Содержание

Часть II
Управление базой данных

Часть IV
Oracle в сетях

Введение

Если вы опытный администратор, или новичок, или разработчик приложения, вы должны знать, как использовать новые возможности Oracle11g, чтобы наилучшим образом удовлетворять потребности клиентов. В этой книге рассмотрены новые возможности, а также способы объединения этих функций в управлении базой данных Oracle. Основное внимание уделено успешному и эффективному управлению возможностями базы данных для получения качественного продукта. Итогом будет гибкая, надежная, безопасная и расширяемая база данных.

Для достижения этих целей особенно важны некоторые компоненты. Все они будут рассмотрены подробно в Части 1 данной книги, когда мы расскажем об архитектуре Oracle, осветим вопросы апгрейда до Oracle 11g и планирования табличных пространств. Хорошо продуманная логическая и физическая архитектура базы данных позволит повысить производительность и упростить администрирование. В Части 2 этой книги будет рассмотрена стратегия мониторинга, безопасности и настройки автономных и сетевых баз данных. Здесь описано обеспечение резервного копирования и восстановление баз данных, чтобы обеспечить возможность восстановления базы данных. В каждом разделе освещены особенности каждой области, планирование и методы управления ею.

В Части 3 этой книги вопросы высокой доступности рассматриваются применительно к Real Application Clusters (RAC), Recovery Manager (RMAN), и Oracle Data Guard.

Подробно рассматриваются вопросы работы в сети и управления распределенными базами данных и базами, созданными в архитектуре клиент/сервер. В Части 4 этой книги описаны Oracle Net, сетевые конфигурации, материализованные представления, расположение прозрачности — все необходимое для успешной реализации распределения базы данных, работающей в архитектуре клиент/сервер. Приведены реальные примеры для каждой основной конфигурации.

Помимо команд, необходимых для работы DBA, описаны экраны Oracle Enterprise Manager, с которых можно выполнять аналогичные функции. При использовании описанных в данной книге методов можно спроектировать и реализовать хорошие системы, и их настройка сведется к минимуму. Администрирование баз данных станет проще, пользователи получат более качественный продукт, а база данных будет работать очень хорошо.

Часть I

Архитектура базы данных

ГЛАВА 1

Знакомство с архитектурой Oracle

Oracle Database 11g стал эволюционным, если не сказать революционным шагом по сравнению с предыдущим выпуском – Oracle 10g. В свою очередь Oracle 10g был поистине революционным шагом от Oracle9i, поскольку позволяет администратору базы данных Oracle (АБД) «установить и забыть» о ней. Oracle 11g продолжает традицию расширения функций, делая управление памятью еще более автоматизированным, добавив несколько новых помощников, существенно повысив готовность к работе и отказоустойчивость.

В части I данной книги содержится обзор архитектуры Oracle и создания фундамента для развертывания успешной инфраструктуры Oracle, а также даны практические советы по инсталляции новой системы, по апгрейду предыдущей версии Oracle. Чтобы обеспечить надежность фундамента программного обеспечения Oracle11g, в соответствующих разделах рассмотрены вопросы, относящиеся к аппаратным средствам сервера и используемым операционным системам.

В части II рассмотрены вопросы повседневного сопровождения и функционирования баз данных Oracle 11g. В главе 5 приведены требования, которые нужно выполнить до загрузки первого установочного диска на сервер. В следующих главах речь идет об управлении дисковой памятью и использовании центрального процессора (ЦП), а также настройке параметров Oracle с целью оптимизировать ресурсы сервера при использовании инструментальных средств, имеющихся в распоряжении АБД для мониторинга производительности базы данных. Управление транзакциями значительно упрощается благодаря использованию AUM (технологии автоматического управления отменами транзакций) — эта опция впервые появилась в Oracle9i и значительно усовершенствована в Oracle 10g и Oracle 11g.

В части III внимание будет сфокусировано на вопросах повышения доступности Oracle 11g. Речь идет об использовании диспетчера восстановления Oracle (RMAN) для выполнения и автоматизации

создания резервных копий базы данных и ее восстановления, а также некоторых других возможностях Oracle (например, технологии Oracle Guard) для обеспечения надежных и легких способов восстановления базы данных после выхода из строя. И, наконец, будет рассмотрено, как с помощью технологии Oracle Real Application Clusters можно одновременно обеспечить среде базы данных отличную масштабируемость и возможности прозрачного восстановления после сбоев. Даже если вы не используете особенности RAC в Oracle 11g, функция ожидания делает Oracle 11g почти такой же доступной, как кластерное решение. Возможность легкого переключения между режимом ожидания и первичной базой данных, как и запрос физической резервной базы данных, обеспечивает высокую доступность до тех пор, пока вы не будете готовы к реализации базы данных RAC.

В части IV описано использование Oracle в сетевой среде. Здесь будет показано, как конфигурировать Oracle Net в N-уровневой среде и управлять большими и распределенными базами данных, которые могут быть физически размещены в близлежащих городах или разбросаны по всему миру.

Мы рассмотрим основы и функции Oracle Database 11g, а также правила установки Oracle 11g с использованием Oracle Universal Installer (OUI) и базы данных Configuration Assistant (DBCA). Мы также будем рассматривать элементы, из которых состоит Oracle 11g, начиная от структуры памяти до дисковой структуры, параметры инициализации, таблиц, индексов и PL/SQL. Каждый из этих элементов играет большую роль в обеспечении высокой масштабируемости, доступности и безопасных условий эксплуатации Oracle 11g.

Обзор баз данных и экземпляров

Хотя довольно часто термины «база данных» и «экземпляр» используются как взаимозаменяемые, на самом деле, они имеют совершенно разные значения. Как будет показано в дальнейшем, в центре обработки данных Oracle они представляют различные сущности.

Базы данных

База данных — это совокупность (collection) данных на диске, хранящаяся в одном или нескольких файлах на сервере базы данных, в которых осуществляется сбор и сопровождение связанной информации. База данных состоит из логических и физических структур. Из всех логических структур базы данных наиболее важна таблица, содержащая столбцы и строки, в которых хранятся родственные данные. База данных должна иметь в своем составе, как минимум, таблицы для хранения полезной информации. На рис. 1.1 приведен пример таблицы, содержащей четыре строки и три столб-

ца. Данные в каждой строке таблицы связанные: каждая строка содержит информацию о конкретном служащем компании.

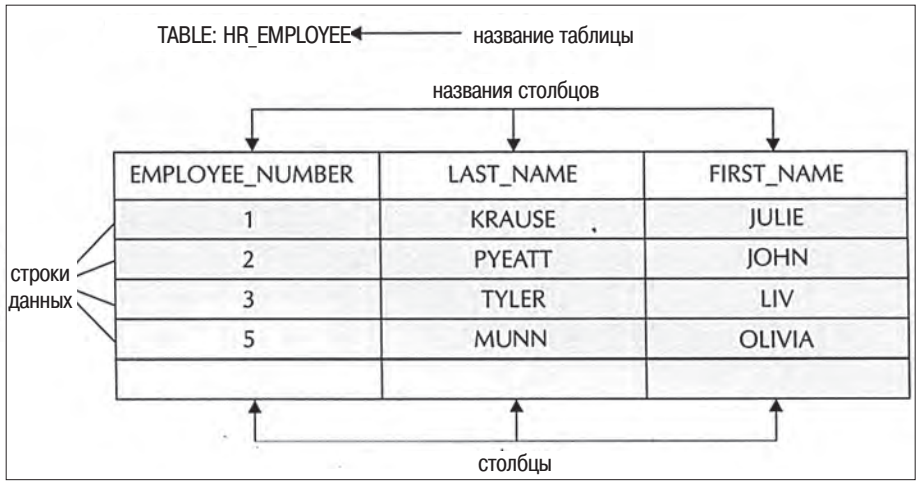

Рис. 1.1. *Пример таблицы базы данных*

В дополнение к этому база данных предлагает уровень защиты, позволяющий предотвратить несанкционированный доступ к данным. Oracle Database 11g предлагает много механизмов для облегчения установления защиты, чтобы конфиденциальные данные оставались таковыми (подробнее об Oracle Security и управлении доступом см. главу 9).

Файлы, из которых состоит база данных, можно разделить на две категории: файлы базы данных и файлы, не относящиеся к файлам базы данных. Они различаются хранящимися в них данными. Файлы базы данных содержат данные и метаданные; файлы, не относящиеся к базе данных, — параметры инициализации, информацию о регистрации (протоколировании) и т. п. Файлы базы данных являются критичными для ежедневного непрекращающегося функционирования базы данных (подробнее физические структуры памяти обсуждаются в разделе «Oracle Физические структуры памяти»).

Экземпляры

Главными компонентами типичного корпоративного сервера являются один или несколько ЦП, дисковая память и оперативная память. Если база данных Oracle хранится на дисках сервера, то экземпляр Oracle хранится в оперативной памяти сервера. Экземпляр Oracle состоит из большого блока памяти, выделенного в области System Global Area (SGA — системная глобальная область), а также из некоторого числа фоновых процессов, осуществляющих интерактивное взаимодействие между SGA и файлами базы данных на дисках.

В Oracle Real Application Clusters (RAC) несколько экземпляров могут использовать одну базу данных. Хотя экземпляры, совместно использующие базу данных, могут находиться на одном сервере, более вероятно, что эти экземпляры размещены на различных серверах, соединенных так называемым высокоскоростным межузловым соединением (interconnect), и обращаются к базе данных, размещенной на специализированной дисковой подсистеме, поддерживающей RAID (дисковый массив) (подробнее о конфигурировании инсталляции RAC см. главу 10).

Логические структуры памяти Oracle

Файлы данных в базе данных Oracle объединяются в одно или несколько табличных пространств (tablespaces). Внутри каждого табличного пространства такие логические структуры базы данных, как таблицы и индексы, являются сегментами, которые далее делятся на экстенты и блоки. Такое логическое подразделение памяти позволяет Oracle более эффективно контролировать использование дисковой памяти. На рис. 1.2 показано отношение между логическими структурами памяти в базе данных.

Табличные пространства

Табличное пространство Oracle состоит из одного или нескольких файлов базы данных; файл базы данных может быть частью только одного табличного пространства. При инсталляции Oracle 11g должно быть создано не менее двух табличных пространств: табличное пространство SYSTEM и табличное пространство SYSAUX. При установке по умолчанию Oracle 11g создает шесть табличных пространств (см. приложение «Установка и настройка» для выборочной установки Oracle 11g).

Oracle 11g позволяет создавать особый вид табличных пространств, называемых табличными пространствами вида BIGFILE (с большим файлом данных), которые могут достигать размера 128TB (терабайт). При использовании табличных пространств такого вида управление табличными пространствами становится для АБД совершенно прозрачным; другими словами, АБД может управлять табличным пространством как неким целым, не учитывая размеры и структуру файлов данных, из которых оно состоит.

Использование Oracle Managed Files (OMF — управляемые Oracle файлы) позволяет облегчить управление файлами данных табличных пространств. С помощью OMF АБД определяет одно или несколько мест в файловой системе, где будут размещены файлы данных, а также управляющие и журнальные файлы, и Oracle автоматически назначает этим файлам имена, а затем управляет ими (подробнее об OMF см. главу 4).

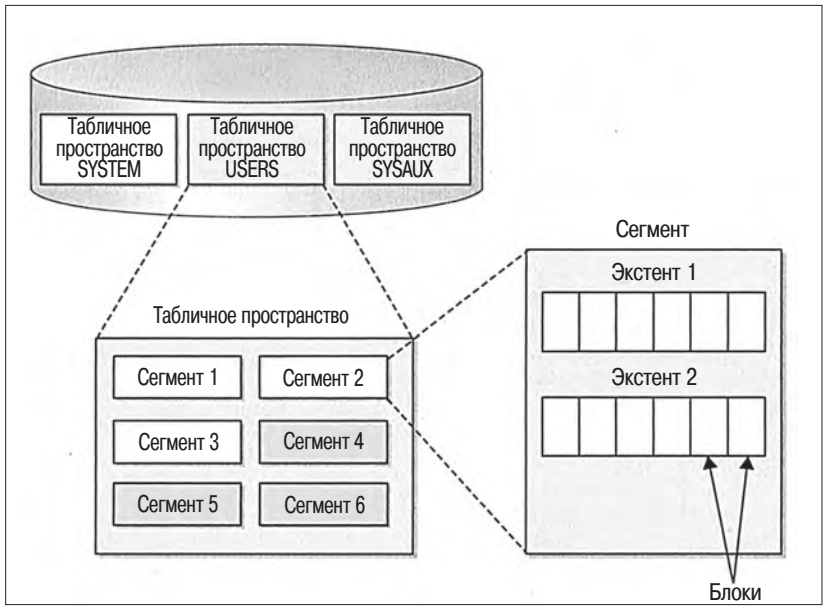

Рис. 1.2. *Логические структуры памяти*

Если табличное пространство *временное* (temporary), то, несмотря на название, само по себе табличное пространство постоянное, а временные только сегменты, хранящиеся в этом табличном пространстве. Временное табличное пространство можно использовать для операций сортировки и как рабочую область при построении индексов. Выделение табличного пространства для подобных операций помогает сократить число конфликтных ситуаций ввода/вывода между временными и постоянными сегментами, хранящимися в другом табличном пространстве, например, с таблицами.

Табличные пространства могут быть либо *управляемыми по словарю*, либо *локально управляемыми*. В табличном пространстве, управляемом по словарю, все операции по управлению экстентами записываются в таблицах словаря данных. Следовательно, если таблицы приложения размещены в табличном пространстве USER, к табличному пространству SYSTEM все равно будут обращения для управления операциями DML над таблицами приложения. Получается, что все пользователи и все приложения должны использовать для управления экстентами табличное пространство SYSTEM. Это создает потенциальное узкое место для приложений с высокой интенсивностью операций записи. В локально управляемом табличном пространстве для каждого файла данных табличного пространства Oracle создает битовую карту для отслеживания доступности памяти. Через словарь данных осуществляется только управление квотами, что существенно сокращает число конфликтных ситуаций для словаря данных.

Начиная с Oracle9i, если табличное пространство SYSTEM локально управляемое, то и все остальные табличные пространства должны быть локально управляемыми, если для них разрешены операции чтения и записи. В базах данных с локально управляемым табличным пространством SYSTEM табличные пространства, управляемые по словарю, должны быть открыты только для чтения.

Блоки

Минимальной единицей хранения для базы данных Oracle является блок базы данных. Размер блока представляет собой конкретное число байтов памяти внутри заданного табличного пространства в базе данных.

Блок обычно кратен размеру блока операционной системы. Это делается для повышения эффективности дискового ввода/вывода. Используемый по умолчанию размер блока определяется параметром инициализации DB_BLOCK_SIZE. Кроме того, другие табличные пространства базы данных могут иметь блоки четырех размеров, хотя в табличных пространствах SYSTEM, SYSAUX и во всех временных табличных пространствах блоки должны иметь размер DB_BLOCK_SIZE.

Экстенты

Следующим уровнем логического группирования в базе данных являются экстенты. Экстент состоит из одного или нескольких блоков базы данных. При увеличении размера объекта базы данных добавляемое объекту пространство выделяется в виде экстента.

Сегменты

Следующим уровнем логического группирования в базе данных является сегмент. Сегментом называется группа экстентов, из которых состоит объект базы данных, рассматриваемый как единое целое, например таблица или индекс. В результате сегмент становится самой малой единицей памяти, с которой имеет дело конечный пользователь базы данных. В базах данных Oracle можно найти четыре типа сегментов: сегменты данных, индексные сегменты, временные сегменты и сегменты отката.

Сегменты данных

Каждая таблица базы данных размещается в одном сегменте данных, который состоит из одного или нескольких экстентов. Более одного сегмента назначается таблице, если она секционированная или кластеризованная. Секционированные и кластеризованные таблицы будут обсуждаться далее в этой главе. Данные сегменты

включают LOB (большие объекты) сегменты, хранящие данные, на которые ссылается столбец LOB в табличном сегменте (если LOB не встроен в таблице).

Индексные сегменты

Каждый индекс хранится в собственном индексном сегменте. Как и в случае с секционированными таблицами, каждый раздел секционированного индекса хранится в собственном сегменте. К этой категории относятся индексные сегменты LOB, таблицы без LOB столбцов, таблицы с LOB столбцами и связанные с LOB индексы, которые могут находиться в собственном табличном пространстве для увеличения производительности.

Временные сегменты

Когда для завершения работы SQL-оператора конечного пользователя требуется дисковая память, например при операциях сортировки, которым не хватает отведенной для них памяти, происходит выделение временного сегмента. Временные сегменты существуют только во время выполнения SQL-оператора.

Сегменты отката

Начиная с Oracle10g, сегменты отката существуют только для табличного пространства SYSTEM, причем обычно АБД не требуется обязательно обслуживать сегмент отката SYSTEM. В предшествующих выпусках Oracle сегменты отката создавались для сохранения «предыдущих» значений операций DML над базой данных, если для таких операций будет выполнен откат. Еще одно применение сегментов отката — показать исходный вид информации (до обновления) с целью единого представления данных для всех пользователей, обращающихся к таблице. Кроме того, сегменты отката используются при восстановлении базы данных для отката всех незавершенных транзакций, которые были активны в момент возникновения аварийной ситуации с экземпляром базы данных или его непредвиденного завершения.

В Oracle 10g для автоматического выделения сегментов отката и управления ими в табличных пространствах отката используется автоматическое управление пространством отката (Automatic Undo Management — AUM). Внутри табличного пространства отката управляемые автоматически сегменты отката (undo segments) структурированы аналогично сегментам отката, управляемым вручную (rollback segments), за одним исключением: первыми управляет, как следует из названия, сама системаOracle, а вторыми вручную управляет АБД (причем, далеко не всегда эффективно). Управляемые автоматически сегменты отката впервые появились в Oracle9i, но в Oracle 10g можно по-прежнему пользоваться управляемыми вручную сегментами отката. Однако их использование в Oracle 10g уже

не приветствуется, а в последующих выпусках их поддержка будет полностью прекращена (о технологии Automatic Undo Management см. главу 7). В Oracle 11g автоматическое управление пространством отката включено по умолчанию, кроме того предусмотрена процедура PL/SQL, чтобы помочь изменять размер табличного пространства UNDO, когда вам это необходимо. Отключение автоматического управления пространством будет подробно обсуждаться в главе 7.

Логические структуры базы данных Oracle

В этом разделе рассматриваются основные моменты всех главных логических структур базы данных, начиная с таблиц и индексов, а также типы данных, которые можно использовать при определении столбцов таблицы. После создания таблицы со столбцами можно ввести для столбцов таблицы ограничения (restrictions) или ограничивающие условия (constraints).

Одной из причин применения Relational Database Management System (RDBMS — система управления реляционными базами данных) для управления данными является широкое использование возможностей аудита и защиты данных базы данных Oracle. Ниже рассматриваются способы разделить доступ к базе данных для различных пользователей или для объектов, к которым они обращаются.

Кроме того, будут рассмотрены другие логические структуры, которые могут быть определены либо АБД, либо пользователем, в том числе синонимы, связи с внешними файлами и ссылки на другие базы данных.

Таблицы

Таблица — базовый механизм сохранения информации в базе данных Oracle. Без таблиц база данных не имела бы никакой ценности для предприятия. Вне зависимости от типа таблицы, данные в ней содержатся в строках и столбцах, подобно тому, как хранятся данные в электронных таблицах. Но на этом сходство заканчивается. Надежность таблицы базы данных, гарантируемая распространяющейся на окружающую таблицу среду базы данных Oracle надежностью, целостностью и масштабируемостью, делает электронные таблицы всего лишь слабым вторым вариантом при выборе решения о месте хранения стратегически важной информации.

В этом разделе рассматриваются различные таблицы базы данных Oracle и как они позволяют удовлетворить почти все потребности хранения данных в вашей организации (подробнее о выборе между этими различными типами таблиц для конкретного приложения и управлении ими см. главы 5 и 8).

Реляционные таблицы

Реляционная таблица — наиболее распространенный тип таблиц базы данных. Эта таблица организована по принципу «кучи»; другими словами, для хранения строк таблицы не существует никакого определенного порядка. В команде *create table* можно специфицировать фразу *organization heap*, посредством которой определяется организованная по принципу «кучи» таблица, но так как режим «кучи» принят по умолчанию, эта фраза обычно опускается.

Каждая строка таблицы содержит один или несколько столбцов, у каждого столбца есть тип данных и длина. Начиная с Oracle8, в столбцах могут также содержаться определенные пользователем типы данных — вложенные таблицы или массивы VARRAY. Кроме того, таблица может быть определена как объектная таблица. Объекты и объектные таблицы будут рассмотрены ниже. Встроенные типы данных Oracle перечислены в табл. 1.1.

Таблица 1.1. *Встроенные типы данных Oracle*

Встроенный тип данных Oracle	Описание
VARCHAR2 (размер) [BYTE | CHAR]	Символьное поле переменной длины с максимальной длиной до 4000 байтов, минимальная длина 1 байт. Модификатор CHAR означает, что для определения длины строки используется так называемая символьная семантика; BYTE означает, что для этой цели используется байтовая семантика.
NVARCHAR2 (размер)	Поле переменной длины с максимальным размером 4000 байтов.
NUMBER (p, s)	Числовой столбец с заданными разрядностью (p) и масштабом (s). Разрядность может изменяться в диапазоне от 1 до 38, а масштаб — от –84 до 127.
LONG	Поле данных переменной длины, размер поля до 2 Гб (231–1).
DATE	Значения дат, начиная с 1 января 4712 г. до н.э. и заканчивая 31 декабря 9999 г.
BINARY_FLOAT	32-битовое число с плавающей точкой.
BINARY_DOUBLE	64-битовое число с плавающей точкой.

Таблица 1.1 *(Продолжение)*

Встроенный тип данных Oracle	Описание
TIMESTAMP (точность_секунд) WITH TIME ZONE	Содержит значение TIMESTAMP, к которому добавлено смещение, связанное с часовым поясом. Часовой пояс может отсчитываться от UTC (всемирное время), например '–6:00' или указываться названием региона, например 'US/Central'.
TIMESTAMP (точность_секунд) WITH LOCAL TIME ZONE	Этот тип данных похож на TIMESTAMP WITH TIMEZONE, но (1) при сохранении в базе данных дата приводится к часовому поясу базы данных и (2) при выборке данных пользователь будет видеть эти данные в пересчете на часовой пояс сеанса.
INTERVAL YEAR (разрядность_года) TO MONTH	Хранит значение интервала времени, выраженное в годах и месяцах. Значение разрядность_года определяет число цифр в поле YEAR даты.
INTERVAL DAY (разрядность_дней) TO SECOND (точность_секунд)	Хранит значение интервала времени, выраженное в днях, часах, минутах, секундах и долях секунды. Значение разрядность_дней лежит в интервале от 0 до 9. Значение по умолчанию равно 2. Значение точность_секунд аналогично значению точность_секунд для типа данных TIMESTAMP. Оно также лежит в интервале от 0 до 9, а значение по умолчанию равно 6.
RAW (размер)	Поле переменной длины (до 2000 байтов).
LONG RAW	Поле переменной длины (до 2 Гб), используемое для хранения двоичных данных.
ROWID	Представление (в кодировке base-64)в виде строки уникального адреса строки соответствующей таблицы. Этот адрес должен быть уникальным для всей базы данных.
UROWID [(размер)]	Строка (в кодировке base-64), представляющая логический адрес строки индекс-таблицы; длина до 4000 байтов.

Таблица 1.1 *(Продолжение)*

Встроенный тип данных Oracle	Описание
NCHAR (размер)	Поле для размещения символов фиксированной длины (состоящих из нескольких байтов). Максимальный размер поля — 2000 байт. Значение по умолчанию — 1 байт.
CLOB	Символьный большой объект, содержащий однобайтовые или многобайтные символы; поддерживает как однобайтные, так и многобайтные наборы символов. Максимальный размер объекта равен (4 Гбайт - 1)*DB_BLOCK_SIZE.
NCLOB	Аналогичен CLOB, но в базу данных вместо символов из наборов символов фиксированной или переменной длины записываются символы Unicode. Максимальный размер объекта равен (4 Гбайт - 1)*DB_BLOCK_SIZE.
BLOB	Двоичный большой объект. Максимальный размер объекта равен (4 Гбайт - 1)*DB_BLOCK_SIZE.
BFILE	Указатель на большой двоичный файл, хранящийся вне базы данных. Двоичные файлы должны быть доступны с сервера, на котором выполняется экземпляр Oracle. Максимальный размер объекта равен 4 Гбайт.

Кроме того, Oracle поддерживает типы данных, сопоставимые с типами данных ANSI. Соответствие между типами данных Oracle и типами данных ANSI показано в табл. 1.2.

Временные таблицы

Начиная с Oracle8i, в базе данных Oracle появились временные таблицы. Временными они являются в том смысле, что сохраняется описание временной таблицы, но не хранящиеся в ней данные. Создается временная таблица с помощью команды *create global temporary table.*

При условии, что все прочие пользователи также имеют полномочия на эту таблицу, они могут выполнять над временной таблицей операторы *select* или такие команды языка манипулирования данными (Data Manipulation Language — DML), как, например, *insert,*

update или *delete*. Однако каждая строка данных в таблице видна только для того пользователя, который вставил эту строку. Когда пользователь удаляет временную таблицу, будут удалены только ранее вставленные им строки.

Данные, содержащиеся во временной таблице, относятся либо к сеансу, либо к транзакции. Продолжительность жизни временной таблицы зависит от ключевых слов, которые используются в конструкции *on commit* для этой временной таблицы. Если указано *on commit delete rows*, то при появлении команд *commit* или *rollback*, будут удалены все строки временной таблицы, а в случае задания *on commit preserve rows*, строки таблицы будут сохранены и после окончания транзакции. Но по окончании сеанса пользователя все строки временной таблицы будут удалены.

Таблица 1.2. *Типы данных Oracle, эквивалентные типам данных ANSI*

Тип данных SQL ANSI	Тип данных Oracle
CHARACTER(n) CHAR(n)	CHAR(n)
CHARACTER VARYING(n) CHAR VRYING(n)	VARCHAR(n)
NATIONAL CHARACTER(n) NATIONAL CHAR(n) NCHAR(n)	NCHAR(n)
NATIONAL CHARACTER VARYING(n) NATIONAL CHAR VARYING(n) NCHAR VARYING(n)	NVARCHAR2(n)
NUMERIC(p,s) DECIMAL(p,s)	NUMBER(p,s)
INTEGER I NT SMALLINT	NUMBER(38)
FLOAT(b) DOUBLE PRECISION REAL	NUMBER

Есть еще несколько моментов, которые следует иметь в виду при использовании временных таблиц. Хотя можно создать индекс для временной таблицы, все записи индекса будут удалены вместе со строками данных, как для обычной таблицы. К тому же, вследствие временности данных из временной таблицы в журнале не будет сгенерирована информация об операциях DML над временными таблицами; однако информация в табличном пространстве отката все-таки создается.

Организация индекс-таблицы

Создание индекса делает поиск конкретной строки таблицы более эффективным. Однако при этом возникают дополнительные накладные расходы, так как теперь база данных должна помимо сопровождения строк самой таблицы осуществлять и сопровождение записей индекса таблицы. А если в таблице не много столбцов и обращения к таблице происходят преимущественно по одному из столбцов? В таком случае правильным решением может оказаться так называемая индекс-таблица (IOT — index organized table). В IOT строки таблицы хранятся в индексе со структурой B-дерева, где в каждом узле этого индекса содержится ключевой (индексированный) столбец вместе с одним или несколькими неиндексированными столбцами.

Одно из наиболее очевидных преимуществ IOT в том, что теперь необходимо поддерживать и сопровождать всего одно представление данных в памяти вместо двух. Аналогично, значения первичного ключа таблицы в IOT будут сохранены только один раз, а в обычной таблице это происходит дважды.

Однако при применении индекс-таблиц обнаруживаются и некоторые недостатки. Некоторым таблицам, например таблицам для протоколирования событий, нужен для этого первичный (или любой другой подходящий) ключ; IOT же обязательно должна иметь первичный ключ. Кроме того, IOT не может быть членом кластера. И, наконец, IOT может оказаться далеко не лучшим решением для таблицы, если в ней много столбцов, и при выборке из этой таблицы информации (строк) обращения к ней производятся по многим столбцам.

Объектные таблицы

Начиная с Oracle8, база данных Oracle поддерживает многие объектно-ориентированные возможности. С помощью определенных пользователем типов и методов можно добиться прозрачной реализации в Oracle объектно-ориентированного прикладного проекта.

У объектных таблиц имеются строки, которые сами выступают объектами или инстанциями (экземплярами реализации) определений типов. На строки объектной таблицы можно ссылаться, используя для этого объектные идентификаторы (object identifier — OID), в противоположность первичному ключу обычной (реляционной) таблицы. Однако объектная таблица может иметь оба ключа — и первичный, и уникальный, как обычная таблица.

Предположим, нужно создать кадровую систему (HR) «с нуля», так что есть все возможности для создания полностью основанного на объектно-реляционной точке зрения проекта. Первым шагом должно стать определение объекта (или типа) — служащий (employee). Это можно сделать следующим образом:

- `create type PERS_TYP as object`

```
(Last_Name varchar2(45),
First_Name varchar2(30),
Middle_Initial char(1),
Surname varchar2(10),
SSN varchar2(15));
```

В этом конкретном случае для типа PERS_TYP не создается ни один метод, но по умолчанию Oracle создает для типа метод конструктора, который получает то же самое имя, что и сам тип (в данном случае PERS_TYP). Для создания объектной таблицы как коллекции объектов типа PERS_TYP можно использовать известный синтаксис *create table*, а именно:

- `create table pers of pers_typ;`

Для добавления объекта к объектной таблице необходимо указать в команде *insert* метод конструктора:

- ```
  insert into pers
    values (pers_typ('Graber', 'Martha', 'E', 'Ms.' '123-45-6789'));
  ```

Начиная с версии Oracle10g, метод конструктора не понадобится, если таблица состоит из единичного объекта. В этом случае упрощенный синтаксис выглядит так:

- `insert into pers values('Graber', 'Martha', 'E', 'Ms.', '123-45-6789');`

Ссылки на экземпляры объекта PERS_TYP могут быть сохранены в других таблицах как объекты типа REF (ссылки), а данные из таблицы PERS могут быть выбраны без прямых ссылок на саму таблицу PERS.

Новые примеры использования объектов для реализации объектно-ориентированных проектов можно найти в главе 5.

Внешние таблицы

Внешние таблицы были впервые введены в Oracle9i. Внешние таблицы (external tables) позволяют пользователю получать доступ к источнику данных, например, к текстовому файлу, как если бы он был таблицей базы данных. Метаданные таблицы хранятся в словаре данных Oracle, но содержимое таблицы хранится вне базы.

Определение внешней таблицы содержит две части. В первой и наиболее хорошо известной части содержится определение таблицы с точки зрения пользователя базы данных. Это определение выглядит так же, как и любое другое типичное определение, с которым приходится сталкиваться в операторе *create table*.

Во второй части содержится то, что отличает внешнюю таблицу от обычной. Именно здесь хранится отображение столбцов внешней таблицы на данные из внешнего файла — с какого столбца начинаются элементы данных, какова ширина (размер)
  ```

столбца, а также символьный или цифровой формат внешнего столбца. Синтаксис ORACLE_LOADER, используемый для типа по умолчанию внешней таблицы, идентичен синтаксису управляющего файла утилиты SQL*Loader. В этом одно из преимуществ внешних таблиц: чтобы получить доступ к внешнему файлу, пользователю необходимо знать, как осуществляется доступ к таблицам базы данных.

Но наряду с преимуществами при использовании внешних таблиц имеются и недостатки. Для внешних таблиц нельзя создавать индексы и нельзя использовать команды вставки (*insert*), обновления (*update*) и удаления (*delete*). Но эти недостатки незначительные по сравнению с преимуществами использования внешних таблиц для загрузки «родных» таблиц базы данных, например в средах хранилищ данных.

Кластеризованные таблицы

Если с двумя (или с большим числом) таблиц часто работают совместно (например, в случае таблицы заказов и детальной таблицы элементов заказа), неплохим решением для повышения производительности запросов, обращающихся к этим таблицам, может оказаться создание кластеризованной таблицы. В случае таблицы заказов и ассоциированной с ней детальной таблицы элементов заказа информация из заголовка документа может храниться в том же блоке, что и информация записей с деталями заказа, что позволит уменьшить число операций ввода/вывода, необходимых для выборки всей информации о заказе.

Кластеризация таблиц позволяет сократить объем памяти, необходимой для хранения столбцов, общих для обеих таблиц, совокупность которых называют значением кластерного ключа. Значение кластерного ключа хранится и в так называемом кластерном индексе. Кластерный индекс работает аналогично обычному традиционному индексу, так как он повышает эффективность запросов к кластеризованным таблицам при доступе к ним по значению кластерного ключа. В примере с заказами и элементами заказа номера заказов будут сохранены только один раз, вместо того, чтобы повторяться в каждой строке с элементами заказа.

Преимущества кластеризованных таблиц существенно сокращаются, когда для них часто приходится выполнять операции вставки, обновления и удаления. Кроме того, частые запросы к отдельным таблицам, входящим в состав кластера, также приводят к сокращению преимуществ кластеризованных таблиц перед некластеризованными.

Хеш-кластеры

Особый тип кластеризованных таблиц — хеш-кластеры — работает во многом аналогично обычной кластеризованной таблице, но

вместо использования обычного кластерного индекса в хеш-кластерах для определения физической локализации места, где следует сохранить строку (или откуда ее следует выбирать), используются функции хеширования кластерного ключа строки. Вся память, необходимая для таблицы, выделяется при ее создании при условии, что известно число хеш-ключей, указываемое при создании кластера. В условиях данного примера (заказ и его элементы) предположим, что базе данных Oracle необходимо отобразить унаследованную систему ввода данных, в которой номера заказов периодически повторяются. Кроме того, номера заказов в обязательном порядке должны быть шестизначными числами. В таком случае можно создать кластер для заказов следующим образом:

```
create cluster order_cluster (order_number (6))
     size 50
     hash is order_number hashkeys 1000000;

create table cust_order (
     order_number number(6)     primary key,
     order_date     date,
     customer_number     number)
cluster order_cluster(order_number);
```

Наибольшие преимущества хеш-кластеров связаны с запросами, использующими операции сравнения по равенству, как показано в следующем примере:

```
select order_number, order_date from cust_order
     where order_number = 196811;
```

Как правило, такой тип запроса будет находить необходимую строку, используя всего одну операцию ввода/вывода, если число *hashkeys* достаточно велико, а фраза *hash is*, содержащая функцию хеширования, продуцирует равномерно распределенные ключи хеширования.

Сортированные хеш-кластеры

Сортированные хеш-кластеры стали новинкой Oracle 10g. Они похожи на обычные хеш-кластеры тем, что для нахождения строки в таблице используется функция хеширования. Однако в дополнение к этому сортированные хеш-кластеры позволяют хранить строки таблицы упорядоченными (в порядке возрастания) по значению одного или нескольких столбцов таблицы. Это позволяет быстрее обрабатывать данные для приложений, которые отдают предпочтение обработке типа «первый пришел, первым обслужен» (FIFO).

Для создания сортированных хеш-кластеров используется такой же синтаксис, как для обычных кластеризованных таблиц, но с добавлением после описания столбцов кластера позиционного па-

раметра SORT. Ниже приведен пример создания таблицы в составе сортированного хеш-кластера:

```
create table order_detail (
     order_number     number,
     order_timestamp     timestamp sort,
     customer_number     number)
cluster order_detail_cluster (
     order_number,
     order_timestamp);
```

Благодаря FIFO-природе сортированного хеш-кластера, когда обращения к заказам осуществляются по столбцу *order_number*, самые старые заказы будут выбраны первыми, для чего будет использовано значение столбца *order_timestamp*.

Секционированные таблицы

Секционирование таблицы (или индекса; см. следующий раздел) помогает повысить управляемость больших таблиц. Таблица может быть разбита на разделы, которые, в свою очередь, могут быть разбиты на подразделы, т. е. на более мелкие части. С точки зрения приложения секционирование остается прозрачным (это означает, что ни в каких из задаваемых конечным пользователем операторах SQL не должно появиться никаких явных ссылок на разделы таблицы). Единственный эффект, который может обнаружить конечный пользователь, заключается в том, что запросы к секционированным таблицам, использующие критерии во фразе *where*, которые соответствуют схеме секционирования, выполняются намного быстрее.

С точки зрения АБД в секционировании таблиц обнаруживается множество преимуществ. Если один из разделов секционированной таблицы размещен на испортившемся дисковом томе, другие разделы этой таблицы по-прежнему останутся доступны для запросов пользователей на время восстановления поврежденного тома. Можно делать резервные копии таблицы по разделам, по одному разделу, а не создавать единую резервную копию для всей таблицы.

Разделы могут быть одного из трех видов: секционированные по диапазонам ключей, хэш-секционированные или начиная с Oracle9i и реализовано в Oracle11g можно распределить отношения parent/child, с применением управляемого разбиения, со множеством комбинаций основных типов разделов, включая список-хеш (list-hash), список-список (list-list), список-диапазон (list-range) и диапазон-диапазон (range-range). Каждая строка в секционированной таблице может принадлежать только одному разделу. Ключ раздела служит для направления строк таблицы в соответствующий раздел; этот ключ может быть составным, состоящим не более чем из 16 столбцов таблицы. Есть несколько несущественных ограничений на типы таблиц, которые можно секционировать; например, нельзя секционировать таблицы, содержащие

столбцы типа LONG или LONG RAW. Ограничения LONG не должны доставить больших проблем, большие объекты (LOB) (CLOB и BLOB, символьный большой объект и двоичный большой объект) гораздо более гибкие и охватывают все особенности типов данных LONG и LONG RAW.

СОВЕТ Корпорация Oracle настоятельно рекомендует тщательно рассмотреть вопрос о секционировании всех таблиц, размер которых превышает 2 Гбайт.

Неважно, какой из типов схемы секционирования используется, каждый член (участник) секционированной таблицы должен иметь те же самые логические атрибуты, например имена столбцов, типы данных, ограничивающие условия и т.д. Физические атрибуты для каждого из разделов могут отличаться в зависимости от размеров и места нахождения дискового устройства. Все дело в том, что секционированная таблица не должна быть логически противоречивой с точки зрения приложения или конечного пользователя.

Раздел по диапазону ключей. Для этого раздела (range partition) значение ключа попадает в определенный диапазон. Например, посещения корпоративного web-сайта электронной торговли можно относить к определенному разделу на основании даты посещения сайта, причем будет создаваться по одному разделу в квартал. Посещение web-сайта, состоявшееся 25 мая 2004 г., будет зафиксировано в разделе FY2004Q2, а посещение, состоявшееся 2 декабря 2004 г., — в разделе FY2004Q4.

Раздел по списку значений ключа. Ключ этого раздела (list partition) попадает в одну из групп, состоящих из различных значений. К примеру, при обработке продаж по различным регионам США может быть создан один раздел для штатов NY, CN, MA и VT и второй — для IL, WI, IA и MN. Все продажи из всех остальных регионов (США и всего остального мира) будут попадать в отдельный раздел, если для них не указан код штата.

Хеш-разделы. Алгоритм хеш-раздела относит строку к тому или иному разделу в зависимости от значения функции хеширования, в которой специфицирован столбец (или столбцы), используемый функцией хеширования, но не назначает раздел явно, а лишь указывает, сколько разделов возможно. Oracle будет назначать разделу строку и обеспечивать сбалансированное распределение строк в каждом разделе.

Хеш-разделы полезны, если отсутствует четкий список или схема разбиения на разделы по диапазону ключей при заданных типах столбцов таблицы или если относительные размеры разделов часто изменяются и необходимо совершенствовать схемы разбиения на разделы (вручную).

Композитные разделы. Дальнейшее совершенствование процесса разбиения на разделы становится возможным при использовании композитных разделов. Можно разбить таблицу на разделы по диапазону ключей, а внутри каждого диапазона разбить на подразделы по списку значений ключа или с помощью хеширования. Новые комбинации в Oracle 11g включают разделы список-хеш (list-hash), список-список (list-list), список-диапазон (list-range) и диапазон-диапазон (range-range).

Секционированные индексы

Индексы для таблицы также можно секционировать либо в соответствии со схемой секционирования индексируемой таблицы (локальное индексирование), либо независимо от схемы секционирования таблицы (глобальное индексирование). Преимущество локально секционированных индексов в том, что их применение повышает доступность индекса при выполнении операций над разделами; например, архивирование и удаление раздела FY2004Q4 и его локального индекса не повлияет на доступность индекса для других разделов таблицы.

Ограничения целостности

Ограничением целостности (constraint) в Oracle называется правило или правила, которые могут быть определены для одного или нескольких столбцов таблицы, чтобы помочь принудительному выполнению бизнес-правил. Так, ограничение целостности может провести в жизнь выполнение бизнес-правила, гласящего, что стартовая зарплата служащего не может быть меньше $25 000 долл. в год. Другой пример ограничения целостности, проводящего в жизнь бизнес-правило: если вновь принятому сотруднику был назначен отдел (хотя он и не должен быть сразу приписан к какому-то отделу), назначенный номер отдела должен быть допустимым и присутствовать в таблице DEPT (таблица отделов).

К столбцам таблицы могут быть применены шесть типов правил целостности данных (data integrity): правило применения значений NULL, уникальные значения столбцов, значения первичных ключей, значения ссылочной целостности, сложная встроенная целостность и целостность на базе триггеров. Мы кратко рассмотрим каждый из них в следующих разделах.

Все ограничения целостности для таблицы определяются либо во время создания таблицы, либо при изменении таблицы на уровне столбцов за исключением триггеров, которые определяются на основании того, какие операции DML выполняются над таблицей. Ограничения целостности могут быть активированы или деактивированы при создании или в любой последующий момент времени. Когда ограничение целостности активируется

или отключается (для этого используются ключевые слова *enable* и *disable*), имеющиеся в таблице данные с помощью ключевых слов *validate* и *novalidate* могут быть подвергнуты или не подвергнуты проверке на достоверность относительно ограничения целостности, зависящего от действующих бизнес-правил.

Например, для таблицы, входящей в состав базы данных производителей автомобилей, которая называется CAR_INFO и содержит сведения о новых автомобилях, требуется новое ограничение для столбца AIRBAG_QTY, в соответствии с которым этот столбец не может иметь пустого (NULL) значения, а его значение не может быть меньше 1 для всех новых автомобилей. Однако в этой таблице содержатся данные о моделях автомобилей, выпущенных в период, когда подушки безопасности для автомобилей еще не требовались, в результате в этом столбце будет стоять либо 0, либо NULL. Одним из решений для такого случая будет создание ограничения целостности для столбца AIRBAG_QTY, обеспечивающего проведение в жизнь нового правила для вновь вводимых в таблицу строк, но не проверка выполнения этого правила для уже имеющихся в таблице строк.

Ниже приведен пример создания таблицы со всеми типами ограничений целостности. Каждый из типов ограничений будет затем рассмотрен в соответствующем подразделе.

```
create table CUST_ORDER
    (Order_Number           NUMBER (6)     PRIMARY KEY,
    Order_Date              DATE           NOT NULL,
    Delivery_Date           DATE,
    Warehouse_Number        NUMBER         DEFAULT 12,
    Customer_Number         NUMBER         NOT NULL,
    Order_Line_Item_Qty     NUMBER         CHECK (Order_Line_Item_Qty < 100),
    UPS_Tracking_Number     VARCHAR2 (50)  UNIQUE,
       foreign key (Customer_Number) references CUSTOMER(Customer_
       Number));
```

Правило применения значений NULL

Ограничения целостности NOT NULL не позволяют ввести в базу данных пустые значения столбцов Order_Date или Customer_Number. С точки зрения бизнес-правил в этом заключен глубокий смысл: для каждого заказа должна быть известна дата, и заказ не имеет никакого смысла, пока он не будет размещен заказчиком.

Значение NULL в столбце не означает, что это значение пробельное или нулевое; правильнее сказать, что значение не существует. Значение NULL не равно ничему, даже другому значению NULL. Эта концепция важна при применении SQL-запросов к столбцам, в которых имеются значения NULL.

Уникальные значения столбцов

Ограничение целостности UNIQUE (уникальное значение) гарантирует, что столбец или группа столбцов (в композитных ограничениях) является уникальным в таблице. В предыдущем примере столбец UPS_Tracking_Number не будет содержать повторяющихся значений.

Для принудительного выполнения этого ограничения целостности Oracle создаст для столбца UPS_Tracking_Number так называемый индекс UNIQUE (индекс по уникальному ключу). Oracle будет использовать этот индекс для принудительного обеспечения выполнения ограничений целостности.

Столбец с ограничением UNIQUE может быть так же объявлен как NOT NULL. Если столбец не объявлен с ограничением NOT NULL, то любое количество строк, может содержать значения NULL, до тех пор, пока остальные строки имеют уникальные значения в этом столбце.

В композитном ограничении по уникальности, которое разрешает иметь значение NULL в одном или нескольких столбцах, именно имеющие не пустое (NOT NULL) значение столбцы определяют, будет ли выполнено ограничение. Столбцы NULL всегда удовлетворяют ограничению, поскольку значение NULL не равно ничему.

Значения первичных ключей

Ограничение целостности PRIMARY KEY относится к числу наиболее распространенных ограничений, которые можно найти в базах данных. Для таблицы может существовать только одно ограничение первичного ключа. Столбец или столбцы, составляющие первичный ключ, не могут иметь значение NULL.

В предыдущем примере первичным ключом является столбец Order_Number. Для выполнения этого ограничения создается индекс по уникальному ключу; если пригодный к использованию индекс для этого столбца уже существует, в ограничении первичного ключа будет использован именно этот индекс.

Значения ссылочной целостности

Ограничения ссылочной целостности FOREIGN KEY более сложные, чем ранее рассмотренные, так как они связаны с другой таблицей, определяющей, какие значения можно ввести в столбец, на который наложено ограничение ссылочной целостности.

В предыдущем примере ограничение FOREGN KEY объявлено для столбца Customer_Number; любые значения, вводимые в этот столбец, должны существовать и в столбце Customer_Number другой таблицы (в данном случае таблицы Customer).

Как и в случае с другими ограничениями, допускающими значения NULL, столбец с ограничением по ссылочной целостности мо-

жет быть NULL, При этом необязательно, чтобы в столбце, на который делается ссылка, содержалось значение NULL.

Более того, ограничение FOREIGN KEY может быть рефлексивным (self-referential, т. е. ссылающимся само на себя). В таблице EMPLOYEE, первичным ключом которой является столбец Employee_Number, для столбца с именем Manager_Number может быть объявлено ограничение FOREIGN KEY по столбцу Employee_Number в той же таблице. Это позволяет создать в таблице EMPLOYEE иерархию подчиненности.

Для повышения производительности рекомендуется всегда создавать индексы для столбцов, объявленных с ограничением FOREIGN KEY. Единственное исключение из этого правила: если столбец, на который делается ссылка, первичный или уникальный ключ в родительской таблице и никогда не подвергался удалению или обновлению.

Сложная встроенная целостность

Для принудительной реализации более сложных бизнес-правил на уровне столбца может быть использовано ограничение CHECK. В предыдущем примере значение столбца Order_Line_Item_Qty не должно превосходить 99.

В ограничении CHECK для вычисления значения ограничения могут использоваться другие столбцы этой (обновляемой или вставляемой) строки. Например, ограничение на столбец STATE_CD разрешает использовать в нем значения NULL только в том случае, если в столбце COUNTRY_CD указано не USA. Помимо значений столбцов в ограничении могут использоваться литеральные значения и встроенные функции, например, TO_CHAR или TO_DATE, при условии, что аргументами этих функций являются литералы или столбцы таблицы.

Для столбца разрешается использовать несколько ограничений CHECK. Для разрешения включения вводимого значения в столбец необходимо, чтобы при вычислении каждого из ограничений CHECK было получено значение TRUE. Например, можно модифицировать предыдущее ограничение CHECK, введя в него в дополнение к имеющемуся условию еще одно: чтобы Order_Line_Item_Qty было больше нуля, но меньше 100.

Целостность на базе триггеров

Если бизнес-правила слишком сложны для реализации с помощью ограничений уникальности, можно создать для таблицы так называемый триггер базы данных, используя команду *create trigger* и блок кода PL/SQL для проведения в жизнь этого бизнес-правила.

Триггеры требуются для принудительного выполнения ограничений ссылочной целостности, когда таблица, на которую делаются ссылки, находится в другой базе данных. Полезны триггеры и для множества других действий, далеких от области проверки выпол-

нения ограничений (например, для аудита доступа к таблице). Подробнее о триггерах базы данных см. главу 17.

Индексы

Индекс Oracle позволяет обеспечить более быстрый доступ к строкам таблицы, когда из таблицы должно быть отобрано лишь небольшое подмножество входящих в нее строк. В индексе хранится значение столбца или столбцов, по которым он строится, а также физическое значение RowID строки, где содержится индексируемое значение, за исключением индекс-таблиц (IOT), использующих первичный ключ в качестве логического RowID. Как только в индексе будет найдено совпадение, RowID из индекса непосредственно укажет на точное положение (адрес) строки таблицы: в каком файле, в каком блоке внутри этого файла и в какой строке внутри блока.

Индексы создаются по одному или нескольким столбцам. Записи индекса Oracle хранятся в структуре B-дерева, поэтому для обхода индекса, требующегося для нахождения ключевого значения для строки, нужно очень мало операций ввода/вывода. Кроме того, уникальные индексы могут служить для еще одной цели: они не только увеличивают скорость поиска строки, но и могут (при желании) обеспечивать выполнение ограничения уникального или первичного ключа для индексируемого столбца. Записи индекса автоматически обновляются при выполнении над содержимым таблицы любой операции вставки, обновления или удаления. При удалении таблицы все созданные для нее индексы также автоматически удаляются.

В Oracle несколько типов индексов, каждый из которых более всего подходит для определенного вида таблиц, метода доступа к ним или прикладной среды. Основные моменты и особенности типов индексов рассмотрим в следующих разделах.

Индексы по уникальному ключу

Индексы по уникальному ключу (индексы UNIQUE) — наиболее распространенная форма индексов со структурой B-дерева. Они часто используются, чтобы обеспечить выполнение для таблицы ограничения первичного ключа. Использование индексов по уникальному ключу гарантирует, что в индексируемом столбце (столбцах) не встретятся повторяющиеся (дублирующие друг друга) значения. Например, индекс UNIQUE для таблицы EMPLOYEE может быть создан для столбца Social Security Number (SSN — номер системы социального обеспечения), так как в этом поле не должно быть повторений (дубликатов). Однако некоторые служащие могут не иметь SSN, поэтому для этого столбца следует разрешить значение NULL.

Индексы по неуникальному ключу

Индексы по неуникальному ключу (индексы non-UNIQUE) помогают ускорить доступ к таблице, не обеспечивая при этом уникальности.

Можно создать индекс non-UNIQUE для столбца Last_Name таблицы EMPLOYEE, чтобы ускорить поиск по фамилиям сотрудников, но в этом случае в столбце может встречаться сколько угодно дубликатов для любой из фамилий.

Если в команде CREATE INDEX не задано никакого другого ключевого слова, для столбца автоматически создается индекс по неуникальному ключу в структуре B-дерева.

Индекс по реверсированному ключу

Индекс по реверсированному ключу (reverse key index) — особый вид индекса, который, как правило, применяется в средах OLTP (оперативной обработки транзакций). В таком индексе порядок данных ключа перед сохранением меняется на противоположный (реверсируется). Ключевое слово *reverse* определяет реверсированный порядок ключа индекса в команде *create index*. Ниже приведен пример создания индекса с реверсированным ключом:

- ```
  create index IE_LINE_ITEM_ORDER_NUMBER
       on LINE_ITEM(Order_Number) REVERSE;
  ```

Если размещается заказ с номером 123459, индекс с реверсированным ключом сохранит этот номер как 954321. Вставки записей с такими индексами будут распределены по всем ключам-листьям в индексе, сокращая число конфликтов между несколькими пользователями, пытающимися записать данные (вставить новые строки) в таблицу. Кроме того, индекс по реверсированному ключу уменьшает количество так называемых «горячих точек» в средах OLTP, если заказы модифицируются (или к ним делаются запросы) вскоре после размещения.

Индекс по ключу-функции

Индексы на базе функций похожи на стандартные индексы со структурой B-дерева. Отличие следующее: в этом индексе хранится некоторое преобразование столбца или столбцов на основе определенных пользователем выражений (функций), а не значение из самого столбца.

Индексы по ключу-функции могут быть полезны, например, когда какие-то названия или адреса хранятся в базе данных как смесь прописных и строчных букв (верхнего и нижнего регистров). Обычный индекс по столбцу, содержащему значение 'SmiTh' не возвратит никакого значения, если критерий поиска задан как 'Smith'. С другой стороны, если фамилии в индексе хранятся полностью в верхнем регистре (SMITH), то для всех поисков придется задавать фамилию в верхнем регистре. Ниже приведен пример создания индекса по ключу-функции для столбца Last_Name таблицы EMPLOYEE:

- ```
  create index up_name on employee (upper(Last_Name));
  ```

В результате для поисков с использованием запросов типа приве-

денного ниже, вместо полного просмотра таблицы будет использоваться только что созданный нами индекс:

- ```
  select Employee_Number, Last_Name, First_Name, from employee
  where upper(Last_Name) = 'SMITH';
  ```

Битовые индексы

По структуре битовые индексы отличаются от индекса со структурой B-дерева в узлах индекса. В битовом индексе хранится по одной строке битов для каждого возможного значения (кардинальности) подлежащего индексации столбца. Длина строки битов равна числу строк в подлежащей индексации таблице.

Помимо существенной экономии памяти по сравнению с традиционными индексами, битовый индекс может обеспечить значительное сокращение времени отклика, так как Oracle может быстро удалять потенциальные строки из запроса, содержащего несколько фраз *where,* задолго до того, как потребуется организовать доступ к самой таблице. В многочисленных битовых картах могут использоваться логические операторы *and* или *or* для определения строк, которые нужно отобрать из таблицы.

Хотя битовые индексы могут быть созданы для любого столбца таблицы, их применение наиболее эффективно, когда подлежащий индексации столбец имеет низкую кардинальность (число возможных значений). Для столбца Gender (пол) таблицы PERS возможны три значения: NULL (нет значения), F (женский) или M (мужской). Поэтому в битовом индексе по столбцу Gender будет храниться всего три битовые карты. Битовый индекс для столбца Last_Name будет содержать практически столько же строк (битовых карт), сколько строк в самой таблице. Вероятнее всего, запросы на получение конкретной фамилии из таблицы с использованием битового индекса потребуют столько же времени, если не больше, чем при полном просмотре таблицы. В этом случае имеет смысл создать традиционный индекс по неуникальному ключу со структурой B-дерева.

В вариации битовых индексов, которая называется битовым индексом соединения (bitmap join index), битовый индекс создается для столбца таблицы, по которому часто производится соединение данной таблицы с одной или несколькими другими таблицами. Это приводит к огромным преимуществам в средах хранилищ данных, где битовые индексы соединения создаются для таблицы фактов и одной или нескольких таблиц измерений, обеспечивая предварительное соединение этих таблиц и экономя ресурсы ЦП и ввода/вывода при выполнении фактического соединения.

ПРИМЕЧАНИЕ Битовые индексы доступны только для версии Enterprise Edition базы данных Oracle 11g.

  ```

Представления

Представления позволяют пользователям видеть специально подготовленное в соответствии с требованиями заказчика представление данных из одной или даже нескольких (используя соединение) таблиц. Представление известно также под именем хранимого запроса (stored query) — детали определяющего представление запроса обычно скрыты от пользователей представления. Однако обычное представление не содержит данные, а только определение запроса, и лежащий в основе представления запрос выполняется всякий раз, когда происходит обращение к представлению. Расширение обычных представлений, которое называется материализованным представлением (materialized view), позволяет для ускорения обработки вместе с определением запроса сохранить и его результаты. Помимо уже названного ускорения обработки у материализованных представлений есть и другие преимущества. Объектные представления, подобно традиционным представлениям, скрывают детали лежащих в их основе соединений таблиц и обеспечивают объектно-ориентированную разработку и обработку даже в тех случаях, когда самые главные таблицы остаются в традиционном реляционном формате.

В следующих разделах рассмотрим основы типов представлений обычного пользователя базы данных, разработчика и администратора баз данных, а также, как будут создаваться и использоваться базы данных.

Обычные представления

Для обычных представлений не требуется выделения какой-либо памяти. В словаре данных сохраняется только определение представления, или запрос. Таблицы запроса, составляющего основу представления, называются базовыми таблицами; каждая базовая таблица представления может быть впоследствии определена как представление.

Представления имеют множество преимуществ. Например, они скрывают сложность данных: старший аналитик может определить представление, в котором будут объединены данные из таблиц EMPLOYEE, DEPARTMENT и SALARY. Это делается с целью облегчить высшему руководству компании выборку информации о зарплате служащих с помощью оператора *select* из объекта, который будет казаться руководству таблицей. На самом деле это будет представлением, содержащим запрос, в котором выполняется объединение таблиц EMPLOYEE, DEPARTMENT и SALARY.

Представления могут быть использованы для обеспечения безопасности. Например, представление таблицы EMPLOYEE, которое называется EMP_INFO, может содержать все столбцы таблицы, за исключением столбца с зарплатой, причем это представление может быть реализовано в режиме *«только для чтения»,* чтобы предотвратить обновление таблицы: