Откройте папку Remote Servers.

3 Щелкните правой кнопкой мыши на удаленном сервере, который необходимо удалить, и выберите Delete во всплывающем меню.

❖ **Удаление удаленного сервера (SQL)**

1 Выполните подключение к базе данных хоста в Interactive SQL с помощью драйвера jConnect (для связи на основе JDBC).

2 Выполните оператор DROP SERVER.

Для получения дополнительной информации см. раздел "Оператор DROP SERVER" (DROP SERVER statement) на стр. 373 в документе "*Справочник по SQL для ASA" (ASA SQL Reference Manual)*.

Пример

С помощью следующего оператора удаляется сервер с именем ***testasa***.

```
DROP SERVER testasa
```

Изменение удаленных серверов

Для изменения атрибутов сервера можно использовать Sybase Central или оператор ALTER SERVER. Это изменение не вступает в силу до следующего подключения к удаленному серверу.

Для изменения сервера необходимо иметь полномочия RESOURCE.

❖ **Изменение свойств удаленного сервера (Sybase Central)**

1 Выполните подключение к базе данных хоста в Sybase Central.

2 Откройте папку Remote Servers для этой базы данных.

3 Щелкните правой кнопкой мыши на удаленном сервере и выберите Properties во всплывающем меню.

4 Установите требуемым образом свойства удаленного сервера.

❖ **Изменение свойств удаленного сервера (SQL)**

1 Выполните подключение к базе данных хоста в Interactive SQL с помощью драйвера jConnect (для связи на основе JDBC).

2 Выполните оператор ALTER SERVER.

Пример

С помощью следующего оператора класс сервера с именем *ASEserver* заменяется на класс *aseodbc*. В этом примере именем источника данных для сервера является *ASEserver*.

```
ALTER SERVER ASEserver
CLASS 'aseodbc'
```

Оператор ALTER SERVER также можно использовать для включения/отключения некоторых возможностей сервера.

Для получения дополнительной информации см. раздел "Оператор ALTER SERVER" (ALTER SERVER statement) на стр. 206 в документе *"Справочник по SQL для ASA" (ASA SQL Reference Manual)*.

Вывод списка удаленных таблиц на сервере

При конфигурировании Adaptive Server Anywhere может потребоваться список удаленных таблиц, доступных на конкретном сервере. Список таблиц на сервере может быть получен с помощью процедуры *sp_remote_tables*.

```
sp_remote_tables имя сервера
                 [,имя таблицы]
                 [, владелец ]
                 [, база данных]
```

Если введено *имя таблицы*, *владелец* или *база данных*, то в список включаются только те таблицы, которые соответствуют этим параметрам.

Например, для получения списка всех рабочих таблиц Microsoft Excel, доступных в источнике данных ODBC с именем *excel*, введите

```
sp_remote_tables  excel
```

Для получения списка всех таблиц в базе данных *production* в ASE с именем *asetest*, владельцем которых является 'fred', введите

```
sp_remote_tables  asetest, null, fred, production
```

Для получения дополнительной информации см. раздел "Системная процедура sp_remote_tables" (sp_remote_tables system procedure) на стр. 670 в документе *"Справочник по SQL для ASA"(ASA SQL Reference Manual)*.

Вывод списка возможностей удаленного сервера

Информация о возможностях удаленного сервера выводится с помощью процедуры *sp_servercaps*. В Adaptive Server Anywhere эта информация о возможностях используется для определения того, сколько операторов SQL может быть передано удаленному серверу.

Системные таблицы, в которых должна содержаться информация о возможностях сервера, не заполняются до тех пор, пока Adaptive Server Anywhere не подключится к удаленному серверу. Эта информация поступает из системных таблиц SYSCAPABILITY и SYSCAPABILITYNAME. Указанное имя сервера (*servername*) должно совпадать с

именем сервера, введенным в операторе CREATE SERVER.

Выполните хранимую процедуру *sp_servercaps*:

```
sp_servercaps  servername
```

Для получения дополнительной информации см. раздел "Системная процедура sp_servercaps" (sp_servercaps system procedure) на стр. 671 в документе "*Справочник по SQL для ASA*"*(ASA SQL Reference Manual)*.

Работа с внешними регистрационными данными

По умолчанию в Adaptive Server Anywhere используются имена и пароли клиентов во всех случаях, когда производится подключение к удаленному серверу со стороны этих клиентов. Однако это поведение по умолчанию может быть переопределено путем создания внешних регистрационных данных. Внешние регистрационные данные – это альтернативные имена и пароли для входа, которые используются при связи с удаленным сервером.

Если используются интегрированные регистрационные данные, то имя IQ и пароль клиента IQ совпадают с кодом и паролем пользователя базы данных, которые через код пользователя IQ отображаются в системных регистрационных данных (syslogin).

Для получения дополнительной информации см. раздел "Использование интегрированных регистрационных данных" (Using integrated logins) на стр. 83 в документе "*Руководство по администрированию баз данных ASA" (ASA Database Administration Guide)*.

Создание внешних регистрационных данных

Создать внешние регистрационные данные пользователя можно с помощью Sybase Central или с использованием оператора CREATE EXTERNLOGIN.

Добавление или изменение внешних регистрационных данных возможно только при использовании регистрационного имени и учетной записи администратора БД.

❖ Создание внешних регистрационных данных (Sybase Central)

1. Выполните подключение к базе данных хоста в Sybase Central.
2. Откройте папку Remote Servers для этой базы данных и выберите удаленный сервер.
3. Щелкните правой кнопкой мыши на удаленном сервере и выберите Properties во всплывающем меню.
4. На закладке External Logins в окне свойств нажмите New и установите требуемые параметры в появившемся диалоге.
5. Нажмите OK для сохранения изменений.

❖ Создание внешних регистрационных данных (SQL)

1. Выполните подключение к базе данных хоста в Interactive SQL с помощью драйвера jConnect (для связи на основе JDBC).
2. Выполните оператор CREATE EXTERNLOGIN.

Пример

С помощью следующего оператора локальный пользователь **fred** получает доступ к серверу **ASEserver**, при этом используется имя для

удаленной регистрации **frederick** и пароль **banana**.

```
CREATE EXTERNLOGIN fred
TO ASEserver
REMOTE LOGIN frederick
IDENTIFIED BY banana
```

Для получения дополнительной информации см. раздел "Оператор CREATE EXTERNLOGIN" (CREATE EXTERNLOGIN statement) на стр. 275 в документе "*Справочник по SQL для ASA*" *(ASA SQL Reference Manual)*.

Удаление внешних регистрационных данных

Для удаления внешних регистрационных данных из системных таблиц Adaptive Server Anywhere можно использовать Sybase Central или оператор DROP EXTERNLOGIN.

Удаление внешних регистрационных данных возможно только при использовании регистрационного имени и учетной записи администратора БД.

❖ Удаление внешних регистрационных данных (Sybase Central)

1. Выполните подключение к базе данных хоста в Sybase Central.
2. Откройте папку Remote Servers.
3. Щелкните правой кнопкой мыши на удаленном сервере и выберите Delete во всплывающем меню.

❖ Удаление внешних регистрационных данных (SQL)

1. Выполните подключение к базе данных хоста в Interactive SQL с помощью драйвера jConnect (для связи на основе JDBC).
2. Выполните оператор DROP EXTERNLOGIN.

Пример

С помощью следующего оператора выполняется удаление внешних регистрационных данных для локального пользователя *fred*, созданных в примере выше:

```
DROP EXTERNLOGIN fred TO ASEserver
```

См. также:

- раздел "Оператор DROP EXTERNLOGIN" (DROP EXTERNLOGIN statement) на стр. 370 в документе *"Справочник по SQL для ASA" (ASA SQL Reference Manual)*.

Работа с таблицами прокси

Прозрачность местоположения удаленных данных достигается путем создания локальной **таблицы прокси**, которая обеспечивает отображение в удаленном объекте. Для создания таблицы прокси следует использовать один из следующих операторов:

- Если таблица в удаленном местоположении уже существует, используйте оператор CREATE EXISTING TABLE. С помощью этого оператора определяется таблица прокси для таблицы, существующей на удаленном сервере.
- Если таблица в удаленном местоположении не существует, используйте оператор CREATE TABLE. С помощью этого оператора на удаленном сервере создается новая таблица, а также определяется таблица прокси для вновь созданной таблицы.

Указание местоположения таблицы прокси

Ключевое слово AT, определяющее местоположение существующего объекта, используется как в операторе CREATE TABLE, так и в операторе CREATE EXISTING TABLE. Строка местоположения состоит из четырех элементов, отделяемых друг от друга точкой или точкой с запятой. Если в качестве разделителя используется точка с запятой, то в полях владельца и базы данных могут использоваться имена и расширения файлов.

Раздел AT имеет следующий синтаксис:

```
... AT 'сервер.база-данных.владелец.имя-таблицы'
```

- **Сервер.** Имя, под которым сервер фигурирует в текущей базе данных; это имя определяется в операторе CREATE SERVER. Это поле обязательно для всех источников удаленных данных.
- **База данных.** Назначение этого поля зависит от источника данных. В некоторых случаях это поле не используется и должно быть оставлено пустым. Однако разделители-точки должны быть введены.

  В Adaptive Server Enterprise в поле *базы данных* определяется база данных, в которой существует таблица, например, *master* или *pubs2*.

  В Adaptive Server Anywhere это поле не используется и должно быть оставлено пустым.

  В Excel, Lotus Notes и Access необходимо указать имя файла, содержащего таблицу. Если в имени файла содержится точка, то в качестве разделителя следует использовать точку с запятой.
- **Владелец.** Если в базе данных поддерживается концепция владения, в этом поле указывается имя владельца. Это поле необходимо только в том случае, когда несколько владельцев имеют таблицы с одинаковыми именами.

- **Имя-таблицы.** Здесь указывается имя таблицы. В случае электронной таблицы Excel здесь вводится имя листа рабочей книги. Если имя таблицы не введено, то предполагается, что имя удаленной таблицы совпадает с именем локальной таблицы прокси.

Примеры

В приведенных ниже примерах иллюстрируется использование строк местоположений.

- Adaptive Server Anywhere:

```
'testasa..DBA.employee'
```

- Adaptive Server Enterprise:

```
'ASEServer.pubs2.dbo.publishers'
```

- Excel:

```
'excel;d:\pcdb\quarter3.xls;;sheet1$'
```

- Access:

```
'access;\\server1\production\inventory.mdb;;parts'
```

Создание таблиц прокси (Sybase Central)

Для создания таблицы прокси используется Sybase Central или оператор CREATE EXISTING TABLE.

С помощью оператора CREATE EXISTING TABLE создается таблица прокси, которая отображается в существующей таблице на удаленном сервере. Adaptive Server Anywhere получает информацию об атрибутах столбцов и об индексах из удаленного объекта.

❖ Создание таблицы прокси (Sybase Central)

1. Выполните подключение к базе данных хоста в Sybase Central.
2. Выполните одно из следующих действий:
   - В папке Tables дважды щелкните Add Proxy Table.
   - В папке Remote Servers щелкните правой кнопкой мыши на удаленном сервере и выберите Add Proxy Table во всплывающем меню.
   - Выберите папку Tables и затем выберите File➤New➤Proxy Table.
3. Выполняйте указания мастера.

Совет

Список таблиц прокси выводится в папке Remote Servers ниже названия соответствующего удаленного сервера. Эти таблицы также представлены в папке Tables. От других таблиц их отличает символ P в значках таблиц.

Создание таблиц прокси с использованием оператора CREATE EXISTING TABLE

С помощью оператора CREATE EXISTING TABLE создается таблица прокси, которая отображается в существующей таблице на удаленном сервере. Adaptive Server Anywhere получает информацию об атрибутах столбцов и об индексах из удаленного объекта.

❖ Создание таблицы прокси с использованием оператора CREATE EXISTING TABLE (SQL)

1 Выполните подключение к базе данных хоста.

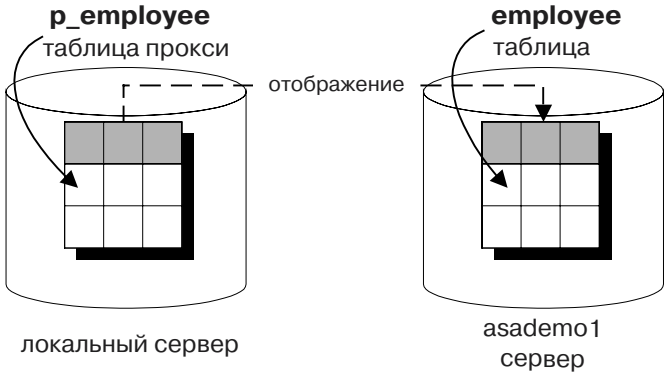
2 Выполните оператор CREATE EXISTING TABLE.

Для получения дополнительной информации см. раздел "Оператор CREATE EXISTING TABLE" (CREATE EXISTING TABLE statement) на стр. 272 в документе "*Справочник по SQL для ASA*" *(ASA SQL Reference Manual)*.

Пример 1

Для создания на данном сервере таблицы прокси с именем *p_employee*, которая отображается в удаленной таблице с именем *employee* на сервере с именем *asademo1*, используйте следующий синтаксис:

```
CREATE EXISTING TABLE p_employee
AT 'asademo1..DBA.employee'
```



Пример 2

С помощью следующего оператора обеспечивается отображение таблицы прокси *a1* в файле mydbfile.mdb Microsoft Access. В этом примере в ключевом слове AT в качестве разделителя используется точка с запятой (;). Сервер, определенный для Microsoft Access, имеет имя *access*.

```
CREATE EXISTING TABLE a1
AT'access;d:\mydbfile.mdb;;a1'
```

Создание таблицы прокси с использованием оператора CREATE TABLE

С помощью оператора CREATE TABLE на удаленном сервере создается новая таблица, а также, при использовании параметра AT, определяется таблица прокси для этой созданной таблицы. При вводе оператора CREATE TABLE используются типы данных Adaptive Server Anywhere. Adaptive Server Anywhere выполняет автоматическое преобразование типов данных во внутренние типы данных удаленного сервера.

Если для создания локальной и удаленной таблиц используется оператор CREATE TABLE, а затем с помощью оператора DROP TABLE сбрасывается таблица прокси, то при этом также сбрасывается удаленная таблица. Однако если удаленную таблицу необходимо оставить, то с помощью оператора DROP TABLE можно сбросить только таблицу прокси, созданную с использованием оператора CREATE EXISTING TABLE.

Для получения дополнительной информации см. раздел "Оператор CREATE TABLE" (CREATE TABLE statement) на стр. 321 в документе *"Справочник по SQL для ASA" (ASA SQL Reference Manual)*.

❖ Создание таблицы прокси с использованием оператора CREATE EXISTING TABLE (SQL)

1 Выполните подключение к базе данных хоста.

2 Выполните оператор CREATE TABLE.

Для получения дополнительной информации см. раздел "Оператор CREATE TABLE" (CREATE TABLE statement) на стр. 321 в документе *"Справочник по SQL для ASA" (ASA SQL Reference Manual)*.

Пример

С помощью следующего оператора на удаленном сервере *asademo1* создается таблица с именем *employee*, а также создается таблица прокси с именем *members*, отображаемая в удаленном местоположении.

```
CREATE TABLE members
( membership_id INTEGER NOT NULL,
member_name CHAR(30) NOT NULL,
office_held CHAR( 20 ) NULL)
AT 'asademo1..DBA.employee'
```

Вывод списка столбцов удаленной таблицы

Если необходимо ввести оператор CREATE EXISTING и задать список столбцов, то в таком случае целесообразно получить список столбцов, доступных в удаленной таблице. Системная процедура *sp_remote_columns* обеспечивает вывод списка столбцов удаленной таблицы и описаний соответствующих типов данных.

```
sp_remote_columns  имя-сервера, имя-таблицы
[, владелец ] [, база-данных]
```

Если введены имя таблицы, владелец или имя базы данных, то в выводимый список включаются только те столбцы, которые соответствуют этим параметрам.

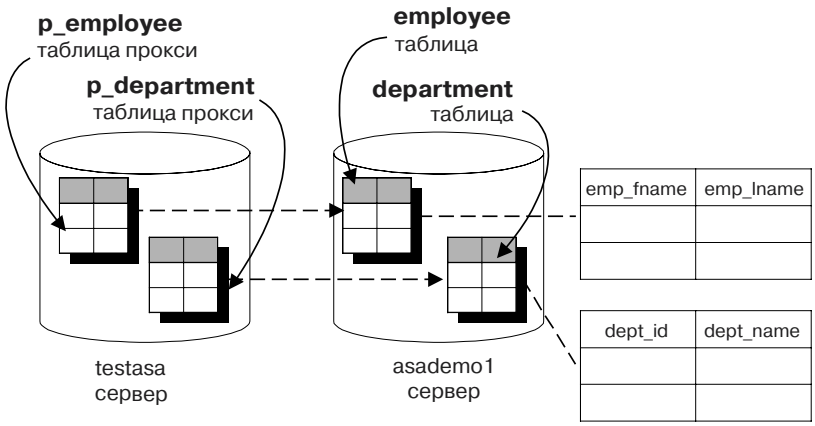
Например, в результате выполнения следующей процедуры выводится список столбцов таблицы *sysobjects* в базе данных *production* на сервере Adaptive Server Enterprise с именем *asetest*:

```
sp_remote_columns  asetest, sysobjects, null, production
```

Для получения дополнительной информации см. раздел "Системная процедура sp_remote_columns" (sp_remote_columns system procedure) на стр. 668 в документе "*Справочник по SQL для ASA*"(*ASA SQL Reference Manual*).

Соединение удаленных таблиц

На приведенном ниже рисунке иллюстрируется, как удаленные таблицы Adaptive Server Anywhere *employee* и *department* демострационной базы данных отображаются на локальном сервере с именем *testasa*.



В реальной ситуации возможно использование соединений между таблицами, существующими в разных базах данных Adaptive Server Anywhere. Здесь для пояснения основных принципов описывается простой случай, когда используется только одна база данных.

❖ Соединение двух удаленных таблиц (SQL)

1 Создайте новую базу данных с именем *empty.db*.

Данные в этой базе данных отсутствуют. Эта база данных будет использоваться только с целью определения удаленных объектов и обращения к демонстрационной базе данных.

2 Запустите *empty.db* и демонстрационную базу данных на сервере базы данных. Это можно выполнить с помощью следующей командной строки, выполняемой из папки установки:

```
dbeng8 asademo empty
```

3 Выполните подключение к *empty.db* из Interactive SQL с использованием кода пользователя **DBA** и пароля **SQL**.

4 В новой базе данных создайте удаленный сервер с именем *testasa*. Класс этого сервера – *asaodbc*, а информация о подключении – **'ASA 8.0 Sample'**:

```
CREATE SERVER testasa
CLASS 'asaodbc'
USING 'ASA 8.0 Sample'
```

5 В этом примере для удаленной базы данных и для локальной базы данных используются одинаковые код пользователя и пароль, поэтому внешние регистрационные данные не требуются.

6 Определите таблицу прокси *employee*:

```
CREATE EXISTING TABLE employee
AT 'testasa..DBA.employee'
```

7 Определите таблицу прокси *department:*

```
CREATE EXISTING TABLE department
AT 'testasa..DBA.department'
```

8 Для выполнения соединения воспользуйтесь оператором SELECT с таблицами прокси.

```
SELECT emp_fname, emp_lname, dept_name
FROM employee JOIN department
ON employee.dept_id = department.dept_id
ORDER BY emp_lname
```

Соединение таблиц из нескольких локальных баз данных

На сервере Adaptive Server Anywhere может использоваться несколько локальных баз данных, работающих одновременно. Посредством определения таблиц в других локальных базах данных Adaptive Server Anywhere как удаленных таблиц можно установить перекрестные соединения между базами данных.

Для получения дополнительной информации об указании нескольких баз данных см. раздел "Значение параметра USING в операторе CREATE SERVER" на стр. 478.

Пример

Предположим, что при использовании базы данных *db1* необходимо получить доступ к данным в таблицах в базе данных *db2*. Для этого потребуется создать определения таблиц прокси, указывающие на таблицы в базе данных *db2*. Например, на сервере Adaptive Server Anywhere с именем *testasa* существует три базы данных – *db1*, *db2* и *db3*.

- Если используется ODBC, создайте имя источника данных ODBC для каждой базы данных, к которой должен быть получен доступ.
- Выполните подключение к одной из баз данных, из которой будет выполняться установление соединений. Например, выполните подключение к базе данных *db1*.
- Выполните CREATE SERVER для всех других локальных баз данных, к которым должен быть получен доступ. Тем самым создается подключение **с возвратом** к серверу Adaptive Server Anywhere.

  ```
  CREATE SERVER local_db2
  CLASS 'asaodbc'
  USING 'testasa_db2'

  CREATE SERVER local_db3
  CLASS 'asaodbc'
  USING 'testasa_db3'
  ```

  Если используется JDBC:

  ```
  CREATE SERVER local_db2
  CLASS 'asajdbc'
  USING 'mypc1:2638/db2'

  CREATE SERVER local_db3
  CLASS 'asajdbc'
  USING 'mypc1:2638/db3'
  ```

- С использованием CREATE EXISTING создайте определения таблиц прокси, отображаемых в таблицы в других базах данных, к которым должен быть получен доступ.

  ```
  CREATE EXISTING TABLE employee
  AT 'local_db2...employee'
  ```

Посылка внутренних операторов в удаленные серверы

Для посылки в удаленный сервер одного или нескольких операторов, имеющих внутренний синтаксис этого удаленного сервера, используется оператор FORWARD TO. Этот оператор может использоваться двумя способами:

- посылка оператора в удаленный сервер;
- перевод Adaptive Server Anywhere в режим ретрансляции для посылки серии операторов в удаленный сервер.

Если подключиться к заданному серверу невозможно, пользователь получает соответствующее сообщение с указанием причины. Если подключение выполнено, то все результирующие данные преобразуются в данные, которые могут быть распознаны клиентской программой.

Оператор FORWARD TO может использоваться для проверки правильности конфигурирования сервера. Если после посылки оператора удаленному серверу сообщение об ошибке от Adaptive Server Anywhere не возвращается, это означает, что удаленный сервер сконфигурирован правильно.

Для получения дополнительной информации см. раздел "Оператор FORWARD TO" (FORWARD TO statement) на стр. 400 в документе "*Справочник по SQL для ASA" (ASA SQL Reference Manual)*.

Пример 1

Следующий оператор проверяет связь с сервером *ASEserver* путем выбора строки версии:

```
FORWARD TO ASEserver {SELECT @@version}
```

Пример 2

Следующие операторы поддерживают сеанс ретрансляции с сервером *ASEserver*:

```
FORWARD TO ASEserver
select * from titles
select * from authors
FORWARD TO
```

Использование вызовов удаленных процедур (RPC)

Пользователи Adaptive Server Anywhere могут направлять вызовы процедур тем удаленным серверам, которые поддерживают данную возможность.

Эта возможность поддерживается в Sybase Adaptive Server Anywhere, Sybase Adaptive Server Enterprise, Oracle и DB2. Вызов удаленной процедуры выполняется примерно так же, как вызов локальной процедуры.

Создание удаленных процедур

Для вызова удаленной процедуры используется Sybase Central или оператор CREATE PROCEDURE.

Для создания удаленной процедуры необходимо обладать полномочиями администратора БД.

❖ **Вызов удаленной процедуры (Sybase Central)**

1 Выполните подключение к базе данных хоста в Sybase Central.

2 Откройте папку Remote Servers.

3 Щелкните правой кнопкой мыши на удаленном сервере, для которого необходимо создать удаленную процедуру, и выберите Properties в меню File.

4 На закладке Remote Procedures нажмите New, после этого выполняйте указания мастера.

> **Совет**
>
> Добавить удаленную процедуру также можно, щелкнув правой кнопкой мыши на удаленном сервере и выбрав Add Remote Procedure во всплывающем меню.

❖ **Вызов удаленной процедуры (SQL)**

1 Предварительно определите процедуру для Adaptive Server Anywhere.

Используемый синтаксис аналогичен синтаксису в определении локальной процедуры, но за исключением того, что вместо операторов SQL, составляющих тело вызова, используется строка местоположения, которая определяет местоположение процедуры.

```
CREATE PROCEDURE remotewho ()
AT 'bostonase.master.dbo.sp_who'
```

2 Выполните процедуру следующим образом:

```
call remotewho()
```

Для получения дополнительной информации см. раздел

"Оператор CREATE PROCEDURE" (CREATE PROCEDURE statement) на стр. 284 в документе "*Справочник по SQL для ASA*" *(ASA SQL Reference Manual)*.

Пример

Ниже приводится пример с использованием параметра:

```
CREATE PROCEDURE remoteuser (IN uname char(30))
AT 'bostonase.master.dbo.sp_helpuser'
call remoteuser('joe')
```

Типы данных для удаленных процедур

Ниже перечислены типы данных, разрешенные для параметров RPC. Использование других типов данных не допускается.

- [UNSIGNED] SMALLINT
- [UNSIGNED] INT
- [UNSIGNED] BIGINT
- TINYINT
- REAL
- DOUBLE
- CHAR
- BIT

Типы данных NUMERIC и DECIMAL разрешены для параметров IN, но не для параметров OUT и INOUT.

Сброс удаленных процедур

Для сброса (удаления) удаленной процедуры используется Sybase Central или оператор DROP PROCEDURE.

Для сброса удаленной процедуры необходимо обладать полномочиями администратора БД.

❖ Сброс удаленной процедуры (Sybase Central)

1. Откройте папку Remote Servers.
2. Щелкните правой кнопкой мыши на удаленном сервере и выберите Properties в меню File.
3. На закладке Remote Procedures выберите удаленную процедуру и нажмите Delete.

❖ Сброс удаленной процедуры (SQL)

1. Выполните подключение к базе данных.
2. Выполните оператор DROP PROCEDURE.

Для получения дополнительной информации см. раздел "Оператор DROP" (DROP statement) на стр. 366 в документе "*Справочник по SQL для ASA*" (*ASA SQL Reference Manual*).

Пример

Сбросьте удаленную процедуру с именем *remoteproc*.

```
DROP PROCEDURE remoteproc
```

Управление транзакциями и удаленные данные

Транзакции позволяют сгруппировать операторы SQL так, чтобы операторы могли рассматриваться как единый блок. Все действия, выполняемые этими операторами, либо полностью подтверждаются для базы данных, либо ни одно из этих действия не подтверждается.

В Adaptive Server Anywhere управление транзакциями с использованием удаленных таблиц в целом аналогично управлению транзакциями для локальных таблиц, но есть и некоторые отличия. Эти вопросы рассматриваются в следующем разделе.

Общие сведения о транзакциях приведены в разделе "Использование транзакций и уровней изоляции" на стр. 89.

Обзор управления транзакциями при использовании удаленных серверов

В управлении транзакциями, в которых участвуют удаленные серверы, используется двухфазный протокол подтверждения. В Adaptive Server Anywhere реализована стратегия, обеспечивающая целостность транзакций в большинстве сценариев. Тем не менее, если в транзакции участвует несколько удаленных серверов, существует вероятность того, что некоторый распределенный блок операций останется в неопределенном состоянии. Несмотря на то, что используется двухфазный протокол подтверждения, какой-либо процесс восстановления не предусмотрен.

Общая логика управления пользовательской транзакцией заключается в следующем:

1 Adaptive Server Anywhere информирует удаленный сервер о начале работы посредством передачи уведомления BEGIN TRANSACTION.

2 Когда транзакция готова к подтверждению, Adaptive Server Anywhere направляет уведомление PREPARE TRANSACTION на все удаленные серверы, участвующие в транзакции. Тем самым обеспечивается готовность удаленного сервера к подтверждению транзакции.

3 Если один из запросов PREPARE TRANSACTION выполнен неудачно, все удаленные серверы получают инструкцию об откате текущей транзакции.

   Если все запросы PREPARE TRANSACTION выполнены успешно, сервер направляет запрос COMMIT TRANSACTION на все удаленные серверы, участвующие в транзакции.

Любой оператор, которому предшествует BEGIN TRANSACTION, может инициировать транзакцию. Все остальные операторы, направляемые удаленному серверу, воспринимаются как отдельный удаленный рабочий блок.

Ограничения на управление транзакциями

На управление транзакциями накладываются следующие ограничения:

- Точки сохранения в удаленные серверы не переносятся.
- Если в транзакции, в которой участвуют удаленные серверы, имеются вложенные операторы BEGIN TRANSACTION и COMMIT TRANSACTION, то обрабатывается набор этих операторов только на самом верхнем уровне вложения. Набор операторов BEGIN TRANSACTION и COMMIT TRANSACTION на самом нижнем уровне вложения удаленным серверам не передается.

Внутренние операции

В этом разделе описываются основные операции на удаленных серверах, выполняемые средствами Adaptive Server Anywhere согласно полученным от клиентских приложений инструкциям.

Разбор запроса

Оператор, получаемый от клиента, подвергается синтаксическому разбору. Если оператор не является допустимым оператором SQL Adaptive Server Anywhere, выводится сообщение об ошибке.

Нормализация запроса

Следующий этап называется нормализацией запроса. На этом этапе проверяются ссылочные объекты, а также совместимость некоторых типов данных.

Например, рассмотрим следующий запрос:

```
SELECT *
FROM t1
WHERE c1 = 10
```

На этапе нормализации запроса проверяется тот факт, что в системных таблицах существует таблица *t1* со столбцом *c1*. Также проверяется совместимость типа данных столбца *c1* со значением 10. Если типом данных столбца является, например, *datetime*, то данный оператор отклоняется.

Предварительная обработка запроса

Предварительная обработка запроса используется для подготовки запроса к оптимизации. В результате этой обработки вид оператора может быть изменен таким образом, что сгенерированный оператор SQL Adaptive Server Anywhere, предназначенный для передачи удаленному серверу, синтаксически отличается от исходного оператора.

При предварительной обработке выполняется расширение представления, в результате чего запрос может применяться к таблицам, на которые ссылается данное представление. Возможны переупорядочение выражений и преобразование подзапросов для повышения эффективности обработки. Например, некоторые подзапросы могут быть преобразованы в соединения.

Возможности сервера

Предыдущие этапы обработки выполняются в отношении всех запросов, как локальных, так и удаленных.

Рассмотренные ниже этапы обработки зависят от типа оператора SQL и от возможностей задействованных удаленных серверов.

Каждый удаленный сервер, определенный в Adaptive Server Anywhere, располагает соответствующим набором возможностей. Информация об этих возможностях хранится в системной таблице *syscapabilities*. Эти возможности инициализируются при первом подключении к удаленному серверу. Универсальный класс сервера *odbc* в большой степени базируется на той информации, которая возвращается драйвером ODBC и используется для определения возможностей. Другие классы сервера, например, *db2odbc*, базируются на более подробной информации о возможностях удаленного сервера данного типа; эта информация дополняет информацию, возвращаемую драйвером.

После инициализации *syscapabilities* для сервера информация о возможностях извлекается только из этой системной таблицы. Пользователь может изменять известные возможности сервера.

Поскольку удаленный сервер не всегда поддерживает все средства того или иного оператора SQL, в Adaptive Server Anywhere оператор должен разбиваться на более простые компоненты до той степени, при которой запрос может быть воспринят удаленным сервером. Средства SQL, не распознаваемые удаленным сервером, должны анализироваться непосредственно в Adaptive Server Anywhere.

Например, в запросе может содержаться оператор ORDER BY. Если удаленный сервер не может выполнить ORDER BY, то ORDER BY в удаленный сервер не передается, и Adaptive Server Anywhere самостоятельно выполняет обработку полученного результата оператором ORDER BY, после чего конечный результат передается пользователю. Таким образом, пользователь имеет в своем распоряжении все функциональные возможности SQL Adaptive Server Anywhere независимо от того, какими средствами располагают другие компоненты.

Полная ретрансляция оператора

Наиболее эффективный способ обработки оператора обычно заключается в том, что максимально возможный объем исходного оператора передается на обработку в задействованный удаленный сервер. Adaptive Server Anywhere всегда передает оператор в максимально возможном объеме. Во многих случаях переданный оператор представляет собой полный оператор, изначально направленный в Adaptive Server Anywhere.

Adaptive Server Anywhere ретранслирует оператор полностью, если:

- все таблицы, указанные в операторе, находятся на одном и том же удаленном сервере;
- удаленный сервер может обработать весь синтаксис данного

оператора.

В относительно редких случаях более эффективным может быть такой вариант, когда ретрансляция оператора выполняется не полностью, и часть обработки выполняет Adaptive Server Anywhere. Например, алгоритм сортировки в Adaptive Server Anywhere может отличаться большей производительностью. В таком случае целесообразно изменить возможности удаленного сервера. Это изменение выполняется с помощью оператора ALTER SERVER.

Для получения дополнительной информации см. раздел "Оператор ALTER SERVER" (ALTER SERVER statement) на стр. 206 в документе "*Справочник по SQL для ASA" (ASA SQL Reference Manual)*.

Частичная ретрансляция оператора

Если в операторе содержатся ссылки на несколько серверов, или оператор использует средства SQL, которые не поддерживаются удаленным сервером, то в таком случае запрос разбивается на более простые компоненты.

Выбор

Разбивка операторов SELECT осуществляется путем удаления той части, которая не может быть передана в удаленный сервер, и обработку которой Adaptive Server Anywhere производит собственными средствами. Например, предположим, что удаленный сервер не может обработать функцию atan2() в следующем операторе:

```
select a,b,c where atan2(b,10) > 3 and c = 10
```

В этом случае в удаленный сервер передается следующий преобразованный оператор:

```
select a,b,c where c = 10
```

Затем Adaptive Server Anywhere на своем уровне применяет функцию "where atan2(b,10) > 3" к промежуточному результирующему набору.

Соединения

При соединении двух таблиц одна из этих таблиц становится так называемой "внешней таблицей". Сканирование внешней таблицы осуществляется с использованием условий WHERE. По каждой обнаруженной строке с указанием сканируется другая таблица, называемая внутренней таблицей, с целью нахождения строки, соответствующей условию соединения.

Этот же алгоритм используется и в случае ссылок на удаленные таблицы. Поскольку затраты на поиск в удаленной таблице обычно значительно выше затрат на поиск в локальной таблице (вследствие затрат на сетевой ввод/вывод), то предпринимается все возможное для того, чтобы удаленная таблица являлась самой внешней таблицей в соединении.

Обновление и удаление

Если Adaptive Server Anywhere не может полностью передать оператор UPDATE или DELETE удаленному серверу, то при обнаружении строка с указанием этот оператор должен быть заменен сканированием таблицы, содержащим максимально возможный объем исходного раздела WHERE, за которым следует позиционированный UPDATE или DELETE типа "Where current of cursor".

Например, если функция **atan2** не поддерживается удаленным сервером, то

```
UPDATE t1
SET a = atan2(b, 10)
WHERE b > 5
```

преобразуется в

```
SELECT a,b
FROM t1
WHERE  b > 5
```

Всякий раз при нахождении строки Adaptive Server Anywhere вычисляет новое значение *a* и выдает:

```
UPDATE t1
SET a = 'new value'
WHERE CURRENT OF CURSOR
```

Если *a* уже имеет значение, которое равно значению "new value", позиционированный оператор UPDATE не требуется и в удаленный сервер не передается.

Для обработки оператора UPDATE или DELETE, для которого требуется сканирование таблицы, удаленный источник данных должен поддерживать возможность выполнения позиционированного UPDATE или DELETE ("Where current of cursor"). Некоторые источники данных эту возможность не поддерживают.

Обновление временных таблиц невозможно

В этом релизе Adaptive Server Anywhere выполнение оператора UPDATE или DELETE невозможно, если в Adaptive Server Anywhere требуется промежуточная временная таблица. Это происходит в запросах с ORDER BY и в некоторых запросах с подзапросами.

Устранение неполадок при доступе к удаленным данным

В этом разделе содержатся рекомендации по поиску и устранению неполадок при работе с удаленными серверами.

Возможности, не поддерживаемые для удаленных данных

Ряд возможностей Adaptive Server Anywhere для удаленных данных не поддерживается. Попытки использования следующих средств могут привести к возникновению проблем:

- применение оператора ALTER TABLE по отношению к удаленным таблицам;
- триггеры, определенные в таблицах прокси, не запускаются;
- SQL Remote;
- типы данных Java;
- внешние ключи, ссылающиеся на удаленные таблицы, игнорируются;
- функции READTEXT, WRITETEXT и TEXTPTR;
- позиционированные операторы UPDATE и DELETE;
- операторы UPDATE и DELETE, требующие создания промежуточной временной таблицы;
- обратная прокрутка относительно курсоров, открытых для удаленных данных (операторами выборки должны быть NEXT или RELATIVE 1);
- Если имя столбца в удаленной таблице является ключевым словом в удаленном сервере, доступ к данным в столбце невозможен. В Adaptive Server Anywhere не могут быть известны все слова, зарезервированные на удаленном сервере. Можно выполнить оператор CREATE EXISTING TABLE и импортировать определение, но выбор этого столбца не допускается.

Учет регистра

Установка учета регистра, выполненная в базе данных Adaptive Server Anywhere, должна соответствовать установке, выполненной для каждого из удаленных серверов.

По умолчанию базы данных Adaptive Server Anywhere создаются без учета регистра. В такой конфигурации может быть получен непредсказуемый результат, если выборка осуществляется из той базы данных, в которой регистр учитывается. Кроме того, получаемый результат варьируется в зависимости от того, передаются ли строковые значения и ORDER BY на

удаленный сервер или вычисляются локально в Adaptive Server Anywhere.

Проблемы связи

Проверить возможность подключения к удаленному серверу можно следующим образом:

- Перед конфигурированием Adaptive Server Anywhere убедитесь, что подключение к удаленному серверу возможно при использовании клиентского инструментального средства, такого как Interactive SQL.
- Для проверки связи и проверки конфигурации удаленного входа выполните простой оператор ретрансляции для удаленного сервера. Например:

```
FORWARD TO testasa {select @@version}
```

- Включите удаленную трассировку для отслеживания взаимодействия с удаленными серверами.

```
SET OPTION cis_option = 2
```

Общие проблемы, связанные с запросами

Для устранения некоторых проблем, связанных с тем, как Adaptive Server Anywhere обрабатывает запрос в отношении удаленной таблицы, особенное значение имеет понимание того, как Adaptive Server Anywhere выполняет такой запрос. Можно получить информацию удаленной трассировки, а также описание плана выполнения запроса следующим образом:

```
SET OPTION cis_option = 6
```

Самоблокировка запросов

Если с одного сервера Adaptive Server Anywhere осуществляется доступ к нескольким базам данных, может потребоваться увеличение числа потоков, используемых сервером базы данных под Windows NT. Для этого используется ключ *-gx* командной строки.

Для поддержки ряда отдельных задач, выполняемых запросом, необходимо иметь достаточное количество потоков. Отсутствие поддержки для выполнения этих задач может привести к самоблокировке запроса.

ГЛАВА 16

Классы серверов для доступа к удаленным данным

Об этой главе

В этой глава описывается взаимодействие Adaptive Server Anywhere с различными классами серверов. Рассматривается следующее:

- типы серверов, поддерживаемые каждым классом серверов;
- значение раздела USING оператора CREATE SERVER для каждого класса сервера;
- специальные требования к конфигурации.

Содержание

Обзор

Классом сервера, указываемым в операторе CREATE SERVER, определяется поведение удаленного подключения. Классы серверов используются в Adaptive Server Anywhere для получения подробной информации о возможностях серверов. Формат операторов SQL в Adaptive Server Anywhere устанавливается в соответствии с возможностями конкретного сервера.

Существует две категории классов серверов:

- классы серверов с использованием JDBC;
- классы серверов с использованием ODBC.

С каждым классом сервера связан набор уникальных характеристик. Эти характеристики должны быть известны администраторам БД и программистам при конфигурировании сервера для обеспечения доступа к удаленным данным.

При чтении этой главы ознакомьтесь как с разделом, описывающим категорию классов серверов (классы с использованием JDBC или ODBC), так и с разделом, посвященным конкретному классу серверов.

Классы серверов с использованием JDBC

Классы серверов с использованием JDBC применяются в том случае, когда в Adaptive Server Anywhere для подключения к удаленному серверу используются виртуальная Java-машина и jConnect 4.0. Существуют следующие классы серверов с использованием JDBC:

- **asajdbc** – Adaptive Server Anywhere (версия 6 и выше);
- **asejdbc** – Adaptive Server Enterprise и SQL Server (версия 10 и выше).

Замечания по конфигурированию JDBC-классов

При обращении к удаленным серверам, которые относятся к JDBC-классам, необходимо учитывать следующее:

- В локальной базе данных должно быть разрешено использование Java.

Для получения дополнительной информации см. раздел "Разрешение использования Java в базе данных" (Java-enabling a database) на стр. 89 в документе *"Руководство по программированию ASA" (ASA Programming Guide)*.

- Для загрузки и выполнения jConnect виртуальной Java-машине требуется больше памяти, чем предусмотрено по умолчанию. Установите следующие значения параметров памяти (как минимум):

```
SET OPTION PUBLIC.JAVA_NAMESPACE_SIZE = 3000000
SET OPTION PUBLIC.JAVA_HEAP_SIZE = 1000000
```

- Поскольку jConnect 4.0 автоматически устанавливается вместе с Adaptive Server Anywhere, то установка дополнительных драйверов не требуется.
- Для повышения производительности Sybase рекомендует использовать класс серверов с использованием ODBC (*asaodbc* или *aseodbc*).
- Любой удаленный сервер, доступ к которому осуществляется с использованием класса серверов *asejdbc* или *asajdbc*, должен поддерживать работу с клиентом, основанным на jConnect 4.x. Имеются следующие сценарии установки jConnect: *SQL_anywhere.SQL (*для Adaptive Server Anywhere) и *SQL_server.SQL* (для Adaptive Server Enterprise). Выполните соответствующий сценарий для удаленного сервера, который предполагается использовать.

Класс серверов asajdbc

Сервер, относящийся к классу серверов *asajdbc*, представляет собой Adaptive Server Anywhere (версии 6 и выше). Какие-либо специальные

требования по конфигурированию источника данных Adaptive Server Anywhere отсутствуют.

Значение параметра USING в операторе CREATE SERVER

Для каждой базы данных Adaptive Server Anywhere, к которой предполагается получить доступ, следует выполнить отдельный оператор CREATE SERVER. Например, если сервер Adaptive Server Anywhere с именем *testasa* выполняется на машине '*banana*' и владеет тремя базами данных (*db1*, *db2*, *db3*), то локальную систему Adaptive Server Anywhere необходимо сконфигурировать следующим образом:

```
CREATE SERVER testasadb1
CLASS 'asajdbc'
USING 'banana:2638/db1'
CREATE SERVER testasadb2
CLASS 'asajdbc'
USING 'banana:2638/db2'
CREATE SERVER testasadb2
CLASS 'asajdbc'
USING 'banana:2638/db3'
```

Если значение */databasename* не задано, то при удаленном подключении используется база данных по умолчанию удаленного сервера Adaptive Server Anywhere.

Для получения дополнительной информации об операторе CREATE SERVER см. раздел "Оператор CREATE TABLE" (CREATE TABLE statement) на стр. 300 в документе *"Справочник по SQL для ASA" (ASA SQL Reference Manual)*.

Класс серверов asejdbc

Сервер, относящийся к классу серверов asejdbc, представляет собой Adaptive Server Enterprise, SQL Server (версии 10 и выше). Какие-либо специальные требования по конфигурированию источника данных Adaptive Server Enterprise отсутствуют.

Преобразования типов данных

При выполнении оператора CREATE TABLE в Adaptive Server Anywhere осуществляется автоматическое преобразование типов данных в соответствующие типы данных Adaptive Server Enterprise. Информация о преобразовании типов данных Adaptive Server Anywhere в типы данных Adaptive Server Enterprise приводится в таблице 2-1.

Тип данных Adaptive Server Anywhere	Тип данных ASE по умолчанию
bit	bit

tinyint	tinyint
smallint	smallint
int	int

Тип данных Adaptive Server Anywhere	Тип данных ASE по умолчанию
integer	integer
decimal [по умолчанию: p=30, s=6]	numeric(30,6)
decimal(128,128)	не поддерживается
numeric [по умолчанию: p=30 s=6]	numeric(30,6)
numeric(128,128)	не поддерживается
float	real
real	real
double	float
smallmoney	numeric(10,4)
money	numeric(19,4)
date	datetime
time	datetime
timestamp	datetime
smalldatetime	datetime
datetime	datetime
char(n)	varchar(n)
character(n)	varchar(n)
varchar(n)	varchar(n)
character varying(n)	varchar(n)
long varchar	text
text	text
binary(n)	binary(n)

long binary	image
image	image
bigint	numeric(20,0)

Классы серверов с использованием ODBC

Существуют следующие классы серверов с использованием ODBC:

- asaodbc;
- aseodbc;
- db2odbc;
- mssodbc;
- oraodbc;
- odbc.

Определение внешних ODBC-серверов

Наиболее распространенным способом определения ODBC-сервера является привязка этого сервера к источнику данных ODBC. Для этого следует создать источник данных с помощью ODBC Administrator.

Если источник данных уже определен, в разделе USING в операторе CREATE SERVER должно использоваться соответствующее имя источника данных ODBC.

Например, сконфигурируйте сервер DB2 с именем **mydb2** и с именем источника данных этого сервера **mydb2** следующим образом:

```
CREATE SERVER mydb2
CLASS 'db2odbc'
USING 'mydb2'
```

Для получения дополнительной информации о создании источников данных см. раздел "Создание источника данных ODBC" (Creating an ODBC data source) на стр. 53 в документе *"Руководство по администрированию баз данных ASA" (ASA Database Administration Guide)*.

Использование строк подключения вместо источников данных

Альтернативным вариантом, позволяющим избежать использования источников данных, является ввод строки подключения в разделе USING оператора CREATE SERVER. Для этого должны быть известны параметры подключения для используемого драйвера ODBC. Например, подключение к ASA может быть следующим:

```
CREATE SERVER testasa
CLASS 'asaodbc'
USING 'driver=adaptive server anywhere
8.0;eng=testasa;dbn=sample;links=tcpip{}'
```

Таким способом определяется подключение к серверу базы данных Adaptive Server Anywhere с именем **testasa**, базой данных **sample** и с использованием протокола TCP-IP.

Дополнительная информация

Для получения информации о конкретных классах ODBC-серверов см. следующие разделы:

- "Класс серверов asaodbc" на стр. 481;

- "Класс серверов aseodbc" на стр. 481;
- "Класс серверов db2odbc" на стр. 483;
- "Класс серверов oraodbc" на стр. 485;
- "Класс серверов mssodbc" на стр. 486;
- "Класс серверов odbc" на стр. 488.

Класс серверов asaodbc

Сервер, относящийся к классу серверов *asaodbc*, представляет собой Adaptive Server Anywhere версии 5.5 или выше. Какие-либо специальные требования по конфигурированию источника данных Adaptive Server Anywhere отсутствуют.

Для обращения к серверам Adaptive Server Anywhere, поддерживающим несколько баз данных, следует создать имя источника данных ODBC, определяющее подключение к каждой базе данных. Выполните оператор CREATE SERVER для каждого из этих имен источников данных ODBC.

Класс серверов aseodbc

Сервер, относящийся к классу серверов *aseodbc*, представляет собой Adaptive Server Enterprise, SQL Server (версии 10 и выше). Если требуется подключение к удаленному Adaptive Server с использованием класса aseodbc, то для Adaptive Server Anywhere необходимо установить драйвер ODBC Adaptive Server Enterprise и библиотеки связи Open Client. При этом, по сравнению с классом asejdbc, достигается более высокая производительность системы.

Примечания

- Open Client должен быть версии 11.1.1, EBF 7886 или выше. Установку Open Client и проверку связи с Adaptive Server следует проводить до установки ODBC и конфигурирования Adaptive Server Anywhere. Драйвер ODBC Sybase должен быть версии 11.1.1, EBF 7911 или выше.
- С помощью Configuration Manager сконфигурируйте источник данных User со следующими атрибутами:
  - На закладке General:

    Введите любое значение для имени источника данных (Data Source Name). Это значение должно использоваться в разделе USING оператора CREATE SERVER.

    Имя сервера должно соответствовать имени сервера в файле интерфейсов Sybase.
  - На закладке Advanced установите флаг Application Using Threads, а также установите флаг Enable Quoted Identifiers.
  - На закладке Connection:

    Установите такое значение в поле символов (charset), которое соответствует набору символов в Adaptive Server Anywhere.

    В поле языка (language) выберите предпочтительный язык для вывода сообщений об ошибках.

- На закладке Performance:
  Установите значение "2-Full." для Prepare Method.

  Для повышения производительности установите максимально возможное значение Fetch Array Size. Это приводит к увеличению требуемого объема памяти, поскольку этим значением определяется число строк, кэшируемых в памяти. Sybase рекомендует установить значение 100.

  Установите значение "0-Cursor." для Select Method.

  Установите максимально возможное значение Packet Size. Sybase рекомендует установить значение -1.

  Установите значение "1" для Connection Cache.

Преобразования типов данных

При выполнении оператора CREATE TABLE в Adaptive Server Anywhere осуществляется автоматическое преобразование типов данных в соответствующие типы данных Adaptive Server Enterprise. Информация о преобразовании типов данных Adaptive Server Anywhere в типы данных Adaptive Server Enterprise приводится в следующей таблице.

Тип данных Adaptive Server Anywhere	Тип данных Adaptive Server Enterprise по умолчанию
Bit	bit
Tinyint	tinyint
Smallint	smallint
Int	int
Integer	integer
decimal [по умолчанию: p=30, s=6]	numeric(30,6)
decimal(128,128)	не поддерживается
numeric [по умолчанию: p=30 s=6]	numeric(30,6)
numeric(128,128)	не поддерживается
Float	real
Real	real
Double	float
Smallmoney	numeric(10,4)

Тип данных Adaptive Server Anywhere	Тип данных Adaptive Server Enterprise по умолчанию
Money	numeric(19,4)
Date	datetime
Time	datetime
Timestamp	datetime
Smalldatetime	datetime
Datetime	datetime
char(n)	varchar(n)
Character(n)	varchar(n)
varchar(n)	varchar(n)
Character varying(n)	varchar(n)
long varchar	text
Text	text
binary(n)	binary(n)
long binary	image
Image	image
Bigint	numeric(20,0)

Класс серверов db2odbc

Сервер, относящийся к классу серверов db2odbc, представляет собой IBM DB2.

Примечания

- Sybase подтверждает возможность использования продукта IBM DB2 Connect версии 5 с пакетом исправлений WR09044. Установите и проверьте конфигурацию ODBC с помощью инструкций, прилагаемых к данному продукту. В Adaptive Server Anywhere не предъявляются какие-либо особые требования к конфигурации источников данных DB2.
- Ниже приводится пример оператора CREATE EXISTING TABLE для сервера DB2 с источником данных ODBC с именем ***mydb2***:

```
CREATE EXISTING TABLE ibmcol
AT 'mydb2..sysibm.syscolumns'
```

Преобразования типов данных

При выполнении оператора CREATE TABLE в Adaptive Server Anywhere осуществляется автоматическое преобразование типов данных в соответствующие типы данных DB2. Информация о преобразовании типов данных Adaptive Server Anywhere в типы данных DB2 приводится в следующей таблице.

Тип данных Adaptive Server Anywhere	Тип данных DB2 по умолчанию
Bit	smallint
Tinyint	smallint
Smallint	smallint
Int	int
Integer	int
Bigint	decimal(20,0)
char(1–254)	varchar(n)
char(255–4000)	varchar(n)
char(4001–32767)	long varchar
Character(1–254)	varchar(n)
Character(255–4000)	varchar(n)
Character(4001–32767)	long varchar
varchar(1–4000)	varchar(n)
varchar(4001–32767)	long varchar
Character varying(1–4000)	varchar(n)
Character varying(4001–32767)	long varchar
long varchar	long varchar
text	long varchar
binary(1–4000)	varchar (для двоичных данных)
binary(4001–32767)	long varchar (для двоичных данных)
long binary	long varchar (для двоичных данных)
image	long varchar (для двоичных данных)
decimal [по умолчанию: p=30, s=6]	decimal(30,6)
numeric [по умолчанию: p=30 s=6]	decimal(30,6)
decimal(128, 128)	НЕ ПОДДЕРЖИВАЕТСЯ
numeric(128, 128)	НЕ ПОДДЕРЖИВАЕТСЯ

Тип данных Adaptive Server Anywhere	Тип данных DB2 по умолчанию
real	real
float	float
double	float
smallmoney	decimal(10,4)
money	decimal(19,4)
date	date
time	time
smalldatetime	timestamp
datetime	timestamp
timestamp	timestamp

Класс серверов oraodbc

Сервер, относящийся к классу серверов **oraodbc**, представляет собой сервер Oracle версии 8.0 или выше.

Примечания

- Sybase подтверждает возможность использования драйвера ODBC Oracle версии 8.0.03.
- Установите и проверьте конфигурацию ODBC с помощью инструкций, прилагаемых к данному продукту.
- Ниже приводится пример оператора CREATE EXISTING TABLE для сервера Oracle с именем *myora*:

  CREATE EXISTING TABLE employees
  AT 'myora.database.owner.employees'
- Вследствие ограничений, накладываемых драйвером ODBC Oracle, выполнение оператора CREATE EXISTING TABLE для системных таблиц невозможно. Выводится сообщение, информирующее о том, что не удается найти таблицу или столбцы.

Преобразования типов данных

При выполнении оператора CREATE TABLE в Adaptive Server Anywhere осуществляется автоматическое преобразование типов данных в соответствующие типы данных Oracle. Информация о преобразовании типов данных Adaptive Server Anywhere в типы данных Oracle приводится в следующей таблице.

Тип данных Adaptive Server	Тип данных Oracle
Anywhere	
bit	number(1,0)
tinyint	number(3,0)
smallint	number(5,0)
int	number(11,0)
bigint	number(20,0)
decimal(prec, scale)	number(prec, scale)
numeric(prec, scale)	number(prec, scale)
float	float
real	real
smallmoney	numeric(13,4)
money	number(19,4)
date	date
time	date
timestamp	date
smalldatetime	date
datetime	date
char(n)	if (n > 255) long else varchar(n)
varchar(n)	if (n > 2000) long else varchar(n)
long varchar	long
binary(n)	if (n > 255) long raw else raw(n)
varbinary(n)	if (n > 255) long raw else raw(n)
long binary	long raw

Класс серверов mssodbc

Сервер, относящийся к классу серверов **mssodbc**, представляет собой Microsoft SQL Server версии 6.5 с Service Pack 4.

Примечания

- Sybase подтверждает возможность использования драйвера ODBC Microsoft SQL Server версии 3.60.0319 (этот драйвер входит в состав релиза MDAC 2.0). Установите и проверьте конфигурацию ODBC с помощью инструкций, прилагаемых к данному продукту.

- Ниже приводится пример оператора CREATE EXISTING TABLE для Microsoft SQL Server с именем *mymssql*:

```
CREATE EXISTING TABLE accounts,
AT 'mymssql.database.owner.accounts'
```

Преобразования типов данных

При выполнении оператора CREATE TABLE в Adaptive Server Anywhere осуществляется автоматическое преобразование типов данных в соответствующие типы данных Microsoft SQL Server. Информация о преобразовании типов данных Adaptive Server Anywhere в типы данных Microsoft SQL Server приводится в следующей таблице.

Тип данных Adaptive Server Anywhere	Тип данных Microsoft SQL Server по умолчанию
bit	bit
tinyint	tinyint
smallint	smallint
int	int
bigint	numeric(20,0)
decimal [по умолчанию: p=30, s=6]	decimal(prec, scale)
numeric [по умолчанию: p=30 s=6]	numeric(prec, scale)
float	if (prec) float(prec) else float
real	real
smallmoney	smallmoney
money	money
date	datetime
time	datetime
timestamp	datetime
smalldatetime	datetime
datetime	datetime
char(n)	if (length > 255) text else varchar(length)
character(n)	char(n)

Тип данных Adaptive Server Anywhere	Тип данных Microsoft SQL Server по умолчанию
varchar(n)	if (length > 255) text else varchar(length)
long varchar	text
binary(n)	if (length > 255) image else binary(length)
long binary	image
double	float

Класс серверов odbc

Источники данных ODBC, не имеющие собственный класс серверов, относятся к классу серверов **odbc**. Допускается использование любого драйвера ODBC, который соответствует ODBC версии 2.0 с уровнем совместимости 1 или выше. Sybase подтверждает возможность использования следующих источников данных ODBC:

- "Microsoft Excel" (см. стр. 488);
- "Microsoft Access" (см. стр. 489);
- "Microsoft FoxPro" (см. стр. 490);
- "Lotus Notes" (см. стр. 490).

Последние версии драйверов Microsoft ODBC могут быть получены из дистрибутива Microsoft Data Access Components (MDAC), размещенного на *www.microsoft/data/download.htm*. Указанные ниже версии драйверов Microsoft входят в состав MDAC 2.0.

В нижеследующих разделах рассматриваются способы получения доступа к этим источникам данных.

Microsoft Excel (Microsoft 3.51.171300)

При работе с Excel каждая рабочая книга Excel на логическом уровне рассматривается как база данных, содержащая несколько таблиц. Эти таблицы отображаются в листах рабочей книги. При указании имени источника данных ODBC (в диспетчере драйвера ODBC) следует ввести имя рабочей книги по умолчанию, связанное с этим источником данных. Однако когда вводится оператор CREATE TABLE, имя по умолчанию может быть переопределено путем ввода имени рабочей книги в текстовой строке местоположения. Таким способом обеспечивается возможность использования только одного имени DSN ODBC для обращения ко всем рабочим книгам Excel.

В приведенном ниже примере предполагается, что создан источник данных ODBC с именем **excel**. Создайте рабочую книгу с именем *work1.xls*, включающую лист (таблицу) с именем **mywork**:

```
CREATE TABLE mywork (a int, b char(20))
AT 'excel;d:\work1.xls;;mywork'
```

Для создания второго листа (или таблицы) выполните следующий оператор:

```
CREATE TABLE mywork2 (x float, y int)
AT 'excel;d:\work1.xls;;mywork2'
```

Существующие рабочие листы могут быть импортированы в Adaptive Server Anywhere с использованием оператора CREATE EXISTING. При этом предполагается, что в первой строке электронной таблицы содержатся имена столбцов.

```
CREATE EXISTING TABLE mywork AT'excel;d:\work1;;mywork'
```

Если Adaptive Server Anywhere сообщает, что таблица не найдена, то в таком случае может потребоваться явное указание диапазона столбцов и строк, в которых должны быть отображены импортируемые листы. Например:

```
CREATE EXISTING TABLE mywork
AT 'excel;d:\work1;;mywork$'
```

Добавление символа $ к имени листа означает, что рабочий лист должен быть выбран полностью.

Обратите внимание на то, что в строке местоположения, введенной в AT, в качестве разделителя полей используется точка с запятой, а не точка. Это необходимо, поскольку точки используются в именах файлов. Excel не поддерживает поле с именем владельца, поэтому не заполняйте это поле.

Удаления не поддерживаются. Кроме того, не могут быть выполнены некоторые обновления, поскольку драйвер Excel не поддерживает позиционированные обновления.

Microsoft Access (Microsoft 3.51.171300)

Базы данных Access хранятся в файле *.mdb*. С помощью диспетчера ODBC создайте источник данных ODBC и отобразите его в одном из этих файлов. С помощью диспетчера ODBC может быть создан новый файл *.mdb*. Этот файл базы данных становится файлом по умолчанию, если только при создании таблицы в Adaptive Server Anywhere не будет задан другой файл по умолчанию.

Предположим, что существует источник данных ODBC с именем **access**.

```
CREATE TABLE tab1 (a int, b char(10))
AT 'access...tab1'
```

или

```
CREATE TABLE tab1 (a int, b char(10))
AT 'access;d:\pcdb\data.mdb;;tab1'
```

или

```
CREATE EXISTING TABLE tab1
AT 'access;d:\pcdb\data.mdb;;tab1'
```

Access не поддерживает указание имени владельца, поэтому это имя вводить не следует.

Microsoft FoxPro (Microsoft 3.51.171300)

Таблицы FoxPro можно сохранить все вместе в одном файле базы данных FoxPro (.dbc), либо можно сохранить каждую таблицу в собственном отдельном файле .dbf. Если используются файлы .dbf, то следует вводить имя файла в строке местоположения; в противном случае будет использоваться папка, из которой был осуществлен запуск Adaptive Server Anywhere.

```
CREATE TABLE fox1 (a int, b char(20))
AT 'foxpro;d:\pcdb;;fox1'
```

В результате выполнения этого оператора создается файл с именем *d:\pcdb\fox1.dbf* (при том условии, что в диспетчере драйвера ODBC выбран параметр "free table directory").

Lotus Notes SQL 2.0 (2.04.0203)

Получить этот драйвер можно на Web-сайте Lotus. Для получения информации о том, как данные Lotus Notes отображаются в реляционных таблицах, ознакомьтесь с документацией, сопровождающей этот программный продукт. Отображение таблиц Adaptive Server Anywhere в формах Lotus Notes является достаточно простой операцией.

Ниже приводится пример выполнения установок в Adaptive Server Anywhere для получения доступа к типовому адресному файлу.

- С помощью драйвера NotesSQL создайте источник данных ODBC. Базой данных является типовой файл names (*c:\notes\data\names.nsf*). Должен быть включен параметр Map Special Characters. В этом примере именем источника данных является *my_notes_dsn*.
- Создайте сервер в Adaptive Server Anywhere:

```
CREATE SERVER names
CLASS 'odbc'
USING 'my_notes_dsn'
```

- Отобразите форму Person в таблице Adaptive Server Anywhere:

```
CREATE EXISTING TABLE Person
AT 'names...Person'
```

- Выполните запрос таблицы:

```
SELECT * FROM Person
```

Отказ от запроса на ввод пароля

Lotus Notes не поддерживает посылку имени пользователя и пароля через ODBC API. При попытке обращения к Lotus Notes с использованием защищенного паролем кода на той машине, на которой выполняется Adaptive Server Anywhere, появляется окно с запросом на ввод пароля. Избежать такого поведения можно при работе в среде многопользовательского сервера.

Для получения автоматического доступа к Lotus Notes (без запроса на ввод пароля) следует использовать идентификатор, который не защищен паролем. Удалить парольную защиту идентификатора можно путем сброса пароля (File➤Tools➤User ID➤Clear Password); это разрешается при том условии, что пароль не был потребован администратором Domino при создании идентификатора. В противном случае сброс пароля невозможен.

ЧАСТЬ 5

Добавление логики к базе данных

В этой части описывается процесс внедрения логики в базу данных с использованием хранимых процедур и триггеров SQL. Наличие в базе данных сохраненной логики автоматически делает базу данных доступной для всех приложений, обеспечивающей преимущества согласованности, эффективности и защиты. Отладчик хранимой процедуры – мощное инструментальное средство для отладки всех видов логики.

ГЛАВА 17

Использование процедур, триггеров и пакетов

Об этой главе

Процедуры и триггеры размещают процедурные операторы SQL в базе данных для использования всеми приложениями. Они позволяют расширить защиту, эффективность и стандартизацию баз данных. Определяемые пользователем функции – это один вид процедур, возвращающих значение среде вызова для использования в запросах и других операторах SQL. Пакеты представляют собой наборы операторов SQL, направляемые на сервер базы данных в виде группы. Многие средства, доступные в процедурах и триггерах (например, управляющие операторы), также доступны в пакетах.

☞ Для многоцелевого назначения серверный JDBC предоставляет более гибкий метод интегрирования логики в базу данных, чем хранимые процедуры SQL. Для получения информации о JDBC см. раздел "Доступ к данным через JDBC" (Data Access Using JDBC) на стр. 129 в документе *"Руководство по программированию ASA" (ASA Programming Guide)*.

Содержание

Обзор процедур и триггеров

Процедуры и триггеры размещают процедурные операторы SQL в базе данных для использования их любыми приложениями. Они могут включать в себя управляющие операторы, разрешающие повторение (оператор LOOP) и условное выполнение операторов SQL (оператор IF и оператор CASE).

Процедуры вызываются оператором CALL и используют параметры для получения и возвращения значений среде вызова. Процедуры могут возвращать результирующие наборы вызывающему оператору, вызывать другие процедуры или запускать триггеры. Например, определяемая пользователем функция представляет собой тип хранимой процедуры, возвращающей одиночное значение в среду вызова. Определяемые пользователем функции не изменяют переданные им параметры, а, скорее, расширяют область функций, доступных для запросов и других операторов SQL.

Триггеры связаны с заданными таблицами базы данных. Они запускаются автоматически всякий раз при вставке, удалении или обновлении строк связанной с ними таблицы. Триггеры могут вызывать процедуры и запускать другие триггеры, но они не имеют своих параметров и не могут быть вызваны оператором CALL.

Отладчик процедур

При помощи отладчика Sybase можно выполнять отладку хранимых процедур и триггеров. Для получения дополнительной информации см. раздел "Отладка логики базы данных" на стр. 557.

Преимущества процедур и триггеров

Определения процедур и триггеров появляются в базе данных независимо от любого приложения базы данных. Такое разделение дает множество преимуществ.

Стандартизация

Процедуры и триггеры стандартизируют действия, выполняемые более чем одной прикладной программой. После однократного кодирования действия и его размещения в базе данных для будущего использования приложениям остается только вызвать процедуру или запустить триггер для многократного получения требуемых результатов. Поскольку изменения происходят только в одном месте, все приложения, использующие это действие, автоматически приобретают новые функциональные возможности при изменении реализации этого действия.

Эффективность

Процедуры и триггеры, используемые в сетевой среде сервера базы данных, могут обращаться к данным в базе данных, не требуя сетевого взаимодействия. Это означает, что они выполняются быстрее и с меньшим воздействием на эффективность сети, чем при их реализации в приложении на одной из клиентских машин.

При создании процедуры или триггера происходит автоматическая проверка правильности синтаксиса, после чего размещаются в системных таблицах. Когда приложение в первый раз вызывает процедуру или запускает триггер, они компилируются из системных таблиц в виртуальную память сервера и выполняются оттуда. Поскольку одна копия процедуры или триггера остается в памяти после первого выполнения, то немедленно происходит повторное выполнение этой же процедуры или триггера. Таким же образом несколько приложений могут одновременно использовать процедуру или триггер. Кроме того, одно приложение может использовать их рекурсивно.

Процедуры, содержащие простые запросы и большое количество аргументов, менее эффективны. Использование процедур более эффективно в случае сложных запросов.

Защита

Процедуры и триггеры обеспечивают защиту, ограничивая доступ пользователей к данным в таблицах таким образом, что их непосредственное изучение или изменение становится невозможным.

В частности, триггеры выполняются согласно полномочиям для таблиц владельца связанной с триггером таблицы, однако их может запустить любой пользователь с полномочиями на вставку, удаление или обновление строк таблицы. Аналогично, процедуры (включая определяемые пользователем функции) выполняются согласно полномочиям владельца процедуры, но их может вызвать любой пользователь, обладающий соответствующими полномочиями. Это означает, что процедуры и триггеры могут иметь (и обычно имеют) иные полномочия, нежели код вызвавшего их пользователя.

Введение в процедуры

Для использования процедур необходимо рассмотреть следующие вопросы:

- создание процедур;
- вызов процедур из приложения базы данных;
- сброс или удаление процедур;
- управление полномочиями на использование процедур.

В данном разделе рассматриваются как вышеуказанные аспекты использования процедур, так и некоторые другие области их применения.

Создание процедур

Adaptive Server Anywhere предоставляет различные инструментальные средства для создания новых процедур.

В Sybase Central для получения необходимой информации используется мастер, после чего создание кода завершается в универсальном редакторе кода. Sybase Central также предоставляет шаблоны процедур (в каталоге Procedures & Functions), которые можно открывать и изменять.

В качестве альтернативного решения для создания процедур можно использовать оператором CREATE PROCEDURE. Однако необходимо иметь полномочия RESOURCE. Место ввода оператора зависит от применяемого инструментального средства.

❖ **Создание новой процедуры (Sybase Central)**

1. Выполните подключение к базе данных с использованием полномочий администратора БД или RESOURCE.
2. Откройте папку Procedures & Functions базы данных.
3. В правой области окна дважды щелкните Add Procedure (Wizard).
   - Выполняйте указания мастера.
   - После открытия Code Editor завершите редактирование кода процедуры.
   - Для выполнения и сохранения процедуры выберите File➤Save Procedure.
   - Для сохранения процедуры без выполнения выберите File➤Save File As.

В папке Procedures & Functions появляется новая процедура.

❖ **Создание новой удаленной процедуры (Sybase Central)**

1 Выполните подключение к базе данных с использованием полномочий администратора БД.

2 Откройте папку Procedures & Functions базы данных.

3 В правой области окна дважды щелкните Add Remote Procedure (Wizard).

4 После открытия Code Editor завершите редактирование кода процедуры.

5 Выполняйте указания мастера.

Совет

Создать удаленную процедуру также можно щелчком правой кнопки мыши на удаленном сервере в папке Remote Servers и последующим выбором Add Remote Procedure из всплывающего меню.

❖ **Создание процедуры с использованием другого инструментального средства**

- Следуйте инструкциям к выбранному инструментальному средству. Перед вводом оператора CREATE PROCEDURE, возможно, потребуется изменить разделитель команд с точки с запятой на другой.

☞ Для получения дополнительной информации о подключении см. раздел "Подключение к базе данных" (Connecting to a Database) на стр. 37 в документе *"Руководство по администрированию баз данных ASA" (ASA Database Administration Guide)*.

Пример

В следующем примере создается процедура *new_dept*, выполняющая INSERT (вставку) в таблицу отделов демонстрационной базы данных и создающая новый отдел.

```
CREATE PROCEDURE new_dept
  IN id INT,
  IN name CHAR(35),
  IN head_id INT )
BEGIN
INSERT
INTO DBA.department ( dept_id,
  dept_name,
  dept_head_id )
VALUES ( id, name, head_id );
END
```

Тело процедуры является составным оператором. Составной оператор начинается оператором BEGIN и заканчивается оператором END. В случае с *new_dept* составным оператором является только INSERT, находящийся между операторами BEGIN и END.

Параметры для процедур отмечены как IN, OUT или INOUT. Все параметры для процедуры *new_dept* являются параметрами IN, поскольку они не изменяются процедурой.

☞ Для получения дополнительной информации см. раздел "Оператор CREATE PROCEDURE" (CREATE PROCEDURE statement) на стр. 284 в документе *"Справочник по SQL для ASA" (ASA SQL Reference Manual),* раздел "Оператор ALTER PROCEDURE" (ALTER PROCEDURE statement) на стр. 203 в документе *"Справочник по SQL для ASA" (ASA SQL Reference Manual)* и раздел "Использование составных операторов" на стр. 521 настоящего документа.

Изменение процедур

Вносить изменения в существующую процедуру можно в Sybase Central или Interactive SQL. Необходимо обладать полномочиями администратора БД или являться владельцем процедуры.

В Sybase Central нельзя непосредственно переименовывать существующие процедуры. Вместо этого нужно создать новую процедуру с новым именем, скопировать в нее предыдущий код, после чего удалить старую процедуру.

В качестве альтернативного решения, для внесения изменений в существующую процедуру можно воспользоваться оператором ALTER PROCEDURE. В этот оператор необходимо включить целиком новую процедуру (с использованием того же синтаксиса, как в операторе CREATE PROCEDURE, создавшем эту процедуру). Также необходимо переназначить полномочия пользователей на эту процедуру.

☞ Для получения дополнительной информации об изменении свойств объектов базы данных см. раздел "Установка свойств объектов базы данных" на стр. 34.

☞ Для получения дополнительной информации о предоставлении или возобновлении полномочий для процедур см. раздел "Предоставление полномочий на процедуры" (Granting permissions on procedures) на стр. 359 в документе *"Руководство по администрированию баз данных ASA" (ASA Database Administration Guide)* и раздел "Возобновление полномочий пользователей" (Revoking user permissions) на стр. 362 в документе *“Руководство по администрированию баз данных ASA” (ASA Database Administration Guide).*

❖ Изменение кода процедуры (Sybase Central)

1. Откройте папку Procedures & Functions.
2. Щелкните правой кнопкой мыши на требуемой процедуре.
3. Во всплывающем меню выполните одно из следующих действий:
   - Выберите Open as Watcom-SQL для редактирования кода в диалекте Watcom-SQL.
   - Выберите Open as Transact-SQL для редактирования кода в диалекте Transact-SQL.
4. Отредактируйте код процедуры в Code Editor.
5. Для выполнения кода в базе данных выберите File➤Save/Execute in Database.

❖ **Изменение кода процедуры (SQL)**

1 Выполните подключение к базе данных.

2 Выполните оператор ALTER PROCEDURE. Включите всю новую процедуру в этот оператор.

☞ Для получения дополнительной информации см. раздел "Оператор ALTER PROCEDURE" (ALTER PROCEDURE statement) на стр. 203 в документе *"Справочник по SQL для ASA" (ASA SQL Reference Manual),* раздел "Оператор CREATE PROCEDURE" (CREATE PROCEDURE statement) на стр. 284 в документе *"Справочник по SQL для ASA" (ASA SQL Reference Manual)* и раздел "Создание процедур" на стр. 499 настоящего документа.

Вызов процедур

Процедуры вызываются операторами CALL. Процедуры можно вызвать прикладной программой или другими процедурами и триггерами.

☞ Для получения дополнительной информации см. раздел "Оператор CALL" (CALL statement) на стр. 238 в документе *"Справочник по SQL для ASA" (ASA SQL Reference Manual)*.

Следующий оператор вызывает процедуру *new_dept* для вставки отдела Eastern Sales:

```
CALL new_dept( 210, 'Eastern Sales', 902 );
```

После этого вызова можно проверить таблицу отделов на предмет добавления нового отдела.

Все пользователи, обладающие полномочиями EXECUTE для этой процедуры, могут вызвать процедуру *new_dept,* даже не имея полномочий на таблицу *department* (таблицу отделов).

☞ Для получения дополнительной информации о полномочиях EXECUTE см. раздел "Оператор EXECUTE [ESQL]" (EXECUTE statement [ESQL]) на стр. 384 в документе *"Справочник по SQL для ASA" (ASA SQL Reference Manual)*.

Копирование процедур в Sybase Central

В Sybase Central можно копировать процедуры между базами данных. Для этого выберите процедуры в правой области окна Sybase Central и перетащите их в папку Procedures & Functions другой подключенной базы данных. После этого создается новая процедура, в которую копируется код первоначальной процедуры.

Следует отметить, что в новую процедуру копируется только код процедуры. Другие свойства процедуры (полномочия и т. д.) не копируются. В одну базу данных можно скопировать любую процедуру, если последней присвоено новое имя.

Удаление процедур

После создания процедуры она остается в базе данных до тех пор, пока не будет явно удалена. Только владелец процедуры или пользователь с полномочиями администратора БД может удалить процедуру из базы данных.

❖ **Удаление процедуры (Sybase Central)**

1. Выполните подключение к базе данных с использованием полномочий администратора БД или в качестве владельца процедуры.
2. Откройте папку Procedures & Functions.
3. Щелкните правой кнопкой мыши на требуемой процедуре и выберите Delete из всплывающего меню.

❖ **Удаление процедуры (SQL)**

1. Выполните подключение к базе данных с использованием полномочий администратора БД или в качестве владельца процедуры.
2. Выполните оператор DROP PROCEDURE.

Пример

Следующий оператор удаляет процедуру *new_dept* из базы данных:

```
DROP PROCEDURE new_dept
```

☞ Для получения дополнительной информации см. раздел "Оператор DROP" (DROP statement) на стр. 366 в документе *"Справочник по SQL для ASA" (ASA SQL Reference Manual)*.

Возвращение результатов процедуры в виде параметров

Процедуры возвращают результаты в среду вызова одним из следующих способов:

- отдельные значения возвращаются как параметры OUT или INOUT;
- могут быть возвращены результирующие наборы;
- отдельный результат может быть возвращен при помощи оператора RETURN.

В этом разделе описывается возвращение результатов из процедур в виде параметров.

Следующая процедура в демонстрационной базе данных возвращает среднюю заработную плату служащих в виде параметра OUT.

```
CREATE PROCEDURE AverageSalary( OUT avgsal
    NUMERIC (20,3) )
BEGIN
    SELECT AVG( salary )
    INTO avgsal
    FROM employee;

END
```

❖ **Выполнение данной процедуры и отображение ее вывода (SQL)**

1 Выполните подключение к демонстрационной базе данных в Interactive SQL с использованием кода пользователя **DBA** и пароля **SQL**. Для получения дополнительной информации о подключении см. раздел "Подключение к базе данных" (Connecting to a Database) на стр. 37 в документе *"Руководство по администрированию баз данных ASA" (ASA Database Administration Guide)*.

2 В окне SQL Statements введите вышеуказанный код процедуры.

3 Создайте переменную для хранения вывода процедуры. В этом случае переменная вывода является числом с тремя десятичными разрядами. Поэтому следует создать переменную следующим образом:

```
CREATE VARIABLE Average NUMERIC(20,3)
```

4 Вызовите процедуру, используя созданную переменную для сохранения результата:

```
CALL AverageSalary (Average)
```

Если процедура создана и выполнена правильно, то в окне Interactive SQL Messages нет указаний на наличие ошибок.

5 Выполните оператор SELECT Average для просмотра значения переменной.

Посмотрите на значение переменной вывода Average. На закладке Results в окне Results для этой переменной показано значение 49988.623 – средняя заработная плата служащего.

Возвращение результатов процедуры в виде результирующих наборов

Помимо возвращения результатов в среду вызова в виде отдельных параметров, процедуры могут возвращать информацию в виде результирующих наборов. Обычно результирующий набор является результатом запроса. Следующая процедура возвращает результирующий набор, содержащий заработную плату каждого служащего данного отдела:

```
CREATE PROCEDURE SalaryList ( IN department_id INT)
RESULT ( "Employee ID" INT, Salary NUMERIC(20,3) )
BEGIN
    SELECT emp_id, salary
    FROM employee
    WHERE employee.dept_id = department_id;
END
```

Если эту процедуру вызывает Interactive SQL, то имена в разделе RESULT сочетаются с результатами запроса и используются как заголовки столбцов в отображаемых результатах.

Для проверки этой процедуры из Interactive SQL ее можно вызвать (с помощью оператора CALL) с указанием одного из отделов компании. В Interactive SQL результаты появляются на закладке Results в окне Results.

Пример

Для составления списка заработных плат служащих научно-исследовательского отдела (код отдела 100) введите следующее:

```
CALL SalaryList (100)
```

Employee ID	Salary
102	45700
105	62000
160	57490
243	72995
...	...

Если этот параметр активизирован на закладке Results диалога Options, Interactive SQL может возвращать только множественные результирующие наборы. Каждый результирующий набор появляется на отдельной закладке в окне Results.

☞ Для получения дополнительной информации см. раздел "Возвращение множественных результирующих наборов из процедур" на стр. 530.

Введение в определяемые пользователем функции

Определяемые пользователем функции представляют собой класс процедур, возвращающих в среду вызова одиночное значение. Adaptive Server Anywhere обрабатывает все определяемые пользователем функции как идемпотент: функция возвращает согласованный результат для тех же параметров и не имеет побочных эффектов. Другими словами, сервер предполагает, что два последовательных вызова той же функции с теми же параметрами возвратят одинаковый результат и не будут иметь нежелательных побочных эффектов в семантике запроса.

В данном разделе рассматривается создание, использование и удаление определяемых пользователем функций.

Создание определяемых пользователем функций

Для создания определяемых пользователем функций используется оператор CREATE FUNCTION. Однако необходимо иметь полномочия RESOURCE.

В следующем примере создается функция, объединяющая две строки с пространством для формирования полного имени из имени и фамилии.

```
CREATE FUNCTION fullname (firstname CHAR(30),
    lastname CHAR(30))
RETURNS CHAR(61)
BEGIN
    DECLARE name CHAR(61);
    SET name = firstname || ' ' || lastname;
    RETURN ( name );
END
```

❖ **Создание данного примера с использованием Interactive SQL**

1 Выполните подключение к демонстрационной базе данных в Interactive SQL с использованием кода пользователя **DBA** и пароля **SQL**. Для получения дополнительной информации о подключении см. раздел "Подключение к базе данных" (Connecting to a Database) на стр. 37 в документе *"Руководство по администрированию баз данных ASA" (ASA Database Administration Guide)*.

2 Введите вышеуказанный код функции в окне SQL Statements.

Примечание

При использовании иного инструментального средства, нежели Interactive SQL или Sybase Central, возможно, потребуется изменить разделитель команд с точки с запятой на другой.

☞ Для получения дополнительной информации см. раздел "Оператор CREATE FUNCTION" (CREATE FUNCTION statement) на стр. 277 в документе *"Справочник по SQL для ASA" (ASA SQL Reference Manual)*.

Синтаксис CREATE FUNCTION немного отличается от синтаксиса оператора CREATE PROCEDURE. Имеются следующие различия:

- Не требуются ключевые слова IN, OUT или INOUT, поскольку все параметры являются параметрами IN.
- Для указания типа возвращаемых данных необходим раздел RETURNS.
- Для указания возвращаемого значения необходим оператор RETURN.

Вызов определяемых пользователем функций

При наличии соответствующих полномочий определяемая пользователем функция может использоваться в любом месте, в котором использовалась бы встроенная неагрегатная функция.

Следующий оператор в Interactive SQL возвращает полное имя из двух столбцов, содержащих имя и фамилию:

```
SELECT fullname (emp_fname, emp_lname)
FROM employee;
```

Fullname (emp_fname, emp_lname)
Fran Whitney
Matthew Cobb
Philip Chin
...

Следующий оператор в Interactive SQL возвращает полное имя из предоставленных имени и фамилии:

```
SELECT fullname ('Jane', 'Smith');
```

Fullname ('Jane','Smith')
Jane Smith

Любой пользователь, обладающий полномочиями EXECUTE для данной функции, может использовать функцию *fullname*.

Пример

Следующая определяемая пользователем функция иллюстрирует локальные объявления переменных.

Таблица *customer* (таблица клиентов) включает в себя нескольких клиентов из Канады, обнаружившихся среди клиентов из США, но столбец *country* (страна) отсутствует. Определяемая пользователем функция *nationality* (гражданство) использует тот факт, что американский почтовый индекс является числовым значением, тогда как канадский почтовый код начинается с буквенного символа, чтобы отличить канадских и американских клиентов.

```
CREATE FUNCTION nationality( cust_id INT )
RETURNS CHAR( 20 )
BEGIN
    DECLARE natl CHAR(20);
    IF cust_id IN ( SELECT id FROM customer
                WHERE LEFT(zip,1) > '9') THEN
            SET natl = 'CDN';
    ELSE
        SET natl = 'USA';
    END IF;
    RETURN ( natl );
END
```

Этот пример объявляет переменную *natl* для хранения строки гражданства, использует оператор SET для задания значения переменной и возвращает значение строки *natl* в среду вызова.

Следующий запрос перечисляет всех клиентов из Канады в таблице *customer:*

```
SELECT * FROM customer
WHERE nationality(id) = 'CDN'
```

Объявления курсоров и исключений рассматриваются в разделах ниже.

Тот же запрос, вновь заявленный без функции, был бы выполнен быстрее, особенно при наличии почтового индекса. Например,

```
Select *
FROM customer
WHERE zip > '99999'
```

Примечания

Эта функция приведена здесь для наглядности; она не выполняется должным образом при использовании в SELECT с большим количеством строк. Если, например, в таблице, содержащей 100 000 строк, использовался запрос SELECT, и 10 000 строк возвращены, то функция будет вызвана 10 000 раз. Если эта функция используется в разделе WHERE того же запроса, то она будет вызвана 100 000 раз.

Удаление определяемых пользователем функций

После создания определяемой пользователем функции она остается в базе данных до тех пор, пока не будет явно удалена. Только владелец функции или пользователь с полномочиями администратора БД может удалить функцию из базы данных.

Следующий оператор удаляет функцию *fullname* из базы данных:

```
DROP FUNCTION fullname
```

Полномочия на выполнение определяемых пользователем функций

Монопольное использование определяемой пользователем функции принадлежит создавшему ее пользователю, и этот пользователь может выполнять ее без полномочий. Владелец определяемой пользователем функции может выдать полномочия другим пользователям с помощью команды GRANT EXECUTE.

Например, создатель функции *fullname* может выдать другому пользователю *(another_user)* полномочия на использование *fullname* с помощью оператора:

```
GRANT EXECUTE ON fullname TO another_user
```

Следующий оператор возобновляет полномочия на использование функции:

```
REVOKE EXECUTE ON fullname FROM another_user
```

☞ Для получения дополнительной информации об управлении полномочиями пользователей при работе с функциями см. раздел "Предоставление полномочий на процедуры" (Granting permissions on procedures) на стр. 359 в документе *"Руководство по администрированию баз данных ASA" (ASA Database Administration Guide)*.

Введение в триггеры

Триггеры используются всякий раз при недостаточных ссылочной целостности и других декларативных ограничениях.

☞ Для получения дополнительной информации о ссылочной целостности см. раздел "Обеспечение целостности данных" (Ensuring Data Integrity) на стр. 65 и раздел "Оператор CREATE TABLE" (CREATE TABLE statement) на стр. 321 в документе *"Справочник по SQL для ASA" (ASA SQL Reference Manual)*.

Не исключено, что пользователю потребуется задействовать более сложную структуру ссылочной целостности, включающую более тщательную проверку, либо принудительную проверку новых данных без соблюдения ограничений в уже существующих данных. Другое назначение триггеров заключается в регистрации действий с таблицами базы данных, независимо от приложений, использующих эту базу данных.

Полномочия на выполнение триггера

Триггеры выполняются с использованием полномочий владельца связанной таблицы, а не кода пользователя, действия которого запускают триггер. Триггер может изменить строки в таблице, которые пользователь не может изменить непосредственно.

Триггеры могут быть определены по одному или более инициируемых триггером действий:

Действие	Описание
INSERT	Вызывает триггер всякий раз при вставке новой строки в связанную с триггером таблицу.
DELETE	Вызывает триггер всякий раз при удалении строки связанной с триггером таблицы.
UPDATE	Вызывает триггер всякий раз при обновлении строки связанной с триггером таблицы.
UPDATE OF список-столбцов	Вызывает триггер всякий раз при обновлении строки связанной с триггером таблицы таким образом, что столбец в списке-столбцов изменяется.

Триггер может являться триггером **уровня строки** или триггером **уровня оператора.** Триггеры уровня строки выполняются до (BEFORE) или после (AFTER) каждой строки, измененной вставкой вызова, обновлением или удалением изменения операции. Триггеры уровня оператора выполняются после того, как операция выполнена целиком.

Гибкость времени выполнения триггера особенно полезна для триггеров, полагающихся на действия, связанные со ссылочной целостностью, такие как каскадные обновления или удаления, осуществляемые (или не осуществляемые) во время их выполнения.

При возникновении ошибки во время выполнения триггера операция, запустившая триггер, прерывается. INSERT, UPDATE и DELETE - атомарные операции (см. раздел "Атомарные составные операторы" на стр. 521). При их прерывании все результаты действия оператора (включая действия триггеров и любые процедуры, вызванные триггерами) возвращаются назад к состоянию до запуска триггера.

Создание триггеров

Триггеры создаются с использованием Sybase Central или Interactive SQL. В Sybase Central код можно составить в Code Editor. В Interactive SQL можно использовать оператор CREATE TRIGGER. Для создания триггера при помощи этих инструментальных средств необходимо иметь полномочия администратора БД или RESOURCE, а также полномочия ALTER для связанной с триггером таблицы.

Тело триггера содержит составной оператор - набор разделенных точкой с запятой операторов SQL, находящихся между операторами BEGIN и END.

В пределах триггера нельзя использовать операторы COMMIT, ROLLBACK и ROLLBACK TO SAVEPOINT.

☞ Для получения дополнительной информации см. список перекрестных ссылок в конце этого раздела.

❖ **Создание нового триггера для данной таблицы (Sybase Central)**

1. Откройте папку Triggers требуемой таблицы.
2. В правой области окна дважды щелкните Add Trigger.
3. Выполняйте указания мастера.
4. Когда мастер закончит инструкции и откроет Code Editor, завершите редактирование кода триггера.
5. Для выполнения кода в базе данных выберите File➤Save/Execute in Database.

❖ **Создание нового триггера для данной таблицы (SQL)**

1. Выполните подключение к базе данных.
2. Выполните оператор CREATE TRIGGER.

Пример 1: триггер INSERT уровня строки

Следующий триггер - пример триггера INSERT уровня строки. Этот триггер проверяет правильность даты рождения, введенной для нового служащего:

```
CREATE TRIGGER check_birth_date
    AFTER INSERT ON Employee REFERENCING NEW AS new_employee
FOR EACH ROW
```

```
BEGIN
    DECLARE err_user_error EXCEPTION
    FOR SQLSTATE '99999';
    IF new_employee.birth_date > 'June 6, 2001' THEN
        SIGNAL err_user_error;
    END IF;
END
```

Этот триггер запускается после вставки любой строки в таблицу *employee* (таблица служащих). Он обнаруживает и запрещает любые новые строки, соответствующие датам рождения позже 6 июня 2001 года.

Фраза REFERENCING NEW AS *new_employee* позволяет операторам в коде триггера ссылаться на данные в новой строке, используя псевдоним *new_employee*.

Передача сигналов об ошибке отменяет оператор с триггером, а также любые предыдущие результаты действия триггера.

Для оператора INSERT, добавляющего несколько строк в таблицу служащих, триггер *check_birth_date* запускается один раз для каждой новой строки. Если для какой-то строки триггер дает сбой, все результаты действия оператора INSERT возвращаются для повторного выполнения.

Можно указать, что триггер запускается перед вставкой строки, а не после, путем изменения первой строки примера следующим образом:

```
CREATE TRIGGER mytrigger BEFORE INSERT ON Employee
```

Раздел REFERENCING NEW ссылается на вставленные значения строки; он не зависит от выбора времени (BEFORE или AFTER) триггера.

Не исключено, что в некоторых случаях проще принудительно задействовать ограничения, используя ссылочную целостность объявления или ограничения CHECK, чем триггеры. Например, реализация вышеупомянутого примера с ограничением CHECK для столбца оказывается более эффективной и сжатой:

```
CHECK (@col <= 'June 6, 2001')
```

Пример 2: триггер DELETE уровня строки

Следующий оператор CREATE TRIGGER определяет триггер DELETE уровня строки:

```
CREATE TRIGGER mytrigger BEFORE DELETE ON employee REFERENCING
OLD AS oldtable
FOR EACH ROW
BEGIN
   ...
END
```

Раздел REFERENCING OLD позволяет коду триггера удаления сослаться на значения в удаляемой строке, используя псевдоним *oldtable*.

Можно определить, что триггер запускается после удаления троки, а не после, путем изменения первой строки примера следующим образом:

```
CREATE TRIGGER check_birth_date AFTER DELETE ON employee
```

Раздел REFERENCING OLD не зависит от выбора времени (BEFORE или AFTER) триггера.

Пример 3: триггер UPDATE уровня оператора

Следующий оператор CREATE TRIGGER соответствует триггерам UPDATE уровня оператора:

```
CREATE TRIGGER mytrigger AFTER UPDATE ON employee
REFERENCING NEW AS table_after_update
        OLD AS table_before_update
FOR EACH STATEMENT
BEGIN
...
END
```

Разделы REFERENCING NEW и REFERENCING OLD позволяют коду триггера UPDATE ссылаться как на старые, так и на новые значения обновляемых строк. Псевдоним таблицы *table_after_update* ссылается на столбцы в новой строке, а псевдоним таблицы *table_before_update* ссылается на столбцы в старой строке.

Разделы REFERENCING NEW и REFERENCING OLD имеют несколько разное значение для триггеров уровня строки и уровня оператора. Для триггеров уровня оператора псевдонимы REFERENCING NEW или OLD являются псевдонимами таблицы, в то время как в триггерах уровня строки они ссылаются на изменяемую строку.

☞ Для получения дополнительной информации см. раздел "Оператор CREATE TRIGGER" (CREATE TRIGGER statement) на стр. 333 в документе *"Справочник по SQL для ASA" (ASA SQL Reference Manual)* и раздел "Использование составных операторов" на стр. 521.

Выполнение триггеров

Триггеры выполняются автоматически всякий раз при выполнении в таблице, именованной в триггере, операций INSERT, UPDATE или DELETE. Для каждой связанной строки триггер уровня строки запускается только один раз, в то время как триггер уровня оператора запускается один раз для всего оператора.

Когда операции INSERT, UPDATE или DELETE запускают триггер, действия выполняются в следующем порядке:

1. Запускаются триггеры BEFORE.
2. Выполняются ссылочные действия.
3. Выполняется сама операция.
4. Запускаются триггеры AFTER.

Если на каком-то из шагов происходит ошибка, не обработанная в пределах процедуры или триггера, то предыдущие шаги (как и последующие) не выполняются, и операция, запустившая триггер, прерывается.

Изменение триггеров

Существующий триггер можно изменить, используя Sybase Central или Interactive SQL. Для этого пользователь должен являться владельцем таблицы, для которой определен триггер, или администратором БД, либо обладать полномочиями полномочия ALTER на таблицу и полномочиями RESOURCE.

В Sybase Central нельзя непосредственно переименовывать существующие триггеры. Вместо этого необходимо создать новый триггер с новым именем, скопировать в него предыдущий код, после чего удалить старый триггер.

В качестве альтернативного решения, для изменения существующего триггера можно воспользоваться оператором ALTER TRIGGER. Новый триггер должен быть целиком включен в этот оператор (с тем же синтаксисом, что и в операторе CREATE TRIGGER, создавшем этот триггер).

☞ Для получения дополнительной информации об изменении свойств объектов базы данных см. раздел "Установка свойств для объектов базы данных" на стр. 34.

❖ Изменение кода триггера (Sybase Central)

1. Откройте папку Triggers требуемой таблицы.
2. Щелкните правой кнопкой мыши на требуемом триггере.
3. Во всплывающем меню выполните одно из следующих действий:
   - ♦ Для редактирования кода в диалекте Watcom SQL выберите Open as Watcom SQL.
   - ♦ Для редактирования кода в диалекте Transact SQL выберите Open as Transact SQL.
4. Отредактируйте код триггера в Code Editor.
5. Для выполнения кода в базе данных выберите File➤Save/Execute in Database.

❖ Изменение кода триггера (SQL)

1. Выполните подключение к базе данных.
2. Выполните оператор ALTER TRIGGER. Включите весь новый триггер в этот оператор.

☞ Для получения дополнительной информации см. раздел "Оператор ALTER TRIGGER" (ALTER TRIGGER statement) на стр. 225 в документе *"Справочник по SQL для ASA" (ASA SQL Reference Manual)*.

Удаление триггеров

После создания триггера он остается в базе данных до тех пор, пока не будет явно удален. Для удаления триггера необходимо иметь полномочия ALTER на таблицы, связанные с этим триггером.

❖ **Удаление триггера (Sybase Central)**

1. Откройте папку Triggers требуемой таблицы.
2. Щелкните правой кнопкой мыши на требуемом триггере и выберите Delete из всплывающего меню.

❖ **Удаление триггера (SQL)**

1. Выполните подключение к базе данных.
2. Выполните оператор DROP TRIGGER.

Пример

Следующий оператор удаляет триггер *mytrigger* из базы данных:

```
DROP TRIGGER mytrigger
```

☞ Для получения дополнительной информации см. раздел "Оператор DROP" (DROP statement) на стр. 366 в документе *"Справочник по SQL для ASA" (ASA SQL Reference Manual)*.

Полномочия на выполнение триггера

Предоставление полномочий на выполнение триггера невозможно. Пользователи не могут выполнять триггеры, поскольку в Adaptive Server Anywhere они запускаются в ответ на совершаемые в базе данных действия. Однако при выполнении любого триггера этот триггер имеет связанные с ним полномочия, определяющие его права на выполнение определенных действий.

Триггеры выполняются с использованием полномочий владельца таблицы, в которой они определены, а не полномочий пользователя, запустившего или создавшего триггер.

Если триггер ссылается на таблицу, он использует членство создателя таблицы в группе с целью определения местонахождения таблиц без явного указания имени владельца. Например, если триггер на таблицу A пользователя 1*(user_1. Table_A)* ссылается на таблицу B *(Table_B)* и не указывает владельца таблицы B, тогда таблица B либо должна быть создана пользователем 1 *(user_1),* либо пользователь 1 должен быть членом той же группы (прямо или косвенно), что и владелец таблицы B. Если ни одно из этих условий не выполнено, то при запуске триггера появляется сообщение table not found (таблица не найдена).

Кроме того, пользователь 1 должен иметь полномочия на выполнение операций, указанных в триггере.

Введение в пакеты

Простой пакет состоит из набора операторов SQL, разделенных точками с запятой или отдельной строкой с единственным словом "**go**". Использование слова "**go**" рекомендуется. Например, следующий набор операторов формирует пакет, создающий отдел Eastern Sales и переводящий всех торговых представителей из штата Массачусетс в этот отдел.

```
INSERT
INTO department ( dept_id, dept_name )
VALUES ( 220, 'Eastern Sales' )
go
UPDATE employee
SET dept_id = 220
WHERE dept_id = 200
AND state = 'MA'
go
COMMIT
go
```

Этот набор операторов можно включить в любое приложение и выполнять их вместе.

Interactive SQL и пакеты

В Interactive SQL выполняется разбор списка разделенных точкой с запятой операторов (ср. приведенный выше) перед их отправкой на сервер. В этом случае **Interactive** SQL посылает каждый оператор на сервер индивидуально, а не пакетом. Если в используемом приложении отсутствует код для разбора, то операторы будут посылаться и обрабатываться как пакет. Если в начале и конце набора операторов помещены операторы BEGIN и END, то Interactive SQL обрабатывает их как пакет.

Многие операторы, используемые в процедурах и триггерах, также могут использоваться в пакетах. В пакетах можно использовать управляющие операторы (CASE, IF, LOOP и т.д.), включая составные операторы (BEGIN и END). Составные операторы могут включать в себя объявления переменных, исключений, временных таблиц или курсоров.

Следующий пакет создает таблицу только в том случае, если таблицы с таким именем не существует:

```
IF NOT EXISTS (
    SELECT * FROM SYSTABLE
    WHERE table_name = 't1' ) THEN
        CREATE TABLE t1 (
        firstcol INT PRIMARY KEY,
        secondcol CHAR( 30 )
        )
        go
ELSE
```

```
        MESSAGE 'Table t1 already exists' TO CLIENT;
    END IF
```

Если дважды запустить этот пакет в Interactive SQL, то при первом запуске он создаст таблицу, а при втором запуске отобразит сообщение о том, что такая таблица уже существует, в окне сообщений Interactive SQL.

Управляющие операторы

В теле процедуры, триггера или пакета содержатся операторы для управления логическим потоком и принятием решений. Доступны следующие управляющие операторы:

Управляющие операторы	Синтаксис

Составные операторы
☞ Для получения дополнительной информации см. раздел "Оператор BEGIN" (BEGIN statement) на стр. 232 в документе *"Справочник по SQL для ASA" (ASA SQL Reference Manual)*.
	`BEGIN [ ATOMIC ]`
	`    Список-операторов`
	`END`

Условное выполнение: IF
☞ Для получения дополнительной информации см. раздел "Оператор IF" (IF statement) на стр. 422 в документе *"Справочник по SQL для ASA" (ASA SQL Reference Manual)*.
	`IF условие THEN`
	`    Список-операторов`
	`ELSEIF условие THEN`
	`    Список-операторов`
	`ELSE`
	`    Список-операторов`
	`END IF`

Условное выполнение: CASE
☞ Для получения дополнительной информации см. раздел "Оператор CASE" (CASE statement) на стр. 240 в документе *"Справочник по SQL для ASA" (ASA SQL Reference Manual)*.
	`CASE выражение`
	`WHEN значение THEN`
	`    Список-операторов`
	`WHEN значение THEN`
	`    Список-операторов`
	`ELSE`
	`    Список-операторов`
	`END CASE`

Повторение: WHILE, LOOP
☞ Для получения дополнительной информации см. раздел "Оператор LOOP" (LOOP statement) на стр. 446 в документе *"Справочник по SQL для ASA" (ASA SQL Reference Manual)*.
	`WHILE условие LOOP`
	`    Список-операторов`
	`END LOOP`

Повторение: цикл FOR курсора
☞ Для получения дополнительной информации см. раздел "Оператор FOR" (FOR statement) на стр. 398 в документе *"Справочник по SQL для ASA" (ASA SQL Reference Manual)*.
	`FOR имя-цикла`
	`    AS имя-курсора`
	`  CURSOR FOR выбор оператора`
	`DO`
	`    Список-операторов`
	`END FOR`

Разрыв: LEAVE ☞ Для получения дополнительной информации см. раздел "Оператор LEAVE" (LEAVE statement) на стр. 435 в документе *"Справочник по SQL для ASA" (ASA SQL Reference Manual)*.	`LEAVE метка`
CALL ☞ Для получения дополнительной информации см. раздел "Оператор CALL" (CALL statement) на стр. 238 в документе *"Справочник по SQL для ASA" (ASA SQL Reference Manual)*.	`CALL имя-процедуры (аргумент, ... )`

☞ Для получения дополнительной информации о каждом операторе см. раздел “Операторы SQL” (SQL Statements) на стр. 189 в документе *"Справочник по SQL для ASA" (ASA SQL Reference Manual)*.

Использование составных операторов

Составной оператор начинается с ключевого слова BEGIN и заканчивается ключевым словом END. Тело процедуры или триггера является **составным оператором.** Составные операторы можно использовать в пакетах. Составные операторы могут быть вложены и объединены с другими управляющими операторами для определения потока выполнения в процедурах, триггерах или пакетах.

Составной оператор позволяет сгруппировать набор операторов SQL и использовать их в виде блока. В пределах составного оператора для операторов SQL используются разделители в виде точки с запятой.

☞ Для получения дополнительной информации о составных операторах см. раздел "Оператор BEGIN" (BEGIN statement) на стр. 232 в документе *“Справочник по SQL для ASA” (ASA SQL Reference Manual)*.

Объявления в составных операторах

Локальные объявления в составном операторе следуют сразу за ключевым словом BEGIN. Локальные объявления существуют только в пределах данного составного оператора. В пределах составного оператора можно объявить следующее:

- переменные;
- курсоры;
- временные таблицы;
- исключения (идентификаторы ошибок).

На локальные объявления может ссылаться любой оператор в пределах составного оператора или в пределах вложенного в него составного оператора. Локальные объявления не видимы для других процедур, вызываемых из составного оператора.

Атомарные составные операторы

Атомарным оператором называется оператор, выполняемый полностью или не выполняемый вообще. Например, при выполнении оператора UPDATE, обновляющего несколько тысяч строк, может возникнуть ошибка, когда большая часть строк уже будет обновлена. Если оператор выполняется не до конца, то происходит возврат всех изменений к первоначальному состоянию. Поэтому оператор UPDATE является атомарным оператором.

Все несоставные операторы SQL атомарны. Составной оператор можно сделать атомарным путем добавления ключевого слова ATOMIC после ключевого слова BEGIN.

```
BEGIN ATOMIC
    UPDATE employee
    SET manager_ID = 501
    WHERE emp_ID = 467;
    UPDATE employee
    SET birth_date = 'bad_data';
END
```

В этом примере два оператора обновления являются частью атомарного составного оператора. Оба эти оператора должны быть либо выполнены полностью, либо не выполнены вообще. Выполнение первого оператора будет успешным. Второй оператор вызывает ошибку преобразования данных, поскольку значение, указанное в столбце **birth_date**, не может быть преобразовано в дату.

Успешного завершения выполнения атомарного составного оператора не происходит, и промежуточные результаты обоих операторов UPDATE не сохраняются. Даже если транзакция, выполняющаяся в данный момент, в конечном счете будет подтверждена, ни один из этих операторов в атомарном составном операторе не окажет влияния на ее результат.

В пределах атомарного составного оператора нельзя использовать операторы COMMIT, ROLLBACK и некоторые операторы ROLLBACK TO SAVEPOINT (см. раздел "Транзакции и точки сохранения в процедурах и триггерах" на стр. 546).

Однако возможны случаи, когда выполняются только некоторые операторы, входящие в состав атомарного составного оператора. Это происходит тогда, когда обработчик исключений в пределах составного оператора сталкивается с ошибкой.

☞ Для получения дополнительной информации см. раздел "Использование обработчиков исключений в процедурах и триггерах" на стр. 541.

Структура процедур и триггеров

Тело процедуры или триггера содержит составной оператор, как описано в разделе "Использование составных операторов" на стр. 521. Составной оператор представляет набор операторов SQL, заключенных между BEGIN и END. Операторы разделяются точкой с запятой.

Операторы SQL, разрешенные в процедурах и триггерах

В процедурах и триггерах можно использовать почти все операторы SQL, в том числе следующие:

- SELECT, UPDATE, DELETE, INSERT и SET VARIABLE;
- оператор CALL для выполнения других процедур;
- управляющие операторы (см. раздел "Управляющие операторы" на стр. 519);
- операторы курсора (см. раздел "Использование курсоров в процедурах и триггерах" на стр. 533);
- операторы обработки исключений (см. раздел "Использование обработчиков исключений в процедурах и триггерах" на стр. 541);
- оператор EXECUTE IMMEDIATE.

Операторы SQL, которые нельзя использовать в процедурах и триггерах, включают в себя следующие:

- оператор CONNECT;
- оператор DISCONNECT.

Операторы COMMIT, ROLLBACK и SAVEPOINT могут использоваться в процедурах и триггерах при наличии некоторых ограничений (см. раздел "Транзакции и точки сохранения в процедурах и триггерах" на стр. 546).

☞ Для получения дополнительной информации см. раздел "Использование" для каждого оператора SQL в главе "Операторы SQL" (SQL Statements) на стр. 189 в документе *“Справочник по SQL для ASA” (ASA SQL Reference Manual)*.

Объявление параметров для процедур

Параметры процедуры указываются в виде списка в операторе CREATE PROCEDURE. Имена параметра должны соответствовать правилам для других идентификаторов базы данных, таких как заголовки столбцов. Они должны иметь действующие типы данных (см. раздел "Типы данных SQL" (SQL Data Types) на стр. 51 в документе *"Справочник по SQL для ASA" (ASA SQL Reference Manual)),* и перед ними должно быть введено одно из ключевых слов: IN, OUT или INOUT. Эти ключевые слова имеют следующие значения:

- **IN.** Аргументом является выражение, предоставляющее значение процедуре.
- **OUT.** Аргументом является переменная, которой процедура может назначить значение.
- **INOUT.** Аргументом является переменная, которая предоставляет значение процедуре, и которой процедура может предоставить новое значение.

В операторе CREATE PROCEDURE параметрам процедуры можно назначить значения по умолчанию. Значение по умолчанию должно быть константой и может быть NULL. Например, следующая процедура по умолчанию использует значение NULL для параметра IN во избежание выполнения запроса, не имеющего смысла:

```
CREATE PROCEDURE
CustomerProducts( IN customer_id
                  INTEGER DEFAULT NULL )
RESULT ( product_id INTEGER,
         quantity_ordered INTEGER )
BEGIN
    IF customer_id IS NULL THEN
       RETURN;
    ELSE
       SELECT product.id,
              sum( sales_order_items.quantity )
       FROM product,
            sales_order_items,
            sales_order
       WHERE sales_order.cust_id = customer_id
       AND sales_order.id = sales_order_items.id
       AND sales_order_items.prod_id = product.id
       GROUP BY product.id;
    END IF;
END
```

Следующий оператор назначает DEFAULT NULL, и процедура вместо запроса выполняет возврат.

```
CALL customerproducts();
```

Передача параметров процедурам

Можно воспользоваться значениями по умолчанию для параметров хранимой процедуры, включенными при выполнении любой из двух форм оператора CALL.

Если дополнительные параметры расположены в конце списка аргументов в операторе CREATE PROCEDURE, то их можно исключить из оператора CALL. В качестве примера рассмотрим процедуру с тремя параметрами INOUT:

```
CREATE PROCEDURE SampleProc( INOUT var1 INT
                                 DEFAULT 1,
                     INOUT var2 int DEFAULT 2,
                     INOUT var3 int DEFAULT 3 )
...
```

Предполагается, что среда вызова установила три переменные для хранения значений, переданных процедуре:

```
CREATE VARIABLE V1 INT;

CREATE VARIABLE V2 INT;

CREATE VARIABLE V3 INT;
```

Можно вызвать процедуру *SampleProc*, представляющую только первый параметр, следующим образом:

```
CALL SampleProc( V1 )
```

В этом случае значения по умолчанию используются для *var2* и *var3*.

Более гибким методом вызова процедур с дополнительными аргументами является передача параметров по имени. Процедуру *SampleProc* можно вызвать следующим образом:

```
CALL SampleProc( var1 = V1, var3 = V3 )
```

или так:

```
CALL SampleProc( var3 = V3, var1 = V1 )
```

Передача параметров функциям

Определяемые пользователем функции не вызываются оператором CALL, но используются так же, как встроенные функции. Например, для получения имен служащих следующий оператор использует функцию *fullname*, описанную в разделе "Создание определяемых пользователем функций" на стр. 506.

❖ Получение списка имен служащих

- Введите следующее:

```
SELECT fullname(emp_fname, emp_lname) AS Name FROM employee
```

Name
Fran Whitney
Matthew Cobb
Philip Chin
Julie Jordan
...

Примечания

- При вызове функций могут использоваться параметры по умолчанию. Однако параметры нельзя передавать функциям по имени.
- Параметры передаются по значению, а не по ссылке. Даже если функция изменяет значение параметра, это изменение не возвращается в среду вызова.
- Выходные параметры нельзя использовать в определяемых пользователем функциях.
- Определяемые пользователем функции не могут возвращать результирующие наборы.

Возвращение результатов из процедур

Процедуры могут возвращать результаты в форме одной или нескольких строк данных. Результаты, состоящие из одной строки данных, могут быть переданы назад процедуре в качестве аргументов. Результаты, состоящие из нескольких строк данных, возвращаются назад как результирующие наборы. Процедуры могут также возвратить отдельное значение, указанное в операторе RETURN.

✍ Простые примеры возвращения результатов из процедур представлены в разделе "Введение в процедуры" на стр. 499. Для получения более подробной информации см. разделы ниже.

Возвращение значения с использованием оператора RETURN

Оператор RETURN возвращает единственное целочисленное значение в среду вызова с немедленным выходом из процедуры. Оператор RETURN имеет следующую форму:

```
RETURN выражение
```

Значение предоставленного выражения возвращается в среду вызова. Для сохранения возвращаемого значения в переменной воспользуйтесь расширением оператора CALL:

```
CREATE VARIABLE returnval INTEGER ;
returnval = CALL myproc() ;
```

Возвращение результатов как параметров процедуры

Процедуры могут возвращать результаты в среду вызова в виде параметров процедуры.

В пределах процедуры параметрам и переменным могут быть назначены значения с использованием:

- оператора SET;
- оператора SELECT с разделом INTO.

Использование оператора SET

Следующая, в каком-то смысле искусственная процедура возвращает значение в параметр оператора OUT, назначенный с использованием оператора SET:

```
CREATE PROCEDURE greater ( IN a INT,
                           IN b INT,
                          OUT c INT)
BEGIN
    IF a > b THEN
        SET c = a;
    ELSE
         SET c = b;
     ND IF ;
END
```

Использование операторов SELECT отдельной строки

Однострочные запросы получают максимум одну строку из базы данных. В этом типе запросов используется оператор SELECT с разделом INTO. Раздел INTO следует за списком выбора и предшествует разделу FROM. В нем содержится список переменных для получения значения для каждого

элемента списка выбора. Количество переменных должно соответствовать количеству элементов в списке выбора.

При выполнении оператора SELECT сервер извлекает результаты оператора SELECT и помещает их в переменные. Если результаты запроса содержат больше одной строки, то сервер возвращает сообщение об ошибке. Для запросов, возвращающих более одной строки, необходимо использовать курсоры. Для получения информации о возвращении более чем одной строки из процедуры см. раздел "Возвращение результирующих наборов из процедур" на стр. 529.

Если в результате запроса не выделяется ни одна строка, то появляется предупреждение: **row not found** (строка не найдена).

Следующая процедура возвращает результаты оператора SELECT отдельной строки в параметры процедуры.

❖ **Возврат количества заказов, размещенных определенным клиентом**

♦ Введите следующее:

```
CREATE PROCEDURE OrderCount (IN customer_ID INT,
                                 OUT Orders INT)
BEGIN
    SELECT COUNT(DBA.sales_order.id)
        INTO Orders
    FROM DBA.customer
        KEY LEFT OUTER JOIN "DBA".sales_order
    WHERE DBA.customer.id = customer_ID;
END
```

Эту процедуру можно проверить в Interactive SQL с использованием следующих операторов, которые показывают количество заказов, размещенных клиентом с кодом 102:

```
CREATE VARIABLE orders INT;
CALL OrderCount ( 102, orders );
SELECT orders;
```

Примечания

- Параметр *customer_ID* объявлен как параметр IN. Этот параметр содержит код клиента, передаваемый процедуре.
- Параметр *Orders* объявлен как параметр OUT. В нем содержится значение переменной заказов (Order), возвращаемое в среду вызова.

- Нет необходимости в использовании оператора DECLARE для переменной *Orders,* поскольку она объявлена в списке аргументов процедуры.
- Оператор SELECT возвращает единственную строку и размещает ее в переменной *Orders.*

Возвращение результирующих наборов из процедур

Результирующие наборы позволяют процедуре возвращать более одной строки результатов в среду вызова.

Следующая процедура возвращает список разместивших заказы клиентов, а также общее количество размещенных заказов. Процедура не включает в этот список клиентов, которые не размещали заказы.

```
CREATE PROCEDURE ListCustomerValue ()
RESULT ("Company" CHAR(36), "Value" INT)
BEGIN
    SELECT company_name,
        CAST( sum( sales_order_items.quantity *
                   product.unit_price)
                   AS INTEGER ) AS value
    FROM customer
        INNER JOIN sales_order
        INNER JOIN sales_order_items
        INNER JOIN product
    GROUP BY company_name
    ORDER BY value DESC;
END
```

- Введите следующее:

```
CALL ListCustomerValue ()
```

Company	Value
Chadwicks	8076
Overland Army Navy	8064
Martins Landing	6888
Sterling & Co.	6804
Carmel Industries	6780
...	...

Примечания

- Количество переменных в списке RESULT должно соответствовать количеству элементов списка SELECT. Если типы данных не совпадают, то там, где это возможно, выполняется автоматическое преобразование типа данных.
- Раздел RESULT является частью оператора CREATE PROCEDURE и не имеет разделителя команды.
- Имена элементов списка SELECT не должны соответствовать таковым из списка RESULT.

- При проверке этой процедуры по умолчанию Interactive SQL отображает только первый результирующий набор. Можно настроить конфигурацию Interactive SQL так, чтобы отображалось более одного результирующего набора путем активизации параметра “Show multiple result sets” на закладке Results в диалоге Options.
- В результирующие наборы процедуры можно вносить изменения, если эти результирующие наборы не сгенерированы из представления. Для внесения изменений в результаты процедуры пользователю, вызывающему процедуру, необходимо обладать соответствующими полномочиями на используемую таблицу. Эти полномочия отличаются от обычных полномочий на выполнение процедуры, где полномочия на таблицу должен иметь владелец процедуры.

☞ Для получения информации об изменении результирующих наборов в Interactive SQL см. раздел "Редактирование значений таблицы в Interactive SQL" (Editing table values in Interactive SQL) на стр. 83 в документе *“Введение в ASA” (ASA Getting Started)*.

Возвращение нескольких результирующих наборов из процедур

Для того чтобы в Interactive SQL стал возможен возврат множественных результирующих наборов, необходимо активизировать соответствующий параметр на закладке Results в диалоге Options. По умолчанию этот параметр выключен. Изменение этой настройки вступает в силу для новых установленных подключений (например, с новыми окнами).

❖ **Включение возврата множественных результирующих наборов**

1. Выберите Tools➤Options.
2. В открывшемся диалоге Options выберите закладку Results.
3. Выберите переключатель Show Multiple Result Sets.

После активизации данного параметра процедура может возвращать в среду вызова более одного результирующего набора. Если задействован раздел RESULT, то результирующие наборы должны быть совместимыми: в них должно быть одинаковое число элементов в списках SELECT, и типы данных должны быть такими, чтобы иметь возможность автоматического преобразования в типы данных, перечисленные в списке RESULT.

В следующей процедуре выводятся имена всех служащий, клиентов и контактных лиц, присутствующих в базе данных:

```
CREATE PROCEDURE ListPeople()
RESULT ( lname CHAR(36), fname CHAR(36) ) BEGIN
    SELECT emp_lname, emp_fname FROM employee;
    SELECT lname, fname
    FROM customer;
    SELECT last_name, first_name
    FROM contact;
END
```

Примечания

- Для проверки этой процедуры в Interactive SQL введите следующий оператор в окне SQL Statements:

```
CALL ListPeople ()
```

Возвращение различных результирующих наборов из процедур

В процедурах раздел RESULT является дополнительным. Исключение раздела результатов позволяет создавать процедуры, возвращающие различные результирующие наборы с разным количеством или типами столбцов в зависимости от того, как они выполняются.

Если не используется возможность возвращения различных результирующих наборов, по соображениям производительности следует включить раздел RESULT.

Например, следующая процедура возвращает два столбца, если входная переменная - Y, но только один столбец имеет другое значение:

```
CREATE PROCEDURE names( IN formal char(1))
BEGIN
    IF formal = 'y' THEN
        SELECT emp_lname, emp_fname
        FROM employee
    ELSE
        SELECT emp_fname
        FROM employee
    END IF
END
```

В зависимости от интерфейса клиентского приложения, на использование различных результирующих наборов в процедурах могут налагаться определенные ограничения.

- **Embedded SQL.** Для получения результирующего набора надлежащей формы необходимо описать (с использованием DESCRIBE) вызов процедуры после открытия курсора для результирующего набора, но до возвращения каких-либо строк.

  Для получения дополнительной информации об операторе DESCRIBE см. раздел "Оператор DESCRIBE [ESQL]" (DESCRIBE statement [ESQL]) на стр. 361 в документе *"Справочник по SQL для ASA" (ASA SQL Reference Manual)*.

- **ODBC.** Процедуры с возвращением различных результирующих наборов могут использоваться приложениями ODBC. Драйвер ODBC Adaptive Server Anywhere выполняет надлежащее описание различных результирующих наборов.

- **Приложения Open Client.** Приложения Open Client могут использовать процедуры с возвращением различных результирующих наборов. Adaptive Server Anywhere выполняет надлежащее описание различных результирующих наборов.

Использование курсоров в процедурах и триггерах

Курсоры извлекают по одной строке для каждого запроса или хранимой процедуры, и при этом результирующие наборы содержат несколько строк. Курсор является дескриптором или идентификатором запроса или процедуры, а также указывает текущую позицию в пределах результирующего набора.

Обзор управления курсором

Управление курсором подобно управлению файлом в языке программирования. Управление курсором осуществляется следующим образом:

1. Объявите курсор для определенного оператора SELECT или процедуры с помощью оператора DECLARE.
2. Откройте курсор с помощью оператора OPEN.
3. Для получения результатов по одной строке из курсора воспользуйтесь оператором FETCH.
4. Предупреждение Row Not Found (строка не найдена) выдается по достижении конца результирующего набора.
5. Закройте курсор с помощью оператора CLOSE.

По умолчанию курсоры автоматически закрываются в конце транзакции (по операторам COMMIT или ROLLBACK). Курсоры открываются с использованием раздела WITH HOLD и остаются открытыми для последующих транзакций, пока не будут закрыты явно.

☞ Для получения дополнительной информации о позиционировании курсоров см. раздел "Позиционирование курсора" (Cursor positioning) на стр. 19 в документе *“Руководство по программированию ASA” (ASA Programming Guide)*.

Использование курсоров для операторов SELECT в процедурах

Следующая процедура использует курсор для оператора SELECT. Основанная на том же запросе, что и в процедуре ListCustomerValue, описанной в разделе "Возвращение результирующих наборов из процедур" на стр. 529, она иллюстрирует несколько возможностей языка хранимых процедур.

```
CREATE PROCEDURE TopCustomerValue
    ( OUT TopCompany CHAR(36),
      OUT TopValue INT )
BEGIN
     -- 1. Объявление исключения “ошибка не найдена“
     DECLARE err_notfound
     EXCEPTION FOR SQLSTATE ’02000’;
-- 2. Объявление переменных для хранения
-- всех имен организаций и соответствующих значений
DECLARE ThisName CHAR(36);
DECLARE ThisValue INT;
-- 3. Объявление курсора ThisCompany
-- для запроса
DECLARE ThisCompany CURSOR FOR
```

```
SELECT company_name,
    CAST( sum( sales_order_items.quantity *
          product.unit_price ) AS INTEGER )
    AS value
FROM customer
    INNER JOIN sales_order
    INNER JOIN sales_order_items
    INNER JOIN product
GROUP BY company_name;
-- 4. Инициализация значений TopValue
SET TopValue = 0;
-- 5. Открытие курсора
OPEN ThisCompany;
-- 6. Создание цикла к строкам запроса
CompanyLoop:
LOOP
    FETCH NEXT ThisCompany
    INTO ThisName, ThisValue;
    IF SQLSTATE = err_notfound THEN
        LEAVE CompanyLoop;
    END IF;
    IF ThisValue > TopValue THEN
        SET TopCompany = ThisName;
        SET TopValue = ThisValue;
    END IF;
END LOOP CompanyLoop;
-- 7. Закрытие курсора
CLOSE ThisCompany;
END
```

Примечания

Процедура *TopCustomerValue* имеет следующие особенности:

- Объявляется исключение " error not found" (ошибка не найдена). Позже в процедуре это исключение указывает на завершение цикла результатов запроса.

Для получения дополнительной информации об исключениях см. раздел "Ошибки и предупреждения в процедурах и триггерах" на стр. 536.

- Заявлено, что две локальные переменные - *ThisName* и *ThisValue* - хранят результаты каждой строки запроса.
- Объявлен курсор *ThisCompany.* Оператор SELECT составляет список названий организаций и общего количества заказов, размещенных организацией.
- Значение *TopValue* установлено на начальное значение 0 для последующего использования в цикле.

- Открывается курсор *ThisCompany*.
- Оператор LOOP осуществляет цикл по каждой строке запроса, помещая по очереди каждое название организации в переменные *ThisName* и *ThisValue*. Если *ThisValue* больше, чем текущее высшее значение, то *TopCompany* и *TopValue* сбрасываются на *ThisName* и *ThisValue*.
- Курсор закрывается в конце процедуры.
- Эту процедуру также можно создать без цикла, добавляя значение ORDER BY раздела DESC к оператору SELECT. Тогда необходимо получение только первой строки курсора.

Конструкция LOOP в процедуре *TopCompanyValue* – это стандартная форма, обеспечивающая выход после обработки последней строки. Эту процедуру можно перезаписать в более компактной форме, используя цикл FOR. Оператор FOR объединяет несколько аспектов вышеприведенной процедуры в отдельный оператор.

```
CREATE PROCEDURE TopCustomerValue2(
                  OUT TopCompany CHAR(36),
                  OUT TopValue INT )
BEGIN
    -- Инициализация переменной TopValue
    SET TopValue = 0;
    -- Выполнение цикла For
    FOR CompanyFor AS ThisCompany
        CURSOR FOR
        SELECT company_name AS ThisName ,
            CAST( sum( sales_order_items.quantity *
                  product.unit_price ) AS INTEGER )
            AS ThisValue
        FROM customer
            INNER JOIN sales_order
            INNER JOIN sales_order_items
            INNER JOIN product
        GROUP BY ThisName
    DO
        IF ThisValue > TopValue THEN
        SET TopCompany = ThisName;
        SET TopValue = ThisValue;
        END IF;
    END FOR;
END
```

Ошибки и предупреждения в процедурах и триггерах

После выполнения приложением оператора SQL оно может проверить **код состояния**. Этот код состояния (или код возврата) указывает либо на успешное выполнение оператора, либо на его неуспешное выполнение с указанием причины последнего. Для указания успешного или неуспешного выполнения оператора CALL для процедуры можно воспользоваться тем же механизмом.

При сообщении об ошибке используется описание состояния SQLCODE или SQLSTATE. Полные описания ошибок и предупреждений SQLCODE и SQLSTATE, а также их значения, приведены в разделе "Сообщения об ошибках базы данных" (Database Error Messages) на стр. 1 в документе *"Справочник по ошибкам ASA" (ASA Errors Manual)*. Всякий раз при выполнении оператора SQL в специальных переменных процедуры, называемых SQLSTATE и SQLCODE, появляется значение. Это значение указывает, встречались ли какие-либо необычные состояния во время выполнения оператора. Значение SQLSTATE или SQLCODE можно проверить в операторе IF, следующим за оператором SQL, и предпринять соответствующие меры в зависимости от того, было выполнение оператора успешным или нет.

Например, переменную SQLSTATE можно использовать для указания того, успешно ли был завершен выбор строки. В процедуре TopCustomerValue, описанной в разделе "Использование курсоров для операторов SELECT в процедурах" на стр. 533, проверка SQLSTATE использовалась для подтверждения обработки всех строк оператора SELECT.

Обработка по умолчанию ошибок в процедурах и триггерах

В этом разделе описывается процесс обработки в Adaptive Server Anywhere ошибок, возникающих во время выполнения процедуры, в том случае, если режим обработки ошибок не встроен в процедуру.

☞ При другом поведении можно использовать обработчики исключений, описанные в разделе "Использование обработчиков исключений в процедурах и триггерах" на стр. 541. Обработка предупреждений немного отличается от обработки ошибок: см. описание в разделе "Обработка по умолчанию предупреждений в процедурах и триггерах" на стр. 540.

Существует два способа обработки ошибок без явного использования режима обработки ошибок:

- **Обработка ошибок по умолчанию.** При неуспешном выполнении процедуры или триггера в среду вызова возвращается код ошибки.
- **ON EXCEPTION RESUME.** Если в операторе CREATE PROCEDURE присутствует раздел ON EXCEPTION RESUME, процедура продолжает выполняться после ошибки, начиная с оператора, следующего за содержащем ошибку оператором.

✍ Точное поведение для процедур, использующих ON EXCEPTION RESUME, управляется установкой параметра ON_TSQL_ERROR. Для получения дополнительной информации см. раздел "Параметр ON_TSQL_ERROR" (ON_TSQL_ERROR option) на стр. 571 в документе *"Руководство по администрированию баз данных ASA" (ASA Database Administration Guide)*.

Обработка ошибок по умолчанию

Как правило, при неуспешном выполнении оператор SQL в процедуре или триггере выполнение триггера или процедуры останавливается, и управление возвращается к прикладной программе с соответствующей настройкой значений SQLCODE и SQLSTATE. Это происходит даже в том случае, когда ошибка, имевшая место в процедуре или триггере, была прямо или косвенно вызвана этой процедурой. В случае триггера операция, запустившая этот триггер, также не выполняется, в приложение возвращается ошибка.

Следующие процедуры иллюстрируют ситуацию, когда приложение вызывает процедуру *OuterProc*, а процедура *OuterProc*, в свою очередь, вызывает процедуру *InnerProc*, которая затем вызывает ошибку.

```
CREATE PROCEDURE OuterProc()
BEGIN
    MESSAGE 'Hello from OuterProc.' TO CLIENT;
    CALL InnerProc();
    MESSAGE 'SQLSTATE set to ',
        SQLSTATE,' in OuterProc.' TO CLIENT
END
CREATE PROCEDURE InnerProc()
    BEGIN
        DECLARE column_not_found
            EXCEPTION FOR SQLSTATE '52003';
        MESSAGE 'Hello from InnerProc.' TO CLIENT;
        SIGNAL column_not_found;
        MESSAGE 'SQLSTATE set to ',
            SQLSTATE, ' in InnerProc.' TO CLIENT;
    END
```

Примечания

- Оператор DECLARE в InnerProc объявляет символическое имя одного из определенных ранее значений SQLSTATE, связанных с уже известными серверу условиями ошибки.
- Оператор MESSAGE передает сообщение в окно сообщений Interactive SQL.
- Оператор SIGNAL генерирует условие ошибки из процедуры *InnerProc*.

Следующий оператор выполняет процедуру *OuterProc:*

```
CALL OuterProc();
```

В окне сообщений Interactive SQL отображается следующее:

Hello from OuterProc. (Сообщение процедуры OuterProc)

Hello from InnerProc. (Сообщение процедуры InnerProc)

Ни один из операторов, следующих за оператором SIGNAL в *InnerProc*, не выполняется: *InnerProc* немедленно передает управление обратно среде вызова, которой в этом случае является процедура *OuterProc*. Не выполняется ни один из операторов, следующих за оператором CALL в *OuterProc*. Условие ошибки возвращается в среду вызова для обработки. Например, Interactive SQL обрабатывает ошибку путем вывода окна сообщений с описанием ошибки.

Функция TRACEBACK предоставляет список операторов, выполнявшихся в момент ошибки. Функцию TRACEBACK можно использовать из Interactive SQL путем ввода следующего оператора:

```
SELECT TRACEBACK(*)
```

Обработка ошибок при помощи ON EXCEPTION RESUME

Если в операторе CREATE PROCEDURE присутствует раздел ON EXCEPTION RESUME, то при возникновении ошибки процедура проверяет следующий оператор. Если оператор обрабатывает ошибку, то процедура продолжает выполняться с оператора, следующего за вызвавшим ошибку оператором. При возникновении ошибки процедура не передает управление среде вызова.

☞ Поведение процедур, использующих ON EXCEPTION RESUME, можно изменить настройкой параметра ON_TSQL_ERROR. Для получения дополнительной информации см. разделе "Параметр ON_TSQL_ERROR" (ON_TSQL_ERROR parameter) на стр. 571 в документе *"Руководство по администрированию баз данных ASA" (ASA Database Administration Guide)*.

Операторы обработки ошибок следующие:

- IF
- SELECT @variable =
- CASE
- LOOP
- LEAVE
- CONTINUE
- CALL
- EXECUTE
- SIGNAL
- RESIGNAL
- DECLARE
- SET VARIABLE

Следующий пример иллюстрирует процесс работы.

Сброс процедур

Перед продолжением выполнения примеров не забудьте удалить обе процедуры *InnerProc* и *OuterProc* вводом следующих команд в окне SQL Statements:

```
DROP PROCEDURE OuterProc;
DROP PROCEDURE InnerProc
```

Следующие процедуры иллюстрируют ситуацию, когда приложение вызывает процедуру *OuterProc*, а процедура *OuterProc*, в свою очередь, вызывает процедуру *InnerProc*, которая затем вызывает ошибку. Эти демонстрационные процедуры основаны на используемых выше в этом разделе процедурах:

```
CREATE PROCEDURE OuterProc()
ON EXCEPTION RESUME
BEGIN
    DECLARE res CHAR(5);
    MESSAGE 'Hello from OuterProc.' TO CLIENT;
    CALL InnerProc();
    SELECT @res=SQLSTATE;
    IF res='52003' THEN
        MESSAGE 'SQLSTATE set to ',
            res, ' in OuterProc.' TO CLIENT;
    END IF
END;

CREATE PROCEDURE InnerProc()
ON EXCEPTION RESUME
BEGIN
    DECLARE column_not_found
        EXCEPTION FOR SQLSTATE '52003';
    MESSAGE 'Hello from InnerProc.' TO CLIENT;
    SIGNAL column_not_found;
    MESSAGE 'SQLSTATE set to ',
    SQLSTATE, ' in InnerProc.' TO CLIENT;
END
```

Следующий оператор выполняет процедуру *OuterProc:*

```
CALL OuterProc();
```

Затем в окне сообщений Interactive SQL отображается следующее:

Hello from OuterProc. (Сообщение процедуры OuterProc)

Hello from InnerProc. (Сообщение процедуры InnerProc)

SQLSTATE set to 52003 in OuterProc. (В OuterProc значение SQLSTATE - 52003)

Порядок выполнения следующий:

1 Выполняется *OuterProc* и вызывает *InnerProc*.

2 В InnerProc оператор SIGNAL сообщает об ошибке.

3 Оператор MESSAGE не является оператором-обработчиком ошибок, поэтому управление передается OuterProc, и сообщение не отображается.

4 В OuterProc оператор, следующий за ошибкой, назначает значение SQLSTATE переменной **res**. Эта переменная является оператором обработки ошибок, поэтому выполнение продолжается, и появляется сообщение OuterProc.

Обработка по умолчанию предупреждений в процедурах и триггерах

Ошибки и предупреждения обрабатываются по-разному. Если по умолчанию для ошибок задается значение для переменных SQLSTATE и SQLCODE, и в случае ошибки происходит возвращение управления среде вызова, то для предупреждений по умолчанию после задания значений SQLSTATE и SQLCODE выполнение процедуры продолжается.

Сброс процедур

Перед продолжением выполнения примеров не забудьте удалить обе процедуры *InnerProc* и *OuterProc* вводом следующих команд в окне SQL Statements:

```
DROP PROCEDURE OuterProc; DROP PROCEDURE InnerProc
```

Следующие демонстрационные процедуры иллюстрируют обработку предупреждений по умолчанию. Эти демонстрационные процедуры основаны на используемых в разделе "Обработка ошибок по умолчанию в процедурах и триггерах" на стр. 536. В этом случае оператор SIGNAL генерирует условие *row not found* (строка не найдена), которое является скорее предупреждением, а не ошибкой.

```
CREATE PROCEDURE OuterProc()
BEGIN
    MESSAGE 'Hello from OuterProc.' TO CLIENT;
    CALL InnerProc();
    MESSAGE 'SQLSTATE set to ',
        SQLSTATE,' in OuterProc.' TO CLIENT;
END


CREATE PROCEDURE InnerProc()

BEGIN
    DECLARE row_not_found
        EXCEPTION FOR SQLSTATE '02000';
    MESSAGE 'Hello from InnerProc.' TO CLIENT;
    SIGNAL row_not_found;
    MESSAGE 'SQLSTATE set to ',

SQLSTATE, ' in InnerProc.' TO CLIENT;
END
```

Следующий оператор выполняет процедуру *OuterProc:*

```
CALL OuterProc();
```

Затем в окне сообщений Interactive SQL отображается следующее:

```
Hello from OuterProc. (Сообщение процедуры OuterProc)
Hello from InnerProc. (Сообщение процедуры InnerProc)
SQLSTATE set to 02000 in InnerProc. (В InnerProc значение SQLSTATE - 02000)
SQLSTATE set to 00000 in OuterProc. (В OuterProc значение SQLSTATE - 00000)
```

Выполнение обеих процедур продолжено после генерации предупреждения с заданным для него SQLSTATE (02000).

Выполнение второго оператора MESSAGE в InnerProc сбрасывает предупреждение. Успешное выполнение любого оператора SQL сбрасывает SQLSTATE до 00000, а SQLCODE - до 0. Если процедура должна сохранить состояние ошибки, то задание значения должно быть осуществлено немедленно после выполнения оператора, вызвавшего предупреждение об ошибке.

Использование обработчиков исключений в процедурах и триггерах

Часто требуется перехватывать ошибки некоторых типов и обработать их в пределах процедуры или триггера, вместо того, чтобы возвращать ошибку в среду вызова. Это выполняется при помощи **обработчика исключений.**

Обработчик исключений определяется разделом EXCEPTION составного оператора (см. раздел "Использование составных операторов" на стр. 521). Обработчик исключений выполняется всякий раз при возникновении ошибки в составном операторе. В отличие от ошибок, предупреждения не вызывают выполнения кода обработки исключений. Код обработки исключений также выполняется при появлении ошибки во вложенном составном операторе, либо в процедуре или триггере, вызванных в пределах составного оператора.

Сброс процедур

Перед продолжением выполнения примеров не забудьте удалить обе процедуры *InnerProc* и *OuterProc* вводом следующих команд в окне SQL Statements:

```
DROP PROCEDURE OuterProc; DROP PROCEDURE InnerProc
```

Демонстрационные процедуры, используемые для иллюстрации обработки исключений, основаны на описанных в разделе "Обработка по умолчанию ошибок в процедурах и триггерах" на стр. 536. В этом случае дополнительный код обрабатывает ошибку "column not found" (столбец не найден) в процедуре *InnerProc*.

```
CREATE PROCEDURE OuterProc()
BEGIN
    MESSAGE 'Hello from OuterProc.' TO CLIENT;
    CALL InnerProc();
    MESSAGE 'SQLSTATE set to ',
        SQLSTATE,' in OuterProc.' TO CLIENT
END
CREATE PROCEDURE InnerProc()
BEGIN
    DECLARE column_not_found
        EXCEPTION FOR SQLSTATE '52003';
    MESSAGE 'Hello from InnerProc.' TO CLIENT;
    SIGNAL column_not_found;
```

```
        MESSAGE 'Line following SIGNAL.' TO CLIENT;
        EXCEPTION
            WHEN column_not_found THEN
                 MESSAGE 'Column not found handling.' TO
CLIENT;
            WHEN OTHERS THEN
        RESIGNAL ;
END
```

Оператор EXCEPTION объявляет выполнение самого обработчика исключений. Строки, следующие за оператором EXCEPTION, не выполняются, если ошибки не возникает. Каждый раздел WHEN определяет имя исключения (объявляемого с помощью оператора DECLARE), а также оператор или операторы, которые должны быть выполнены в случае возникновения этого исключения. Раздел WHEN OTHERS THEN определяет оператор или операторы, которые должны быть выполнены, если произошедшее исключение не появляется в предшествующих разделах WHEN.

В этом примере оператор RESIGNAL передает исключение обработчику исключений более высокого уровня. RESIGNAL является действием по умолчанию в случае, если раздел WHEN OTHERS THEN в обработчике исключений не определен.

Следующий оператор выполняет процедуру *OuterProc:*

```
CALL OuterProc();
```

Затем в окне сообщений Interactive SQL отображается следующее:

```
Hello from OuterProc. (Сообщение процедуры OuterProc)
Hello from InnerProc. (Сообщение процедуры InnerProc)
Column not found handling. (Обработка ненайденного столбца)
SQLSTATE set to 00000 in OuterProc. (В OuterProc значение SQLSTATE - 00000)
```

Примечания

- Вместо строк, следующих за оператором SIGNAL в *InnerProc,* выполняются операторы EXCEPTION.
- Поскольку произошедшая ошибка является ошибкой "column not found" (столбец не найден), выполняется оператор MESSAGE, включенный для обработки ошибки, и SQLSTATE обнуляется (указывает на отсутствие ошибок).
- После выполнения кода обработки исключений управление передается процедуре *OuterProc,* выполнение которой продолжается, как при отсутствии ошибок.
- ON EXCEPTION RESUME не должен использоваться с явной обработкой исключений. При наличии ON EXCEPTION RESUME код обработки исключений не выполняется.

- Если код обработки ошибок для исключения “column not found” (столбец не найден) является просто оператором RESIGNAL, то управление передается процедуре *OuterProc* с SQLSTATE, имеющей значение 52003. Эта ситуация не отличается от той, какая возникает при отсутствии кода обработки ошибок в *InnerProc*. Если *OuterProc* не содержит кода обработки ошибок, выполнение процедуры прекращается.

Обработка исключений в атомарных составных операторах

Когда исключение обрабатывается в пределах составного оператора, выполнение составляющий его операторов завершается без активного исключения, и изменения, сделанные до появления исключения, становятся необратимыми. Это же относится и к атомарным составным операторам. Если ошибка происходит в пределах атомарного составного оператора и обрабатывается явно, то в атомарном составном операторе выполняются некоторые, но не все операторы.

Вложенные составные операторы и обработчики исключений

Код, следующий за оператором, вызывающим ошибку, выполняется только в случае наличия в определении процедуры раздела EXCEPTION RESUME.

Вложенные составные операторы можно использовать для обеспечения более тщательного контроля за тем, какие операторы, следующие за ошибкой, выполняются, а какие - нет.

Сброс процедур

Перед продолжением выполнения примеров не забудьте удалить обе процедуры *InnerProc* и *OuterProc* вводом следующих команд в окне SQL Statements:

```
DROP PROCEDURE OuterProc; DROP PROCEDURE InnerProc
```

Следующая демонстрационная процедура иллюстрирует процесс использования вложенных составных операторов для управления потоком. Данная процедура основана на процедуре, используемой в качестве примера в разделе "Обработка по умолчанию ошибок в процедурах и триггерах" на стр. 536.

```
CREATE PROCEDURE InnerProc()
BEGIN
    BEGIN
         DECLARE column_not_found
              EXCEPTION FOR SQLSTATE VALUE '52003';
              MESSAGE 'Hello from InnerProc' TO CLIENT;
              SIGNAL column_not_found;
                  MESSAGE 'Line following SIGNAL' TO CLIENT
                  EXCEPTION
              WHEN column_not_found THEN
                  MESSAGE 'Column not found handling' TO
                  CLIENT;
              WHEN OTHERS THEN
                  RESIGNAL;
    END;
```

```
        MESSAGE 'Outer compound statement' TO CLIENT;
    END
```

Следующий оператор выполняет процедуру *InnerProc:*

```
CALL InnerProc();
```

Затем в окне Interactive SQL отображается следующее:

```
Hello from InnerProc (Сообщение процедуры InnerProc)
Column not found handling (Обработка ненайденного столбца)
Outer compound statement (Внешний составной оператор )
```

При обнаружении оператора SIGNAL, вызывающего ошибку, управление переходит к обработчику исключений для составного оператора, и выводится сообщение "Column not found handling" (Обработка ненайденного столбца). Затем управление передается назад внешнему составному оператору, и выводится сообщение “ Outer compound statement ” (Внешний составной оператор).

При обнаружении во внутреннем составном операторе иной ошибки, нежели “column not found”, обработчик исключений выполняет оператор RESIGNAL. Оператор RESIGNAL передает управление напрямую в среду вызова, и остальная часть внешнего составного оператора не выполняется.

Использование оператора EXECUTE IMMEDIATE в процедурах

Оператор EXECUTE IMMEDIATE позволяет конструировать операторы в пределах процедур с использованием комбинации литеральных строк (в кавычках) и переменных.

Например, следующая процедура включает в себя оператор EXECUTE IMMEDIATE, создающий таблицу.

```
CREATE PROCEDURE CreateTableProc
           IN tablename char(30) )
BEGIN
     EXECUTE IMMEDIATE 'CREATE TABLE ' || tablename ||
'(column1 INT PRIMARY KEY)'
END
```

В АТОМАРНЫХ составных операторах нельзя использовать оператор EXECUTE IMMEDIATE, вызывающий оператор COMMIT, поскольку COMMIT в этом контексте недопустим.

Оператор EXECUTE IMMEDIATE не поддерживает выполнение операторов или запросов, возвращающих результирующие наборы.

☞ Для получения дополнительной информации об операторе EXECUTE IMMEDIATE см. раздел “Оператор EXECUTE [SP]" (EXECUTE statement [SP]) на стр. 386 в документе *“Справочное описание по ASA SQL” (ASA SQL Reference Manual)*.

Транзакции и точки сохранения в процедурах и триггерах

Операторы SQL в процедуре или триггере являются частью текущей транзакции (см. раздел "Использование транзакций и уровней изоляции" на стр. 89). В пределах одной транзакции можно вызвать несколько процедур или разместить несколько транзакций в одной процедуре.

Использование операторов COMMIT и ROLLBACK недопустимо в пределах любого атомарного оператора (см. раздел "Атомарные составные операторы" на стр. 521). Помните, что триггеры запускаются благодаря INSERT, UPDATE или DELETE, которые являются атомарными операторами. В триггере или в любых процедурах, вызываемых триггером, использование операторов COMMIT и ROLLBACK недопустимо.

В пределах процедуры или триггера можно использовать точки сохранения (см. раздел "Точки сохранения внутри транзакций" на стр. 93), но оператор ROLLBACK TO SAVEPOINT не может использовать точку сохранения до того, как будет запущена атомарная операция. Кроме того, по завершении атомарной операции все точки сохранения в ее пределах освобождаются.

Несколько советов по написанию процедур

В данном разделе представлены некоторые указания по созданию процедур.

Проверка необходимости изменения разделителя команд

При написании процедур изменять разделитель команд в Interactive SQL или Sybase Central не нужно. Однако если процедуры и триггеры создаются и проверяются с помощью какого-либо другого средства просмотра, то, возможно, потребуется изменение разделителя команд с точки с запятой на другой символ.

В пределах процедуры каждый оператор заканчивается точкой с запятой. Для разбора самого оператора CREATE PROCEDURE в некоторых средствах просмотра необходимо изменение разделителя команд с точки с запятой на другой символ.

При использовании приложения, для которого необходимо изменить символ разделителя команд, целесообразно использовать в качестве нового символа две точки с запятой (;;) или вопросительный знак (?), если система не допускает использования многознакового разделителя.

Необходимое разграничение операторов в процедуре

Каждый оператор в пределах процедуры должен заканчиваться точкой с запятой. Точки с запятой можно оставлять для завершения последнего оператора в списке, однако, целесообразно использовать их после каждого оператора.

Оператор CREATE PROCEDURE содержит описание RESULT и составной оператор, формирующие его тело. После ключевых слов BEGIN и END или после раздела RESULT точки с запятой не ставятся.

Использование полностью определенных имен таблиц в процедурах

Если в процедуре имеются ссылки на таблицы, то именам последних всегда должно предшествовать имя владельца (создателя) той или иной таблицы.

Когда процедура ссылается на таблицу, то она использует членство создателя процедуры членство в какой-либо группе для определения местонахождения таблиц без явно указанного имени владельца. Например, если процедура, созданная *user_1,* ссылается на *Table_B* и не определяет владельца *Table_B*, то тогда *Table_B* должна быть создана *user_1*, либо *user_1* должен быть членом группы (прямо или косвенно), являющейся владельцем *Table_B.* Если ни одно из этих условий не соблюдено, то при вызове процедуры появляется сообщение “table not found” (таблица не найдена).

Неудобство длинных, полностью определенных имен можно свести к минимуму путем использования корреляционных имен для обозначения таблиц в пределах оператора. Для получения информации о корреляционных именах см. раздел "Раздел FROM" (FROM clause) на стр. 402 в документе *"Справочник по SQL для ASA" (ASA SQL Reference Manual)*.

Определение дат и времени в процедурах

Даты и время направляются из процедур в базу данных в виде строк. Часть строки, содержащая дату, интерпретируется согласно текущей настройке параметра DATE_ORDER базы данных. Поскольку разные подключения могут изменить настроенные значения этого параметра, некоторые строки могут быть ошибочно преобразованы в даты, либо база данных не сможет преобразовать строку в дату.

При использовании строк дат в пределах процедур необходимо пользоваться однозначным форматом даты: *yyyy-mm-dd* (гггг-мм-дд) или *yyyy/mm/dd* (гггг/мм/дд). Сервер однозначно воспринимает эти строки как даты, независимо от настройки параметра DATE_ORDER.

☞ Для получения дополнительной информации о датах и времени см. раздел "Типы данных даты и времени" (Date and time data types) на стр. 65 в документе *"Справочник по SQL для ASA" (ASA SQL Reference Manual)*.

Верификация передачи входных аргументов процедуры

Одним из способов верификации входных аргументов является отображение значения параметра в окне сообщений Interactive SQL при помощи оператора MESSAGE. Например, следующая процедура просто отображает значение входного параметра *var:*

```
CREATE PROCEDURE message_test (IN var char(40))
BEGIN
    MESSAGE var TO CLIENT;
END
```

Можно также использовать отладчик хранимых процедур.

Операторы, разрешенные в пакетах

Для использования в пакетах допустимы все операторы SQL (включая операторы определения данных, такие как CREATE TABLE, ALTER TABLE и т. д.), за исключением следующих:

- оператор CONNECT или DISCONNECT;
- оператор ALTER PROCEDURE или ALTER FUNCTION;
- оператор CREATE TRIGGER;
- команды Interactive SQL, такие как INPUT или OUTPUT.
- Хост-переменные в пакетах использовать нельзя.

Оператор CREATE PROCEDURE допустим, если он является последним оператором в пакете. Таким образом, в пакетах может содержаться только один оператор CREATE PROCEDURE.

Использование операторов SELECT в пакетах

В пакет можно включить один или более операторов SELECT.

Ниже приведен допустимый пакет:

```
IF EXISTS( SELECT *
           FROM SYSTABLE
           WHERE table_name='employee' )
THEN
    SELECTemp_lname AS LastName,
             emp_fname AS FirstName
    FROM employee;
    SELECT lname, fname
    FROM customer;
    SELECT last_name, first_name
    FROM contact;
END IF
```

Наличие псевдонима результирующего набора необходимо только в первом операторе SELECT, поскольку для описания результирующего набора сервер использует первый оператор SELECT в пакете.

Оператор RESUME должен следовать за каждым запросом для получения следующего результирующего набора.

Вызов внешних библиотек из процедур

Во внешней библиотеке можно вызвать функцию из хранимой процедуры или определяемой пользователем функции. В DLL функции можно вызывать в Windows, в NLM - в NetWare и в разделяемом объекте - в UNIX. В Windows CE вызывать внешние функции нельзя.

В данном разделе описывается выполнение в процедурах вызовов внешней библиотеки.

Внешние библиотеки, вызванные из процедур, совместно используют память сервера. Если из процедуры вызывается DLL, содержащая ошибки обработки памяти, то существует опасность выхода из строя сервера или повреждения базы данных. Перед развертыванием библиотек в рабочей базе данных убедитесь, что они тщательно проверены.

Старший API заменяется API, описанным в этом разделе. Библиотеки, записанные в API версий до 7.0, поддерживаются, но в новой разработке необходимо использовать новый API.

Adaptive Server Anywhere включает в себя набор системных процедур, использующих эту возможность, например, для отправки сообщений электронной почты с использованием MAPI.

☞ Для получения дополнительной информации о системных процедурах см. раздел "Системные процедуры и функции" (System Procedures and Functions) на стр. 643 в документе *"Справочник по SQL для ASA" (ASA SQL Reference Manual).*

Создание процедур и функций с внешними вызовами

В данном разделе представлено несколько примеров процедур и функций с внешними вызовами.

Необходимость полномочий администратора БД

Для создания процедур или функций, ссылающихся на внешние библиотеки, необходимо обладать полномочиями администратора БД. Это более строгое требование, чем необходимость наличия полномочий RESOURCE для создания других процедур или функций.

Синтаксис

Можно создать процедуру, вызывающую функцию *function_name* в DLL *library.dll,* следующим образом:

```
CREATE PROCEDURE dll_proc ( список-параметров ) EXTERNAL NAME
'function_name@library.dll'
```

Если вызвать внешнюю DLL из процедуры, то эта процедура не сможет выполнять никаких других задач; она только сформирует обертку вокруг DLL.

Аналогичный оператор CREATE FUNCTION имеет следующий вид:

```
CREATE FUNCTION dll_func ( список-параметров ) RETURNS тип-
данных EXTERNAL NAME 'function_name@library.dll'
```

В этих операторах *function_name* - экспортируемое имя функции в динамически загружаемой библиотеке, а *library.dll* - имя библиотеки. Аргументы в *списке-параметров* должны соответствовать по типу и порядку аргументу, ожидаемому в соответствии с функцией библиотеки. Функция библиотеки обращается к аргументу процедуры, используя API, описанный в разделе "Прототипы внешних функций" на стр. 552.

Любое значение, возвращенное внешней функцией, в свою очередь возвращается процедурой в среду вызова.

Какие-либо другие операторы недопустимы

Процедура, которая ссылается на внешнюю функцию, не может включать в себя другие операторы: единственной ее целью является принять аргументы функции, вызывать эту функцию и возвратить любое значение и возвращенные аргументы из функции в среду вызова. В вызове процедуры можно использовать параметры IN, OUT или INOUT так же, как и для других процедур: входные значения передаются внешней функции, и любые измененные функцией параметры возвращаются в среду вызова в параметрах OUT или INOUT.

Зависимые от системы вызовы

Вызовы, зависящие от операционной системы, можно определить так, что процедура будет вызывать одну функцию при работе в одной операционной системе, и другую функцию (рассматриваемую как аналогичная) - при работе в другой операционной системе. Синтаксис для таких вызовов включает в себя добавление имени функции к имени операционной системы.

Например:

```
CREATE PROCEDURE dll_proc ( список-параметров )
EXTERNAL NAME
'Windows95:95_fn@95_lib.dll;WindowsNT:nt_fn@nt_lib.dll'
```

Идентификатор операционной системы должен быть **WindowsNT**, **Windows95**, **UNIX** или **NetWare.**

Если в списке функций не содержится записи для операционной системы, под которой работает сервер, но в нем содержится запись без указания операционной системы, тогда сервер базы данных вызывает функцию для этой записи.

Вызовы NetWare имеют немного другой формат, нежели при других операционных системах. Все символы глобально известны под NetWare, так что любой экспортируемый символ (например, имя функции) должен быть уникальным для всех NLM в системе. Следовательно, присутствие имени NLM в вызове не является необходимым, и вызов имеет следующий синтаксис:

```
CREATE PROCEDURE dll_proc ( список-параметров )
EXTERNAL NAME 'NetWare:nw_fn'
```

Указывать имя библиотеки нет необходимости. Если оно указано, то система его игнорирует.

☞ Для получения дополнительной информации о синтаксисе оператора CREATE PROCEDURE см. раздел "Оператор CREATE PROCEDURE" (CREATE PROCEDURE statement) на стр. 284 в документе *"Справочник по SQL для ASA" (ASA SQL Reference Manual)*.

☞ Для получения дополнительной информации о синтаксисе оператора CREATE FUNCTION см. раздел "Оператор CREATE FUNCTION" (CREATE FUNCTION statement) на стр. 277 в документе *" Справочник по SQL для ASA" (ASA SQL Reference Manual)*.

Прототипы внешних функций

В данном разделе описывается API для функций во внешних библиотеках.

API определяется заголовочным файлом с именем *extfnapi.h* в подпапке *h* папки установки SQL Anywhere Studio. Этот заголовочный файл управляет возможностями прототипов внешних функций, зависимыми от платформы.

Объявление версии API

С целью уведомления сервера базы данных о том, что библиотека написана не с использованием старой версии API, функцию необходимо представить следующим образом:

uint32 extfn_use_new_api( )

Функция возвращает беззнаковое 32-разрядное целое значение. Если возвращаемое значение отлично от нуля, то сервер базы данных предполагает, что старая версия API не используется.

Если функция не экспортируется DLL, сервер базы данных предполагает, что используется старая версия API. При использовании новой версии API возвращенное значение должно быть номером версии API, определенным в *extfnapi.h*.

Прототипы функции

Имя функции должно совпадать с именем ссылки в операторе CREATE PROCEDURE или CREATE FUNCTION. Объявление функции должно быть следующим:

void *имя-функции* **(an_extfn_api ****api,* **void *** *дескриптор-аргументов* **)**

Функция должна возвратиться пустой, принять в качестве аргументов структуру, использованную для передачи аргументов, и дескриптор к аргументам, предоставленный процедурой SQL.

Структура **an_extfn_api** имеет следующую форму:

```
typedef struct an_extfn_api {
    short (SQL_CALLBACK *get_value)(
             void * arg_handle,
             a_SQL_uint32 arg_num,
             an_extfn_value *value
             );
    short (SQL_CALLBACK *get_piece)(
             void * arg_handle,
             a_SQL_uint32 arg_num,
             an_extfn_value *value,
             a_SQL_uint32 offset
             );
    short (SQL_CALLBACK *set_value)(
             void * arg_handle,
             a_SQL_uint32 arg_num,
             an_extfn_value *value
             short append
             );
    void (SQL_CALLBACK *set_cancel)(
             void * arg_handle,
             void * cancel_handle
             );
    } an_extfn_api;
```

Структура **an_extfn_value** имеет следующую форму:

```
typedef struct an_extfn_value {
    void * data;
    a_SQL_uint32 piece_len;
    union {
            a_SQL_uint32 total_len;
            a_SQL_uint32 remain_len;
    } len;
    a_SQL_data_type type;
} an_extfn_value;
```

Примечания

Вызов **get_value** для параметра OUT возвращает тип данных аргумента, а также возвращает данные как NULL.

Функцию **get_piece** для любого данного аргумента можно вызвать только сразу после функции **get_value** для того же самого аргумента,

Для возврата NULL в **an_extfn_value** установите **data** на NULL.

Поле **append** в **set_value** определяет, замещаются представленные данные (FALSE) или добавляются (TRUE) к существующим данным. Необходимо вызвать **set_value** с `append=FALSE` до его вызова с `append=TRUE` для того же

самого аргумента. Для типов данных фиксированной длины поле **append** игнорируется.

Сам заголовочный файл содержит некоторые дополнительные примечания.

Следующая таблица показывает условия, при которых функции, определенные в **an_extfn_api**, возвращают значение FALSE:

Функция	Возвращает 0, когда следующее истинно; либо возвращает 1
get_value()	- **arg_num** недопустим; например, **arg_num** больше, чем число аргументов в **ext_fn**. - Вызывается до необходимой инициализации вызова внешней функции.
get_piece()	- **arg_num** недопустим; например, **arg_num** не соответствует последнему **get_value**. | - Смещение превышает полную длину значения для аргумента arg_num. - Вызывается до необходимой инициализации вызова внешней функции.
set_value()	- **arg_num** недопустим; например, **arg_num** больше, чем число аргументов в **ext_fn**. - Аргумент **arg_num** - только ввод. - Тип предоставленного значения не соответствует типу аргумента **arg_num.** - Вызывается до необходимой инициализации вызова внешней функции.

☞ Для получения дополнительной информации о значениях, которые можно ввести в поле a_sql_data_type, см. раздел "Типы данных Embedded SQL" (Embedded SQL data types) на стр. 173 в документе *"Руководство по программированию ASA" (ASA Programming Guide)*.

☞ Для получения дополнительной информации о передаче параметров внешним функциям см. раздел "Передача параметров внешним функциям" на стр. 555.

Реализация обработки отмены

Внешняя функция, которая предположительно будет отменена, должна проинформировать об этом сервер базы данных вызовом функции API **set_cancel.** Для активизации отмены внешних операций необходимо экспортировать специальную функцию. Эта функция должна иметь следующую форму:

void an_extfn_cancel(void * *cancel_handle* **)**

Если DLL не экспортирует эту функцию, сервер базы данных игнорирует любые пользовательские прерывания для функций в DLL. В этой функции *cancel_handle* - указатель, предоставленный отменяемой функцией серверу базы данных при каждом вызове внешней функции при помощи функции API *set_cancel,* приведенной в структуре **an_extfn_api** выше.

Передача параметров внешним функциям

Типы данных

Внешней библиотеке можно передать следующие типы данных SQL:

Тип данных SQL	Тип C
CHAR	Символьные данные с заданной длиной
VARCHAR	Символьные данные с заданной длиной
LONG VARCHAR	Символьные данные с заданной длиной
BINARY	Символьные данные с заданной длиной
LONG BINARY	Символьные данные с заданной длиной
TINYINT	1-байтовое целое
[UNSIGNED] SMALLINT	2-байтовое целое [беззнаковое]
[UNSIGNED] INT	4-байтовое целое [беззнаковое]
[UNSIGNED] BIGINT	8-байтовое целое [беззнаковое]
VARBINARY	Двоичные данные с заданной длиной
REAL	Число с плавающей точкой одинарной точности
DOUBLE	Число с плавающей точкой двойной точности

Нельзя использовать типы данных даты или времени, а также нельзя точные числовые типы данных.

Для предоставления значений для параметров INOUT или OUT используйте функцию API **set_value.** Для прочтения параметров IN и INOUT используйте функцию API **get_value.**

Передача NULL

NULL можно передать в качестве допустимого значения для всех аргументов. Функции во внешних библиотеках могут предоставить NULL как возвращаемый тип для любого типа данных.

Возвращаемые типы внешней функции

В следующей таблице перечислены поддерживаемые возвращаемые типы и их соответствие возвращаемому типу функции SQL или процедуры.

Тип данных C	Тип данных SQL
void	Используется для внешних процедур.
char *	Возвращаемый тип функции CHAR()
long	Возвращаемый тип функции INTEGER
float	Возвращаемый тип функции FLOAT
double	Возвращаемый тип функции DOUBLE.

Если функция во внешней библиотеке возвращает NULL, и было объявлено, что внешняя функция SQL должна возвратить CHAR(), то возвращаемое значение расширенной функции SQL – NULL.

ГЛАВА 18

Отладка логики базы данных

Об этой главе

В данной главе описывается использование отладчика Sybase для помощи в разработке Java-классов, хранимых процедур SQL, триггеров и обработчиков событий.

Содержание

Введение в отладку базы данных

При наличии в базе данных Java в базу данных можно добавить сложные классы. Другим способом добавления логики в базу данных является использование хранимых процедур и триггеров SQL. Отладчик позволяет проверить эти классы и процедуры, выявить и устранить встречающиеся в них неполадки.

В данной главе описан процесс установки и использования отладчика.

Возможности отладчика

Отладчик Sybase позволяет выполнять различные задачи, в том числе:

- **Отладка Java-классов.** Можно выполнить отладку Java-классов, хранящихся в базе данных.
- **Отладка процедур и триггеров.** Можно выполнить отладку хранимых процедуры и триггеров SQL.
- **Отладка обработчиков событий.** Обработчики событий - это расширение хранимых процедур SQL. Материал данной главы об отладке хранимых процедур в равной степени применим к отладке обработчиков событий.
- **Просмотр классов и хранимых процедур.** Можно просмотреть исходный код установленных классов и процедур SQL.
- **Выполнение трассировки.** Можно построчно проверить код Java-класса или хранимой процедуры, запущенных в базе данных. Пользователь имеет возможность просмотра списка всех вызванных функций.
- **Установка контрольных точек.** Можно выполнить код до определенной контрольной точки и затем остановить его выполнение в этой точке.
- **Установка условий прерывания.** Контрольные точки включают в себя строки кода, но можно задать условия, при которых выполнение кода должно завершиться. Например, можно остановиться на строке на десятый раз ее выполнения, либо только на определенном значении переменной. Остановиться также можно всякий раз, когда в приложение Java сбрасывается определенное исключение.
- **Просмотр и изменение локальных переменных.** Когда выполнение приостановлено в контрольной точке, можно просмотреть значения локальных переменных и изменить их значение.
- **Просмотр и прерывание на выражениях.** Когда выполнение приостановлено в контрольной точке, можно просмотреть значения большого количества выражений.
- **Просмотр и изменение строковых переменных.** Строковые переменные являются значениями OLD и NEW триггеров уровня строки. Эти значения можно проверить и установить.
- **Выполнение запросов.** Когда выполнение приостановлено в контрольной точке в пределах процедуры SQL, можно выполнить запросы. Это позволяет просмотреть промежуточные результаты,

хранящиеся во временных таблицах, а также проверочные значения в базовых таблицах.

Требования по использованию отладчика

Отладчик запускается на клиентской машине и подключается к базе данных, используя драйвер Sybase jConnect JDBC.

Для использования отладчика потребуется следующее:

- **Java-in-the-database**. Теперь это отдельное лицензируемое средство. Для получения дополнительной информации см. раздел "Введение в Java в базе данных" (Introduction to Java in the Database) на стр. 49 в документе *"Руководство по программированию ASA" (ASA Programming Guide)*.
- **Полномочия.** Для использования отладчика необходимо обладать полномочиями администратора БД или полномочиями для работы с группой SA_DEBUG. Эта группа добавляется ко всем базам данных при их создании.
- **Исходный код.** Отладчику должен быть доступен исходный код используемого приложения. Для Java-классов исходный код содержится в папке на жестком диске. Для хранимых процедур исходный код содержится в базе данных.
- **Параметры компиляции**. Для выполнения отладки Java-классов они должны быть откомпилированы так, чтобы содержать информацию отладки. Например, при использовании компилятора JDK *javac.exe* от Sun Microsystems Java-классы должны быть откомпилированы с использованием параметра `-g` командной строки.
- **Поддержка jConnect**. База данных, к которой необходимо подключиться, должна иметь установленную поддержку jConnect.

Информация об отладчике

В этой главе представлены три учебных раздела, предназначенных для подготовки к использованию отладчика. В отладчике доступна интерактивная справка, основанная на выборе задач и содержащая информацию о каждом окне.

Учебный раздел по началу работы с отладчиком

В данном учебном разделе описана процедура запуска отладчика, его подключения к базе данных, выполнения отладки простейшей хранимой процедуры и Java-класса.

Урок 1: Подключение к базе данных

В данном учебном разделе описан процесс запуска отладчика, подключения к базе данных и начало использования подключения для отладки.

Запуск отладчика

Отладчик запускается на клиентской машине.

Отладчик можно запустить из командной строки или в Sybase Central.

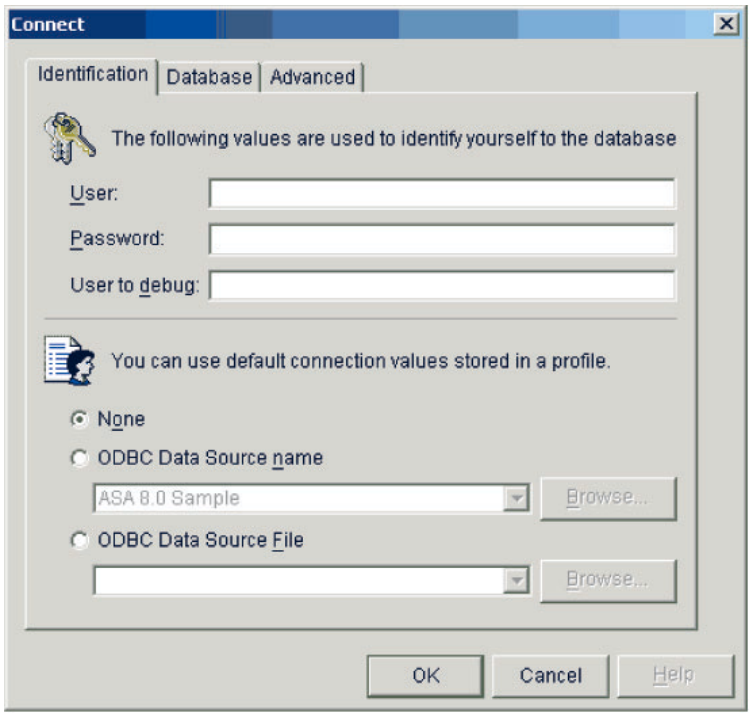
❖ **Запуск отладчика**

1. Для выполнения действий, описанных в этом учебном разделе, закройте демонстрационную базу данных и все приложения Adaptive Server Anywhere.
2. В Sybase Central выберите Tools➤Adaptive Server Anywhere 8➤Open Database Object Debugger.

   или

   В системной командной строке перейдите в папку, содержащую программное обеспечение Adaptive Server Anywhere для данной операционной системы, и введите **dbprdbg**.

Появляется окно Connect:

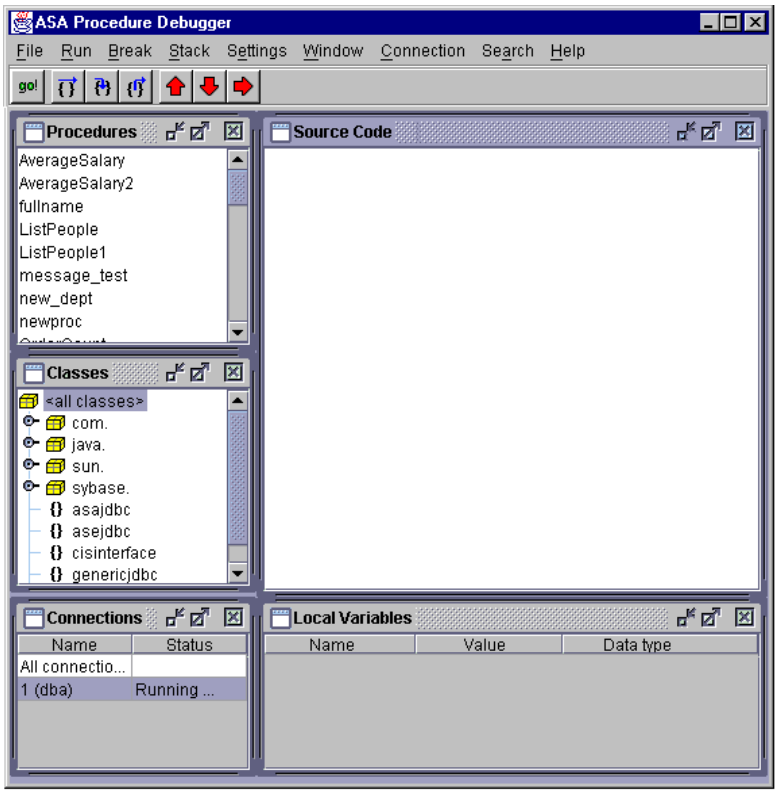


Теперь можно выполнить подключение к базе данных из отладчика.

3 В окне Login отладчика введите следующую информацию:

- **User (Пользователь)** - введите код пользователя **DBA**.
- **Password (Пароль)** - введите пароль **SQL**.
- **User to debug (Отладка для пользователей)** - введите “звездочку” (*) для указания на необходимость выполнения отладки подключений всех пользователей.
- **Host (Хост)** - оставьте хост **localhost**.
- **Port (Порт)** - оставьте порт **2638**.

4 Нажмите OK для выполнения подключения к базе данных. Появляется интерфейс отладчика, отображающий список хранимых процедур и триггеров, а также список Java-классов, установленных в базе данных.



Для получения дополнительной информации о допустимых параметрах подключения для отладчика см. раздел "Поставка URL для сервера" (Supplying a URL for the server) на стр. 137 в документе "*Руководство по программированию ASA*" (*ASA Programming Guide*).

Урок 2: Отладка хранимой процедуры

В этом учебном разделе описан типовой сеанс отладки хранимой процедуры. Он является продолжением раздела "Урок 1: Подключение к базе данных" на стр. 560.

В этом учебном разделе вызывается хранимая процедура *sp_customer_products*, являющаяся частью демонстрационной базы данных.

Процедура *sp_customer_products* выполняет следующий запрос к демонстрационной базе данных:

```
CREATE PROCEDURE DBA.sp_customer_products(
    INOUT customer_id INTEGER )
RESULT(id integer,quantity_ordered integer)
BEGIN
  SELECT product.id,sum(sales_order_items.quantity)
  FROM product,sales_order_items,sales_order
  WHERE sales_order.cust_id = customer_id
  AND sales_order.id = sales_order_items.id
  AND sales_order_items.prod_id = product.id
  GROUP BY product.id
END
```

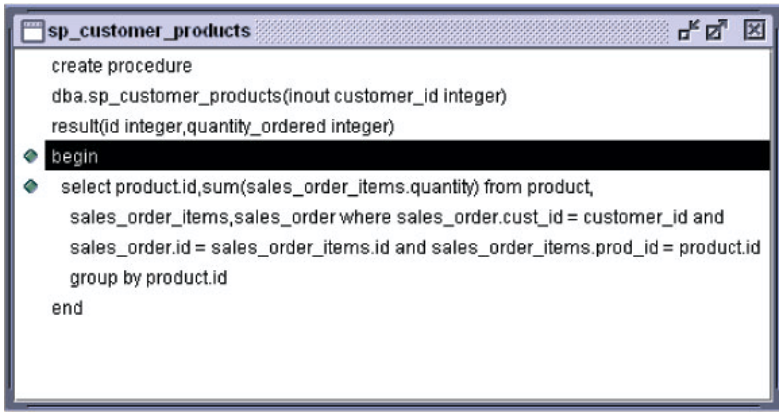
В качестве ввода берется код определенного клиента, в результирующем наборе возвращается список кодов товаров и количеств, заказанных данным клиентом.

Отображение исходного кода хранимой процедуры в отладчике

Исходный код хранимой процедуры сохранен в базе данных. Исходный код хранимой процедуры можно отобразить в окне исходного кода (Source Code).

❖ **Отображение исходного кода хранимой процедуры в отладчике**

- В интерфейсе отладчика дважды щелкните на хранимой процедуре *sp_customer_products*. Исходный код для процедуры появляется в окне Source Code.



Установка контрольной точки

В теле процедуры *sp_customer_products* можно задать контрольную точку. При выполнении процедуры выполнение прерывается в данной контрольной точке.

❖ **Установка контрольной точки в хранимой процедуре**

1 В окне Source Code определите местонахождение строки с помощью запроса:

```
select product.id,…
```

2 Нажмите на зеленый ромб слева от строки, чтобы его цвет изменился на красный. Повторные нажатия на индикатор переключают его состояние.

Запуск процедуры

Хранимую процедуру можно вызвать из Interactive SQL и затем проверить, как происходит прерывание ее выполнения в контрольной точке.

❖ **Вызов процедуры в Interactive SQL**

1 Запустите Interactive SQL. При необходимости выполните подключение к демонстрационной базе данных с использованием кода пользователя **DBA** и пароля **SQL**.

Выполненное подключение появляется в списке в окне Connections.

2 Для вызова процедуры с использованием кода клиента 122 выполните следующую команду в Interactive SQL:

```
CALL sp_customer_products( 122 )
```

Запрос не завершен. Вместо этого выполнение прервано в контрольной точке в отладчике. В Interactive SQL кнопка "Interrupt the SQL Statement" (Прервать оператор SQL) активна. В окне Source Code отладчика желтая стрелка указывает текущую строку.

3 Перейдите к следующей строке, выбрав Run➤Step Over. Также можно нажать F7.

✍ Для более длинных процедур можно использовать другие способы перемещения по коду. Дополнительные примеры см. в разделе "Урок 3: Отладка Java-класса" на стр. 565.

Для следующего урока оставьте в отладчике состояние прерывания на строке SELECT.

Просмотр и изменение переменных

В отладчике можно просмотреть значения переменных.

Просмотр локальных переменных

Для лучшего понимания ситуации просмотреть значения локальных переменных можно в процедуре в процессе прохождения переменных исходного кода.

❖ **Просмотр и изменение значения переменной**

1 Если окно Local Variables (Локальные переменные) не открыто, выберите Window➤Local Variables для его отображения.

   В окне Local Variables показаны две локальные переменные: сама хранимая процедура (не имеющая возвращаемого значения, и поэтому приведенная как NULL) и **customer_id** (код клиента), переданный процедуре.

2 В окне Local Variables дважды щелкните на элементе столбца Value (значение) для **customer_id** и введите "125", чтобы изменить значение кода клиента, используемое в запросе.

3 В окне Source Code нажмите F5 для завершения выполнения запроса и окончания данного учебного раздела.

На закладке Results в окне Results в Interactive SQL отображается список кодов товаров для клиента 125.

Просмотр строчных переменных триггера

Помимо локальных переменных можно отобразить другие переменные, например, значения OLD и NEW триггера уровня строки в окне отладчика Row Variables.

Урок 3: Отладка Java-класса

В этом учебном разделе описан типовой сеанс отладки хранимой процедуры. Он является продолжением раздела "Урок 1: Подключение к базе данных" на стр. 560.

В этом учебном разделе из Interactive SQL вызывается метод **JDBCExamples.Query(),** прерывается выполнение в отладчике, и в исходном коде осуществляется поиск данного метода.

Метод **JDBCExamples.Query()** выполняет следующий запрос к демонстрационной базе данных:

```
SELECT id, unit_price FROM product
```

Затем он осуществляет циклический просмотр всех строк результирующего набора и возвращает одну с самой высокой ценой за единицу товара.

Компилирование Java-классов для отладки

Для выполнения отладки классов их необходимо откомпилировать с использованием параметра *javac -g*. Типовые классы откомпилированы для отладки.

Подготовка базы данных

При выполнении примеров с Java, таких как "Урок 3: Отладка Java-класса" на стр. 565, необходимо установить примеры Java-классов в демонстрационную базу данных.

☞ Для получения дополнительной информации об установке примеров Java см. раздел "Установка примеров Java" (Setting up Java examples) на стр. 86 в документе *"Руководство по программированию ASA" (ASA Programming Guide)*.

☞ Для получения дополнительной информации о классе **JDBCExamples** и его методах см. раздел "Доступ к данным с использованием JDBC" (Data Access Using JDBC) на стр. 129 в документе *"Руководство по программированию ASA" (ASA Programming Guide)*.

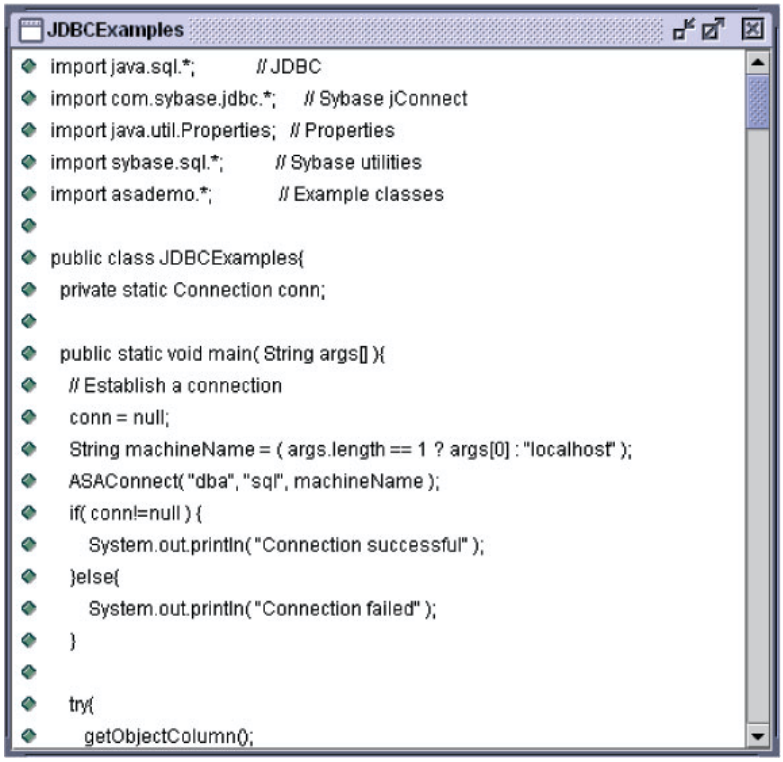
Отображение исходного кода Java в отладчике

В наборе местоположений отладчик производит поиск файлов исходного кода (с расширением *.java*). В папке установки в список местоположений нужно добавить подпапку *jxmp* для того, чтобы выполняемый в данный момент в базе данных код был доступен отладчику.

❖ **Отображение исходного кода Java в отладчике**

1. В интерфейсе отладчика выберите File➤Edit Source Path. Появляется окно Source Path.
2. Введите путь к папке *jxmp* в папке установки Adaptive Server Anywhere. Например, если Adaptive Server Anywhere установлен в *c:\asa8*, то ввести пришлось бы следующее:

   ```
   c:\asa8\jxmp
   ```
3. Нажмите Apply и закройте окно.
4. В окне Classes дважды щелкните **JDBCExamples**. Исходный код для класса **JDBCExamples** появляется в окне Source Code.



Примечания относительно расположения исходного кода Java

Окно Source Path содержит список папок, в которых отладчик производит поиск исходного кода Java. Применяются правила Java для поиска пакетов. Отладчик также осуществляет поиск текущего CLASSPATH для исходного кода.

Например, если к исходному пути добавляются пути *c:\asa8\jxmp* и *c:\Java\src,* и отладчик должен найти класс, названный *asademo.Product*, он ищет исходный код в *c:\asa8\jxmp\asademo\Product.Java* и *c:\Java\src\my\asademo\Product.Java*

Установка контрольной точки

Задать контрольную точку можно в начале метода Query(). При вызове этого метода выполнение прерывается в данной контрольной точке.

❖ **Установка контрольной точки в Java-классе**

1 В окне Source Code выполните прокрутку вниз до начала метода **Query()**. Этот метод располагается ближе к концу класса и начинается со следующей строки:

```
public static int Query() {
```

2 Нажмите на зеленый индикатор слева от первой строки метода, чтобы его цвет изменился на красный. Первая строка метода:

```
int max_price = 0;
```

Повторные нажатия на индикатор переключают его состояние.

Запуск метода

Метод **Query()** можно вызвать из Interactive SQL и затем проверить, как происходит прерывание его выполнения в контрольной точке.

❖ **Вызов метода в Interactive SQL**

1 Запустите Interactive SQL. При необходимости выполните подключение к демонстрационной базе данных с использованием кода пользователя **DBA** и пароля **SQL**.

Выполненное подключение появляется в списке в окне Connections.

2 Для вызова метода введите следующую команду в Interactive SQL:

```
SELECT JDBCExamples.Query()
```

Запрос не завершен. Вместо этого выполнение прервано в контрольной точке в отладчике. В Interactive SQL кнопка Stop активна. В окне Source Code отладчика красная стрелка указывает текущую строку.

Теперь можно последовательно проверить таким образом исходный код и выполнить операцию отладки в отладчике.

Последовательная проверка исходного кода

В данном разделе иллюстрируются некоторые способы последовательной проверки исходного кода в отладчике.

В соответствии с предыдущим разделом отладчик должен был остановить выполнение **JDBCExamples.Query()** на первом операторе метода.

Примеры

Ниже приведены некоторые шаги, которые можно выполнить:

1 **Переход к следующей строке**. Для перехода к следующей строке текущего метода выберите Run➤Step Over или нажмите F7. Пробуйте выполнить это два или три раза.

2 **Переход к выбранной строке**. Выберите следующую строку при помощи мыши и выберите элемент Run➤Run To Selected или нажмите F6 для перехода на эту строку и прерывания выполнения:

```
max_price = price;
```

Красная стрелка перемещается на эту строку.

3 **Установка контрольной точки и выполнение кода до нее**. Выберите следующую строку (строка 292) и нажмите F9 для задания контрольной точки на этой строке:

```
return max_price;
```

В левом столбце появляется “звездочка”, отмечающая контрольную точку. Нажмите F5 для выполнения до этой контрольной точки.

4 **Выполнение различных действий**. Попробуйте применить разные способы проверки кода. Завершите выполнение нажатием F5.

По завершении выполнения в окне Results в Interactive SQL на закладке Results отображается значение 24.

Параметры

Полный набор параметров для проверки исходного кода содержится в меню ***Run***. Более подробная информация представлена в интерактивной справке отладчика.

Просмотр и изменение переменных

В отладчике можно просмотреть значения как локальных переменных (объявленных в методе), так и статических переменных класса.

Просмотр локальных переменных

Для лучшего понимания ситуации просмотреть значения локальных переменных можно в процессе проверки исходного кода. Для этого класс должен был быть откомпилированным с использованием параметра *javac* `-g`.

❖ **Просмотр и изменение значения переменной**

1 Установите контрольную точку на первой строке метода **JDBCExamples.Query.** Эта строка выглядит следующим образом:

```
int max_price = 0
```

2 Для выполнения метода еще раз введите следующий оператор в Interactive SQL:

```
SELECT JDBCExamples.Query()
```

Запрос выполняется только до контрольной точки.

3 Для перехода на следующую строку нажмите F7. Теперь была объявлена и инициализирована к нулю переменная **max_price.**

4 Если окно Local Variables не открыто, выберите Window➤Local Variables.

В окне Local Variables показаны несколько локальных переменных. Переменная **max_price** имеет нулевое значение. Все остальные переменные перечислены с указанием *variable not in scope,* что означает, что они еще не инициализированы.

5 В окне Local Variables дважды щелкните на записи в столбце Value для **max_price** и введите 45 для изменения значения **max_price** на 45.

   Значение 45 превышает любую другую цену. Вместо возврата 24 теперь в качестве максимальной цены запрос возвратит 45.

6 Для проверки кода несколько раз нажмите F7 в окне Source Code. По мере выполнения проверки значения переменных будут появляться в окне Local Variables. Продолжайте проверку, пока у переменных **stmt** и **result** не появятся значения.

7 Разверните объект **result** нажатием на значок рядом с ним, либо переставьте курсор на строку и нажмите ENTER. Таким образом отображаются значения полей в объекте.

8 По окончании выполнения различных действий по проверке и изменению переменных нажмите F5 для завершения выполнения запроса и окончания данного учебного раздела.

Просмотр статических переменных

Помимо локальных переменных можно отобразить переменные уровня класса (статические переменные) в окне Static отладчика и просмотреть их значения в окне Inspection. Дополнительная информация представлена в интерактивной справке отладчика.

Общие задачи отладчика

При необходимости выполнить...	Выберите...
Добавление папки в путь поиска исходного файла Java	File➤Add Source Path
Отображение контекста вызывающей программы	Stack➤Down
Отображение контекста вызывающей программы	Stack➤Up
Активизация фиксации подключений отладчиком	Connection➤Enable capture
Выход из отладчика	File➤Exit
Нахождение строки в выбранном окне	Search➤Find
Игнорирование регистра слова при поиске	Search➤Ignore Case
Загрузка параметров настройки отладчика процедур из файла	Settings➤Load From File
Регистрация в базе данных нового подключения	Connection➤Login
Выход из базы данных	Connection➤Logout
Перезапуск выполнения программы	Run➤Restart
Запуск сценария отладки	File➤Run Script Для получения дополнительной информации см. раздел "Написание сценариев отладчика" на стр. 583.
Запуск программы построчно, с построчной проверкой процедур и триггеров	Run➤Step Into
Запуск программу построчно, с переключением при необходимости между программой Java и хранимыми процедурами в других средах	Run➤Step Through

При необходимости выполнить...	Выберите...
Запуск программы построчно, без построчной проверки процедур и триггеров. Если какая-либо процедура или триггер содержат контрольную точку, то контрольная точка будет игнорироваться.	Run➤Step Over
Запуск программы или возобновление выполнения программы с контрольной точки	Run➤Go
Запуск программы до возвращения текущей процедуры/метода	Run➤Step Out
Сохранение контрольных точек процедур в пределах процедур	Settings➤Remember Breakpoints
Сохранение параметров настройки отладчика процедур	Settings➤Save
Сохранение параметров настройки отладчика процедур в файле	Settings➤Save to File
Сохранение расположения окон отладчика процедур, шрифтов и т.д. автоматически при выходе из отладчика	Settings➤Save on Exit
Сохранение расположения окон отладчика процедур, шрифтов и т.д. с параметрами настройки	Settings➤Remember Windows Атрибуты
Задание пути к исходному файлу Java	File➤Edit Source Path
Просмотр исходного кода для процедуры или класса	File➤Open

Запуск отладчика

Отладчик осуществляет мониторинг хранимых процедур и Java-классов, выполняемых другими подключениями к той же базе данных.

Отладчик можно запустить из командной строки или в Sybase Central.

❖ **Запуск отладчика (Sybase Central)**

1. Запустите Sybase Central.
2. Откройте папку Utilities дополнения к Adaptive Server Anywhere.
3. Дважды щелкните на Debug Database Objects в правой области окна.

❖ **Запуск отладчика (командная строка)**

1. В папке установки Adaptive Server Anywhere перейдите в подпапку, содержащую исполняемый файл сервера базы данных для используемой операционной системы.
2. Выполните следующую команду: `dbprdbg`

   Если была задана переменная среды SQLCONNECT, то отладчик использует ее для выполнения подключения к базе данных.

Подключение к базе данных

Отладчик выполняет подключение к базе данных так же, как любое клиентское приложение. Процедуры и Java-классы, отладку которых он выполняет, выполняются не из самого отладчика, а из других подключений.

❖ **Подключение к базе данных**

1. Запустите базу данных, отладку которой необходимо выполнить.
2. Запустите отладчик. Если не задана переменная среды SQLCONNECT или выбрано Connection➤Login, то при запуске отладчик отображает диалог входа в систему.

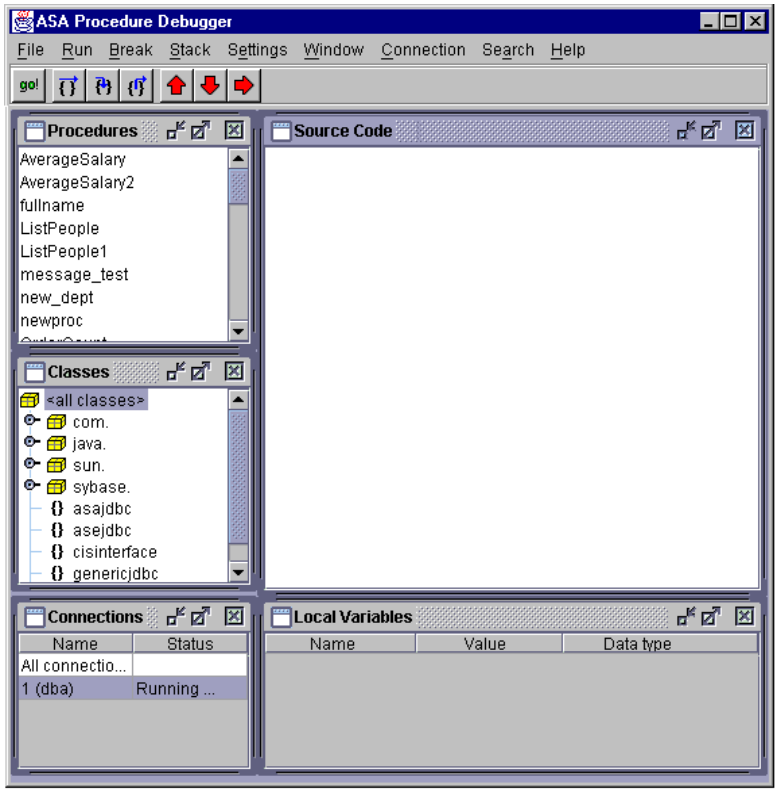
   ☞ Для получения дополнительной информации см. раздел "Запуск отладчика" на стр. 573.
3. В диалоге входа в систему введите следующую информацию:

♦ **Информация подключения к базе данных.** Обычная информация для подключения, как при подключении из Interactive SQL или Sybase Central.

Для получения дополнительной информации см. раздел "Подключение из Sybase Central или Interactive SQL" (Connecting from Sybase Central or Interactive SQL) на стр. 42 в документе *"Руководство по администрированию баз данных ASA" (ASA Database Administration Guide)*.

- **Отладка для пользователя.** Вводится код пользователя для подключения, отладку которого необходимо выполнить. Для выполнения отладки всех подключений к базе данных поле можно оставить незаполненным или ввести "звездочку" (*).

4 Нажмите OK для выполнения подключения. Появляется интерфейс отладчика, отображающий список хранимых процедур и триггеров, а также список Java-классов, установленных в базе данных.



Выбор подключения для отладки

В автоматически обновляемом окне Connection отображены все подключения к базе данных. Начать отладку подключения можно следующими способами.

- **Прикрепление клиентского приложения к базе данных**. При прикреплении клиента его подключение появляется в окне Connection. Выбрав Connection➤Auto Attach, отладчику можно задать функцию автоматического прикрепления к подключениям по мере их появления. Теперь в этом подключении можно задать контрольную точку или выбрать Run➤Interrupt (или нажать CTRL-C) для прерывания выполнения при следующем запуске любого оператора хранимой процедуры. Теперь можно предпринять любое необходимое действие для того, чтобы клиент вызвал эту хранимую процедуру.

  ☞ Для получения дополнительной информации см. раздел "Проверка переменных" на стр. 582.

- **Предварительная установка контрольной точки.** Иногда между временем подключения клиента и выполнением процедуры нет паузы. В этом случае контрольную точку следует установить заранее. В окне Connection выберите **All connections** (Все подключения). Установите контрольную точку в хранимой процедуре, отладку которой необходимо выполнить. Эта контрольная точка применяется ко всем текущим и будущим подключениям до момента выхода из базы данных.

Установка активного подключения

❖ **Установка активного подключения**

1 Откройте окно Connection.

2 Дважды щелкните на подключении, которое необходимо сделать активным.

Специальный параметр **All connections** (Все подключения) рассматривается как отдельное подключение.

Требования по использованию отладчика

Для использования отладчика необходимо наличие следующего:

- **Полномочия**. Для использования отладчика необходимо обладать полномочиями администратора БД или полномочиями группы SA_DEBUG. Эта группа добавляется ко всем базам данных при их создании.

- **Исходный код**. Отладчику должен быть доступен исходный код используемого приложения. Для Java-классов исходный код хранится в папке на жестком диске. Для хранимых процедур исходный код хранится в базе данных и становится доступным автоматически.

- **Параметры компиляции**. Для выполнения отладки Java-классов они должны быть откомпилированы так, чтобы содержать информацию отладки. Например, при использовании компилятора JDK *javac.exe* от Sun Microsystems Java-классы должны быть откомпилированы с использованием параметра `-g` командной строки.

Настройка отладчика

Отладчик процедур Adaptive Server Anywhere можно настроить с целью упростить его использование. В меню Settings можно задать способ отображения окон, определить способ использования контрольных точек, а также отображения символов.

Сохранение расположения окон

В отладчике процедур можно сохранять расположение окон.

❖ **Сохранение расположения окон**

1. Выберите Settings➤Remember windows positions.
2. Выберите Settings➤Save.

Сохранение контрольных точек

Отладчик процедур позволяет сохранять контрольные точки для активного подключения или подключений.

❖ **Сохранение контрольных точек для подключения**

1. В окне Connections выберите подключение, для которого необходимо сохранить контрольные точки для их использования отладчиком.
2. Выберите Settings➤Remember breakpoints.
3. Выберите Settings➤Save.

Таким образом сохраняются контрольные точки для выбранного подключения; в самих процедурах контрольные точки не сохраняются. Для сохранения контрольных точек для всех текущих подключений выберите **All Connections** в окне Connections.

Использование полных имен переменных при развертывании объектов и массивов

По умолчанию отладчик процедур отображает краткую форму имен переменных при развертывании объектов и массивов. Эту настройку можно изменить следующим образом:

❖ **Использование длинных имен переменных для объектов и массивов**

- Выберите Settings➤Use long variable names.

Изменение формата отображения значений переменных

Отладчик процедур позволяет задавать формат отображения значений переменных.

❖ **Изменение формата значений переменных**

- В меню Settings выберите формат, в котором должны отображаться переменные.

Доступны следующие форматы:

- Display 'char' in hex (отображение 'char' в шестнадцатеричной форме);
- Display 'char' as int (отображение 'char' как целое);
- Display 'byte' in hex (отображение 'byte' в шестнадцатеричной форме);
- Display 'short' in hex (отображение 'short' в шестнадцатеричной форме);
- Display 'int' in hex (отображение 'int' в шестнадцатеричной форме);
- Display 'long' in hex (отображение 'long' в шестнадцатеричной форме).

Активизация клавиш VI в окнах с текстом

По умолчанию клавиши в окнах редактирования текста, например, в окне Source Code, имеют те же функции, что и в Notepad. Отладчик позволяет выбрать возможность редактирования текста в этих окнах способом, похожим на редактирование текста в VI.

❖ **Активизирование некоторых клавиш VI в текстовых окнах**

- Выберите Settings➤VI.

После проверки настройки VI активизируются следующие клавиши VI:

- / поиск
- **n** найти далее
- **N** найти ранее
- **j** курсор вверх
- **k** курсор вниз
- **CTRL-b** страница вверх
- **CTRL-f** страница вниз

Автоматическое сохранение всех изменений настроек при выходе из отладчика

По умолчанию при выходе из отладчика все изменения настроек, внесенные после последнего явного сохранения, сбрасываются.

❖ **Сохранение настроек при выходе из отладчика**

♦ Выберите Settings➤Save on exit.

Параметры настройки сохранены в файле *ProcDebugDefaults.rc* в папке, содержащей отладчик процедур. При следующем открытии отладчика активными параметрами настройки будут сохраненные в файле *ProcDebugDefaults.rc*.

Сохранение параметров настройки в файле

Отладчик процедур позволяет сохранять параметры настройки в файле *ProcDebug.rc*.

❖ **Сохранение параметров настройки в файле**

♦ Выберите Settings➤Save to file.

Загрузка сохраненных параметров настройки из файла

Сохраненные параметры настройки можно получить из файла *ProcDebug.rc* следующим образом:

❖ **Загрузка параметров настройки из файла**

♦ Выберите Settings➤Load from file.

Работа с контрольными точками

В данном разделе описывается использование контрольных точек в случаях, когда отладчик прерывает выполнение исходного кода.

Установка контрольных точек

Контрольная точка дает отладчику команду прервать выполнение активного подключения на указанной строке.

Устанавливаемая контрольная точка применяется только к активному подключению.

☞ Для получения дополнительной информации см. раздел "Установка активного подключения" на стр. 575.

❖ **Установка контрольной точки (окно Source code)**

1 Активизируйте подключение, отладку которого необходимо выполнить.

2 Откройте окно Source code и отобразите код, в котором необходимо задать контрольную точку.

❖ **Установка контрольной точки (меню Break)**

1 Активизируйте подключение, отладку которого необходимо выполнить.

2 Выберите Break➤New. Введите адрес контрольной точки в диалог. Для хранимой процедуры введите адрес формы *procedure:line*.

Для Java-класса введите адрес формы *class:line* или *class.method*.

Отключение и включение контрольных точек

Состояние контрольной точки можно изменить в окне Source code или с помощью меню Break.

❖ **Изменение состояния контрольной точки (окно Source Code)**

1 Активизируйте подключение, отладку которого необходимо выполнить.

2 Откройте окно Source Code и отобразите процедуру, содержащую контрольную точку, состояние которой необходимо изменить.

3 Нажмите индикатор контрольной точки слева от строки, которую необходимо отредактировать. Состояние строки переключается только с контрольной точки на активное подключение, на состояние контрольной точки для всех подключений, на отключенную контрольную точку и на удаление контрольной точки.

❖ **Изменение состояния контрольной точки (окно Breakpoints)**

1 Выделите подключение, отладку которого необходимо выполнить, чтобы сделать его активным.

2 Откройте окно Breakpoints.

3 Нажмите на индикатор контрольной точки слева. Состояние контрольной точки изменяется с активного на неактивное.

В качестве альтернативы, контрольную точку можно вставить нажатием INSERT и заданием адреса контрольной точки. Контрольную точку можно удалить ее выбором и нажатием DELETE.

❖ **Изменение состояния контрольных точек (меню Break)**

1 Активизируйте подключение, отладку которого необходимо выполнить.

2 В меню Break выберите требуемое поведение:

- **Clear All** - сброс всех контрольных точек;
- **Disable All** - отключение всех контрольных точек;
- **Enable All** - включение всех контрольных точек.

Редактирование условий контрольной точки

Контрольным точкам можно добавить условия для передачи отладчику команды прервать выполнение в контрольной точке только при соблюдении определенного условия или счетчика.

❖ **Добавление условия или счетчика контрольной точке (окно Breakpoints)**

1 Активизируйте подключение, отладку которого необходимо выполнить.

2 Откройте окно Breakpoints. Контрольные точки для этого подключения отображены в списке.

Для Java-класса условие должно быть выражением булевской переменной Java. Для процедур и триггеров оно должно быть условием поиска SQL. Условие вычисляется в контексте активного подключения.

Примеры

В Java-классе *JDBCExamples.Query* можно использовать условие контрольной точки

```
(price < 10)
```

на строке

```
if (max.price == price) or (price == 10)
```

В процедуре *sp_contacts* можно использовать контрольную точку

```
contact.id = contact.old_id
```

на строке

```
DELETE FROM contact WHERE contact.id = contact.old_id
```

Прерывание на исключении

Подобно контрольным точкам, прерывание на исключении применяется к активному подключению. Вместо прерывания на определенной строке отладчику можно задать команду прерывания выполнения после того, как из Java-класса сброшено исключение.

❖ **Установка условия прерывания**

1 Активизируйте подключение, отладку которого необходимо выполнить.

Для получения информации об установке активного подключения см. раздел "Установка активного подключения" на стр. 575.

2 Задайте условие перерыва из меню Break выбором одного из следующих элементов:

- **When value changes** (при изменении значения) **-** прерывание при изменении переменной или поля;
- **When Exception Thrown** (при исключении) - прерывание при сбросе выбранного класса;
- **On Any Exception** (при любом исключении) - прерывание при сбросе любого исключения Java.

Прерывание подключений

Иногда может быть полезным прервать подключение немедленно или в момент выполнения им следующей команды хранимой процедуры или Java. За один раз можно прервать одно подключение.

❖ **Прерывание подключения**

1 Активизируйте подключение, отладку которого необходимо выполнить. При необходимости прерывания подключения, которого еще нет в окне Connection, выберите All Connections.

☞ Для получения дополнительной информации см. раздел "Установка активного подключения" на стр. 575.

2 Выберите Run➤Interrupt. Отладчик прерывает выполнение на активном подключении при следующем выполнении им оператора.

Проверка переменных

Отладчик процедур позволяет изучить поведение переменных при проверке кода.

Локальные переменные

Во время проверки кода можно просмотреть значения переменных.

❖ **Просмотр значений переменных**

1 Установите контрольную точку в процедуре, переменные которой необходимо просмотреть.

Для получения информации об установке контрольных точек см. раздел "Установка контрольных точек " на стр. 579.

2 Откройте окно Locals.

3 Запустите процедуру. Переменные вместе со своими значениями появляются в окне Local Variables.

Глобальные переменные

Глобальные переменные автоматически отображаются в окне глобальных переменных.

Стек вызовов

Иногда требуется просмотреть контекст переменных, когда они используются во вложенных процедурах. Можно просмотреть список процедур, использующих переменные, в окне Calls (Вызовы).

❖ **Отображение контекста переменных**

1 Установите контрольную точку в процедуре, переменные которой необходимо изучить.

2 Откройте окно Calls.

3 В окне Calls отображаются имена процедур, использующих переменные. Текущая процедура показана в самом верху списка. Вызвавшая ее процедура расположена сразу под ней и т.д.

Написание сценариев отладчика

Отладчик позволяет создавать сценарии на языке программирования Java. Сценарий представляет собой Java-класс, развертывающий класс sybase.asa.procdebug.DebugScript (см. соответствующий раздел на стр. 583).

При выполнении отладчиком сценария он загружает класс и вызывает его метод **run**. Первый параметр метода **run** - указатель на экземпляр интерфейса sybase.asa.procdebug.IDebugAPI (см. соответствующий раздел на стр. 584). Этот интерфейс позволяет осуществлять взаимодействие и управлять действиями отладчика.

Окно отладчика представлено в виде интерфейса sybase.asa.procdebug.IdebugWindow (см. соответствующий раздел на стр. 587).

Сценарии можно компилировать при помощи, например, следующей команды:

```
javac -classpath "c:\Program Files\Sybase\SQL Anywhere
8\java\ProcDebug.jar";%classpath% myScript.Java.
```

Класс sybase.asa.procdebug.DebugScript

Сценарии используются для управления поведением отладчика. Сценарии – это классы, развертывающие класс **DebugScript**. Для получения информации о сценариях см. раздел "Написание сценариев отладчика" на стр. 583.

Класс **DebugScript** выглядит следующим образом:

```
// Все сценарии должны наследоваться из этого класса
package sybase.asa.procdebug;
abstract public class DebugScript
{
    abstract public void run( IDebugAPI db, String args[] );
    /*
         Метод run вызывается отладчиков,
        - аргументы включают в себя аргументы командной строки
    */
    public void OnEvent( int event ) throws DebugError {}
    /*
         - Переопределение следующих методов для обработки
           событий отладки
        - Примечание: этот метод не будет вызываться до вызова
           DebugAPI.AddEventHandler( this );
    */
}
```

Интерфейс sybase.asa.procdebug.IDebugAPI

Сценарии используются для управления поведением отладчика. Сценарии – это Java-классы, использующие интерфейс **IDebugAPI** для управления отладчиком. Для получения дополнительной информации о сценариях см. раздел "Написание сценариев отладчика" на стр. 583.

Интерфейсы **IDebugAPI** выглядят следующим образом:

```
package sybase.asa.procdebug;
import java.util.*;
public interface IDebugAPI
{
   // Имитация элементов меню
   IDebugWindow MenuOpenSourceWindow() throws DebugError;
   IDebugWindow MenuOpenCallsWindow() throws DebugError;
   IDebugWindow MenuOpenClassesWindow() throws DebugError;
   IDebugWindow MenuOpenClassListWindow() throws DebugError;
   IDebugWindow MenuOpenMethodsWindow() throws DebugError;
   IDebugWindow MenuOpenStaticsWindow() throws DebugError;
   IDebugWindow MenuOpenCatchWindow() throws DebugError;
   IDebugWindow MenuOpenProcWindow() throws DebugError;
   IDebugWindow MenuOpenOutputWindow() throws DebugError;
   IDebugWindow MenuOpenBreakWindow() throws DebugError;
   IDebugWindow MenuOpenLocalsWindow() throws DebugError;
   IDebugWindow MenuOpenInspectWindow() throws DebugError;
   IDebugWindow MenuOpenRowVarWindow() throws DebugError;
   IDebugWindow MenuOpenQueryWindow() throws DebugError;
   IDebugWindow MenuOpenEvaluateWindow() throws DebugError;
   IDebugWindow MenuOpenGlobalsWindow() throws DebugError;
   IDebugWindow MenuOpenConnectionWindow() throws DebugError
   IDebugWindow MenuOpenThreadsWindow() throws DebugError;
   IDebugWindow GetWindow( String name ) throws DebugError;
```

```
void MenuRunRestart() throws DebugError;
void MenuRunHome() throws DebugError;
void MenuRunGo() throws DebugError;
void MenuRunToCursor() throws DebugError;
void MenuRunInterrupt() throws DebugError;
void MenuRunOver() throws DebugError;
void MenuRunInto() throws DebugError;
void MenuRunIntoSpecial() throws DebugError;
void MenuRunOut() throws DebugError;
void MenuStackUp() throws DebugError;
void MenuStackDown() throws DebugError;
void MenuStackBottom() throws DebugError;
void MenuFileExit() throws DebugError;
void MenuFileOpen( String name ) throws DebugError;
void MenuFileAddSourcePath( String what ) throws DebugError;
void MenuSettingsLoadState( String file ) throws DebugError;
void MenuSettingsSaveState( String file ) throws DebugError;
void MenuWindowTile() throws DebugError;
void MenuWindowCascade() throws DebugError;
void MenuWindowRefresh() throws DebugError;
void MenuHelpWindow() throws DebugError;
void MenuHelpContents() throws DebugError;
void MenuHelpIndex() throws DebugError;
void MenuHelpAbout() throws DebugError;
void MenuBreakAtCursor() throws DebugError;
void MenuBreakClearAll() throws DebugError;
void MenuBreakEnableAll() throws DebugError;
void MenuBreakDisableAll() throws DebugError;
void MenuSearchFind( IDebugWindow w, String what ) throws
DebugError;
void MenuSearchNext( IDebugWindow w ) throws DebugError;
void MenuSearchPrev( IDebugWindow w ) throws DebugError;
void MenuConnectionLogin() throws DebugError;
void MenuConnectionReleaseSelected() throws DebugError;

// Окно вывода
void OutputClear();
void OutputLine( String line );
void OutputLineNoUpdate( String line );
void OutputUpdate();

// Путь к источнику Java
void SetSourcePath( String path ) throws DebugError;
String GetSourcePath() throws DebugError;

// Захват исключений java
Vector GetCatching();
void Catch( boolean on, String name ) throws DebugError;
```

```
// Подключения к базе данных
int ConnectionCount();
void ConnectionRelease( int index );
void ConnectionAttach( int index );
String ConnectionName( int index );
void ConnectionSelect( int index );

// Регистрация в базе данных
boolean LoggedIn();
void Login( String url, String userId, String password, String
userToDebug ) throws DebugError;
void Logout();

// Имитация действий с клавиатурой/мышью
void DeleteItemAt( IDebugWindow w, int row ) throws DebugEr-
ror;
void DoubleClickOn( IDebugWindow w, int row ) throws DebugEr-
ror;

// Контрольные точки
Object BreakSet( String where ) throws DebugError;
void BreakClear( Object b ) throws DebugError;
void BreakEnable( Object b, boolean enabled ) throws DebugEr-
ror;
void BreakSetCount( Object b, int count ) throws DebugError;
int BreakGetCount( Object b ) throws DebugError;
void BreakSetCondition( Object b, String condition ) throws
DebugError;
String BreakGetCondition( Object b ) throws DebugError;
Vector GetBreaks() throws DebugError;

// Сценарии
void RunScript( String args[] ) throws DebugError;
void AddEventHandler( DebugScript s );
void RemoveEventHandler( DebugScript s );
```

```
// Разное
void EvalRun( String expr ) throws DebugError;
void QueryRun( String query ) throws DebugError;
void QueryMoreRows() throws DebugError;
Vector GetClassNames();
Vector GetProcedureNames();
Vector WindowContents( IDebugWindow window ) throws DebugEr-
ror;
boolean AtBreak();
boolean IsRunning();
boolean AtStackTop();
boolean AtStackBottom();
void SetStatusText( String msg );
String GetStatusText();
void WaitCursor();
void OldCursor();
void Error( Exception x );
void Error( String msg );
void Warning( String msg );
String Ask( String title );
boolean MenuIsChecked( String cmd );
void MenuSetChecked( String cmd, boolean on );
void AddInspectItem( String s ) throws DebugError;
// Ограничения для параметра DebugScript.OnEvent
public static final int EventBreak = 0;
public static final int EventTerminate = 1;
public static final int EventStep = 2;
public static final int EventInterrupt = 3;
public static final int EventException = 4;
public static final int EventConnect = 5;
};
```

Интерфейс sybase.asa.procdebug.IDebugWindow

Сценарии используются для управления поведением отладчика. В сценариях окно отладчика представлено интерфейсом **IDebugWindow**. Для получения дополнительной информации о сценариях см. раздел "Написание сценариев отладчика" на стр. 583.

Интерфейсы **IDebugWindow** выглядят следующим образом:

```
// Этот интерфейс представляет окно отладчика
package sybase.asa.procdebug;
public interface IDebugWindow
{
public int GetSelected();
/*
    Получение текущей выбранной строки -1 при
    отсутствии таковой
*/
public boolean SetSelected( int i );
```

```
/*
    Установка текущей выбранной строки. Игнорируется,
    если i < 0 или i > количества строк
*/

public String StringAt( int row );
/*
    Получение представления String строки Nth данного окна.
    Возвращает нуль, если row > количества строк
*/
public java.awt.Rectangle GetPosition();
public void SetPosition( java.awt.Rectangle r );
/*
    Получение/установка расположения окна в рамке
*/
public void Close();
/*
    Закрытие (удаление) окна
*/
}
```

Индекс

Символы

А

Б

В

Г

Д

Е

Ж

З

И

К

П

P

С

Т

У

Ф

X

Ц

Ч

Ш

Э

Я

A

B

D

E

F

G

I

J

L

M

N

O

P

S

T

V

W

X