

# Введение в Oracle

---

У Ч Е Б Н О Е   П О С О Б И Е

■ В. И. КОМАРОВ



# ОГЛАВЛЕНИЕ

|  |           |
|--|-----------|
| <b>ОСНОВНЫЕ ПОНЯТИЯ</b>                              | <b>3</b>  |
| Установка Oracle.....                                | 3         |
| Физическая организация данных.....                   | 4         |
| Логическая организация данных.....                   | 5         |
| Конкурентный доступ к данным.....                    | 6         |
| Манипулирование данными.....                         | 8         |
| <b>SQL. ОПРЕДЕЛЕНИЕ ДАННЫХ</b>                       | <b>9</b>  |
| Типы данных Oracle.....                              | 9         |
| Правила целостности.....                             | 14        |
| Таблицы.....   | 16        |
| Прочие объекты.....                                  | 19        |
| Объектные привилегии.....                            | 21        |
| Зависимости между объектами.....                     | 23        |
| <b>SQL. ВЫБОРКА ДАННЫХ</b>                           | <b>24</b> |
| Оператор выборки.....                                | 24        |
| Соединение таблиц.....                               | 30        |
| Множественные операции, группировка, сортировка..... | 34        |
| Иерархические запросы.....                           | 38        |
| <b>SQL. ИЗМЕНЕНИЕ ДАННЫХ</b>                         | <b>40</b> |
| Вставка данных.....                                  | 40        |
| Обновление данных.....                               | 41        |
| Удаление данных.....                                 | 42        |
| <b>ПРОЦЕДУРНОЕ РАСШИРЕНИЕ SQL — PL/SQL</b>           | <b>43</b> |
| Общие положения.....                                 | 43        |
| Типы данных PL/SQL.....                              | 43        |
| Операторы PL/SQL.....                                | 49        |
| Анонимные блоки PL/SQL.....                          | 51        |
| Хранимые процедуры.....                              | 54        |
| Пакеты.....  | 56        |
| Управление транзакциями.....                         | 58        |
| <b>ТРИГГЕРЫ</b>                                      | <b>60</b> |
| Создание триггеров.....                              | 60        |
| Выполнение триггеров.....                            | 60        |
| Особенности кода триггеров.....                      | 61        |
| <b>ОПТИМИЗАЦИЯ ЗАПРОСОВ</b>                          | <b>64</b> |
| Оптимизатор.....                                     | 64        |
| Подсказки оптимизатору (хинты).....                  | 66        |
| <b>ЛИТЕРАТУРА</b>                                    | <b>68</b> |

Настоящее пособие подготовлено на основе материалов спецкурса «Введение в Oracle», прочитанного автором на кафедре АСВК факультета Вычислительной математики и кибернетики Московского Государственного Университета им. М. В. Ломоносова в рамках спецсеминара «Объектно-ориентированные программные оболочки» под руководством доцента Гуляева А. В.

Пособие содержит краткое описание основных возможностей Oracle 8i, причем весь материал справедлив и для Oracle 7.x, если обратное не оговорено явно. Основной упор сделан на простоту и краткость изложения. Объем материала достаточен для того, чтобы освоивший его ориентировался в возможностях Oracle и мог включаться в разработку приложений, обращаясь в случае необходимости к фирменной документации. Все примеры запросов, приведенные в тексте, проверены и работают.

Пособие не может считаться полноценным учебником по Oracle — за кадром намеренно оставлены многие вещи. В частности, пособие не содержит материала о таких структурах управления памятью, как кластеры (clusters), многосекционные таблицы (partitioned tables) и моментальные копии (snapshots), о работе с большими объектами (LOBs), о нереляционных расширениях Oracle 8, таких как нескаллярные поля таблиц и объектные расширения PL/SQL.

Связаться с автором можно по адресу [vkomarov@yandex.ru](mailto:vkomarov@yandex.ru).

Пользуясь случаем, хочу выразить благодарность Потапову В. В. за ценные замечания и дополнения, высказанные им после прочтения первого варианта рукописи, а также всем моим коллегам по ОСАО «Ингосстрах», передавшим мне частичку своего богатого опыта.

Отдельное спасибо Рогову Е. В., проделавшему огромную работу по редактированию рукописи и проверке приведенных примеров кода.

# ОСНОВНЫЕ ПОНЯТИЯ

*В этой главе изложены основные понятия, которые часто будут употребляться в последующих главах. Предполагается, что читатель знаком с понятиями теории БД, такими как отношение (relation), строка (row), столбец (column). Если это не так, советуем обратиться к книгам [1] или [2].*

## Установка Oracle

Oracle Server (в дальнейшем Oracle) — классическая реляционная СУБД в архитектуре клиент-сервер. Правда, Oracle 8i приобрел многие функции, не свойственные реляционной СУБД, однако в настоящей книге эти функции не рассматриваются.

### Установка серверного ПО

Установка сервера Oracle не должна вызвать проблем. В корне CD-ROM находится файл *setup.exe*, запускающий Oracle Universal Installer — программу, устанавливающую продукты Oracle и ведущую собственный реестр установленных продуктов. Установка сервера занимает от 600 М до 1,3 Г на жестком диске. При установке можно выбрать одну из стандартных конфигураций установки (typical или minimal) или явно указать необходимые компоненты.

После копирования файлов на диск запускаются программы конфигурации сервера — Net 8 Configuration Assistant и Database Configuration Assistant.

Net 8 (в более ранних версиях SQL\*Net) — компонент Oracle, отвечающий за коммуникации между клиентом и сервером. Он скрывает от прикладных программ используемые сетевые протоколы. По умолчанию Net 8 работает с протоколами TCP/IP и Named pipes, но дополнительно можно установить поддержку множества других протоколов, например, IPX/SPX.

Net 8 Configuration Assistant позволяет сконфигурировать листенеры (listeners) — процессы, принимающие пользовательские обращения и соединяющие их с сервером БД. Если отключить листенеры, то сервер будет продолжать обслуживать уже установленные соединения, но установить новое соединение будет невозможно. Поскольку листенеры устанавливаются как отдельные службы Windows, их отключением можно пользоваться, чтобы запретить пользователям обращаться к серверу в аварийной ситуации. Правда, для этой цели есть и регулярный механизм Oracle — специальные сессии (restricted sessions).

Database Configuration Assistant позволяет создать экземпляр Oracle. Экземпляр (instance) называется совокупность процессов и общих областей памяти, обслуживающих БД. На одном сервере может быть запущено несколько экземпляров Oracle, но необходимость в этом возникает крайне редко. Кроме того, в распределенных системах одну БД могут обслуживать несколько экземпляров, запущенных на разных серверах.

При создании экземпляра вам придется ответить на несколько вопросов. Имя экземпляра — влияет только на настройку клиентских программ, традиционно экземпляр называют ORCL. Количество пользователей — от этого зависит конфигурация процесса-сервера (процесса, непосредственно обслуживающего клиентские запросы): если пользователей меньше 15, то для каждого клиентского соединения будет создаваться свой обслуживающий поток, в противном случае один поток будет обслуживать несколько соединений. После этого Database Configuration Assistant предложит параметры создаваемых дисковых файлов. Параметры по умолчанию вполне работоспособны, а конфигурирование параметров дисковой памяти выходит за рамки настоящей книги. Процесс создания экземпляра занимает достаточно много времени и сильно зависит от скорости дисков и объема оперативной памяти.

После окончания этой процедуры сервер готов к работе. Инсталлятор создает несколько служб (service) Windows. Для работы сервера необходимы только две из них — OracleOraHome8TNSListener (процесс-листер; имя состоит из имени листенера, задаваемого при установке, и суффикса TNSListener) и OracleServiceORCL (процесс-сервер; состоит из префикса OracleService и имени экземпляра).

### Установка клиентского ПО

Клиентское программное обеспечение устанавливается с того же компакт-диска, что и сервер, при помощи Oracle Universal Installer и занимает, в зависимости от варианта установки, от 30 до 100 мегабайт. Единственным необходимым для связи с сервером компонентом является Net 8 — сетевой интерфейс Oracle. Кроме него устанавливается Java-машина,

необходимая для запуска утилит настройки, использующих графический интерфейс. Также могут быть установлены SQL\*Plus – консоль БД, SDK – заголовки OCI (Oracle Call Interface), Application Wizard для Visual C++ и др. Если компьютер будет использоваться как рабочее место администратора БД, то устанавливаются также утилиты администрирования.

После установки клиентского ПО необходимо создать псевдоним (alias) – именованный набор параметров, описывающий конкретное соединение. Например, для соединения с сервером по протоколу TCP/IP следует указать IP-адрес сервера (словесный или цифровой), порт (как правило, 1521) и имя экземпляра (как правило, ORCL). Для управления псевдонимами используется утилита Net 8 Easy Config, входящая в комплект клиентского ПО. Информация о псевдонимах хранится в текстовом файле `net8\admin\tnsnames.ora` и может быть изменена в любом текстовом редакторе.

Сразу же после установки клиентского ПО целесообразно выполнить настройку национальных параметров:

- **NLS\_LANG** – язык сообщений сервера и кодовая страница, в которой сервер передает клиенту символьные (char, varchar2 и long) данные. Формат значения этой переменной таков:

Язык\_Территория.КодоваяСтраница

Для России Oracle по умолчанию ставит RUSSIAN\_CIS.CL8MSWIN1251, т. е. сообщения выдаются на русском языке, а символьные данные передаются в кодовой странице 1251. Полный список обозначений языков и кодовых страниц можно найти в [6]<sup>1</sup>. Если вам по каким-либо причинам удобнее читать сообщения сервера и обозначать месяцы, дни недели и т. п. по-английски, установите значение этой переменной в AMERICAN\_AMERICA.CL8MSWIN1251. В дальнейших примерах предполагается именно такая установка. Заметьте, что настройки языка, территории и кодовой страницы не зависят друг от друга. Вполне допустимо, например, значение AMERICAN\_CIS.CL8MSWIN1251;

- **NLS\_DATE\_FORMAT** – формат даты по умолчанию. Если не задавать этот параметр, то его значение соответствует территории, заданной параметром NLS\_LANG. Для территории “CIS” Oracle устанавливает формат по умолчанию 'dd.mm.rr', а для территории “AMERICA” – 'dd-mon-rr', т. е. две цифры обозначают год текущего, следующего или предыдущего столетия (см. главу «Определение данных»). Рекомендуется установить формат 'dd-mon-yy' или 'dd.mm.yy' (год указывается четырьмя цифрами).

Эти и другие переменные хранятся в реестре Windows как символьные значения ключа

HKEY\_LOCAL\_MACHINE\SOFTWARE\ORACLE\HOME0

Командой `alter session` можно изменять национальные настройки для отдельных сессий, а посмотреть настройки текущей сессии можно при помощи системного представления `V$NLS_PARAMETERS`:

```
select * from V$NLS_PARAMETERS
```

## Физическая организация данных

При создании экземпляра Oracle создает файлы для хранения данных. Один или несколько файлов образуют табличное пространство (tablespace). Администратор может добавить файл к табличному пространству или переименовать существующий файл (в т. ч. с переносом на другой диск). Управлять размещением объектов можно в табличных пространствах, но не в файлах.

Oracle должен иметь как минимум одно табличное пространство – SYSTEM, в котором хранится словарь данных. Кроме него при создании экземпляра создаются пространства TEMP (для временных объектов), RBS (для сегментов отката), USERS (для пользовательских таблиц) и INDX (для индексов).

Словарь данных – это набор таблиц, в которых хранится информация о пользовательских объектах. Изменять данные в этих таблицах можно только командами DDL (Data Definition Language) и командами управления сервером. Для более наглядного отображения словаря данных существуют специальные представления, начинающиеся с префиксов USER\_ (объекты текущего пользователя), ALL\_ (все объекты) или DBA\_ (объекты, принадлежащие администраторам). Например, получить информацию о своих таблицах можно получить из представления USER\_TABLES, информацию обо всех пользовательских таблицах – из пред-

<sup>1</sup> Другие русские кодовые страницы – RU8PC866 («альтернативная» кодировка DOS, 866 кодовая страница), CL8ISO8859P5 (ISO-8859-5) и CL8KOI8R (КОИ-8). Все они однобайтовые и включают в себя набор ASCII.

ставления `ALL_TABLES`, а информацию о системных таблицах, в которых хранится словарь данных, — из представления `DBA_TABLES`. Информация о правилах целостности доступна через представление `USER_CONSTRAINTS`, а исходные тексты хранимых процедур — через представление `USER_SOURCE`. Кроме того, в Oracle существуют представления, через которые доступна динамическая информация о текущем состоянии сервера — их имена начинаются с `V$`. Например, информацию об открытых сессиях можно получить через представление `V$SESSION`. По умолчанию пользователь не имеет права читать представления, начинающиеся с `DBA_` и `V$`. Более подробную информацию о системных представлениях можно найти в [7].

Все табличные пространства (и файлы) разделены на блоки. Размер блока задается при создании экземпляра. Он кратен размеру блока операционной системы и имеет ограничение сверху. При расширении объектов дисковая память под них выделяется экстендами (extent) — наборами подряд идущих блоков. Набор экстендов называется сегментом (segment). Для каждого объекта выделяется один сегмент. Сегменты растут, но сами собой не уменьшаются. Если, например, из таблицы удалено много строк, то физически она будет занимать столько же места, сколько до удаления. Освободить занимаемое ею место можно либо удалив таблицу (`drop table`), либо удалив все данные таблицы (`truncate table`), либо принудительно освободив неиспользуемое пространство (`alter table ... deallocate unused`). В последнем случае будут освобождены только те экстенды, в которых нет ни одной строки.

В Oracle существует 4 вида сегментов — сегменты данных (таблиц), индексов, отката и временные сегменты. Временные сегменты используются для сортировки данных при выполнении операций `create index` и выборке данных, требующей сортировки (с фразами `order by`, `group by`, `union`, `distinct` и др.). Сегменты отката (rollback segment) используются для обеспечения конкурентного доступа к данным.

## Логическая организация данных

### Пользователи и схемы

Каждая база данных имеет список пользователей. Для того, чтобы начать работу, сервер должен идентифицировать пользователя. В Oracle возможно два способа идентификации — при помощи пароля, хранимого в самом Oracle, либо при помощи механизма авторизации, встроенного в операционную систему.

Каждый пользователь может хранить в базе данных свои объекты — таблицы (table), представления (view), индексы (index), хранимые процедуры (stored programs) — процедуры (procedure), функции (function) и пакеты (package), синонимы (synonym), последовательности (sequence), триггеры (trigger), а также типы (type), моментальные копии (snapshot или materialized view) и другие объекты. Совокупность всех объектов, принадлежащих пользователю, называется схемой. Имя схемы совпадает с именем пользователя.

Таблицы — основные элементы БД. В них физически находятся все данные пользователя.

Представление — проекция данных, находящихся в одной или нескольких физических таблицах или других представлениях. Представления создаются при помощи SQL-запросов, хранимых в базе данных. Представления можно использовать почти везде, где используются таблицы.

Индексы — специальные структуры на основе таблиц, служащие для ускорения доступа к данным. Индексы никак не влияют на функциональность, они могут создаваться и уничтожаться пользователем в любой момент, однако правильно построенный индекс может значительно уменьшить время выполнения запроса. Кроме того, индексы используются для поддержания правил целостности.

Хранимые процедуры — программы на алгоритмическом языке (PL/SQL, а в Oracle 8i — и Java), которые хранятся и выполняются на сервере. К хранимым процедурам относятся процедуры, функции и пакеты, а также триггеры — процедуры, выполняемые сервером автоматически при наступлении каких-либо событий, например, при изменении данных в таблицах.

Последовательности — механизм, предназначенный для генерации уникальных чисел, которые могут быть использованы при создании первичного ключа. Последовательности никак не связаны с таблицами, поэтому одна и та же последовательность может быть использована для генерации ключей нескольких таблиц. Все значения, сгенерированные с помощью последовательности, будут различными, хотя и не обязательно идущими подряд.

Синоним — дополнительное имя объекта. Синонимы используются для упрощения ко-

манд SQL и маскировки реального имени, владельца и местонахождения объекта. Синонимы бывают частными (private) и общими (public). Частный синоним является объектом схемы, общий доступен всем пользователем и создается администратором базы данных. Примером общего синонима может служить dual.

Полное имя объекта состоит из имени схемы, имени самого объекта и, если объект расположен на другом сервере, имени сервера:

```
schema.object@dblink
```

Например:

```
employee -- объект employee в текущей схеме
bank.course -- объект course в схеме bank
dicti.cities@srv16 -- объект cities в схеме dicti на сервере srv16
```

В Oracle действует следующее правило разрешения имен:

1. объект в текущей схеме;
2. частный синоним;
3. общий синоним;
4. другая схема и поиск в ней объекта после точки.

Если взять в качестве примера имя dicti.cities, то сначала Oracle будет искать в текущей схеме объект (таблицу, представление, хранимую процедуру, пакет) с именем dicti; если такой объект не найден, будет искаться синоним dicti в текущей схеме; затем будет искаться синоним dicti в схеме public; затем будет искаться схема dicti и в ней — объект cities.

## Защита данных

Каждый пользователь имеет свой домен защиты — набор характеристик, определяющих возможности пользователя. К этим характеристикам относятся квоты и привилегии.

Квоты (quota) ограничивают количество ресурсов, которые могут потребляться пользователем. Квоты могут ограничивать использование дискового пространства, количество одновременно открываемых сессий, длительность сессий, время простоя сессии. Кроме того, можно ограничить процессорное время и количество операций ввода/вывода как на уровне сессии, так и на уровне команды.

Привилегии (privilege) — это права на выполнение конкретных команд. Привилегии делятся на объектные и системные. Системные привилегии позволяют выполнять действия над экземпляром (например, соединиться с сервером БД, т. е. создать сессию) или над некоторым типом объектов (например, создавать таблицы). Системные привилегии выдаются администратором БД. Объектные привилегии позволяют выполнять конкретные действия над конкретными объектами, например, выполнять процедуру или выбирать строки из таблицы. Объектные привилегии выдаются владельцами этих объектов. Для обращения к объекту требуются объектные привилегии, выданные на этот объект, или системные привилегии на класс объектов. Например, чтобы выполнить некоторую процедуру, надо иметь объектную привилегию EXECUTE на эту процедуру или системную привилегию EXECUTE ANY PROCEDURE.

Для более гибкого управления привилегиями в Oracle существуют роли (role) — именованные наборы привилегий. Привилегия может быть выдана конкретному пользователю, а может — роли, которая, в свою очередь, выдается пользователю.

При установке Oracle создается набор предопределенных ролей. При создании нового пользователя в Oracle 7.x необходимо выдать ему роли CONNECT (создание сессий) и RESOURCE (создание объектов). Oracle 8 также создает указанные роли при установке, однако в следующих версиях возможен отказ от них.

## Конкурентный доступ к данным

### Команды, транзакции, сессии

Общение пользователя с сервером происходит при помощи команд SQL и PL/SQL. Одна или несколько команд образуют транзакцию (transaction) — неделимую с точки зрения воздействия на БД операцию манипулирования данными. Либо результаты всех команд, входящих в транзакцию, отображаются в БД, либо воздействие всех этих команд на БД полностью отсутствует. Если к моменту выполнения команды транзакция не была открыта, Oracle открывает ее автоматически. Транзакция может быть завершена явно с помощью команд управления транзакциями либо неявно — в результате выполнения нетранзакционных команд.

Соединение пользователя с сервером называется сессией<sup>1</sup> (session). За сессию пользователь может провести ноль или больше транзакций. Пользователь может одновременно открывать несколько сессий с одним и тем же экземпляром, если у него есть такая привилегия.

### Изоляция транзакций

Стандарт SQL92 определяет четыре уровня изоляции транзакций. Их подробное описание можно найти в [3] или в [5].

В Oracle реализовано два уровня изоляции — READ COMMITTED и SERIALIZABLE. При уровне READ COMMITTED команда читает только те данные, которые были зафиксированы (committed) к моменту начала выполнения команды. Вторая команда в той же транзакции может получить другой результат. При уровне SERIALIZABLE все команды видят изменения, либо завершённые к моменту начала транзакции, либо сделанные в этой транзакции. Кроме того, транзакция может быть помечена как READ ONLY. Такая транзакция будет видеть только те изменения, которые были завершены до ее начала. Ни при каких условиях транзакция не может увидеть изменения, сделанные другой незавершённой транзакцией, т. е. уровень READ UNCOMMITTED, допускающий чтение незафиксированных данных (dirty read), в Oracle не поддерживается.

Администратор задает уровень изоляции по умолчанию для всей базы данных. Однако программист может изменить уровень изоляции для сессии и даже для отдельной транзакции.

Если сериализуемая транзакция пытается обновить или удалить строки, модифицированные другими транзакциями, закончившимися после ее начала, возникает ошибка сериализации:

```
ORA-08177: can't serialize access for this transaction
```

### Методы изоляции пользователей

Для обеспечения совместной работы пользователей в Oracle применяются блокировки и поддержка нескольких версий данных.

Блокировка строк происходит автоматически при изменении данных командами insert, update и delete. Oracle никогда не осуществляет эскалацию блокировок, т. е. сколько бы строк ни заблокировала транзакция, это не приводит к монопольной блокировке всей таблицы. Таблицы при выполнении этих операций блокируются в разделяемом (shared) режиме.

Команды DDL (Data Definition Language), такие как alter table, create procedure и т. д. также блокируют таблицы. Команды drop table и alter table требуют монопольной блокировки таблицы, т. е. во время выполнения этих команд ни одна другая сессия получить доступ к таблице не может. Команды create procedure, create trigger и т. п. блокируют в разделяемом режиме таблицы, используемые в создаваемых процедурах или триггерах, т. е. другие транзакции могут читать и изменять данные в таблицах, но Oracle гарантирует, что структура таблиц останется неизменной на протяжении всего времени выполнения команды. Если к моменту выполнения команды DDL была открыта транзакция, Oracle автоматически фиксирует ее. После выполнения команды DDL Oracle также подтверждает транзакцию.

Кроме автоматических блокировок, Oracle позволяет блокировать объекты вручную. Для блокировки строк используется команда select с фразой for update, для блокировки таблиц — команда lock table. В большинстве случаев ручная блокировка данных не требуется.

Обратите внимание, что выборка не блокирует прочитанные строки, а измененные данные по-прежнему доступны для чтения в других транзакциях. Транзакция ждет лишь в том случае, если она пытается изменить строку, измененную в другой незавершённой транзакции. Это достигается при помощи механизма, обеспечивающего несколько версий одних и тех же данных.

Для обеспечения такой целостности Oracle использует сегменты отката (rollback segment). При выполнении команд DML (Data Manipulation Language) изменения пишутся непосредственно в БД, а старые версии данных выталкиваются в сегменты отката. Транзакции (или команды, в зависимости от уровня изоляции), начавшиеся раньше этой команды, будут читать старые версии из сегментов отката. В случае отката транзакции данные из сегментов отката будут переписаны обратно в базу данных. Поэтому откат транзакции занимает почти столько же времени, сколько сама транзакция, в то время как фиксация транзакции происходит практически мгновенно, т. к. при этом Oracle только отмечает, что сегменты

---

<sup>1</sup> В переводе фирменной документации по Oracle, выполненном фирмой РДТех, используется термин «сеанс».

отката данной транзакции больше не нужны. Сегменты отката освобождаются в тот момент, когда завершится занявшая их транзакция и все транзакции, начавшиеся раньше. Если во время интенсивного обновления базы выполняется длительный запрос, может случиться, что все сегменты отката будут заняты, и Oracle не сможет обеспечить целостность данных. При этом возникает ошибка

```
ORA-1555: snapshot too old: rollback segment number <номер> with name "<имя>" too small
```

Избежать этого можно, разбив длинную транзакцию на несколько коротких, создав дополнительные сегменты отката или выполнив ручную блокировку данных.

## Манипулирование данными

Для доступа к данным в Oracle реализован транслятор языка SQL. Реализация SQL в Oracle соответствует начальному (entry) уровню стандарта SQL92. Кроме возможностей, описанных в стандарте, SQL в Oracle имеет много нестандартных фирменных расширений. Существует режим, в котором Oracle будет предупреждать о несоответствии выполняемой команды стандарту.

Все команды SQL делятся на несколько групп. Команды DML (Data Manipulation Language) позволяют читать и изменять данные в существующих объектах. К ним относятся такие команды, как `select`, `insert`, `update`, `delete`, `lock table` и `explain plan`.

Команды управления транзакциями позволяют группировать несколько команд в одну транзакцию, а также устанавливать параметры транзакции. К ним относятся команды `commit`, `rollback`, `savepoint` и `set transaction`.

Команды DDL (Data Definition Language) определяют и изменяют структуру объектов БД, а также уничтожают эти объекты. К ним относятся команды создания, изменения и удаления объектов (`create`, `alter` и `drop`), переименования (`rename`), удаления данных без уничтожения структуры (`truncate`), сбора статистики (`analyze`), выдачи и отбора привилегий (`grant` и `revoke`), управления протоколированием (`audit` и `noaudit`) и добавления комментариев к объектам схемы (`comment`).

Команды управления сессией позволяют изменять параметры конкретной сессии. Это команды `alter session` (изменение параметров) и `set role` (включение и выключение ролей для текущей сессии).

Команда управления системой `alter system` предназначена для управления экземпляром. Она позволяет принудительно закрывать сессии, изменять режим ведения журналов, приостанавливать работу БД и выполнять некоторые другие административные операции.

Кроме транслятора SQL в Oracle входит PL/SQL-машина (PL/SQL-engine) – транслятор фирменного процедурного расширения SQL. PL/SQL – полноценный процедурный язык, включающий в себя операторы ветвления и циклов, позволяющий обрабатывать исключительные ситуации и представляющий возможности управления курсорами. Oracle позволяет создавать хранимые процедуры и триггеры на PL/SQL и хранит их откомпилированный код, что позволяет существенно упростить логику клиентской части приложения, уменьшает сетевой трафик и повышает производительность. Для выполнения команд SQL PL/SQL-машина обращается к транслятору SQL, однако на практике оказывается, что не каждая SQL-команда может быть выполнена из PL/SQL.

# SQL. Определение данных

*Эта глава рассказывает о типах данных Oracle и правилах преобразования типов, о структуре и предназначении объектов Oracle, а также о том, как создавать и изменять эти объекты. Значительное внимание уделено правилам логической целостности данных.*

## Типы данных Oracle

Oracle, как и любая реляционная СУБД, позволяет хранить в таблицах баз данных алфавитно-цифровую и двоичную информацию. Кроме хранения данных скалярных типов, в Oracle 8 появилась возможность создавать свои типы данных (классы, массивы и коллекции) и сохранять объекты этих типов в таблицах БД. В настоящем пособии эти новые возможности не рассматриваются.

### Скалярные типы данных

Oracle поддерживает следующие типы данных:

- `char` — строки фиксированной длины;
- `varchar2` — строки переменной длины;
- `nchar` — строки, содержащие национальные символы, фиксированной длины;
- `nvarchar2` — строки, содержащие национальные символы, переменной длины;
- `number` — числа;
- `date` — дата и время;
- `raw` — двоичная информация фиксированной длины;
- `long` — текстовая информация большого (до 4 Г<sup>1</sup>) объема;
- `long raw` — двоичная информация большого (до 4 Г) объема;
- `bfile` — внешний файл;
- `blob` — двоичная информация, хранимая отдельно от таблицы;
- `clob` — символьная информация, хранимая отдельно от таблицы;
- `nclob` — символьная информация с национальными символами, хранимая отдельно от таблицы;
- `rowid` — физический адрес строки в таблице.

Столбцы типа `char` и `varchar2` позволяют хранить символьные строки длиной до 2000 и 4000 символов соответственно<sup>2</sup>. Данные типа `varchar2` занимают на диске ровно столько места, сколько необходимо для их хранения. Использование `varchar2` вместо `char` может значительно ускорить работу приложения и сэкономить дисковое пространство.

Oracle использует разную семантику для сравнения значений типа `char` и `varchar2` — с добавлением и без добавления пробелов в конце соответственно: в первом случае строки 'a' и 'a ' равны, во втором — вторая строка больше первой. `Varchar` является синонимом `varchar2`, однако Oracle предупреждает, что в более поздних версиях для данных типа `varchar` может использоваться другая семантика сравнения, поэтому в приложениях следует использовать `varchar2`.

`Nchar` и `nvarchar2` предназначены для хранения символьных строк, содержащих национальные (многобайтовые) символы<sup>3</sup>. Их длина не может превышать 2000 и 4000 байт соответственно<sup>4</sup>. Длина строк указывается в символах или в байтах, в зависимости от кодовой страницы, указанной при создании базы данных. Если размер символа постоянный, то длина указывается в символах, если переменный — в байтах. Переменные типов `char` и `nchar` (`varchar2` и `nvarchar2`) несовместимы по присваиванию.

Численные значения хранятся в полях типа `number`. Максимальная точность хранения — 38 знаков. Значением поля типа `number` могут быть положительные числа от  $10^{-130}$  до  $9.99.99 \times 10^{125}$ , отрицательные числа от  $-9.99.99 \times 10^{125}$  до  $-10^{-130}$ , а также 0. При желании для поля типа `number` можно задавать количество цифр и точность (количество знаков после запятой). По умолчанию Oracle хранит столько знаков после запятой, сколько задано при вводе, а количество цифр максимально, т. е. составляет 38 знаков. Можно задавать точность,

<sup>1</sup> В Oracle 7.x максимальный размер полей типа `long` и `long raw` равен 2 Г.

<sup>2</sup> В Oracle 7.x максимальная длина `char` — 255 байт, а `varchar2` — 2000 байт.

<sup>3</sup> Однобайтовые национальные символы (например, в кодировке windows-1251) могут храниться в полях типа `char/varchar2`.

<sup>4</sup> В Oracle 7.x типы данных `nchar` и `nvarchar2` отсутствуют.

не ограничивая количество цифр — в этом случае вместо количества цифр ставится \*. Например, вот как будет храниться число 123456.78 при задании разных форматов поля:

|               |   |
|---------------|---|
| number        | 123456.78   |
| number (*,1)  | 123456.8  |
| number (9)    | 123457  |
| number (9,2)  | 123456.78   |
| number (5)    | ошибка! Максимальное число, которое можно записать в такое поле — 99999 |
| number (4,-2) | 123500  |

При указании отрицательной точности числа округляются до соответствующего количества знаков до запятой. При выводе чисел Oracle по умолчанию использует в качестве разделителя целой и дробной части точку. Изменить этот символ можно при помощи изменения параметра NLS\_NUMERIC\_CHARACTERS — в реестре Windows для экземпляра или для всех сессий на клиентской машине, либо командой `alter session` для конкретной сессии.

Поля типа `date` хранят информацию о дате и времени с точностью до секунды. Oracle поддерживает арифметику дат. К дате можно добавлять числа — число интерпретируется как количество дней. Так, например, выражение `sysdate+1` означает «завтра в то же время», `sysdate-0.5` — 12 часов назад. Складывать даты нельзя, но можно вычитать их друг из друга — в результате получается число, не обязательно целое, показывающее, сколько дней (суток) прошло между двумя датами. Если по каким-либо причинам надо хранить время отдельно от даты, лучше представлять время числом — это упростит вычисления.

Поля типа `raw` используются для хранения двоичной информации. Максимальная длина поля типа `raw` составляет 2000 байт<sup>1</sup>. В отличие от `char` и `varchar2`, при передаче от сервера клиенту поля типа `raw` не подвергаются никаким преобразованиям.

Поля типа `long` и `long raw` используются для хранения больших (до 4 Г) объемов соответственно текстовой и двоичной информации. При передаче данных типа `long` SQL\*Net автоматически выполняет преобразование кодовых страниц (как для `char` и `varchar2`); при передаче данных типа `long raw` преобразование не выполняется (как для `raw`). На столбцы типа `long` и `long raw` накладывается ряд ограничений: в таблице может быть только один столбец типа `long` или `long raw`; столбцы типа `long` или `long raw` нельзя индексировать, нельзя использовать в функциях и выражениях; нельзя использовать во фразах `where`, `group by`, `order by`, `connect by` и `distinct`; нельзя использовать в правилах целостности; нельзя использовать в списках выборки в операторе `select`, если он объединен с другим оператором посредством `union`, `intersect` или `minus` или является частью оператора `create table ... as select`. Допускается выборка полей типа `long` и `long raw` при объединении результатов запросов посредством `union all`.

Типы `blob` (Binary Large Object), `clob` (Character Large Object), `nclob` (National Character Large Object) и `bfile` (Binary file) также предназначены для хранения больших объектов размером до 4 Г<sup>2</sup>. Данные типов `blob`, `clob` и `nclob` хранятся в файлах БД в специальных LOB-сегментах, а данные типа `bfile` — во внешнем файле. Основное отличие LOB-типов от `long` и `long raw` в том, что при выборке поля оператором `select` возвращается не сам объект, а локатор — указатель на объект и позиция в нем. Для доступа к данным используются функции из пакета `DBMS_LOB` или вызовы OCI (Oracle Call Interface).

Кроме перечисленных, Oracle поддерживает ряд обозначений типов, соответствующих стандарту ANSI или обозначениям, принятым в DB2. При создании таблиц эти типы преобразуются во внутренние типы:

| ANSI/DB2   | Oracle                   |
|--|--------------------------|
| <code>character(n)</code>  | <code>char(n)</code>     |
| <code>character varying(n)</code> , <code>char varying(n)</code> , <code>varchar(n)</code> | <code>varchar2(n)</code> |
| <code>float(p)</code> , <code>real</code> , <code>double precision</code>                  | <code>number</code>      |
| <code>integer</code> , <code>int</code> , <code>smallint</code>                            | <code>number(38)</code>  |
| <code>decimal(p,s)</code> , <code>numeric(p,s)</code>                                      | <code>number(p,s)</code> |
| <code>long varchar</code>  | <code>long</code>        |

<sup>1</sup> В Oracle 7.x максимальная длина поля типа `raw` составляет 255 байт.

<sup>2</sup> В Oracle 7.x перечисленные типы отсутствуют.

Типы `time` и `timestamp` в Oracle не поддерживаются — вместо них следует использовать тип `date`.

### Тип `rowid`

Каждая строка таблицы имеет уникальный идентификатор, называемый `rowid` (ROW Identifier). В случае кластеризованных (храняемых вместе) таблиц строки разных таблиц, хранящиеся в одном блоке, могут иметь одинаковый `rowid`. `Rowid` является физическим адресом начала строки данных. Выборка по `rowid` — самый быстрый способ доступа к строке таблицы. Индексы представляют собой таблицы соответствия значений ключевых полей конкретным `rowid`.

При выборке `rowid` Oracle 8 возвращает строку из 18 символов — этот формат называется “extended `rowid`”. Для того, чтобы получить из extended `rowid` информацию о физическом расположении строки, необходимо воспользоваться функциями пакета `DBMS_ROWID`. Oracle 7.x при выборке `rowid` возвращает строку, состоящую из трех чисел, разделенных точкой, — номера файла, номера блока файла и номера строки в блоке. Этот формат в Oracle 8.x называется “restricted `rowid`” и поддерживается лишь для совместимости. Однако не следует рассматривать `rowid` как строку символов определенного формата. `Rowid` — совершенно произвольная структура данных, которая может изменяться от версии к версии, мало того, не существует стандарта на представление `rowid` в виде строки.

Каждая строка каждой таблицы имеет псевдостолбец под названием `rowid`. Псевдостолбец называется так потому, что он физически нигде не хранится, однако может быть извлечен командой `select`. Например:

```
select isn, rowid from employee

isn      rowid
-----
1        AAAAwPAADAAAAADAAA
4        AAAAwPAADAAAAADAAC
```

К `rowid` применимы операции сравнения (`=`, `<>`, `<`, `>`), в частности, результат запроса может быть отсортирован по значению `rowid`, хотя это и не имеет практического смысла.

`Rowid` строки традиционной таблицы остается постоянным с момента создания. Кроме того, в Oracle существуют индексные таблицы (index-organized tables), данные которых хранятся только в ассоциированных с таблицей индексах. Такая таблица обязана иметь первичный ключ, а `rowid` строки в такой таблице остается постоянным, пока не изменен ее первичный ключ.

Если строка удалена, ее `rowid` может быть назначен другой строке. При выгрузке и загрузке БД утилитами `import` и `export` `rowid` не сохраняется. Эти два свойства не позволяют использовать `rowid` в качестве первичного ключа таблицы.

Тем не менее, можно создавать таблицы с полями типа `rowid` и использовать `rowid` для ссылок между таблицами. Однако программист должен обеспечить корректность `rowid` на время выполнения команды, т. е. строки с `rowid`, участвующими в соединении, не должны быть удалены. Выборка по несуществующему `rowid` приводит к ошибке:

```
ORA-01410: invalid ROWID
```

### Неопределенное значение (`null`)

Возможна ситуация, когда значение поля не определено. В этом случае в поле записывается специальное «неопределенное» значение — `null`. К. Дейт в [2] приводит ряд возражений против использования неопределенных значений, однако в реальных базах данных `null` используется весьма широко. Будьте осторожны при использовании неопределенных значений, т. к. значением любого выражения, в котором используется `null`, будет `null`:

```
<число>+null = null
<число>-null = null
<число>/null = null
<число>*null = null
null/<число> = null
<значение> (= | <> | > | < | <= | >=) null = null
false or null = null
true and null = null
```

Из этого правила есть исключения:

```
<строка> || null = <строка>
null || <строка> = <строка>
true or null = true
false and null = false
```

## Преобразование типов

Oracle позволяет преобразовывать данные от одного типа к другому как явно, используя специальные функции преобразования типов, так и неявно, указывая данные иного типа, чем ожидалось по контексту.

К функциям преобразования типа относятся:

- `to_number (<строка>[, <формат>[, <национальные настройки>]]);`
- `to_date (<строка>[, <формат>[, <национальные настройки>]]);`
- `to_char (<дата/время>[, <формат>[, <национальные настройки>]]);`
- `rawtohex (<двоичные данные>);`
- `hextoraw (<строка>);`
- `rowidtochar (<rowid>);`
- `chartorowid (<строка>).`

Функция `to_number` предназначена для преобразования символьных строк в числа. Первым параметром указывается строка, подлежащая преобразованию, вторым — форматная строка. Oracle допускает более 20 различных символов для спецификации формата; их полный список можно найти в [8].

Третий параметр — национальные настройки — позволяет изменить символы, разделяющие группы разрядов и отделяющие целую часть от дробной, язык (для форматирования дат) и другие настройки, связанные с принятой в языке системой обозначений. В подавляющем большинстве случаев достаточно задания переменных `NLS_LANG` и `NLS_DATE_FORMAT`, про которые говорилось в главе 1.

Функция `to_date` предназначена для преобразования строки в дату и время. Для форматирования дат используются следующие символы:

| Символ            | Значение  |
|-------------------|---|
| ., - / : ;        | знаки препинания воспринимаются буквально   |
| D                 | день недели, от 1 до 7, в зависимости от территории:<br>“AMERICA” — воскресенье=1, понедельник=2, суббота=7<br>“CIS” — понедельник=1, суббота=6, воскресенье=7  |
| DD                | день месяца*  |
| DDD               | день года, от 1 до 366  |
| DY                | сокращенное название дня из трех букв, в зависимости от языка, например:<br>“AMERICAN” — «WED»<br>“RUSSIAN” — «СРД»   |
| DAY               | полное название дня недели, дополненное пробелами до максимального количества символов** в названии дня недели в текущем языке, например:<br>“AMERICAN” — «WEDNESDAY»<br>“RUSSIAN” — «СРЕДА » (6 дополнительных пробелов)               |
| HH, HH12          | количество часов, от 01 до 12*  |
| AM, A.M. PM, P.M. | индикатор времени суток, с точками или без. В функции <code>to_date</code> используйте 'AM' или 'A.M.'  |
| HH24              | количество часов, от 00 до 23*  |
| MI                | количество минут, от 00 до 59   |
| MM                | месяц, от 01 до 12*   |
| MON               | сокращенное название месяца из трех букв, в зависимости от языка, например:<br>“AMERICAN” — «JUL»<br>“RUSSIAN” — «ИЮЛ»  |
| MONTH             | полное название месяца, дополненное пробелами до максимального количества символов** в названии месяца в текущем языке, например:<br>“AMERICAN” — «JULY » (5 дополнительных пробелов)<br>“RUSSIAN” — «ИЮЛЬ » (4 дополнительных пробела) |
| RM                | месяц римскими цифрами, от I до XII   |

|            |  |
|------------|--|
| RR         | <p>две цифры года</p> <ul style="list-style-type: none"> <li>■ текущего столетия, если число меньше 50 и две последних цифры текущего года меньше 50 или число больше или равно 50 и две последних цифры текущего года больше или равны 50;</li> <li>■ предыдущего столетия, если число больше или равно 50 и две последних цифры текущего года меньше 50;</li> <li>■ следующего столетия, если число меньше 50 и две последних цифры текущего года больше или равны 50.</li> </ul> <p>Например, «92» будет обозначать «1992» с 1950 по 2049 годы и «2092» с 2050 по 2149 годы<br/>«05» будет обозначать «2005» с 1950 по 2049 годы и «2105» с 2050 по 2149 годы</p> |
| SS         | количество секунд, от 00 до 59*  |
| Y, YY, YYY | последние 1, 2 или 3 цифры года  |
| YYYY       | год  |

\* При преобразовании из строки в дату допускается пропуск лидирующего нуля. При преобразовании даты в строку всегда при необходимости добавляется лидирующий нуль.

\*\* При преобразовании строки в дату можно опустить лишние пробелы. При преобразовании даты в строку пробелы добавляются всегда.

Полный список символов, используемых при форматировании дат, можно найти в [8].

Функция `to_char` является обратной по отношению к `to_date` и `to_number`. Ее первым аргументом является дата или число, вторым — форматная маска. Результатом функции будет символьное представление даты или числа с учетом формата:

```
select to_char (sysdate, 'dd-mon-yyyy hh24:mm:ss') as now from dual
```

```
now
```

```
-----
21-aug-2001 15:08:40
```

```
select to_char (sysdate, 'dd-Mon-rr hh12:mm', 'nls_date_language=RUSSIAN') as now from dual
```

```
now
```

```
-----
21-Авг-01 03:08
```

Конструкция `select ... from dual` используется, когда надо вычислить значение функции, не производя при этом выборки данных. Dual — это системная таблица, состоящая из одного столбца и содержащая одну строку, что гарантирует, что запрос из этой таблицы также вернет одну строку. Обратите также внимание, что на результат оказывает влияние регистр символов форматирования.

Функции `rawtohex` и `hexraw` предназначены для преобразования значений типа `raw` или `long raw` в символьные (`char` и `varchar2`) строки. Каждый байт кодируется двумя шестнадцатеричными цифрами.

Функции `chartorowid` и `rowidtochar` предназначены для преобразования `rowid` в строку и обратно.

В некоторых случаях Oracle преобразует данные от одного типа к другому неявно, т. е. без вызова функций преобразования. Ниже перечислены эти случаи:

- `date` в `varchar2`. Дата/время формируются в соответствии с форматом по умолчанию (заданным параметром `NLS_DATE_FORMAT`);
- `char` и `varchar2` в `date`. Строка распознается как дата/время, если она является записью даты/времени в формате по умолчанию;
- `number` в `varchar`. Число преобразуется в строку следующим образом: целая часть записывается без разделителей, а дробная часть отделяется символом, заданным параметром `NLS_NUMERIC_CHARACTERS`;
- `char` и `varchar` в `number`. Строка преобразуется в число, если она является записью числа в формате по умолчанию — целая часть записана без разделителей, а дробная часть отделена символом, заданным параметром `NLS_NUMERIC_CHARACTERS`;
- `char` и `varchar2` в `rowid` и `rowid` в `varchar2` — автоматически вызываются функции `chartorowid` или `rowidtochar`;
- `char` и `varchar2` в `raw` и `long raw` — автоматически вызывается `hexraw`.

Пусть в таблице `employee` есть поля `isn` типа `number`, `salary` типа `number`, `hiredate` типа `date` и `name` типа `varchar2`. В таком случае все приведенные ниже команды корректны:

```
update employee set hiredate='1-Jan-2001' where isn='33'
```

```
update employee set hiredate=to_date('2001.01.01', 'yyyy.mm.dd') where isn=33
select * from employee where salary>'400'
select name || ' ' || hiredate || ' ' || salary from employee
```

Применять преобразования по умолчанию чисел и дат в строки и обратно не рекомендуется, т. к. во-первых, правила этих преобразований зависят от национальных настроек и могут измениться в следующих версиях Oracle, а во-вторых, преобразования по умолчанию могут сильно сказаться на производительности.

## Правила целостности

Oracle позволяет задать условия, которым должны удовлетворять все данные в БД. Условия задаются в декларативной форме, т. е. программист лишь задает условие, а сервер обеспечивает его проверку при любом изменении данных. Такие условия называются правилами целостности (integrity constraint).

В Oracle поддерживаются четыре правила целостности, налагающих ограничения на данные в пределах одной таблицы: запрет неопределенных значений (not null), проверка условия (check), уникальность значений (unique) и первичный ключ (primary key). Кроме того, для обеспечения ссылочной целостности существуют внешние ключи (foreign key).

### Проверка условия

Правило not null запрещает использование неопределенных значений. Если столбец объявлен как not null, данные в этом столбце не могут содержать значение null.

Более сложные правила целостности можно определить при помощи предложения check. Эти правила представляют собой логические выражения, которые должны выполняться для каждой строки таблицы. Они могут включать в себя выражения, содержащие значения столбцов, операции над ними и вызовы встроенных функций. Выражения в правилах check не могут содержать вызовов пользовательских хранимых функций и операций select.

Пусть, например, в таблице employee определены поля salary, hiredate и firedate. Тогда можно было бы ввести следующие правила check:

```
salary>100 -- минимальная зарплата
hiredate>='1-Jan-2001' -- защита от неправильного ввода
firedate>hiredate -- увольнение позже приема
```

Обратите внимание, что Oracle проверяет не положительный результат проверки, а отсутствие отрицательного результата. Это значит, что если для поля задано правило salary>100, то это поле может принимать значение null, т. к. результатом любого сравнения с null (кроме is [not] null) будет null, а не true или false.

### Уникальность значений

Согласно правилу unique, в таблице не может существовать двух строк с одинаковыми значениями столбца или набора столбцов. При этом неопределенные значения столбцов не считаются одинаковыми, т. е. две и более строк могут содержать неопределенные (null) значения во всех столбцах, объявленных как unique. Пусть в таблице test существует правило целостности unique, наложенное на столбец first, и таблица пуста:

```
insert into test (first) values (1) -- ОК
insert into test (first) values (1) -- ошибка
insert into test (first) values (null) -- ОК
insert into test (first) values (null) -- ОК
```

Пусть теперь правило unique задано для пары столбцов first и second:

```
insert into test (first,second) values (1,2) -- ОК
insert into test (first,second) values (1,2) -- ошибка
insert into test (first,second) values (1,null) -- ОК
insert into test (first,second) values (1,null) -- ошибка
insert into test (first,second) values (null,null) -- ОК
insert into test (first,second) values (null,null) -- ОК
```

Для того, чтобы поддержать правило целостности unique, Oracle использует индекс. Если подходящего индекса нет, то Oracle автоматически создает уникальный индекс по тому набору полей, на который наложено правило unique. Если правило будет уничтожено или запрещено, то автоматически созданный индекс будет уничтожен. Если для поддержания правила unique используется готовый индекс, то этот индекс не обязан быть уникальным. Кроме того, в этом случае при уничтожении или запрещении правила индекс не уничтожается, соответственно, нет необходимости пересоздавать его при включении правила.

### Первичный ключ

Правило primary key задает первичный ключ. Каждая строка таблицы однозначно определяется значениями столбца или набора столбцов, входящих в первичный ключ. Для обеспечения правила целостности primary key Oracle неявно создает правила not null для каждого столбца, входящего в ключ, и правило unique для набора столбцов. Правило primary key также может использовать готовый индекс, не обязательно уникальный. Таблица может иметь не более одного первичного ключа (а может и не иметь его вовсе).

Правило целостности может иметь имя. При создании безымянного правила целостности Oracle автоматически генерирует для него уникальное имя. Если индекс, с помощью которого поддерживается правило unique или primary key, создан автоматически, то его имя совпадает с именем правила целостности. Поскольку индекс не может строиться более чем по 16 полям, одно правило unique или primary key не может включать в себя более 16 столбцов. Кроме того, общая длина полей, входящих в правило primary key или unique, не может превышать 3128 символов.

### Ссылочная целостность

Правила ссылочной целостности (referential integrity) позволяют добавлять или обновлять строки только в том случае, если значение столбца (набора столбцов) совпадает с каким-либо значением в связанной таблице. Примером могут служить таблицы, описывающие сотрудников и отделы: значение отдела, в котором работает сотрудник, должно представлять собой номер существующего отдела. В этом случае таблица отделов называется родительской, а таблица сотрудников — дочерней.

Oracle поддерживает три варианта действий, которые производятся над дочерней таблицей при изменении данных родительской таблицы — “update, delete restrict”, “delete cascade” и “set null”<sup>1</sup>.

В случае “update, delete restrict” запрещается удалять или изменять ключевое значение в родительской таблице, если в дочерней таблице есть строки, ссылающиеся на изменяемую строку. В случае “delete cascade” при удалении строки из родительской таблицы автоматически будут удалены все строки дочерней таблицы, ссылавшиеся на удаленную строку, т. е., в нашем примере, при удалении отдела автоматически будут удалены все сотрудники этого отдела. В случае “set null” при удалении строки из родительской таблицы во всех строках дочерней таблицы, ссылавшихся на нее, значение поля-ключа будет установлено в null.

В Oracle ссылочная целостность определяется при помощи правила foreign key. Оно описывается следующим образом:

```
foreign key (<внешний ключ>) references <родительская таблица> (<ключ>)
[on delete (cascade | set null)]
```

Внешний ключ — это поле или набор полей, перечисленных через запятую, количество и типы которых совпадают с количеством и типом полей первичного ключа в родительской таблице. Поле, объявленное как foreign key, может иметь неопределенное значение (null), если только для него явно не задано правило not null. По умолчанию правило foreign key запрещает удаление строк, на которые есть ссылки, из родительской таблицы, т. е. реализует вариант “update, delete restrict”. Если требуется создать внешний ключ с правилами “delete cascade” и “set null”, это надо указать явно во фразе on delete.

Для поля (или набора полей), объявленных как foreign key, Oracle не создает индекс автоматически. Рекомендуется делать это вручную.

Правило foreign key реализует отношение (dependence) много-к-одному (many-to-one) или один-к-одному (one-to-one). Так, например, отношение «сотрудник работает ровно в одном отделе, но в отделе может работать несколько сотрудников» (много-к-одному) может быть выражено введением в таблицу сотрудников (employee) поля deptisn, содержащего первичный ключ отдела в таблице department:

```
deptisn number(10) not null constraint fk_employee_dept references department(isn)
```

Здесь и далее первичные ключи таблиц называются isn (Internal System Number). Фраза constraint fk\_employee\_dept задает имя правила целостности.

Для того, чтобы реализовать отношение многие-ко-многим, заводится промежуточная таблица, связанная отношениями много-к-одному с двумя исходными. Пусть, например, есть таблицы employee (сотрудники) и project (проекты), и один сотрудник может участвовать в нескольких проектах. Тогда, чтобы связать их отношением многие-ко-многим, создается вспомогательная таблица:

<sup>1</sup> В Oracle 7.x правило “set null” отсутствует.

```
create table empproj (
  empisn number not null,
  projectisn number not null,
  foreign key (empisn) references employee (isn),
  foreign key (projectisn) references project (isn),
  unique (empisn, projectisn)
)
```

Последнее ограничение указывает, что сотрудник не может участвовать в проекте более одного раза. Два правила `not null` и правило `unique` можно было бы выразить одним правилом

```
primary key (empisn, projectisn)
```

### Проверка правил целостности

Все правила целостности могут находиться во включенном или выключенном состоянии. Отключенное правило не оказывает никакого влияния на данные, как если бы его вообще не было; однако таблица, являющаяся родительской для какого-либо правила `foreign key`, не может быть удалена, даже если соответствующее правило отключено. В момент включения (или создания, при условии, что правило создается включенным) происходит проверка правила для всех строк таблицы. Если хотя бы одна строка не удовлетворяет правилу, то оно не будет включено (создано); при этом будет выдана ошибка. Код ошибки зависит от типа включаемого (создаваемого) правила целостности. Можно создать правило отключенным — в этом случае при создании правила проверка производиться не будет.

Если столбцы таблицы имеют значения по умолчанию, то эти значения подставляются до проверки правил целостности, т. е. они должны удовлетворять этим правилам.

Правило целостности может быть нарушено в процессе выполнения команды. Например, пусть таблица связана сама с собой правилом `foreign key`. Первая команда `insert` должна вставить строку, ссылающуюся на саму себя, либо несколько строк, ссылающихся друг на друга. В процессе выполнения команды могут оказаться строки, ссылающиеся на несуществующие ключи, однако к моменту завершения команды все ссылки должны быть корректны. Таким образом, проверка правил целостности происходит после завершения команды.

Oracle 8 позволяет создавать правила целостности с отложенной (`deferred`) проверкой — такие правила проверяются при завершении транзакции. При создании правила целостности можно задать атрибут `deferrable`, наличие которого позволяет откладывать проверку до конца транзакции или `not deferrable` — такое правило всегда будет проверяться непосредственно после выполнения команды. Кроме того, можно указать начальное состояние правила — `initially immediate` (проверка по окончании команды) или `initially deferred` (проверка по окончании транзакции). Переключение режима проверки правил целостности производится командой `alter session` с фразой `set constraints`. Подробнее об использовании отложенных правил целостности можно прочитать в [5], а о синтаксисе создания правил целостности и переключения режимов — в [8].

## Таблицы

### Создание таблиц

Таблицы создаются при помощи команды `create table`:

```
create table <имя> (
  <описание столбца> {,<описание столбца>}
)
[as <подзапрос>]
```

Полный синтаксис команды можно найти в [8]. Помимо описания столбцов, команда `create table` может изменять параметры выделения дисковой памяти для таблицы, разрешать/запрещать параллельное выполнение операций при создании таблицы, управлять кэшированием данных, задавать параметры размещения больших объектов (`blob`, `clob`, `nclob`), если такие столбцы присутствуют в таблице.

Описание столбца состоит из его имени, типа, значения по умолчанию и накладываемых на него правил целостности:

```
<имя столбца> <тип> [default <значение по умолчанию>]{<правило целостности>}
```

Например:

```

create table employee (
    isn number(10) constraint pk_employee primary key,           -- 1
    name varchar2(64) not null,                                   -- 2
    hiredate date default sysdate not null,                       -- 3
    salary number (9,2) check (salary>=83.94),                   -- 4
    deptisn number (10) not null constraint fk_employee_dept references department(isn), -- 5
    mgrisn number (10) references employee(isn) on delete cascade, -- 6
    remark varchar2 (4000),                                       -- 7
    state char (1) default 'N' not null check (state in ('N', 'Y')) -- 8
)

```

1. правило primary key с названием pk\_employee;
2. безымянное правило not null;
3. значения по умолчанию (текущая дата) и безымянное правило not null; значения по умолчанию задаются перед любыми правилами целостности;
4. безымянное правило check. Из него не следует правило not null;
5. правило foreign key с именем fk\_employee\_dept. По умолчанию запрещается удалять строки из department, если есть ссылающиеся на него строки в employee, т. е. действует правило “update, delete restrict”;
6. безымянное правило foreign key, связывающее таблицу саму с собой. Поле mgrisn может иметь значение null; если из таблицы удаляется строка, на которую есть ссылка через mgrisn, то удаляются и все ссылающиеся на нее строки;
7. поле без правил целостности;
8. одновременно заданы правила check и not null.

Если в команде create table указана фраза as <подзапрос>, то имя и тип столбцов можно не задавать — они будут совпадать с именами и типами столбцов, возвращенных подзапросом. Можно задать имена столбцов — в этом случае их количество должно совпадать с количеством столбцов, возвращенных подзапросом. Типы в этом случае указывать нельзя, но можно задавать правила целостности:

```

create table newemployee (
    isn, name not null check (name=upper(name))
) as
select isn, upper(shortname) from employee

```

Если выбранные данные не удовлетворяют правилам целостности, то таблица не будет создана, а Oracle вернет ошибку:

```
ORA-02446: CREATE TABLE ... AS SELECT failed - check constraint violated
```

Можно указывать правила целостности и отдельно от описания столбцов. Так, например, определение таблицы employee можно было бы переписать так:

```

create table employee (
    isn number (10),
    ...
    deptisn number (10) not null,
    ...
    constraint pk_employee primary key (isn),
    foreign key (deptisn) references department (isn)
)

```

Всего в таблице может быть не более 1000 столбцов.<sup>1</sup>

### Модификация таблиц

Для изменения структуры таблицы служит команда alter table. Эта команда позволяет:

- добавлять столбцы;
- изменять тип, размер и/или значение по умолчанию существующих столбцов;
- удалять столбцы из таблицы;<sup>2</sup>
- добавлять, уничтожать, включать и выключать правила целостности;
- запрещать/разрешать выполнение триггеров для таблицы;
- изменять параметры управления дисковой памятью для таблицы.

Ниже приведен синтаксис команды alter table для вышеперечисленных действий, а полный синтаксис можно найти в [8].

Добавление столбца:

```
alter table <имя> add (<описание столбца> {,<описание столбца>})
```

Описание столбца подчиняется тем же правилам, что описание столбца в команде create table. Например,

<sup>1</sup> В Oracle 7.x таблица не может содержать больше 254 столбцов.

<sup>2</sup> Oracle 7.x не позволяет удалять столбцы из таблицы.

```
alter table employee add (firedate date check (firedate>hiredate))
```

Удаление столбца:

```
alter table <имя> drop column <имя столбца>
```

Для изменения столбца используется команда `alter table` с предложением `modify`:

```
alter table <имя> modify (<описание столбца> {,<описание столбца>})
```

например:

```
alter table employee modify (
    remark varchar2 (2000),
    salary number (8,2) not null
)
```

При этом все правила целостности (в частности, `check (salary>=83.94)`) остаются в силе.

Для того, чтобы добавить правило целостности, используется команда

```
alter table <имя> add (<правило целостности> {,<правило целостности>})
```

где описание правила целостности совпадает с описанием правила целостности в команде `create table`. Имейте в виду, что новый столбец с правилом целостности `not null` или `primary key` не может быть создан командой `alter table`, если в таблице есть хоть одна строка. Если вам нужно создать такой столбец, то сначала создайте его без правила целостности, затем заполните его данными и создайте правило.

Для удаления правила целостности используется команда

```
alter table <имя> drop constraint <имя правила целостности>
```

Включение и отключение правил целостности производится командами:

```
alter table <имя> disable constraint <имя правила целостности>
```

и

```
alter table <имя> enable constraint <имя правила целостности>
```

Если имя правила целостности при создании таблицы явно не задавалось, можно посмотреть его в системном представлении `USER_CONSTRAINTS`.

Изменять правила целостности нельзя. Например, если в таблице `employee` надо изменить правило внешнего ключа с “delete cascade” на “set null”, то нужно выполнить две команды:

```
alter table employee drop constraint SYS_C00915 -- имя, сгенерированное системой
alter table employee add (
    foreign key(mgrisn) references employee(isn) on delete set null
)
```

Уничтожается таблица командой `drop`:

```
drop table <имя>
```

При этом уничтожаются все правила целостности, триггеры, индексы и объектные привилегии, связанные с таблицей.

Промежуточное положение между командами DDL и DML занимает команда `truncate table`. Она уничтожает все данные в таблице и может освободить занимаемое таблицей дисковое пространство. Операция `truncate table` не может быть отменена путем отката транзакции. При выполнении этой операции не будут выполняться триггеры, срабатывающие на удаление строки. В то же время команда `truncate table` не будет выполнена, если таблица является родительской для какого-либо правила `foreign key` — перед выполнением этой команды нужно отключить или уничтожить все такие правила, за исключением тех, которые связывают таблицу саму с собой. В простейшем случае команда записывается как

```
truncate table <имя>
```

а полный синтаксис можно найти в [8].

Таблица может быть переименована командой `rename`:

```
rename <имя> to <новое имя>
```

Oracle позволяет хранить в словаре данных комментарии к таблицам или отдельным столбцам. Они записываются при помощи команды `comment`:

```
comment on table <имя таблицы или представления> is '<текст комментария>'
```

и

```
comment on column <имя таблицы или представления>.<имя столбца> is '<текст комментария>'
```

Прочитать комментарии можно при помощи системных представлений `USER_TAB_COMMENTS` (комментарии к таблицам) и `USER_COL_COMMENTS` (комментарии к столбцам).

## Прочие объекты

### Представления

Представление — это сохраненный запрос к БД. Результатом выполнения запроса является отношение, что позволяет использовать представления в командах `insert`, `update`, `delete` и `select` так же, как и таблицы. При изменении данных представления изменяются данные в таблицах, входящих в это представление, а при использовании представления в команде выборки данных Oracle подставляет его текст в запрос и затем разбирает и выполняет запрос целиком. В представлении нельзя определять правила целостности, однако можно определять достаточно сложную логику работы при изменении данных представления — с помощью триггеров `instead of`.

Представления используются для того, чтобы скрыть физическую структуру данных, обеспечив большую гибкость приложения, чтобы упростить запросы или чтобы ограничить доступ к данным. Представление не требует места в БД, кроме места на хранение своего определения в словаре данных.

При использовании представления Oracle использует национальные настройки текущей сессии, если эти настройки не заданы явно в SQL-запросе, образующем представление.

Создается представление командой `create view`:

```
create [or replace] [force] view <имя> [(<имя столбца> {,<имя столбца>})]
as <запрос>
[with (read only | check option [constraint <имя правила>])]
```

Если запрос некорректен (содержит ошибку или пользователь, создающий представление, не имеет привилегий на обращение к указанным в запросе объектам), представление не будет создано. Однако если указать параметр `force`, то представление будет создано в любом случае, а попытка обратиться к нему приведет к ошибке:

```
ORA-04063: view "<имя>" has errors
```

Представление может становиться ошибочным и в результате изменения структуры объектов, к которым оно обращается.

Столбцы представления получают те имена, которые возвращает SQL-запрос. Если же эти имена не устраивают программиста, то можно явно перечислить имена столбцов после имени представления в скобках. Количество имен должно в точности соответствовать количеству столбцов, возвращаемых запросом.

Фраза `with read only` говорит о том, что представление можно использовать только на чтение. Фраза `with check option` заставляет Oracle при выполнении изменения данных представления командами `insert` и `update` проверять новые значения на соответствие условиям запроса. Если запрос содержит подзапросы или изменение данных выполняется триггером `instead of`, то проверка работать не будет.

Уничтожается представление командой `drop`:

```
drop view <имя>
```

### Индексы

Индексы — дополнительные структуры данных, связанные с таблицами. Индексы предназначены для ускорения поиска в таблицах, а в некоторых случаях данные могут читаться прямо из индексов, без чтения таблиц. Например, все данные индексных таблиц хранятся непосредственно в индексах, т. е. эти таблицы не имеют сегмента данных.

Индекс строится по одному или нескольким полям; в индекс может входить до 16 полей. Порядок перечисления полей в составном индексе имеет значение. Если значение всех полей, по которым построен индекс, в некоторой строке не определено (равно `null`), то такая строка в индекс не входит.

Oracle допускает создание уникальных индексов, т. е. индексов, в котором каждый набор значений индексированных полей уникален. Однако создавать такие индексы вручную не рекомендуется, т. к. уникальность значения — понятие логическое, а не физическое. Oracle автоматически создает уникальные индексы при задании правил целостности `unique` и `primary key`.

Создается индекс командой `create index`, синтаксис которой приведен ниже:

```
create [ unique | bitmap ] index <имя> on <имя таблицы> (<имя столбца> {,<имя столбца>})
```

Ключевое слово `unique` позволяет создать уникальный индекс.

Ключевое слово `bitmap` создает индекс, построенный на битовых картах. Такие индексы могут быть значительно эффективнее и компактнее традиционных индексов, основанных на

В-деревьях<sup>1</sup>. В bitmap-индексах индексируются все значения, включая null. Составной bitmap-индекс может быть использован для поиска по любому из полей, входящих в индекс, при этом порядок перечисления полей в составном индексе не имеет значения. Однако, у bitmap-индексов есть ряд недостатков, существенно ограничивающих их применение. Во-первых, они эффективны только при небольшом количестве различных значений столбца. Индексировать, например, даты или суммы проводок следует традиционными индексами. Во-вторых, bitmap-индексы теряют эффективность при частом обновлении данных в таблице.

Более подробно о структуре индексов можно прочитать в [5], а об их использовании — в [9].

Индекс в любой момент может быть уничтожен — это не повлияет на работоспособность приложений, хотя может существенно их замедлить. Уничтожается индекс командой `drop index`:

```
drop index <имя>
```

В некоторых случаях доступ к данным можно ускорить, разрушив индекс и создав его заново. В Oracle 7.x это делается последовательным выполнением команд `drop` и `create`, а в Oracle 8 можно воспользоваться фразой `rebuild` команды `alter index`. Имейте в виду, что создание индекса для заполненной таблицы — весьма ресурсоемкая операция: требуется время и пространство во временных сегментах для сортировки данных.

Oracle позволяет индексировать таблицы по значениям функций от столбцов, а также явно управлять размещением индексов в дисковой памяти. Полный синтаксис команды `create index` можно найти в [8].

Индекс может быть переименован командой `rename`:

```
rename <имя> to <новое имя>
```

### Последовательности

Последовательности — объекты, генерирующие неповторяющиеся номера. Эти объекты полезны для генерации первичных ключей.

Создаются последовательности командой `create sequence`:

```
create sequence <имя> [start with <число>] [increment by <число>]
[maxvalue <число>] [minvalue <число>] [cycle | nocycle]
[cache <число> | nocache] [order | noorder]
```

Значения `start with` и `increment by` задают начальное значение последовательности и шаг, с которым будет увеличиваться значение. Задание начального значения и шага, отличных от 1, может оказаться полезным в распределенной базе данных — если, например, на двух серверах заданы начальные значения 1 и 2 и шаг 10, то ключи, сгенерированные разными серверами, будут уникальны в пределах обоих серверов.

Oracle не следит, чтобы все значения, выбранные из последовательности, были как-то использованы. Если, например, транзакция, выбравшая очередное значение, будет отменена, то выбранное значение будет утеряно.

Параметры `minvalue` и `maxvalue` определяют минимальное и максимальное значения, которые могут быть выбраны из последовательности. Если минимальное (максимальное) значение будет достигнуто, то очередное обращение к последовательности приведет к ошибке:

```
ORA-08004: sequence <имя>.NEXTVAL exceeds MAXVALUE and cannot be instantiated
```

Если указан параметр `cycle`, то по достижении одной из границ последовательность будет снова генерировать числа, начиная с другой границы. Параметр `cache` управляет кэшированием значений. Если этот параметр задан, то обращения к последовательности будут кэшироваться, т. е. за один раз будет выбираться сразу несколько значений. Это ускоряет работу, но приводит к потере большого количества значений, если за сессию происходит мало обращений к последовательности.

В случае, если в команде `create sequence` указано ключевое слово `order`, Oracle гарантирует, что выбранное из последовательности значение будет больше (при отрицательном шаге — меньше), чем все предыдущие. В противном случае порядок может быть нарушен, но, как правило, это неважно.

<sup>1</sup> Для понимания материала данного раздела знание физической структуры индексов не обязательно. Тем не менее, найти описание В-деревьев можно, например, в книге Дональда Е. Кнута «Искусство программирования», т. 3 «Сортировка и поиск», раздел 5.1.6 «Сильноветвящиеся деревья». Битовые карты подробно описаны в фирменной документации Oracle.

Последовательность имеет два атрибута: `nextval` — следующее уникальное значение и `currval` — последнее выбранное значение. Обратиться к атрибуту `currval` можно только в том случае, если в текущей сессии уже было обращение к `nextval`. Таким образом, значение `currval` для каждой сессии свое.

Для обращения к последовательности (а также к любым объектам, не являющимся данными таблицы, например, функциям) применяется конструкция `select ... from dual`:

```
select seq_emp.currval from dual
```

```
ORA-08002: sequence SEQ_EMP.CURRVAL is not yet defined in this session
```

```
select seq_emp.nextval from dual
```

```
nextval
```

```
-----
```

```
1
```

```
select seq_emp.currval from dual
```

```
currval
```

```
-----
```

```
1
```

Если пользователю дается право непосредственно записывать данные в таблицу, то удобно обращаться к последовательности для генерации первичного ключа в триггере. В конце главы, посвященной триггерам, вы найдете соответствующий пример.

### Синонимы

Синоним — дополнительное имя для объекта БД — таблицы, представления, последовательности или хранимой процедуры. Синоним не требует физического пространства, кроме как для своего определения в словаре данных. Создается синоним командой `create synonym`:

```
create [public] synonym <имя> for <имя объекта>
```

Уничтожается синоним командой `drop`:

```
drop synonym <имя>
```

Синоним может быть переименован командой `rename`:

```
rename <имя> to <новое имя>
```

### Хранимые процедуры и триггеры

Кроме данных, Oracle может хранить алгоритмы работы с этими данными — хранимые процедуры и триггеры. Они записываются на языке PL/SQL — фирменном процедурном расширении SQL — и хранятся в виде исходных текстов и откомпилированных образов. Этому языку посвящена отдельная глава настоящего пособия. Кроме того, Oracle 8i имеет собственную Java-машину и позволяет писать хранимые процедуры и триггеры на Java.

Подробнее о создании хранимых процедур и триггеров вы можете прочитать в соответствующих главах.

## Объектные привилегии

Объектные привилегии определяют права на выполнение конкретных операций над конкретными объектами. В Oracle определены 8 видов объектных привилегий.

| Привилегия | Описание   | Таблица | Представление | Последовательность | Хранимая процедура |
|------------|--|---------|---------------|--------------------|--------------------|
| Alter      | Изменение параметров объекта   | x       |               | x                  |                    |
| Delete     | Удаление данных  | x       | x             |                    |                    |
| Execute    | Исполнение   |         |               |                    | x                  |
| Index      | Индексирование   | x       |               |                    |                    |
| Insert     | Вставка данных   | x       | x             |                    |                    |
| References | Использование таблицы в качестве родительской в правилах foreign key | x       |               |                    |                    |
| Select     | Выборка данных   | x       | x             | x                  |                    |

|        |                   |   |   |  |  |
|--------|-------------------|---|---|--|--|
| Update | Обновление данных | x | x |  |  |
|--------|-------------------|---|---|--|--|

Каждый пользователь имеет все объектные привилегии на объекты своей схемы и может выдавать эти привилегии другим пользователям. Делается это командой `grant`:

```
grant (<имя привилегии> {,<имя привилегии>} | all)
on <имя объекта> [(<столбец таблицы> {,<столбец таблицы>})]
to {<имя пользователя> | <имя роли> | public}
[with grant option]
```

Если в команде `grant` вместо перечисления привилегий указано `all`, то пользователю будут выданы все объектные привилегии, допустимые для данного объекта. Привилегии, связанные с таблицами (`select`, `update`, `index` и `references`) можно выдавать не только на таблицу целиком, но и на отдельные столбцы, которые перечисляются после имени таблицы в скобках через запятую.

Объектные привилегии могут выдаваться конкретному пользователю, роли (т. е. всем пользователям, которым выдана эта роль) и специальному пользователю `public`, т. е. всем пользователям БД. Исключение составляют привилегии `index` и `references` — они не могут быть выданы роли, но могут быть выданы всем (т. е. пользователю `public`). Если в команде `grant` указана фраза `with grant option`, то пользователь, получивший привилегию явно или через роль, может передать ее другому пользователю или роли. Роль также может быть назначена другой роли или пользователю `public`.

Отобрать привилегию у пользователя или роли или роль у пользователя можно командой `revoke`. Для ролей синтаксис команды таков:

```
revoke <имя роли> from (<имя пользователя> | <имя роли> | public)
```

Если отобрать у пользователя роль, то пользователь может продолжать пользоваться привилегиями этой роли до тех пор, пока не завершит свою сессию или не выключит роль явно с помощью команды `set role`.

Для объектных привилегий используется более сложный синтаксис отбора:

```
revoke (<имя привилегии> {,<имя привилегии>} | all)
on <имя объекта>
from (<имя пользователя> | <имя роли> | public)
[cascade constraints]
```

Фраза `cascade constraints` применима только тогда, когда отбирается привилегия `references` или все привилегии — в этом случае автоматически уничтожаются все правила целостности `foreign key`, требующие отобранных привилегий.

Для того, чтобы создать роль, необходимо выполнить команду

```
create role <имя> [identified by (<пароль> | externally)]
```

Для создания роли необходима системная привилегия `create role`. Можно указать, что включение роли требует идентификации. Если указан пароль, то идентификация происходит средствами Oracle, если же указано ключевое слово `externally`, то идентификация происходит на уровне операционной системы.

При соединении клиента с сервером сессия получает все привилегии, выданные пользователю явно, а также все привилегии, выданные ролям, в том числе и требующим идентификации, которые включены в список ролей пользователя по умолчанию. Пользователь может включать и выключать роли для сессии командой `set role`:

```
set role (all [except <имя> {,<имя>}] | none | <имя> {,<имя>})
```

Ключевое слово `none` выключает все роли, выданные пользователю. Посмотреть включенные роли можно через системное представление `SESSION_ROLES`.

Выключение роли может понадобиться в случае, когда нужно временно добавить сессии какие-либо права, выполнить определенные действия и снова выключить дополнительные права. Кроме того, количество одновременно включенных ролей для пользователя ограничено.

Уничтожить роль можно командой `drop role`:

```
drop role <имя>
```

Такие объекты, как индексы, триггеры, синонимы не имеют собственных объектных привилегий. Выдача привилегии на доступ к данным в таблице автоматически приводит к выдаче привилегии на исполнение соответствующего триггера или использование индексов. В Oracle не существует объектных привилегий, позволяющих изменять текст представления, хранимой процедуры или триггера. Сделать это может только владелец объекта или администратор, имеющий соответствующую системную привилегию. Код хранимых процедур и триггеров выполняется с правами владельца объекта. Если, например, в теле процедуры

происходит обращение к таблице, на доступ к которой нет привилегий у вызвавшего эту процедуру, но есть привилегии у ее владельца, процедура будет выполнена успешно.

При выдаче привилегий вместо имени объекта можно использовать его синоним. Выданные таким образом привилегии сохраняются, даже если синоним будет уничтожен.

Уничтожение объектов приводит к уничтожению всех выданных на него привилегий. Поэтому если вы хотите изменить текст хранимой программы или представления, сохранив привилегии, используйте вместо пары команд drop и create команду create or replace.

## Зависимости между объектами

Такие объекты, как представления, хранимые процедуры и триггеры в своих определениях содержат ссылки на другие объекты (таблицы, последовательности, другие хранимые процедуры). Oracle автоматически отслеживает зависимости между объектами.

При изменении базового объекта изменяется статус зависимых объектов — они получают статус «недействительный» (invalid). Этот статус означает, что перед обращением к объекту требуется его проверить. Применительно к хранимой процедуре или триггеру это означает компиляцию, применительно к представлению — повторный разбор его записи в словаре данных. Таблицы, последовательности и синонимы действительны всегда.

Изменение статуса — процесс рекурсивный. Это значит, что при пометке объекта как недействительного помечаются также все зависимые от него объекты, а при компиляции объекта компилируются все объекты, от которых он зависит, если они недействительны.

При обращении к недействительному объекту Oracle автоматически предпринимает попытку его перекомпиляции, и если это удастся, то команда, вызвавшая обращение, выполняется успешно. В противном случае возникает ошибка и команда отменяется.

Можно перекомпилировать объекты вручную. Для этого служит команда alter ... compile:

```
alter view <ИМЯ> compile
alter (procedure | function) <ИМЯ> compile
alter package <ИМЯ> compile [body]
alter trigger <ИМЯ> compile
```

При изменении объектных и системных привилегий Oracle помечает зависимые объекты как недействительные только в том случае, если они требуют привилегий, которые были отобраны.

# SQL. Выборка данных

Главная и наиболее мощная возможность языка SQL – это выборка данных, осуществляемая оператором *select*. В качестве хорошего учебника по выборке данных можно порекомендовать [4], а полное формальное описание синтаксиса оператора *select* приведено в [8].

## Оператор выборки

Общий вид команды на выборку данных таков:

<подзапрос> [<команда ручной блокировки>]

Определение подзапроса рекурсивно:

```
select [unique | distinct] <список выражений>
from <список отношений>
[where <условие>]
[group by <список группировки> | <иерархический запрос>]
{<множественная операция> <подзапрос>}
[order by <список сортировки>]
```

### Выражения

Список выражений в операторе *select* представляет собой набор выражений, разделенных запятыми. Выражением в SQL является

- столбец отношения;
- константа;
- псевдостолбец;
- обращение к последовательности;
- операция над выражениями;
- функция от выражения или функция без параметра;
- декодировка;
- обращение к объектам, конструкторы типов и прочие не рассматриваемые здесь выражения<sup>1</sup>;
- подзапрос, возвращающий одну строку и один столбец<sup>2</sup>.

Столбец таблицы задается именем. Если столбец с указанным именем существует в нескольких отношениях, указанных в списке, то надо указать имя отношения и имя столбца через точку.

Псевдостолбцы называются так потому, что физически не хранятся в базе данных, но подчиняются тем же синтаксическим правилам, что и столбцы. В Oracle существуют следующие псевдостолбцы:

- *rowid* — физический адрес (*rowid*) начала строки в таблице;
- *rownum* — порядковый номер, под которым строка попала в результирующее отношение;
- *level* — уровень, на котором была выбрана строка при иерархическом запросе.

Значение *rownum* присваивается непосредственно в момент попадания строки в результирующее отношение. Следовательно, если отношение сортируется, то значения *rownum* в нем могут идти не по порядку. *Rownum* может участвовать и в условии выборки. Так, запрос

```
select * from employee where rownum<=10
```

вернет первые 10 строк (в произвольном порядке) таблицы *employee*, а запрос

```
select * from employee where rownum>10
```

не вернет ни одной строки, т. к. при выборке первой строки из таблицы условие не выполняется (*rownum*=1), следовательно, строка в результирующее отношения не попадет, и *rownum* не увеличится.

Вместо имени столбца в списке выражений можно указать \* — этот символ означает все столбцы отношения, из которого происходит выборка. Если отношений в списке несколько, то \* обозначает все столбцы всех отношений. Запись имени отношения и \* через точку обозначает все столбцы указанного отношения.

```
select * from employee
```

```
select *, rowid from employee -- ошибка
```

<sup>1</sup> В версиях 8.x

<sup>2</sup> Начиная с версии 8.1.5

```
select employee.*, rowid from employee -- правильно

select
  employee.*, -- все столбцы
  sysdate,   -- функция
  rowid,     -- псевдостолбец
  'now'      -- константа
from employee
```

В списке выборки допустимы операции и функции над выражениями. Oracle поддерживает следующие операции (в порядке убывания приоритета):

| Оператор | Значение                     |
|----------|------------------------------|
| - +      | унарный минус и унарный плюс |
| * /      | умножение и деление          |
| + -      | сложение и вычитание         |
|          | конкатенация строк           |

Список встроенных функций Oracle приведен ниже:

### 1. Математические функции

- **abs(x)** — абсолютное значение x:  
abs(-15) = 15
- **acos(x)** — арккосинус x:  
acos(-1) = 3.14159265
- **asin(x)** — арксинус x:  
asin(-1) = -1.57079633
- **atan(x)** — арктангенс x; результат от  $-\pi/2$  до  $\pi/2$ :  
atan(1) = 0.78539816
- **atan2(x,y)** — то же, что atan(x/y); результат от  $-\pi$  до  $\pi$ , в зависимости от знака x и y:  
atan2(5774,10000) = 3.1415914
- **ceil(x)** — наименьшее целое, большее или равное x:  
ceil(15.7) = 16  
ceil(-15.7) = -15
- **cos(x)** — косинус x:  
cos(0) = 1
- **cosh(x)** — гиперболический косинус x:  
cosh(2.71828) = 7.6101113
- **exp(x)** — экспонента x:  
exp(1) = 2.718281828
- **floor(x)** — наибольшее целое, меньшее или равное x:  
floor(15.7) = 15  
floor(-15.7) = -16
- **ln(x)** — натуральный логарифм x:  
ln(2.71828) = 0.99999933
- **log(a,x)** — логарифм x по основанию a:  
log(2,1024) = 10
- **mod(a,b)** — остаток от деления a на b:  
mod(7,3) = 1
- **power(x,y)** — x в степени y:  
power(2,3) = 8
- **round(x[,n])** — округление x до n десятичных разрядов:  
round(128.51) = 129  
round(128.51,1) = 128.5  
round(128.51,-1) = 130
- **sign(x)** — знак x:  
sign(-9) = -1
- **sin(x)** — синус x:  
sin(1.57) = 0.99999968
- **sinh(x)** — гиперболический синус x:

```
sinh(2.71828) = 7.54412319
```

- **sqrt(x)** — квадратный корень x:

```
sqrt(9) = 3
```

- **tan(x)** — тангенс x:

```
tan(3.14159/4) = 0.99999867
```

- **tanh(x)** — гиперболический тангенс x:

```
tanh(1) = 0.761594156
```

- **trunc(x,[n])** — отбрасывание дробной части x, точность n десятичных знаков:

```
trunc(128.51) = 128
```

```
trunc(128.51,1) = 128.5
```

```
trunc(128.51,-1) = 120
```

## 2. Символьные функции

- **ascii(s)** — ASCII-код первого символа строки s:

```
ascii('Abc') = 65
```

- **chr(n)** — ASCII-символ с кодом n:

```
chr(65) = 'A'
```

- **concat(s1,s2)** — конкатенация строк s1 и s2; то же, что s1 || s2:

```
concat('A', 'B') = 'AB'
```

- **initcap(s)** — делает первые буквы всех слов строки s заглавными:

```
initcap('иванов и.и.') = 'Иванов И.И.'
```

- **instr(s,sub[,n[,m]])** — возвращает позицию m-го вхождения строки sub в строку s, начиная с позиции n; если n отрицательна, то позиция считается от конца строки s. По умолчанию m=1 и n=1:

```
instr('Иванов И.И.', 'И') = 1
```

```
instr('Иванов И.И.', 'И', 2, 2) = 10
```

- **length(s)** — длина s в символах:

```
length('длина строки') = 12
```

- **lower(s)** — преобразование строки s в нижний регистр:

```
lower('Нижний') = 'нижний'
```

- **lpad(s1,n[,s2])** — дополняет s1 слева до длины n символами из s2 (если s2 не указана, то пробелами). Если длина s1 больше n, обрезает s1 справа:

```
lpad('семь', 7, '.') = '...семь'
```

- **ltrim(s1[,s2])** — убирает из строки s1 слева все символы, входящие в s2. Если s2 не указана, убирает пробелы:

```
ltrim('   семь') = 'семь'
```

- **replace(s1,s2[,s3])** — заменяет в строке s1 все вхождения s2 на s3. Если s3 не указана, что все вхождения s2 удаляются:

```
replace('Иванов И.И.', 'Иван', 'Петр') = 'Петров И.И.'
```

- **rpad(s1,n[,s2])** — дополняет строку s1 справа до длины n символами из s2 (если s2 не указана, то пробелами). Если длина s1 больше n, обрезает s1 справа:

```
rpad('семь', 7, '.') = 'семь...'
```

- **rtrim(s1[,s2])** — убирает из строки s1 справа все символы, входящие в s2. Если s2 не указана, убирает пробелы:

```
rtrim('семь   ') = 'семь'
```

- **soundex(s)** — возвращает звуковое представление слова s<sup>1</sup>:

```
soundex('MicroSoft') = 'M262'
```

```
soundex('miKroSOVte') = 'M262'
```

- **substr(s,m[,n])** — возвращает подстроку s начиная с позиции m длиной n символов. Если m=0, считается m=1; если m отрицательно, то считается позиция от конца строки. Если n не указано, то возвращается подстрока от m до конца строки:

```
substr('Иванов И.И.', 1, 6) = 'Иванов'
```

```
substr('Иванов И.И.', -4) = 'И.И.'
```

- **translate(s1,s2,s3)** — возвращает s1, в которой все символы из s2 заменены соответст-

<sup>1</sup> Алгоритм построения звукового значения английских слов описан в книге Дональда Е. Кнута «Искусство программирования», т. 3 «Сортировка и поиск». С национальными символами (в частности, с кириллицей) функция не работает.

вующими символами из s3; символы, не входящие в s2, не заменяются; символы, которые есть в s2, но не имеют соответствия в s3, удаляются:

```
translate('Иванов И.И.', 'Ива', '!') = '!нов !..'
```

- **trim( [leading|trailing] с from s)** — удаляет символы с из начала (leading), конца (trailing) или с обоих концов строки s; с представляет собой строку длиной в 1 символ:

```
trim(leading ' ' from ' строка') = 'строка'
```

- **upper(s)** — преобразование строки s в верхний регистр:

```
upper('вЕрХний') = 'ВЕРХНИЙ'
```

### 3. Функции работы с датами

- **add\_months(d,n)** — добавляет n месяцев к дате d с учетом количества дней в месяце:

```
add_months('31-Oct-2001',4) = '28-Feb-2002'
```

- **last\_day(d)** — последний день месяца, которому принадлежит дата d:

```
last_day('3-Oct-2001') = '31-Oct-2001'
```

- **months\_between(d1,d2)** — количество месяцев (необязательно целое) между датами d2 и d1 (если d2>d1, то результат отрицательный):

```
months_between('3-Oct-2001', '2-Nov-2001') = -0.967741935
```

- **next\_day(d,c)** — ближайший следующий за датой d день недели c:

```
next_day('3-Oct-2001', 'wed') = '10-Oct-2001'
```

- **round(d[,fmt])** — округляет дату d<sup>1</sup> до единиц, указанных fmt — форматной строкой для даты. По умолчанию округляет до дня:

```
round(to_date('3-Oct-2001', 'dd-mon-yyyy'), 'YEAR') = '1-Jan-2002'
```

- **sysdate** — текущая дата и время; обратите внимание, что если функция не принимает аргументов, пустые скобки после ее имени не пишутся.

- **trunc(d[,fmt])** — обрезает дату d<sup>2</sup> до единиц, указанных fmt — форматной строкой для даты. По умолчанию обрезает до дня:

```
trunc(to_date('3-Oct-2001', 'dd-mon-yyyy'), 'mm') = '1-Oct-2001'
```

### 4. Разные функции

- **greatest(n1,n2,...,nN)** — наибольшее из чисел:

```
greatest(-1,2,8,-5) = 8
```

- **least(n1,n2,...,nN)** — наименьшее из чисел:

```
least(1,7,4) = 1
```

- **nvl(p1,p2)** — если p1 не null, то p1, иначе — p2:

```
nvl(null,5) = 5
```

- **uid** — идентификатор текущего пользователя.

- **user** — имя текущего пользователя Oracle.

- **userenv(p)** — параметры текущей сессии; p — одна из предопределенных строк. Например, 'language' выдает текущие языковые настройки:

```
userenv('language') = 'RUSSIAN_CIS.CL8MSWIN1251'
```

Полный список допустимых параметров можно найти в [8].

Если символьные функции возвращают значения длиннее, чем требуется (4000 символов для varchar2 и 2000 символов для char), то значение без предупреждения обрезается до нужной длины. Кроме перечисленных функций, в операторе select можно использовать функции преобразования типов, о которых рассказывалось в предыдущей главе, а также функции, написанные самим разработчиком на языке PL/SQL. Более полный список и подробную информацию о встроенных функциях Oracle можно найти в [8], а об использовании пользовательских функций рассказано в главе, посвященной PL/SQL.

Особняком от остальных встроенных функций стоит функция decode:

```
decode(<выражение>, <значение1>, <значение2>{, <значение_2n-1>, <значение_2n>}[, <значение по умолчанию>])
```

Если <выражение> и какое-нибудь из значений <значение\_2i-1> равны, то функция вернет <значение\_2i>. Если выражение не равно ни одному из нечетных значений, то функция

<sup>1</sup> Обратите внимание, что при передаче константы в качестве параметра функции ее надо явно приводить к типу date, т. к. по умолчанию функция пытается преобразовать строку в число.

<sup>2</sup> См. примечание к функции round.

вернет значение по умолчанию, а если оно не указано, то null. Особенность этой функции состоит в том, что она позволяет сравнивать на равенство значения null. Например,

```
decode(status, 'Y', 'выверенный', 'N', 'новый', null, 'удален', 'недопустимое значение')
```

принимает значения

```
'выверенный',      если status='Y'
'новый',           если status='N'
'удален',         если status is null
'недопустимое значение' во всех остальных случаях
```

Кроме столбцов и выражений над ними, в списке выражений оператора select может участвовать подзапрос. Этот подзапрос должен содержать один столбец и возвращать не более одной строки. Если запрос вернул одну строку, то значением выражения-подзапроса становится значение единственного столбца; если запрос не вернул ни одной строки, то null; если же запрос вернул более одной строки, возникает ошибка:

```
ORA-01427: single-row subquery returns more than one row
```

Каждое поле в отношении должно иметь имя. При простой выборке поле получает то же имя, что было у него в том отношении, из которого оно выбрано. Если же в результирующем отношении несколько полей с одинаковыми именами, Oracle переименовывает их по своему усмотрению. Также поля переименовываются, если в его описании участвуют вызовы функций или операторы. Для того, чтобы переименовать поля, надо указать новое имя после выражения, отделив его ключевым словом as или пробелом. Названия полей могут содержать национальные или служебные символы, а также совпадать с зарезервированными словами — в этом случае их надо заключать в двойные кавычки:

```
select
  isn,                -- поле получит имя "isn"
  initcap(name),      -- Oracle сам сформирует имя
  salary-83.5 as difference, -- переименование
  hiredate as "принят", -- новое имя содержит национальные символы
  firedate as "null",  -- новое имя совпадает со служебным словом
  remark "примечание" -- слово as можно опустить
from employee
```

Перед списком выражений можно указать ключевое слово distinct или unique. В этом случае из результирующего отношения будут исключены дубликаты. Так, например, запрос

```
select distinct salary from employee
```

вернет все возможные суммы зарплаты. В списке полей при использовании слова distinct (или unique) не должно быть полей типа long, long raw, bfile, blob, clob, nclob. Имейте в виду, что использование distinct приводит к сортировке результата и, соответственно, замедляет выполнение запроса.

## Отношения

После ключевого слова from в запросе перечисляются отношения, из которых осуществляется выборка. В качестве отношения может выступать таблица, представление или SQL-подзапрос. Oracle не поддерживает «соединенные таблицы» (синтаксис типа employee (inner | left outer | right outer | full outer) join department) — все условия соединения задаются после ключевого слова where.

Каждое отношение должно иметь имя для того, чтобы можно было сослаться на его столбцы во фразе where. На таблицу или представление можно сослаться непосредственно по его имени, но можно и переименовать их, чтобы имя было короче. Новое имя (псевдоним) отношения указывается непосредственно после него через пробел:

```
select e.name as empname, d.name as deptname
from employee e, department d
where deptisn=d.isn
```

В приведенном примере таблице employee присвоен псевдоним e, и запись e.name означает столбец name таблицы employee. Поскольку столбцы с именем name и isn есть в обеих таблицах, обязательно надо указать во фразах select и where, к полям какой именно таблицы происходит обращение. В противном случае Oracle выдает ошибку:

```
ORA-00918: column ambiguously defined
```

Этот же пример можно записать и без переименования:

```
select employee.name, department.name
from employee, department
where deptisn=department.isn
```

Поскольку поле deptisn присутствует только в таблице employee, в запросе не указано, из какого отношения брать это поле. Однако в реальных запросах рекомендуется всегда ука-

зывать имя отношения явно.

Если в списке отношений присутствует подзапрос, то у него всегда должен быть псевдоним:

```
select d.datepay, d.amount
from
  debt d,
  (
    select e.isn,e.name as ename,d.name as dname
    from employee e, department d
    where e.deptisn=d.isn and e.mgrisn is null
  ) emp
where d.remark = emp.ename
```

### Ограничение выборки

Для того, чтобы ограничить выборку, используются условия после ключевого слова `where`. Условия состоят из логических операций над выражениями и операторов `not`, `and` и `or`. Приоритет оператора `not` больше, чем `and`, а приоритет `and` больше, чем `or`. Для указания последовательности вычисления условий можно использовать круглые скобки.

Список логических операций приведен в таблице:

|                                     |  |
|-------------------------------------|--|
| =<br>!= < > ^=<br>< > >= <=         | операторы сравнения; действуют так же, как логические операторы в традиционных языках программирования. Сравнение любого значения с <code>null</code> дает <code>null</code> , т. е. не <code>true</code> и не <code>false</code> . Обратите внимание, что условие «не равно» может быть записано тремя разными способами (приведены во второй строке)   |
| is [not] null                       | проверка, является ли значение неопределенным ( <code>null</code> )  |
| [not] between ... and ...           | принадлежность значения интервалу; <code>x between a and b</code> эквивалентно <code>x&gt;=a and x&lt;=b</code>  |
| [not] like ...<br>[escape <символ>] | соответствие строки шаблону. Каждый символ шаблона соответствует самому себе, за исключением <code>_</code> (подчеркивания), соответствующего любому символу и <code>%</code> , соответствующего любому набору из 0 или более символов. Можно указать «escape-символ», который экранирует следующий за ним <code>%</code> или <code>_</code>   |
| [not] in (...)                      | принадлежность значения множеству. Множество может быть перечислением значений (не более 1000 <sup>1</sup> ) или подзапросом, возвращающим один столбец.<br>Обратите внимание:<br><code>null in (множество)</code> возвращает <code>null</code> (не <code>true</code> и не <code>false</code> ), даже если множество содержит в себе <code>null</code> ;<br><code>... not in (множество)</code> возвращает <code>null</code> , если хотя бы один элемент множества <code>null</code> . |
| exists (подзапрос)                  | <code>true</code> , если подзапрос вернул хотя бы одну строку  |

Приведем несколько примеров, иллюстрирующих ограничение выборки.

Выбрать всех сотрудников, работающих в отделе с кодом 10:

```
select * from employee where deptisn=10
```

Выбрать сотрудников, получающих максимальную зарплату:

```
select * from employee
where salary=(select max(salary) from employee)
```

Выбрать всех сотрудников, принятых на работу в январе 2001 года:

```
select * from employee
where trunc(hiredate) between '1-Jan-2001' and '31-Jan-2001'
```

Обратите внимание, что при некорректно написанном клиентском ПО столбец `hiredate` может содержать даты с ненулевым временем (например, при заполнении его значением `sysdate`). 31 января с ненулевым временем — больше, чем просто 31 января, и такие даты не попали бы в интервал. Поэтому в запросе использована функция `trunc`. Правильно было бы использовать функцию `trunc` при вставке данных в таблицу.

Выбрать сотрудников по фамилии Иванов:

<sup>1</sup> В Oracle 7.x — не более 240.

```
select * from employee
where
  upper(name) like 'ИВАНОВ %' or
  upper(name) like 'ИВАНОВА %'
```

если бы мы задали маску 'ИВАНОВ%', то в выборку мог бы попасть сотрудник с фамилией, например, Иванович.

Выбрать сотрудников, в примечании к которым содержится текст '10%':

```
select * from employee where remark like '%10\%%' escape '\'
```

Выбрать сотрудников, к которым есть особые замечания:

```
select * from employee where remark is not null
```

Выбрать сотрудников, принятых на работу 13 числа:

```
select * from employee where to_char(hiredate, 'DD')='13'
```

Выбрать сотрудников, работавших 13 января 2001 года:

```
select * from employee
where '13-Jan-2001' between hiredate and nvl(firedate, sysdate)
```

Выбрать сотрудников, фамилия которых начинается с гласной буквы:

```
select * from employee
where upper(substr(name,1,1)) in ('А', 'Я', 'О', 'Ё', 'Э', 'Е', 'И', 'Ы', 'У', 'Ю')
```

Предикат `in` в такой форме эквивалентен последовательной записи условий с предикатом `=`, соединенных оператором `or`. Этот же запрос можно переписать следующим образом:

```
select * from employee
where
  upper(substr(name,1,1))='А' or
  upper(substr(name,1,1))='Я' or
  ...
  upper (substr(name,1,1))='Ю'
```

Выбрать сотрудников, вторая буква в фамилии которых — «о»:

```
select * from employee where name like '_o%'
```

Этот же запрос можно переписать с использованием функции `substr`:

```
select * from employee where substr(name,2,1)='o'
```

Обратите внимание, что благодаря неявному преобразованию типов предикат `like` применим не только к строкам, но и к числам. Так, например, условия `mod(isn,10)=5` и `isn like '%5'` эквивалентны, но первое будет вычислено быстрее.

## Соединение таблиц

Соединением таблиц называется отношение, строки которого являются конкатенацией строк (или частей строк) исходных таблиц. Именно операция соединения (`join`) делает язык SQL столь удобным для манипуляции данными.

### Простое соединение

Рассмотрим пример: выдать фамилии сотрудников вместе с названиями отделов:

```
select e.name as emplname, d.name as deptname
from employee e, department d
where e.deptisn=d.isn
```

Результатом выполнения этого запроса могло бы быть отношение

| emplname      | deptname            |
|---------------|---------------------|
| Иванов И.И.   | Бухгалтерия         |
| Петров П.П.   | Бухгалтерия         |
| Сидоров С.С.  | Служба безопасности |
| Воробьев В.В. | Технический отдел   |

В данном примере строки результирующего отношения получаются путем конкатенации частей строк таблицы `employee` (поле `name`) и таблицы `department` (поле `name`). Условие `e.deptisn=d.isn` называется условием соединения. Соединение по условию `=` называется эквисоединением (`equijoin`). Это наиболее важный и часто встречающийся вид соединения, и для его реализации разработано много эффективных алгоритмов.

Фраза `where` может содержать не только условие соединения, но и другие условия. Так, например, если надо выбрать всех сотрудников, кроме работающих в бухгалтерии, запрос можно переписать следующим образом:

```
select e.name as empname, d.name as deptname
from employee e, department d
where
  e.deptisn=d.isn and
  d.name<>'Бухгалтерия'
```

Один из способов построить результирующее отношение (простой, но неэффективный) состоит в том, чтобы построить декартово произведение отношений<sup>1</sup> employee и department, а затем исключить из него те элементы, где не выполняется условие соединения. Если же условие соединения не указано, то результатом соединения будет именно декартово произведение таблиц. Так, если в таблице employee 1000 строк, а в таблице department — 20 строк, то запрос

```
select e.name, d.name from employee e, department d
```

вернет 20x1000=20000 строк. Поэтому необходимо тщательно следить, чтобы каждое из отношений, перечисленных во фразе from, участвовало в условии соединения.

В соединении может участвовать любое количество отношений. Пусть, например, у нас есть таблица project, содержащая текущие проекты, и таблица emprrj, каждая строка в которой соответствует участию сотрудника в проекте (таким образом реализуется отношение многие-ко-многим между таблицами employee и project). Тогда для того, чтобы выбрать сотрудников технического отдела, участвующих в проектах, начавшихся до 1 января 2001 года, и названия этих проектов, нам понадобится соединение четырех таблиц:

```
select
  e.name, p.name, p.startdate
from employee e, department d, emprrj ep, project p
where
  d.name='Технический отдел'
  and d.isn=e.deptisn    -- сотрудники технического отдела
  and e.isn=ep.empisn    -- связь между сотрудниками...
  and p.isn=ep.prjisn    -- ...и проектами
  and p.startdate<'1-Jan-2001'
```

Обратите внимание, что таблицы department и emprrj участвуют в соединении, но данные из них не попадают в результирующее отношение.

Одна и та же таблица может участвовать в соединении несколько раз. Пусть, например, надо выдать фамилии сотрудников вместе с фамилиями их начальников:

```
select e.name as empame, m.name as mgrname
from employee e, employee m
where e.mgrisn=m.isn
```

Здесь и далее будем предполагать, что таблица employee содержит следующие данные:

| isn | name          | hiredate    | salary | mgrisn |
|-----|---------------|-------------|--------|--------|
| 233 | Иванов И.И.   | 28-Sep-2001 | 200    |        |
| 133 | Петров П.П.   | 19-Sep-2001 | 150    | 233    |
| 333 | Сидоров С.С.  | 28-Sep-2001 | 100    | 233    |
| 633 | Воробьев В.В. | 05-Oct-2001 | 90     | 133    |

Тогда результатом запроса будет

| empname       | mgrname     |
|---------------|-------------|
| Петров П.П.   | Иванов И.И. |
| Сидоров С.С.  | Иванов И.И. |
| Воробьев В.В. | Петров П.П. |

Соединение отношения с самим собой называется самосоединение (selfjoin).

### Открытое соединение

В предыдущем примере сотрудник Иванов не попал в выборку, так как значение mgrisn для него (null) не совпадает ни с одним из значений первичного ключа (isn). Для того, чтобы включить в выборку строки отношения, для которых нет подходящих строк другого отношения, существует специальный вид соединения, называемый открытым соединением (outer join). Стандарт SQL92 предусматривает специальные конструкции для обозначения открытого соединения — left outer join, right outer join и full outer join. Oracle не поддерживает такой синтаксис. Более того, Oracle допускает только одностороннее открытое соединение<sup>2</sup>. Признаком открытого соединения служит знак (+). Смысл этого знака можно выразить словами «если есть подходящие». Так, результатом запроса

<sup>1</sup> Т. е. такое отношение, куда входят конкатенации всех возможных пар строк из обоих исходных отношений.

<sup>2</sup> Oracle 9 уже поддерживает ключевое слово join, включая full outer join.

```
select e.name as empname, m.name as mgrname
from employee e, employee m
where e.mgrisn=m.isn(+)
```

будет

| empname       | mgrname     |
|---------------|-------------|
| Петров П.П.   | Иванов И.И. |
| Сидоров С.С.  | Иванов И.И. |
| Воробьев В.В. | Петров П.П. |
| Иванов И.И.   | null        |

В стандарте SQL92 этот запрос выглядел бы так:

```
select e.name as empname, m.name as mgrname
from employee e left outer join employee m on e.mgrisn=m.isn
```

Если во открытом соединении участвуют несколько таблиц, то знак (+) нужно ставить после каждого столбца таблицы, со стороны которой соединение открыто. Так, если надо выбрать сотрудников, начальник которых не Иванов, запрос формулируется так:

```
select e.name as empname, m.name as mgrname
from employee e, employee m
where
  e.mgrisn=m.isn(+)
  and m.name(+)<>'Иванов И.И.'
```

В этом случае результат будет

| empname       | mgrname     |
|---------------|-------------|
| Воробьев В.В. | Петров П.П. |
| Иванов И.И.   | null        |

Если не указать (+) после m.name, то Иванов И. И. в результат не попадет, т. к. выражение null<>'Иванов И.И.' принимает значение null.

Напоследок еще один пример — выдать все пары «сотрудник — проект», включая сотрудников, не занятых ни в одном проекте:

```
select e.name, p.name
from employee e, empprj ep, project p
where
  e.isn=ep.empisn(+)
  and ep.prjisn(+) = p.isn
```

### Подзапросы

Подзапрос — это вложенное предложение select. Именно возможность вкладывать один запрос в другой привело к использованию прилагательного «структурный» в аббревиатуре SQL (Structured Query Language). Обычно подзапросы используются для формирования множества значений и проверки значения на принадлежность этому множеству. Пусть, например, надо вывести фамилии всех сотрудников, участвующих в проекте «Бухгалтерия»:

```
select e.name
from employee e
where e.isn in (
  select ep.empisn from empprj ep, project p
  where ep.prjisn=p.isn and p.name='Бухгалтерия'
)
```

Сначала выполнится подзапрос, заключенный в круглые скобки — он выдаст множество сотрудников, участвующих в проекте, а затем из таблицы employee будут выбраны их имена. В данном случае (и в большинстве других случаев) запрос может быть переформулирован без предиката in:

```
select e.name
from project p, empprj ep, employee e
where
  p.name='Бухгалтерия'
  and p.isn=ep.isn.prjisn
  and ep.empisn=e.isn
```

В таком варианте запрос более эффективен, однако в некоторых случаях, например, в команде обновления данных update, применим только вариант с предикатом in. Напомним, что условие null in (<подзапрос>) имеет значение null даже если одно из значений, возвращенных подзапросом — null.

Подзапрос также может содержать подзапросы. Так, например, выбрать сотрудников, участвующих в проектах, начавшихся после 1 января 2001 года можно было бы запросом

```
select e.name from employee e
where e.isn in (
  select empisn from empprj
  where prjisn in (
    select isn from project where datestart>'1-Jan-2001'
  )
)
```

Разумеется, этот же запрос можно было бы переформулировать и без `in` — менее понятно, зато более эффективно:

```
select distinct e.name
from project p, empprj ep, employee e
where
  p.datestart>'1-Jan-2001'
  and p.isn=ep.prjisn
  and ep.empisn=e.isn
```

В рассмотренных примерах подзапрос выполнялся один раз. Однако возможен запрос, в котором подзапрос выполняется несколько раз. Например, выбрать сотрудников, участвующих в проекте «Бухгалтерия», можно было бы так:

```
select e.name from employee e
where 'Бухгалтерия' in (
  select p.name
  from project p, empprj ep
  where
    ep.empisn=e.isn
    and ep.prjisn=p.isn
)
```

Подзапрос выполняется для каждого значения `employee.isn` — он выдает список проектов, в котором участвует сотрудник. Такой подзапрос называется коррелированным подзапросом (*correlated subquery*).

Наряду с предикатом `in` в SQL существует и предикат `not in`. Как нетрудно догадаться, его значение `true`, если значение не входит во множество значений, `false` — если входит, и `null` — если проверяемое значение `null`. Еще раз обратите внимание, что если хотя бы один элемент множества — `null`, то значением предиката `not in` также будет `null`.

Подзапрос может использоваться не только с предикатом `in`, но и с обычными операторами сравнения. Например, выбрать сотрудников, получающих минимальную зарплату, можно так:

```
select name from employee
where salary=(select min(salary) from employee)
```

В данном случае подзапрос должен возвращать ровно одну строку, иначе возникнет ошибка.

Подзапрос, используемый в операторе сравнения, может быть и коррелированным. Например, следующий запрос выдает фамилию сотрудника, его зарплату и максимальную зарплату среди его сослуживцев (т. е. людей, у которых тот же начальник):

```
select
  name,salary,(select max(salary) from employee where nvl(mgrisn,0)=nvl(e.mgrisn,0))
from employee e
```

### Предикат exists

Предикат `exists` представляет собой квантор существования. Его аргументом является подзапрос; он принимает значение `true`, если подзапрос возвращает хотя бы одну строку. Например, запрос о сотрудниках, участвующих в проекте, можно сформулировать так: для каких сотрудников существует пара сотрудник — проект, где имя проекта — «Бухгалтерия». На языке SQL этот запрос формулируется так:

```
select e.name from employee e
where exists (
  select * from empprj ep, project p
  where
    p.name='Бухгалтерия'
    and p.isn=ep.prjisn
    and ep.isn=e.isn
)
```

Если этот запрос может быть сформулирован и без `exists`, то для ответа на вопрос «какой сотрудник не участвует ни в одном проекте» этот предикат необходим:

```
select e.name from employee
where
  not exists(select * from empprj where empisn=e.isn)
```

В подзапросе, используемом предикатом `exists`, можно использовать символ `*` или явно перечислить одно или несколько полей — это не имеет значения.

## Множественные операции, группировка, сортировка

### Множественные операции

Поскольку отношения — это множества строк, к ним могут быть применены множественные операции. Язык SQL поддерживает следующие операции:

- `union` — объединение;
- `union all` — объединение без исключения дубликатов. Результат этой операции является не множеством, а мультимножеством;
- `intersect` — пересечение;
- `minus` — разность множеств.

Пусть, например, надо выбрать сотрудников, работающих в отделе Иванова или участвующих в проекте «Бухгалтерия». Для простоты предположим, что нам известны первичные ключи сотрудника по фамилии Иванов и проекта «Бухгалтерия»:

```
select name from employee where mgrisn=233
union
select name from employee e
where exists (select * from empprj where prjism=13 and empism=e.isn)
```

Если сотрудник одновременно работает в отделе Иванова и участвует в проекте «Бухгалтерия», то он попадет в результирующее отношение один раз, т. к. оператор `union` исключает дубликаты. В данном случае можно было бы обойтись без `union`, перечислив условия `mgrisn=233` и `exists(...)` через `or`. Однако `union` позволяет объединять отношения, полученные выборкой из разных таблиц. Пусть, например, надо выбрать список отделов и сотрудников:

```
select isn, name, isn as deptisn, 'D' from department
union
select isn, name, deptisn, 'E' from employee
```

Для оператора `union` (и других операторов над множествами) важно, чтобы все множества имели одинаковый тип элементов, т. е. количество и типы столбцов в отношениях совпадали. Имена столбцов могут не совпадать — столбцы результирующего отношения получают те же имена, что и столбцы первого отношения, даже если первое отношение пустое.

Оператор `union` требует сортировки отношения, получившегося в результате объединения для исключения дубликатов. Поэтому там, где по смыслу должен использоваться `union`, но дубликатов быть не может, используйте `union all`. Например, последний запрос будет выполняться значительно быстрее, если переписать его так:

```
select isn, name, isn as deptisn, 'D' from department
union all
select isn, name, deptisn, 'E' from employee
```

Операторы `intersect` и `minus` предъявляют те же требования к отношениям-операндам, что и `union`. Они требуют сортировки обоих отношений, поэтому эффективнее заменить эти операторы условиями выборки перечисленными через `and` (`intersect`) или `and not` (`minus`). Примеры, когда вычитаемые отношения строятся из разных таблиц, достаточно редки и поэтому здесь не приводятся.

### Группировка

Во всех приведенных выше примерах из базы извлекались данные в том виде, в котором они лежат в таблицах. Однако язык SQL позволяет извлекать данные по группам строк. Для этого служит фраза `group by` предложения `select`. Пусть, например, в таблице `debt` хранятся расходы. Информация о расходах и доходах — бухгалтерский субсчет, сумма, дата и примечание. Тогда получить суммы операций по каждому субсчету можно следующим запросом:

```
select subacc, sum(amount) from debt
group by subacc
```

Результатом запроса будут группы, в каждую из которых входят доходы и расходы по одному субсчету. Предположим, что таблица `debt` содержит следующую информацию:

| isn | amount | subacc | datepay     |
|-----|--------|--------|-------------|
| 33  | 1000   | 50001  | 01-Jan-2001 |
| 133 | 500    | 50001  | 09-Jan-2001 |
| 233 | 700    | 50001  | 23-Feb-2001 |
| 333 | 200    | 50002  | 08-Feb-2001 |
| 433 | 200    | 50002  | 08-Jan-2001 |
| 533 | 400    | 50002  | 09-Jan-2001 |

Тогда результатом запроса будет такое отношение:

| subacc | sum(amount) |
|--------|-------------|
| 50001  | 2200        |
| 50002  | 800         |

Поля, по которым группируются строки, должны быть перечислены во фразе `group by`. Если строки группируются по всем полям, то такой запрос эквивалентен указанию ключевого слова `distinct` в операторе `select`. Например, следующие два запроса возвратят одно и то же:

```
select distinct salary from employee
select salary from employee group by salary
```

Все поля, не входящие в список группировки, должны вычисляться с помощью агрегатных функций, т. е. функций, аргументами которых являются целые столбцы. Список агрегатных функций приведен в таблице:

|          |  |
|----------|--|
| min      | минимальное значение   |
| max      | максимальное значение  |
| count    | количество строк в группе  |
| avg      | среднее значение   |
| variance | дисперсия. Вычисляется по формуле<br>$\frac{1}{n-1} \left( \sum_{i=1}^n x_i^2 - \frac{1}{n} \left( \sum_{i=1}^n x_i \right)^2 \right)$ |
| stddev   | среднеквадратичное отклонение; то же, что <code>sqrt(variance(...))</code>   |
| sum      | сумма  |

Аргументами функций `min` и `max` могут быть числа или строки<sup>1</sup>, аргументами `avg`, `sum`, `variance` и `stddev` — только числа. Агрегатные функции могут вычислять значения по всем строкам или только по различным значениям в строках, входящих в группу — для этого надо перед именем поля указать ключевое слово `distinct`.

Функции `count` в качестве аргумента можно передавать `*` — в этом случае она посчитает общее количество строк в группе. Если вместо `*` передать имя поля, то она выдаст количество строк, в которых значение этого поля не `null`. При вычислении остальных функций, а также при вычислении количества различных значений (`count(distinct ...)`) строки, в которых значение поля — `null`, игнорируются.

Приведем несколько примеров.

Вывести количество сотрудников:

```
select count(*) from employee where firedate is null
```

Вывести количество уволенных сотрудников:

```
select count(*) from employee where firedate is not null
```

или, что то же самое,

```
select count(firedate) from employee
```

Вывести количество сотрудников и среднюю зарплату по отделам:

```
select d.name, count(*), avg(e.salary)
from department d, employee e
where d.isn=e.deptisn
group by d.name
```

Вывести количество субсчетов, по которым было движение денег в январе 2001 года:

```
select count(distinct subacc) from debt
where trunc(datepay) between '1-Jan-2001' and '31-Jan-2001'
```

Вывести обороты по месяцам по субсчету 50001:

<sup>1</sup> В Oracle 7.x — только числа.

```
select to_char(datepay, 'yy-mon'), sum(amount)
from debt
where subacc='50001'
group by to_char(datepay, 'yy-mon')
```

Обратите внимание, что выражение во фразе `group by` должно совпадать с выражением во фразе `select` или с аргументом функции, которая используется в выражении. Так, из трех следующих запросов выполнится только первый (хотя его результат будет отличаться от результата предыдущего):

```
select to_char(datepay, 'yy-mon'), sum(amount)
from debt
where subacc='50001'
group by datepay

select to_char(datepay, 'yy-mon'), sum(amount) -- ошибка
from debt
where subacc='50001'
group by to_char(datepay, 'yy-mon-dd')

select datepay, sum(amount) -- ошибка
from debt
where subacc='50001'
group by to_char(datepay, 'yy-mon')
```

Для того, чтобы отфильтровать группы, служит фраза `having`. Пусть, например, надо вывести субсчета, оборот по которым за январь 2001 года превышает 1000 рублей:

```
select subacc, sum(amount) as total
from debt
where trunc(datepay) between '1-Jan-2001' and '31-Jan-2001'
group by subacc
having sum(amount)>1000
```

Можно переписать этот же запрос без использования `having`, применив вложенный `select`:

```
select * from (
  select subacc, sum(amount) as total
  from debt
  where trunc(datepay) between '1-Jan-2001' and '31-Jan-2001'
  group by subacc
) where total>1000
```

Первый вариант предпочтительнее, хотя в Oracle, в отличие от большинства реализаций SQL, второй вариант также будет работать.

### Специальные возможности группировки<sup>1</sup>

Oracle 8 содержит специальные псевдофункции `rollup` и `cube` для обеспечения дополнительных возможностей группировки данных.

Функция `rollup` предназначена для включения в результат запроса промежуточных итогов. Пусть, например, надо вывести все операции за месяц с итогами по субсчетам и общими итогами. В Oracle 7.x задача решается так:

```
select datepay, subacc, amount, to_char(null) as name
from debt
where datepay between '1-Jan-2001' and '31-Jan-2001'
union all
select to_date(null) as datepay, subacc, sum(amount), 'Итого по счету' as name
from debt
where datepay between '1-Jan-2001' and '31-Jan-2001'
group by subacc
union all
select to_date(null) as datepay, to_char(null) as subacc, sum(amount), 'Итого' as name
from debt
where datepay between '1-Jan-2001' and '31-Jan-2001'
```

Обратите внимание, что `null` не имеет типа, поэтому, чтобы могла выполняться операция `union all`, `null` явно приводится к нужному типу с помощью функций преобразования. Поле `name` необходимо, чтобы отличать регулярные строки от итоговых.

Использование `rollup` позволяет записать этот же запрос значительно короче и эффективнее:

---

<sup>1</sup> В Oracle 7.x отсутствуют.

```
select
    datepay, subacc, sum(amount),
    decode(grouping(datepay),1,decode(grouping(subacc),1,'Итого', 'Итого по счету')) as name,
    grouping(subacc), grouping(datepay)
from debt
where datepay between '1-Jan-2001' and '31-Jan-2001'
group by rollup(subacc,datepay)
```

Таким образом, использование `rollup` позволяет добавить к регулярным строкам строки группы. Результат включает в себя регулярные строки и  $n$  уровней группировки, где  $n$  — количество выражений, перечисленных в `rollup`. Первый уровень составляют строки, в которых последнее выражение заменяется на `null` и производится группировка по остальным полям, второй — строки, в которых на `null` заменяются последнее и предпоследнее выражения и т. д. Для того, чтобы различать строки разных уровней группировки, существует псевдофункция `grouping`. Ее значение равно 1, если строка содержит свертку по выражению, записанному в качестве аргумента, и 0 в противном случае. Под сверткой в данном контексте понимается операция замены поля на `null` и группировка по остальным полям. В регулярных строках `grouping` возвращает 0 для любого выражения из списка `rollup`.

Псевдофункция `cube` предназначена для построения кросс-таблиц. Она позволяет группировать данные на основе всех возможных комбинаций выражений, перечисленных в качестве аргументов. Таким образом, полученное отношение содержит, помимо регулярных строк,  $2^n$  уровней группировки. Если бы нас интересовала не только информация о расходах за весь период по субсчету, но и информация о расходах по всем субсчетам за конкретную дату, то в Oracle 7.x к запросу следовало бы при помощи `union all` добавить еще одну секцию:

```
...
union all
select datepay, to_char(null) as subacc, sum(amount), 'Итого на дату' as name
from debt
where datepay between '1-Jan-2001' and '31-Jan-2001'
group by datepay
```

В Oracle 8 следовало бы переписать запрос, используя `cube` вместо `rollup`:

```
select
    datepay, subacc, sum(amount),
    decode(grouping(datepay),1,
        decode(grouping(subacc),1,'Итого', 'Итого по счету'),
        decode(grouping(subacc),1,'Итого на дату',null)
    ) as name,
    grouping(subacc), grouping(datepay)
from debt
where datepay between '1-Jan-2001' and '31-Jan-2001'
group by cube(subacc,datepay)
```

Если фраза `group by` содержит обычные выражения, то сначала выполняется группировка по ним. Полученные в результате группировки строки считаются регулярными, и к ним применяется свертка по `rollup` или `cube`. Более подробно прочитать об использовании `rollup` и `cube` можно в [10] или в документации по СУБД фирмы IBM DB2 Universal Database версии 5.0 и старше.

### Сортировка

По определению отношения как множества, оно не упорядочено. Соответственно, Oracle, как и любая другая СУБД, выдает строки в произвольном порядке — даже один и тот же запрос, выполненный несколько раз, может вернуть строки в разном порядке. В некоторых случаях (например, при группировке, использовании ключевого слова `distinct` в команде `select` или при применении определенных алгоритмов соединения отношений) наборы строк упорядочены определенным образом, однако рассчитывать на это при создании приложений нельзя.

Для сортировки результатов запроса существует фраза `order by` предложения `select`. Она может встречаться в единственном месте — в конце запроса. Ни в подзапросах, ни перед операторами `union`, `union all`, `intersect` и `minus` фраза `order by` не допускается.

После ключевого слова `order by` перечисляются выражения, по которым сортируется результат запроса. Это может быть любое выражение над столбцами соединяемых таблиц, в том числе и не входящими в список выборки, псевдоним столбца или номер столбца. После выражения может записываться одно из ключевых слов `asc` или `desc`. Они задают порядок сортировки — по возрастанию (`ascending`) или по убыванию (`descending`). По умолчанию используется сортировка по возрастанию. Можно перечислить несколько выражений через запятую — в этом случае строки, в которых значения первого выражения совпадают, будут

отсортированы по второму и т. д.

Пусть, например, надо выдать список сотрудников, упорядоченный в порядке убывания зарплаты. Сотрудников с одинаковой зарплатой упорядочить по алфавиту:

```
select name, salary
from employee
order by salary desc, name
```

или, что то же самое,

```
select name, salary
from employee
order by 2 desc, 1
```

Типы выражений во фразе `order by` можно комбинировать. Так, в предыдущем примере можно было бы написать

```
order by 2 desc, name.
```

Если, например, мы не уверены, что при написании имени соблюден регистр букв, а сумму зарплаты при сортировке нужно округлить до рубля, то можно переписать запрос так:

```
...
order by round(salary) desc, upper(name)
```

Значения `null` считаются максимальными — при сортировке по возрастанию они оказываются в конце отношения, а при сортировке по убыванию — в начале. Можно изменить такое предположение — для этого следует после имени или позиции столбца добавить фразу `nulls first` или `nulls last`<sup>1</sup>.

Всего в списке `order by` может быть не более 254 выражений. Этот список не может содержать столбцов типа `blob`, `clob`, `nclob`, `bfile`, `long`, `long raw`, а также столбцов нескаллярных типов.

В случае, если в команде `select` указано слово `distinct`, выражения во фразе `order by` могут содержать только те столбцы, которые указаны в списке выборки:

```
select subacc from debt
order by datepay -- правильно, хотя и бессмысленно

select distinct subacc from debt
order by subacc, datepay -- неправильно

select distinct subacc from debt
order by subacc -- правильно; можно также order by 1
```

В случае, если запрос содержит группировку, выражения могут включать в себя только те столбцы, которые входят в список выборки, причем если столбец вычисляется при помощи агрегатной функции, выражение также должно включать в себя эту функцию. Пусть, например, надо упорядочить отделы по количеству сотрудников, отделы с одинаковой численностью — по средней зарплате, отделы с одинаковой зарплатой — по алфавиту:

```
select d.name as deptname, count(*) as c, avg(salary) as sal
from department d, employee e
where d.isn=e.deptisn
group by d.name
order by c desc, sal desc, deptname
```

Фразу `order by` этого запроса можно переписать любым из следующих способов:

```
order by 2 desc, 3 desc, 1

order by count(*) desc, avg(salary) desc, d.name
```

## Иерархические запросы

Oracle поддерживает специальный тип запросов, называемый иерархическим запросом (*hierarchical query*). Подробнее прочитать о них можно в [10].

При иерархическом запросе из таблицы выбирается набор строк, затем дочерние строки, связанные с ними условием иерархии, затем строки-«внуки», связанные условием иерархии с дочерними строками, и т. д. Результат запроса имеет древовидную структуру. Если образуется цикл, т. е. на очередном уровне иерархии будут выбраны строки, выбранные на одном из предыдущих уровней, возникает ошибка:

```
ORA-01436: CONNECT BY loop in user data
```

Синтаксис иерархического запроса таков:

<sup>1</sup> Начиная с версии 8.1.6

```
select <список столбцов>
from <имя таблицы>
where <условие фильтрации>
start with <условие отбора первого уровня>
connect by <условие иерархии>
```

Пусть, например, нам нужно выбрать всех начальников Воробьева:

```
select level-1, name from employee
where level<>1
start with name='Воробьев В.В.'
connect by prior mgrisn=isn
order by 1
```

При выполнении этого запроса на первом шаге выбирается сотрудник Воробьев (фраза `start with name='Воробьев В.В.'`). Псевдостолбец `level` показывает, на каком шаге была выбрана строка — 1 для корневых строк, 2 — для дочерних и т.д. На втором шаге выбирается непосредственный начальник Воробьева. Фраза `connect by prior mgrisn=isn` значит, что следующий уровень составляют люди, `isn` которых равен `mgrisn` сотрудников старшего (`prior`) уровня. В данном случае на каждом уровне будет появляться одна новая строка, но из дальнейших примеров будет видно, что это не всегда так. Наконец, на третьем шаге будет выбран начальник начальника Воробьева, `mgrisn` которого равен `null`, и на этом построение дерева прекратится. Полученный набор строк фильтруется по условию `level<>1`, сортируется, и в результате получаем:

```
level-1  name
-----
1         Петров П.П.
2         Иванов И.И.
```

Для того, чтобы узнать, сколько начальников стоит над Воробьевым, можно поместить псевдостолбец `level` под агрегатную функцию:

```
select max(level-1) from employee
start with name='Воробьев В.В.'
connect by prior mgrisn=isn
```

Обратите внимание, что сначала строится целиком древовидная структура запроса, а затем она фильтруется по условию `where`. Таким образом, одно и то же условие, помещенное в фразы `where` и `connect by`, в первом случае отсекает узел, а во втором — все поддерево, начинающееся из этого узла. Первый из следующих запросов выберет всех сотрудников, подчиняющихся Иванову, кроме Петрова, второй — кроме Петрова и его подчиненных, а третий — включит в выборку Петрова, но исключит всех его подчиненных:

```
select isn, name from employee
where name<>'Петров П.П.'
start with name='Иванов И.И.'
connect by prior isn=mgrisn

select isn, name from employee
start with name='Иванов И.И.'
connect by prior isn=mgrisn and name<>'Петров П.П.'

select isn, name from employee
start with name='Иванов И.И.'
connect by prior isn=mgrisn and prior name<>'Петров П.П.'
```

Заметьте, что в этих запросах количество новых строк на каждом уровне может быть больше единицы.

Команда, выполняющая иерархический запрос, не может выполнять соединения таблиц и производить выборку из представления, содержащего соединение.

И напоследок еще один пример, разобравшись в котором, вы сможете оценить всю мощь и красоту иерархических запросов. Вывести каждого сотрудника вместе с его высшим начальником:

```
select
  e.name as ename, m.name as mgrname
from employee e, employee m
where exists (
  select * from employee
  start with isn=e.isn
  connect by prior mgrisn=isn
  where isn=m.isn
)
and m.mgrisn is null
```

# SQL. Изменение данных

*Средства, предоставляемые языком SQL для изменения данных, достаточно просты, но в то же время эффективны. Для того, чтобы полноценно ими пользоваться, необходимо знакомство с оператором выборки select.*

## Вставка данных

Вставка данных осуществляется оператором insert:

```
insert into <имя отношения> [(<столбец> { , <столбец> })]
values (<значение> { , <значение> })

или

insert into <имя отношения> [(<столбец> { , <столбец> })] [<псевдоним>]
<подзапрос>
```

Отношение может быть таблицей или представлением. Для того, чтобы представления можно было использовать в операциях изменения (т. н. updateable view), они не должны выполнять соединений, иерархических запросов, множественных операций, группировок, сортировок, а также не должны содержать фразы distinct и подзапросов. Единственный способ выполнить операцию изменения (вставки, обновления или удаления) данных в представлении, не удовлетворяющем перечисленным условиям, — написание триггера instead of.

Имена столбцов, перечисленные после имени отношения, указывают, каким столбцам будут присвоены значения, перечисленные во второй части предложения insert. Все остальные столбцы получают значения по умолчанию (default), если таковые есть, а если их нет — значения null. Если перечисление столбцов после имени отношения опущено, то считается, что перечислены все столбцы в том порядке, в котором они были созданы оператором create table (или create view). Запись без перечисления имен столбцов удобна при интерактивном выполнении команд SQL, но не рекомендуется для использования в приложениях, т. к. при изменении структуры таблицы или представления такие команды придется переписывать.

Вариант оператора insert с фразой values предназначен для вставки одной строки. Значения, перечисленные во фразе values, могут быть константами, вызовами функций, обращениями к последовательности, операциями над перечисленными выражениями, а также подзапросами, содержащими один столбец и возвращающими одну строку.

Пусть, например, надо добавить в таблицу нового сотрудника Галкина Г. Г. с окладом 95 под непосредственное руководство Петрова П. П.:

```
insert into employee (isn, name, mgrisn, salary)
values(
    seq_employee.nextval,
    'Галкин Г.Г.',
    (select isn from employee where name='Петров П.П.'),
    95
)
```

Все выражения должны быть тех же типов, что и соответствующие им столбцы, или приводиться к нужным типам преобразованиями по умолчанию. Поле hiredate (дата приема на работу) получит в приведенном примере значение по умолчанию, т. е. текущую дату (sysdate), а поле remark (примечание) — значение null.

Обратите внимание, что в Oracle нет самоувеличивающихся полей (autoincrement). Для присвоения строке уникального первичного ключа можно пользоваться такой же конструкцией, как в приведенном примере, однако чаще всего значение первичному ключу присваивает триггер — пример кода вы найдете в главе, посвященной триггерам.

Вариант предложения insert с подзапросом предназначен для вставки в таблицу нескольких строк. Подзапрос может иметь произвольную структуру и обращаться к любым представлениям и таблицам, включая ту, в которую записываются данные. На эту таблицу можно ссылаться и по псевдониму, указываемому после имени в команде insert. Пусть, например, надо вставить в таблицу debt расход на выплату зарплаты всем сотрудникам:

```
insert into debt (subacc, amount, datepay, remark)
select '50002', salary, trunc(sysdate), name from employee
```

Количество и типы столбцов, выбранных подзапросом, должны совпадать с количеством и типом столбцов отношения, в которое записываются данные; совпадение имен необязательно.

## Обновление данных

Обновление данных выполняется командой update:

```
update <имя отношения> [<псевдоним>]
set {<столбец>=<выражение> | (<столбец> {,<столбец>})=(<подзапрос>)}
[where <условие>]
```

Отношение должно быть таблицей или представлением, удовлетворяющим тем же требованиям, что и в операторе insert. Условие во фразе where предназначено для отбора строк, которые будут изменены. Оно очень похоже на условие фильтрации в операторе select, выполняющем выборку из одной таблицы. Если для отбора обновляемых строк требуется соединение таблиц, то можно воспользоваться подзапросами в предикатах in или exists. Если условие отбора не указано, то будут обновлены все строки в таблице.

Во фразе set перечисляются столбцы, которые должны быть обновлены. Выражение, присваиваемое столбцу, может быть обращением к последовательности или функции, подзапросом, возвращающим одну строку и один столбец, константой или именем столбца таблицы — в этом случае берется значение столбца до того, как были проведены какие-либо изменения.

Обновить все строки таблицы employee:

```
update employee set last_modified = sysdate
```

Увеличить зарплату Иванову на 20%:

```
update employee set salary = trunc(salary*1.2)
where name='Иванов И.И.'
```

Увеличить зарплату всем подчиненным Петрова, включая самого Петрова, на 10%, если их зарплата меньше 100:

```
update employee set salary = trunc(salary*1.1)
where isn in (
  select isn from employee
  where salary<100
  start with name='Петров П.П.'
  connect by prior isn=mgrisn
)
```

Обратите внимание, что подзапрос будет выполнен целиком до того, как в таблицу будут внесены какие-либо изменения.

Обновить строки, содержащие информацию о выплате зарплаты за последний месяц. Известно, что субсчет «зарплата» имеет код 50002, а поле empisn таблицы debt содержит ссылку на сотрудника, которому выплачена зарплата:

```
update debt d
set
  amount = (select salary from employee where isn=d.empisn),
  datepay = trunc(sysdate)
where
  subacc='50002' and
  datepay>last_day(add_months(sysdate,-1))
```

Можно изменить сразу несколько столбцов при помощи подзапроса, возвращающего одну строку и содержащего столько столбцов, сколько перечислено в круглых скобках — эта возможность иллюстрируется следующим примером. Подчинить Петрова тому же начальнику, которому подчиняется Иванов и назначить ему такую же зарплату, как и у Иванова; кроме того, записать благодарность в примечание:

```
update employee
set
  (mgrisn,salary) = (select mgrisn, salary from employee where name='Иванов И.И.'),
  remark = 'Объявлена благодарность'
where name='Петров П.П.'
```

Если надо, чтобы новая благодарность не уничтожила старую, запрос следует переписать:

```
update employee
set
  (mgrisn,salary)=(select mgrisn, salary from employee where name='Иванов И.И.'),
  remark = remark || 'Объявлена благодарность'
where name='Петров П.П.'
```

Длина поля remark будет расти, пока не достигнет длины, указанной при создании таблицы. При попытке записать в поле более длинное значение возникнет ошибка:

```
ORA-01401: inserted value too large for column
```

## Удаление данных

Удаление данных осуществляется командой delete:

```
delete from <имя отношения>  
[where <условие>]
```

Отношение должно быть таблицей или представлением, удовлетворяющим тем же требованиям, что и представления в операторах insert и update. Условие, как и в операторе update, выбирает те строки, которые будут удалены. Если условие не указано, из отношения будут удалены все строки.

Удалить все строки из таблицы debt:

```
delete from debt
```

Имейте в виду, что пространство, занимаемое таблицей debt, при этом не освободится — оно будет сохранено за этой таблицей. Если вам надо удалить все данные и освободить дисковое пространство, используйте команду truncate table.

Удалить строки, содержащие информацию о расходах по статье 50000:

```
delete from debt where subacc='50000'
```

Удалить всех сотрудников технического отдела:

```
delete from employee  
where deptisn=(select isn from department where name='Технический отдел')
```

Имейте в виду, что удаление строк из таблиц-справочников, таких как employee, — операция весьма дорогостоящая, т. к. как обычно многие таблицы связаны со справочниками правилами foreign key, которые проверяются при удалении. Поэтому хорошей практикой является сохранение в таблицах-справочниках всех строк. Вместо удаления недействительные строки можно помечать специальным значением в поле status.

# Процедурное расширение SQL — PL/SQL

*PL/SQL — процедурное расширение языка SQL, разработанное фирмой Oracle. PL/SQL-машины входят в разные продукты Oracle — например, Oracle Server или Oracle Forms. Команды PL/SQL могут выполняться как на стороне клиента, так и на стороне сервера. Использование PL/SQL на сервере позволяет расширить возможности SQL и значительно сократить сетевой трафик.*

## Общие положения

Программа PL/SQL представляет собой последовательность операторов, каждый из которых заканчивается символом `;`. PL/SQL нечувствителен к регистру букв, за исключением строковых констант.

Идентификатор может иметь длину до 30 символов. Первым символом должна быть буква, остальные — буквы, цифры и символы `_` `$` `#`. Идентификатор можно заключать в двойные кавычки. Тогда он может содержать любые символы и становится зависимым от регистра. При этом он не совпадает с таким же идентификатором, записанным без кавычек, что позволяет при необходимости использовать в качестве идентификатора зарезервированные слова.

Любая последовательность символов, заключенная между `/*` и `*/`, считается комментарием. Кроме того, комментарием считается часть строки от `--` (двух подряд идущих минусов) до конца строки.

## Типы данных PL/SQL

### Скалярные типы

PL/SQL поддерживает стандартные типы SQL, как встроенные (`number`, `char`, `varchar2`, `long` и т. д.), так и ANSI-типы (`numeric`, `decimal`, `varchar`, `integer` и т. д.). Все правила приведения типов SQL справедливы и для PL/SQL. В отличие от SQL, максимальная длина переменных типа `char` и `varchar2` составляет 32767 символов. Значения такой длины могут быть получены в результате строковых операций или при считывании из столбца типа `long`. Эти значения также могут быть сохранены в столбце типа `long`, но в столбцах типа `char` и `varchar2` могут быть сохранены значения не длиннее 2000 и 4000 байт соответственно. В блоках PL/SQL можно объявлять переменные типа `long` или `long raw`, но максимальная длина хранимых в них объектов — 32760 байт, несмотря на то, что хранимые в базе данных значения могут иметь размер до 4 Г. Можно также считывать в переменные типа `char` и `varchar2` значения столбцов типа `raw` и `long raw`. При этом каждый байт представляется двумя шестнадцатеричными цифрами, как при вызове функции `rawtohex`.

Кроме перечисленных типов в PL/SQL имеется тип `boolean`, принимающий значения `true` и `false` (и `null`), и целочисленные типы `binary_integer` и `pls_integer`. Переменные этих типов принимают значения от  $-2^{31}$  до  $2^{31}-1$ . Их отличие в том, что при операциях с переменными типа `pls_integer` используется машинная арифметика, а с переменными типа `binary_integer` — библиотечная. Операции с переменными типа `pls_integer` выполняются быстрее, чем с переменными типа `binary_integer` или `number`. Но имейте в виду, что если при выполнении операции, в которой участвует переменная типа `pls_integer`, произойдет переполнение, то возникнет ошибка, в то время как при результате операции над `binary_integer` при переполнении будет преобразован в `number`. При присваивании и использовании в операторах SQL типы `binary_integer` приводятся к `number`. Над типом `binary_integer` определены следующие подтипы: `natural` ( $0..2^{31}-1$ ), `positive` ( $1..2^{31}-1$ ), `naturaln` ( $0..2^{31}-1$ ; not null), `positiven` ( $1..2^{31}-1$ ; not null), `signtype` ( $-1, 0, 1$ ; допускается null).

Программист может определять собственные подтипы на основе уже имеющегося типа, вводя дополнительные ограничения на диапазон его значений, или на основе типа другого объекта — переменной или столбца таблицы — при помощи атрибута `%TYPE`. Для определения подтипа используется ключевое слово `subtype`:

```

subtype BirthDate is date;
subtype HireDate is date not null; -- переменным данного типа запрещается принимать
значение null
subtype salary_t is employee.salary%TYPE; -- тип совпадает с типом столбца salary таблицы
employee
subtype dos_name is varchar2(8); -- значения не длиннее 8 символов1
fname varchar2(8); -- объявление переменной
subtype dos_name2 is fname%TYPE; -- тип совпадает с типом переменной fname

```

Объявление подтипа на основе типа столбца таблицы гарантирует, что при изменении таблицы код останется корректным. При определении подтипа нельзя указать значение по умолчанию (default), но если тип определен как not null, то переменной этого типа при объявлении обязательно должно быть присвоено начальное значение, иначе код не будет скомпилирован.

Помните, что контроль границ выполняется только во время выполнения. Так, например, следующий код будет успешно скомпилирован:

```

declare
  n positiven := 2; -- обязательно присвоить начальное значение!
begin
  n := -1;
end;

```

однако при выполнении произойдет ошибка:

```
ORA-06502: PL/SQL: numeric or value error ORA-06512: at line 4
```

### Комбинированные типы

Программист на PL/SQL может определять комбинированные типы данных — записи. Для доступа к полям записи используется оператор . (точка). Синтаксис объявления типа приведен ниже:

```
type <имя> is record (<поле> <тип> {, <поле> <тип>});
```

Обратите внимание, что вместо ключевого слова subtype используется слово type. Например,

```

type time is record (
  hours number,
  minutes number,
  seconds number
);

```

Если поле записи имеет символьный тип (char, varchar2 и т. д.), то при объявлении типа обязательно указывать длину поля:

```

type full_name is record (
  name varchar2(31),
  second_name char(1),
  surname varchar2(31)
);

```

При определении комбинированного типа можно при помощи символа := или ключевого слова default указывать значение, принимаемое полем по умолчанию. Можно объявить поле как not null — в этом случае обязательно надо указать значение по умолчанию:

```

type employee_info is record (
  name full_name, -- поле записи может быть записью
  hire_date date not null default sysdate,
  salary number := 400 -- символы := и default совершенно равноправны
);

```

Комбинированный тип может быть определен на основе структуры таблицы или курсора при помощи атрибута %ROWTYPE. Например,

```
subtype employee_info is employee%ROWTYPE;
```

Такое определение гарантирует, что при изменении структуры таблицы employee код с использованием типа employee\_info останется корректным. Обратите внимание, что в этом случае снова используется ключевое слово subtype.

Переменные типа записи можно присваивать друг другу, но только если они одного и того же типа — совпадения структуры записей недостаточно. Переменной типа записи можно присвоить null — это означает присвоение null всем ее полям; если поле является записью, то null, в свою очередь, присваивается ее полям и т. д. рекурсивно. При этом переменные типа записи не подлежат сравнению и проверке на null:

<sup>1</sup> Здесь имеется расхождение с документацией к Oracle 8, в которой утверждается, что нельзя непосредственно в определении подтипа ограничивать размер базового типа. Вместо этого в документации предлагается использовать способ, показанный на следующих двух строках примера.

```

declare
  type r1 is record (a number, b number);
  type r2 is record (a number, b number);
  a r1;
  b r2;
  c r1;
begin
  a := c; -- корректно
  b := c; -- недопустимо
  a := null; -- равнозначно a.a:=null; a.b:=null;
  if a=c then ... end if; -- недопустимо
  if a is null then ... end if; -- недопустимо
end;
```

## Курсоры

Курсором в Oracle называется набор строк, полученный при помощи оператора `select`, и связанный с ним указатель текущей строки. С помощью курсоров осуществляется последовательный, запись за записью, доступ к хранимым данным. Необходимость в таком механизме возникает из-за того, что процедурный PL/SQL не может работать непосредственно с отношениями, как это происходит в реляционном SQL.

Для работы с курсором в PL/SQL существуют операции `open` (открыть курсор), `close` (закрыть курсор) и `fetch` (выбрать очередную строку). Позиционирование указателя в курсоре не предусмотрено. Однако в операторе `select` курсора может присутствовать фраза `order by` — в этом случае `fetch` будет выдавать строки в порядке сортировки.

Курсор объявляется следующим образом:

```

cursor <имя> [(<параметр> {, <параметр>})]
[return <тип>]
is <команда выборки (select)>;
```

Во фразе `return` можно указать тип записи, возвращаемой курсором. Это может быть пользовательский тип, объявленный командой `type`, или ссылка на структуру таблицы при помощи атрибута `%ROWTYPE`. Указанный тип должен соответствовать типу записи, возвращаемой оператором `select`. Например:

```

cursor c is select * from employee;
cursor d return debt%ROWTYPE is select * from debt;
```

Если в операторе `select` указана фраза `for update`, то одновременно с открытием курсора происходит блокировка строк, которая снимается при фиксации или откате транзакции.

В описании курсора можно указывать параметры, что позволяет открывать одни и те же курсоры для разных наборов начальных условий. Параметр объявляется так:

```

<имя параметра> [in] <тип> [:= <значение по умолчанию>];
```

Вместо символа `:=` можно использовать ключевое слово `default`. Параметры курсора всегда передаются только для чтения, что может быть подчеркнуто использованием необязательного слова `in`. Параметры со значениями по умолчанию записываются в конце списка. Например:

```

subtype subacc_t is debt.subacc%type;
cursor c(p subacc_t, d date := sysdate) is
  select * from debt where subacc=p and datepay=d;
```

Для задания фактических параметров при открытии курсора используется позиционная или именованная нотация. При позиционной нотации фактические параметры перечисляются через запятую в том же порядке, в котором перечислены формальные параметры. Для задания параметра по имени используется запись

```

<имя> => <значение>
```

Можно использовать обе нотации одновременно — при этом в начале списка фактических параметров используется позиционная нотация, а в конце — именованная.

Для курсора определены следующие атрибуты:

- `%ISOPEN` — true, если курсор был открыт;
- `%FOUND` — true, если выбрана очередная строка;
- `%NOTFOUND` — true, если очередная строка не найдена;
- `%ROWCOUNT` — номер текущей строки.

Значения атрибутов `%FOUND` и `%NOTFOUND` равны null до тех пор, пока не будет выполнен хотя бы один оператор `fetch`. Если `fetch` не выбрал запись (при этом атрибут `%NOTFOUND` был установлен в true), то значение переменной (переменных), перечисленных после слова `into`, не изменяется — в частности, туда не записывается null.

Для того, чтобы получить набор строк для другого набора параметров, курсор следует закрыть и открыть повторно. Открытие открытого курсора приводит к возникновению исключения `CURSOR_ALREADY_OPEN`, а любое обращение к неоткрытому курсору (в т. ч. повторное закрытие) — к возникновению исключения `INVALID_CURSOR`.

```
declare -- секция описаний, см. раздел "Анонимные блоки"
  cursor ec (pmin number:=null, pmax number:=null) return employee%ROWTYPE is
    select * from employee
      where salary between nvl(pmin, salary) and nvl(pmax,salary);
  cursor ecs (pmin number:=null, pmax number:=null) is
    select name, hiredate from employee
      where salary between nvl(pmin, salary) and nvl(pmax,salary);
  emp ec%ROWTYPE;
  vname emlooyee.name%TYPE;
  vdate date;
  c number;

begin
  /*
   * Открываем курсор и передаем ему параметр, используя именованную нотацию.
   * Второй параметр принимает значение по умолчанию (null)
   */
  open ec(pmax=>800);
  loop -- см. раздел "Циклы"
    /*
     * Извлекаем из курсора очередную запись
     * и помещаем ее в переменную emp
     */
    fetch ec into emp;
    /*
     * прерываем цикл, если записи кончились
     */
    exit when ec%NOTFOUND;
    ...
  end loop;
  close ec; -- закрываем курсор, чтобы потом открыть снова

  /*
   * Открываем курсор и передаем ему параметры, используя позиционную нотацию.
   */
  open ecs(200,400);
  loop
    /*
     * Извлекаем из курсора очередную запись
     * и помещаем ее поля в переменные vname и vdate;
     * количество и типы переменных должны совпадать
     * с количеством и типами полей
     */
    fetch ecs into vname,vdate;
    exit when ec%NOTFOUND;
    ...
  end loop;
  c := ecs%ROWCOUNT; -- сколько записей извлекли?
  close ecs;
  ...
end;
```

Для выполнения команд SQL PL/SQL открывает неявный курсор. Получить к нему доступ можно после выполнения команды, используя идентификатор SQL:

```
update employee set salary=salary*1.05 where hiredate<='1-Jan-1999';
if SQL%ROWCOUNT=0 then
  -- не найдено таких сотрудников
  ...
end if;
```

Кроме того, неявный курсор можно использовать для организации цикла, пробегающего по всем строкам, извлеченным из курсора. Подробнее об этом написано в разделе, посвященном циклам.

В PL/SQL можно объявить переменные-указатели на курсор (ref cursor), называемые также курсорными переменными (cursor variables). Для объявления переменной надо сначала объявить ее тип, а потом — переменную этого типа:

```
type ref_cursor is ref cursor; -- любой курсор
type emp_cur is ref cursor return employee%ROWTYPE; -- курсор для таблицы employee
cur emp_cur; -- объявление переменной
```

Курсорная переменная может быть объявлена локально и возвращена в качестве результата функции. Для того, чтобы связать переменную с курсором, служит команда `open ...`

for<sup>1</sup>:

```
open <имя курсорной переменной> for <запрос>;
```

С курсорной переменной можно работать так же, как с курсором: извлекать строки оператором fetch, закрывать оператором close, проверять атрибуты %FOUND, %NOTFOUND, %ISOPEN, %ROWCOUNT. Курсорной переменной нельзя присвоить null, их нельзя передавать между серверами, к ним неприменимы операции сравнения.

### Таблицы PL/SQL

Для хранения набора однородных объектов в PL/SQL предусмотрено три механизма, называемых таблицами PL/SQL. Они называются: коллекции (nested table), индексированные коллекции (index-by table) и динамические массивы (varying array). Различия между ними сведены в таблице:

|                                     | массив   | коллекция                           | индексированная коллекция                         |
|-------------------------------------|--|-------------------------------------|---|
| Состояние после объявления          | null, требует инициализации  | null, требует инициализации         | пустая коллекция                                  |
| Индексы обязаны идти подряд         | да   | нет                                 | нет   |
| Ограничение на количество элементов | задается при объявлении  | может быть расширена в любой момент | в любой момент можно обращаться к любому элементу |
| Диапазон индексов                   | 1..limit()   | 1..2 <sup>31</sup> -1               | -2 <sup>31</sup> +1..2 <sup>31</sup> -1           |
| Типы элементов (все кроме...)       | nchar, nclob, nvarchar2, ref_cursor, коллекции и массивы<br>binary_integer, pls_integer, boolean,<br>long, long raw, natural, naturaln,<br>positive, positiven, signtype<br>blob, clob, объекты с полями blob и clob |                                     |   |
| Могут храниться в БД                | да, индексы сохраняются  | да, индексы не сохраняются          | нет   |

Ниже приведен синтаксис объявления таблиц PL/SQL:

■ коллекция:

```
type <имя> is table of <тип элемента> [not null];
```

■ индексированная коллекция:

```
type <имя> is table of <тип элемента> [not null] index by binary_integer;
```

■ массив:

```
type <имя> is varray (<макс. размер>) of <тип элемента> [not null];
```

После объявления коллекция или массив имеют значение null, а для того, чтобы инициализировать их, нужно воспользоваться функцией-конструктором, которая создается автоматически и имеет то же имя, что и имя типа. Аргументами конструктора являются начальные значения элементов коллекции или массива, причем при создании массива количество фактических параметров конструктора не должно превышать размера, заданного при объявлении массива:

```
declare
    type num_array is varray(10) of number;
    sample num_array;
begin
    if sample is null then -- условие выполняется
        ...
    end if;
    sample := num_array(1,3,48,56);
    if sample is null then -- условие НЕ выполняется
        ...
    end if;
end;
```

После инициализации все элементы имеют последовательные индексы — от 1 до количества элементов, указанных при вызове конструктора. Конструктору можно не передавать па-

<sup>1</sup> В Oracle 7.x команда select для курсорной переменной не может содержать фразы for update.

раметров — в этом случае массив или коллекция не будут null'ом, но будут содержать 0 элементов. Если элемент коллекции или массива не объявлен как not null, то среди параметров конструктора могут быть null. Можно инициализировать коллекции и массивы непосредственно при объявлении:

```
declare
  type str_list is table of varchar2(32);
  l str_list := str_list('Иванов', 'Петров', 'Сидоров');
  ...
```

Память под массивы PL/SQL отводится не в момент создания, как в традиционных языках программирования, а по мере роста. Если тип массива varray(10), то это не значит, что в нем всегда 10 элементов.

В отличие от массивов и коллекций, индексированная коллекция сама по себе не может иметь значение null — она инициализируется автоматически. По сути своей это ассоциативный массив, ключом в котором служит целое число. В любой момент вы можете присвоить значение элементу индексированной коллекции с любым индексом, в то время как любое обращение к элементу коллекции или массива с индексом вне допустимого диапазона приведет к возникновению исключения SUBSCRIPT\_BEYOND\_COUNT, если индекс больше, чем количество элементов в коллекции, SUBSCRIPT\_OUTSIDE\_LIMIT, если индекс вне допустимого диапазона (отрицательный или больше максимального размера массива), или NO\_DATA\_FOUND, если индекс корректен, но элемента с таким индексом не существует.

Для того, чтобы изменить размер коллекции или массива, используются методы extend и trim:

#### ■ collection.extend[(n[,m])]

Этот метод добавляет один null-элемент в конец коллекции. Если указан параметр n, добавляется n элементов, если указан m, то вместо null добавляются копии m-го элемента. Коллекция может расти неограниченно, в то время как массив — только до своего максимального размера. Если попытаться расширить массив больше, чем это предусмотрено при объявлении типа, будет брошено исключение SUBSCRIPT\_OUTSIDE\_LIMIT.

#### ■ collection.trim[(n)]

Удаляет n (если не указано, то 1) элементов с конца коллекции. Если n больше, чем размер коллекции, то возникает исключение SUBSCRIPT\_BEYOND\_COUNT.

Коллекции и индексированные коллекции имеют метод delete, при вызове которого физический размер коллекции не изменяется:

#### ■ collection.delete(n)

Удаляет элемент с номером n из коллекции.

```
declare
  type num_list is table of number;
  list num_list := (15,13,18);
  l number;
begin
  list.delete(2); -- удаляет элемент с индексом 2
                  -- теперь индексы идут не подряд
  l := list(2);   -- исключение NO_DATA_FOUND
  list.delete(3); -- удален элемент с индексом 3
  list(3) := 19;  -- допустимо
  list(4) := 21;  -- исключение SUBSCRIPT_BEYOND_COUNT
  list.extend;
  list(4) := 21;  -- допустимо
end;
```

Кроме перечисленных методов, существуют методы, общие для всех трех видов таблиц:

#### ■ collection.exists(n)

возвращает true, если есть элемент с индексом n, и false, если индекс вне допустимого диапазона, или элемент не был инициализирован, или элемент был удален методом delete.

#### ■ collection.count

возвращает количество элементов в таблице.

#### ■ collection.limit

возвращает null для коллекций и индексированных коллекций и максимальное количество элементов для массивов.

#### ■ collection.first и collection.last

возвращают индексы соответственно первого и последнего элементов. Если таблица пустая, возвращают null.

#### ■ collection.prior(n) и collection.next(n)

возвращают индекс элемента, предшествующего элементу или следующего за элементом n. Если элемент с индексом n первый (последний), то возвращает null.

Для коллекций и индексированных коллекций единственный способ перебрать все элементы — использование `first` и `next` или `last` и `prior`:

```
e := list.first;
while e is not null loop
  ...
  do_something(list(e));
  ...
  e := list.next(e);
end loop;
```

Таблицы можно присваивать друг другу, но только в том случае, если совпадают их типы — совпадения типов элементов недостаточно:

```
declare
  type list1 is varray(10) of char(1);
  type list2 is varray(10) of char(1);
  l1 list1;
  l11 list1;
  l2 list2;
begin
  ...
  l1 := l11; -- допустимо
  l2 := l11; -- ошибка при компиляции
end;
```

## Операторы PL/SQL

### Присваивание

Для присваивания значения переменным в PL/SQL служит оператор `:=`. Переменные и значения должны быть одного типа, либо присваиваемое значение должно приводиться к типу переменной преобразованием по умолчанию.

Кроме этого, для присваивания значения переменным можно воспользоваться оператором `select ... into`. Например,

```
declare
  cemp employee%ROWTYPE;
...
select * into cemp from employee where isn=7618;
...

или

declare
  name varchar2(64);
  birthday date;
...
select name, birthday into name, birthday from employee where isn=7618;
...

```

Количество и тип возвращаемых значений должны совпадать с типом переменных. Оператор `select` должен возвращать ровно одну строку. Если оператор `select` не возвращает ничего, то возникнет исключение `NO_DATA_FOUND`, если возвращает больше одной строки — исключение `TOO_MANY_ROWS`.

Присваивать значение переменным PL/SQL можно операторами `insert`, `update` и `delete` с фразой `returning`. Например,

```
update employee set salary=800 where isn=7618
returning salary, name
into salary, name;
```

Такой оператор также должен вызывать изменение только одной строки.

К сожалению, не существует способа использовать запись целиком в операторах `update` или `insert`:

```
declare
  type erec is record (name varchar2(64), salary number);
  emp erec;
begin
  emp.name := 'Петров';
  emp.salary := emp.salary+100;
  insert into employee (name, salary)
  values (emp); -- неправильно, ошибка при компиляции
  insert into employee (name, salary)
  values (emp.name, emp.salary) -- правильно
end;
```

PL/SQL запрещает обращение к последовательностям в операторе присваивания. Вместо этого следует использовать оператор `select ... into`:

```
eisn := seq_employee.nextval;           -- ошибка
select seq_employee.nextval into eisn from dual; -- правильно
```

Арифметические и логические операции PL/SQL приведены в таблице по убыванию приоритета:

|   |   |
|---|---|
| ** , NOT  | возведение в степень, логическое «НЕ»   |
| +, -  | унарные «+» и «-»                       |
| *, /  | умножение, деление                      |
| +, -,   | сложение, вычитание, конкатенация строк |
| =, <, >, <=, >=, <>, !=, ^=,<br>is [not] null, [not] like,<br>[not] between, [not] in | сравнение                               |
| AND   | логическое «И»                          |
| OR  | логическое «ИЛИ»                        |

### Условный оператор

Условный оператор в PL/SQL имеет следующую форму:

```
if <условие> then
  <команды>;
{elsif <условие> then
  <команды>;
}
[else
  <команды>;
]
end if;
```

Условие представляет собой логическое выражение. Обратите внимание, что сравнение любого значения с null дает null — действуют те же правила, что и в SQL. Выражение  $a > 5$ , когда  $a = \text{null}$ , ни истинно, ни ложно. Для проверки значения переменной на null используйте оператор `is null`, либо пользуйтесь функцией `nvl`:

```
a := null;
if a > 5 then
  -- это не выполняется
...
elsif a <= 5 then
  -- это тоже не выполняется
...
elsif a is null then
  -- а вот это выполнится
end if;
```

### Операторы цикла

В PL/SQL существует 4 вида циклов:

- бесконечный цикл;
- цикл с предусловием;
- цикл со счетчиком;
- цикл с курсором.

Бесконечный цикл записывается так:

```
[<<метка>>]
loop
...
end loop [метка];
```

Для досрочного выхода из любого цикла служит оператор `exit`. Его синтаксис таков:

```
exit [метка] [when <условие>];
```

Указание метки позволяет прервать внешний из вложенных циклов. Указание условия во фразе `when` эквивалентно записи

```
if <условие> then exit; end if;
```

но при этом нагляднее и короче. Использование оператора `exit` вне тела цикла запрещено.

Цикл с предусловием имеет следующий синтаксис:

```
while <условие> loop
...
end loop;
```

Цикл со счетчиком записывается так:

```
for <переменная> in [reverse] <нижняя граница>..<верхняя граница> loop
...
end loop;
```

Границы цикла могут быть заданы выражениями — они вычисляются один раз перед началом выполнения цикла и не могут быть изменены в его теле. Переменная-счетчик цикла автоматически объявляется как переменная типа `integer` и уничтожается при выходе из цикла. Если в блоке `declare` была объявлена переменная с таким же именем, то переменная-счетчик скрывает ее. Присваивать переменной-счетчику значения внутри цикла запрещено. Значение счетчика последовательно увеличивается (при использовании ключевого слова `reverse` уменьшается) на 1. Обратите внимание, что нижняя граница всегда записывается перед верхней. Если ее значение больше, чем значение верхней границы, то цикл не выполняется ни разу, даже если указано ключевое слово `reverse`.

Цикл с курсором выглядит следующим образом:

```
for <переменная> in <курсор> loop
...
end loop;
```

Переменная цикла автоматически создается оператором цикла и имеет тип `<курсор>%ROWTYPE`. После завершения цикла переменная автоматически уничтожается.

Курсор может быть явно описанным курсором или командой `select`, заключенной в круглые скобки:

```
for c in (select * from employee order by name) loop
  if c.salary>=1000 then
    insert into temp values (c.name, c.salary);
  end if;
end loop;
```

или

```
declare
  cursor emp_cur (pmax number) is
    select * from employee where salary>=pmax;
begin
  for c in emp_cur(1000) loop
    insert into temp values (c.name, c.salary);
  end loop;
end;
```

Оператор цикла автоматически открывает курсор, последовательно выбирает из него строки и закрывает. Если к моменту выполнения оператора `for` явный курсор уже открыт, возникает исключение `CURSOR_ALREADY_OPEN`.

### Прочие операторы

Кроме перечисленных, в PL/SQL существуют операторы `null` и `goto`.

Оператор `null` не выполняет никаких действий. Он требуется для того, чтобы поставить перед ним метку, либо чтобы написать процедуру-заглушку (блок PL/SQL должен состоять как минимум из одного оператора).

Метка подчиняется тем же правилам формирования идентификаторов, что и остальные объекты, и записывается в двойных угловых скобках.

Оператор `goto` передает управление на указанную метку. Как и в других языках программирования, можно передавать управление во внешний блок, но нельзя передавать управление во внутренний блок или в тело цикла:

```
for c in (select * from employee) loop
  if c.salary<1000 then goto continue; end if;
  ...
  <<continue>>
  null; -- оператор необходим, чтобы поставить метку
end loop;
```

## Анонимные блоки PL/SQL

Команды PL/SQL, так же, как и команды SQL, могут выполняться непосредственно из клиентского приложения, в частности, с консоли. Конструкция, передаваемая серверу таким образом, называется анонимным блоком PL/SQL. Его общая форма такова:

```

[<<метка>>] -- метка блока
[declare
  -- описание локальных объектов
]
begin
  -- исполняемые команды
[exception
  -- раздел обработки исключений
]
end [метка];

```

### Описание объектов

В секции описаний должны быть объявлены все объекты, используемые в блоке, такие как типы, константы, переменные, курсоры, процедуры и функции.

Константы объявляются следующим образом:

```
<имя> constant <тип> := <значение>;
```

Константы могут быть только скалярных типов, определенных явно или как тип другого объекта (переменной, константы, поля записи или столбца таблицы), взятый при помощи атрибута %TYPE. Значением константы может быть null. Для константы типа varchar2 обязательно указывать длину (от 1 до 32767), для char длина по умолчанию принимается равной 1.

Похожим образом объявляются переменные:

```
<имя> <тип> [not null] [:= <начальное значение>];
```

Тип переменной также может быть объявлен явно или при помощи атрибутов %TYPE и %ROWTYPE. Если переменная объявлена как not null, то обязательно должно быть задано начальное значение. Помните, что если начальное значение не задано, то переменная инициализируется значением null, и в результате арифметических операций с участием этой переменной также будет получаться null. Вместо символа := можно использовать ключевое слово default. Примеры объявления переменных приведены ниже:

```

total number;
count pls_integer not null := 0;
name employee.name%TYPE default 'unknown';

```

Обратите внимание, что в одном предложении объявляется только одна переменная.

### Исполняемые команды

Область исполняемых команд — единственная обязательная часть PL/SQL-блока. Внутри нее, кроме операторов PL/SQL, могут встречаться вложенные блоки. Это применяется, например, для локализации обработки исключений.

В PL/SQL действуют правила видимости имен, похожие на правила других языков высокого уровня: объект, объявленный в блоке, считается локальным и извне этого блока не виден. Если компилятор не находит объект в текущем блоке, он ищет его в объемлющем блоке. Если объект не найден в самом верхнем по иерархии блоке, он ищется среди глобальных (т. е. описанных в пакете STANDARD) объектов. И, наконец, в случае неудачи компилятор фиксирует ошибку. Если глобальный объект скрыт локальным, то к глобальному можно обратиться, используя метку блока:

```

<<outer_block>>
declare
  foo number := 5;
begin
  declare
    foo number := -3;
    bar number;
  begin
    bar := abs(foo); -- abs - стандартная (глобальная) функция
                    -- значение bar - 3
    bar := bar+outer_block.foo; -- обращение к переменной внешнего блока
                               -- значение bar - 8
  end;
end outer_block;

```

### Обработка исключений

Исключением в PL/SQL называется ошибка времени выполнения. По семантике понятие исключения в PL/SQL близко к понятию исключения в C++, Object Pascal или любом другом современном языке программирования, а по синтаксису PL/SQL больше всего напоминает Ada.

Исключения бывают внутренние (предопределенные) и определяемые пользователем. Предопределенные исключения объявлены как глобальные в пакете STANDARD, их список

можно найти в документации.

Пользовательские исключения должны быть описаны явно. Синтаксис объявления исключений таков:

```
<имя> exception;
```

Исключения подчиняются тем же правилам видимости, что и другие объекты PL/SQL. Для броска пользовательского исключения используется оператор `raise`.

Если исключение не было обработано в текущем блоке, то оно передается во внешний блок. Если не нашлось подходящего обработчика, то исключение возвращается в клиентскую программу.

Для обработки исключений следует описать обработчики в секции `exception`. Синтаксис этого раздела таков:

```
exception
{when <имя> {or <имя>} then <операторы> }
[when others then <операторы> ]
end;
```

При броске исключения выполняется код от `when` с соответствующим типом исключения до следующей фразы `when` или до конца блока. Если не нашлось подходящего обработчика, но есть обработчик `when others`, то выполняется его код. Однако пользоваться этим следует с осторожностью, так как фраза `when others` блокирует все исключения, и можно заблокировать какое-либо содержательное исключение во вложенном блоке или вызываемой функции. Представьте, например, что в вызванной функции возникло исключение `E_ZERO_DIVIDE`!

В обработчике исключения можно пользоваться функциями `sqlcode` и `sqlerrm`. Функция `sqlcode` возвращает код ошибки, а `sqlerrm` — сообщение. `Sqlerrm` с параметром возвращает сообщение для исключения с заданным номером. Для пользовательских исключений `sqlcode` по умолчанию возвращает 1, а `sqlerrm` — 'User-Defined Exception'. Для того, чтобы приписать пользовательскому исключению другой код, используется прагма (`pragma` — директива транслятора)

```
exception_init (<исключение>, <код ошибки>)
```

При броске такого исключения `sqlcode` возвращает приписанный исключению код ошибки, а `sqlerrm` — сообщение, соответствующее ошибке с указанным кодом. Код исключения должен быть отрицательным.

Кроме того, существует глобальная процедура

```
raise_application_error (<код>, <сообщение>)
```

позволяющая бросить исключение с заданным кодом и сообщением. Однако эта процедура предназначена главным образом для передачи ошибок в клиентское приложение, так как бросаемое таким образом исключение не имеет имени и ловится только при помощи фразы `when others`. Код исключения, бросаемого этой процедурой, должен лежать в диапазоне от -20000 до -20999.

```
declare
  NULL_SAL exception; -- объявление пользовательского исключения
  pragma exception_init (NULL_SAL,-20101); -- с определенным кодом
  sal number;
  e1sn number;
  s1sn number;

begin
  begin
    select isn into e1sn from employee where name='Иванов И.И.';
  exception
    when others then
      e1sn := null; -- исключение обрабатывается локально
  end; -- выполнение продолжается
  if e1sn is null then
    raise_application_error(-20102,'не найден сотрудник!');
  end if;
  select amount, isn into sal, s1sn from salary
    where isn=e1sn and datepay between '1-Jan-2001' and '31-Jan-2001';
  if amount is null then
    raise null_sal;
  else
    update salary set amount=sal*1.2 where isn=s1sn;
  end if;
```

```

exception
  when NO_DATA_FOUND or TOO_MANY_ROWS then
    null; -- игнорирует
  when NULL_SAL then
    update salary set amount=0 where isn=sisn;
end; -- остальные исключения возвращаем во внешний блок

```

## Хранимые процедуры

### Создание хранимых процедур

Oracle позволяет создавать процедуры и функции. Их текст и скомпилированный код хранится на сервере. Это дает возможность ускорить выполнение приложения (за счет исключения повторной компиляции) и уменьшить трафик (за счет хранения программ на сервере).

Для создания процедуры служит команда SQL `create procedure`:

```

create [or replace]
procedure <имя> [(<параметр> {,<параметр>})] is
  -- объявление локальных объектов
begin
  -- исполняемые команды
[exception
  -- секция обработки исключений
]
end;

```

Для создания функции используется команда `create function`:

```

create [or replace]
function <имя> [(<параметр> {,<параметр>})] return <тип результата> is
  -- объявление локальных объектов
begin
  -- исполняемые команды
  ...
  return <результат>;
[exception
  -- секция обработки исключений
]
end;

```

Например:

```

create or replace
function str_reverse(pStr in varchar2) return varchar2 is
/* разворачивает строку задом наперед */
rc varchar2(4000) := '';
begin
  for i in reverse 1..length(pStr) loop
    rc := rc || substr(pStr,i,1);
  end loop;
  return rc;
end;

```

Так же, как и для анонимных блоков, обязательной является только секция исполняемых команд.

Oracle не предусматривает операции изменения текста хранимой процедуры. Для того, чтобы изменить текст, процедуру нужно уничтожить и создать заново. Однако уничтожение объекта командой `drop` приводит к потере всех объектных привилегий, с ним связанных. Фраза `or replace` команд `create view`, `create procedure`, `create function` и `create package` позволяет заменить текст существующего объекта (представления или хранимой процедуры), сохранив объектные привилегии.

Параметры определяются как

```
<имя> [in | out | in out ] <тип> [:= <значение по умолчанию>]
```

Если направление передачи параметра не указано, то по умолчанию предполагается `in`. Для типа параметра нельзя явно задавать ограничения размера, (например, `varchar2(8)`), однако тип формального параметра может быть задан с помощью атрибутов `%TYPE` и `%ROWTYPE` или как пользовательский тип.

Если параметр задан как `in`, то он ведет себя как константа — ему нельзя присвоить значение. Параметр типа `out` при вызове функции имеет значение `null`. В теле процедуры ему должно быть присвоено значение. Фактическими параметрами для формальных параметров типа `out` или `in out` должны быть переменные.

Если для параметров заданы значения по умолчанию, то количество фактических параметров может быть меньше количества формальных параметров. При вызове функции мо-

жет использоваться как позиционная, так и именованная нотация. При позиционной нотации порядок формальных и фактических параметров должен совпадать. При именованной нотации параметры записываются так:

<имя> => <значение>

Список может содержать позиционную и именованную (в конце списка) нотации одновременно. Эти правила совпадают с правилами передачи параметров при открытии курсоров.

Процедура (функция) или анонимный PL/SQL блок могут иметь вложенные процедуры (функции), которые должны быть описаны в секции объявлений. Например:

```
declare -- область описаний анонимного блока
function max_root(a number, b number, c number) return number is
/* максимальный из действительных корней квадратного уравнения */
d number;
begin
d := b*b-4*a*c;
if d<0 then return null; end if;
return greatest((-b+sqrt(d))/(2*a), (-b-sqrt(d))/(2*a));
end;
begin -- начало анонимного блока
dbms_output.put_line(max_root(1,-2,1));
end;
```

В приведенном выше примере используется процедура `put_line` стандартного пакета `DBMS_OUTPUT`. Этот пакет предназначен для передачи текстовых сообщений между приложениями. Для того, чтобы читать вывод сервера в `SQL*Plus`, нужно предварительно выдать команду `set serveroutput on`.

Oracle допускает перегрузку подпрограмм, т. е. создание подпрограмм с одинаковыми именами, если эти подпрограммы описаны в одном блоке, в одном пакете, внутри другой подпрограммы или внутри анонимного PL/SQL блока. Создать две хранимых процедуры с одинаковыми именами нельзя. Перегруженные подпрограммы должны отличаться количеством и/или типом формальных параметров.

Процедуры и функции могут принимать в качестве параметров и возвращать значения любого PL/SQL-типа, в том числе записи, таблицы PL/SQL, `ref cursor`, `long` и `long raw`, `blob` и `clob`.

Процедуры и функции могут вызывать друг друга, в том числе и рекурсивно.

### Использование хранимых функций в SQL

Хранимые функции, написанные на PL/SQL, можно использовать в командах SQL таким же образом, как и стандартные функции, например, `initcap` или `abs` — во всех местах, где допустимо выражение. Хранимые функции нельзя вызывать в правилах целостности `check` таблицы и при задании значения столбца по умолчанию. Имейте в виду, что при разрешении имен столбца таблицы имеют больший приоритет, чем вызовы функций без параметров. Если же получилось, что название столбца совпадает с именем функции, используйте для вызова функции запись `<имя схемы>.<имя функции>`. В SQL можно пользоваться только позиционной нотацией при задании параметров, но не именованной. Например, мы можем использовать в запросе функцию `str_reverse`, созданную в предыдущем примере:

```
select str_reverse(name) as sn from employee where rownum<3

sn
-----
.И.И вонавИ
.П.П вортеП
```

Для того, чтобы функцию PL/SQL можно было вызывать в командах SQL, она должна удовлетворять следующим условиям:

1. Все ее параметры должны иметь направление `in`, но не `out` или `in out`;
2. Типы аргументов и возвращаемого значения должны быть внутренними типами Oracle, т. е. `number`, `varchar2`, `date`, но не `boolean`, `ref cursor`, `record`. Если функция возвращает значение целочисленного типа PL/SQL (`natural`, `pls_integer`, `binary_integer` и т. п.), то SQL преобразует результат в тип `number`. Если параметр имеет целочисленный тип, то передаваемое значение типа `number` будет округлено. Если же фактический параметр типа `number` настолько велик, что не помещается в целое число, то при вызове функции произойдет ошибка SQL (в PL/SQL это равнозначно броску безымянного исключения);
3. Функция не должна иметь побочных эффектов.

Под побочными эффектами понимается изменение среды, способное привести к тому, что результаты функций будут зависеть от порядка вызова. Чтобы избежать этого, функция

не должна:

1. Выполнять операции insert, update или delete над любыми таблицами;
2. Читать изменяемую таблицу (если вызвана из команды insert, update или delete);
3. Управлять транзакциями (см. раздел «Управление транзакциями»);
4. Вызывать другие процедуры или функции, выполняющие эти операции.

В то же время функция может сохранять результаты своего выполнения в переменных пакета и читать их оттуда, как это будет показано ниже. Это единственный допустимый побочный эффект и непонятно, почему разработчики Oracle оставили такую возможность.

## Пакеты

### Создание пакетов

PL/SQL, кроме процедур и функций, предоставляет более высокий уровень абстракции — пакеты. Пакеты позволяют сгруппировать логически связанный код и данные. При этом пакеты состоят из двух хранимых частей — спецификации и тела. Спецификация представляет собой интерфейс, который виден из программ, использующих пакет. Тело пакета, скрытое от пользователей, содержит собственно реализацию. Такой подход позволяет менять реализацию пакета без необходимости изменения при этом использующего его кода.

Пакеты Oracle имеют следующие особенности:

- При вызове любой функции пакета весь пакет целиком загружается в память. Это ведет к ускорению выполнения других функций пакета;
- Пакет может сохранять свое состояние между вызовами в переменных, глобальных для пакета.

В спецификации пакета описываются общедоступные объекты — процедуры, функции, переменные, курсоры, типы, исключения, константы — т.е. объекты, которыми можно пользоваться извне. Спецификация пакета пишется следующим образом:

```
create [or replace] package <имя пакета> is
/*
    описание объектов, доступных извне
    (прототипы процедур и функций, курсоры, переменные)
*/
end;
```

Например:

```
create or replace package employee_pack is
    type emp_rec is employee%ROWTYPE; -- экспортируемый тип
    counter number(10); -- экспортируемая переменная; так лучше не делать
    procedure increase_salary (percent number); -- прототип, т.е. заголовок без тела
    function get_name (pIsn number);
end;
```

Обратите внимание, что в заголовке пакета находятся только прототипы процедур, функций и курсоров. Их тела находятся в реализации пакета. Если в пакете нет объявленных процедур, функций и курсоров, то он может не иметь тела.

Для того, чтобы извне обратиться к объекту пакета, используется имя пакета с оператором . (точка). Так, например, чтобы вызвать процедуру increase\_salary пакета emp\_package, надо написать

```
emp_package.increase_salary(10);
```

Тело пакета записывается так:

```
create [or replace] package body <имя пакета> is
/*
    описание объектов (переменных, курсоров, процедур с телами и т.д.), недоступных извне
*/
/*
    тела процедур, функций и курсоров, объявленных в спецификации
*/
[begin
    -- инициализация пакета
]
end;
```

В теле пакета должны быть реализованы все процедуры, функции и курсоры, объявленные в заголовке. Кроме того, в теле могут быть реализованы подпрограммы и объявлены переменные, недоступные внешним модулям. Пакет может (но не обязан) иметь раздел инициализации, который выполняется один раз при загрузке пакета. Пакет загружается в память в тот момент, когда первый раз за сессию происходит обращение к какому-либо из объектов пакета.

Переменные, глобальные для пакета (т. е. описанные вне процедур и функций пакета), сохраняют свое значение между вызовами в пределах сессии:

```
package body access is
count binary_integer:=0;
procedure control is
begin
    count:=count+1;
    ...
end;
end;
```

В приведенном примере значение переменной `access.count` показывает, сколько раз за сессию была вызвана процедура `access.control`. Имейте в виду, что значение одной и той же переменной в разных сессиях никак друг от друга не зависят, а если в приложении используется какой-либо мультиплексор соединений, то значение глобальной переменной пакета непредсказуемо.

Глобальные переменные пакета не могут иметь тип `ref cursor`.

### Использование пакетов в SQL

В SQL можно использовать функции, описанные в пакетах, но нельзя использовать переменные и константы. Для обращения к функции используется запись `<имя пакета>.<имя функции>`. Требования к функциям пакета те же самые, что и к обыкновенным хранимым функциям. Однако транслятор PL/SQL не может определить «чистоту» функции, так как реализация может изменяться. Для того, чтобы сообщить компилятору, что функция «чистая», используется прагма `restrict_references`:

```
pragma restrict_references (<имя функции>, WNDS [,WNPS] [,RNDS] [,RNPS]);
```

- WNDS — “writes no database state” — не выполняет команд `insert`, `update`, `delete`, т. е. не изменяет состояние БД.
- WNPS — “writes no package state” — не изменяет значение переменных пакета (своего или чужого).
- RNDS — “reads no database state” — не читает состояние БД, т. е. не содержит операторов `select`.
- RNPS — “reads no package state” — не читает значения переменных пакета.

Минимально допустимый для использования в SQL уровень чистоты функции — WNDS. Прагма `restrict_references` записывается в спецификации пакета после объявления функции, но необязательно сразу за ним.

Если реализация функции нарушает уровень чистоты, заданный прагмой, компилятор фиксирует ошибку. Все аргументы прагмы независимы — так, из RNPS не следует WNPS.

Имейте в виду, что при вызове любой функции пакет может быть загружен впервые, значит, при этом выполняется раздел инициализации. Следовательно, раздел инициализации должен соответствовать максимальному уровню чистоты функций пакета:

```
create package salary_control is
function get_max_salary (pl date pr date) return number;
pragma restrict_references (get_max_salary, WNDS, WNPS);
function credit_state (id number) return char;
pragma restrict_references (credit_state, WNDS, RNPS);
...
end;
```

В приведенном выше примере раздел инициализации должен удовлетворять уровню чистоты WNDS, WNPS, RNPS. Можно явно указать уровень чистоты раздела инициализации, указав в прагме `restrict_references` вместо имени функции имя пакета.

Обратите внимание, что путем сохранения своего состояния в переменной пакета функция может возвращать различные значения в разные моменты:

```
create or replace package do_not_use is
function bad return number;
pragma restrict_references (bad,WNDS);
end;

create or replace package body do_not_use is
internal_var number := 0;
function bad return number is
begin
    internal_var := internal_var+1;
    return internal_var;
end;
end;
```

В данном примере функция `do_not_use.bad` будет возвращать значения, увеличиваю-

щиеся на единицу. Излишне говорить, что широкое использование этой возможности в приложениях ведет к трудноуловимым логическим ошибкам.

### Стандартные пакеты

С сервером Oracle поставляется набор стандартных пакетов. Реализация этих пакетов скрыта, а заголовки можно посмотреть в подкаталоге `|Ora81|RDBMS\ADMIN` каталога Oracle. Некоторые из стандартных пакетов перечислены ниже.

- **DBMS\_ALERT**. Содержит процедуры, предназначенные для асинхронного оповещения приложений о событиях БД;
- **DBMS\_JOB**. Позволяет запускать процедуры на сервере по расписанию;
- **DBMS\_LOB**. Содержит процедуры и функции для работы с большими объектами (LOB);
- **DBMS\_OUTPUT**. Позволяет передавать текстовые сообщения между приложениями;
- **DBMS\_PIPE**. Позволяет организовывать каналы для передачи сообщений между сессиями;
- **DBMS\_RANDOM**. Содержит генератор случайных чисел;
- **DBMS\_ROWID**. Содержит процедуры для разбора значений rowid;
- **DBMS\_SQL**. Позволяет динамически формировать команды SQL из PL/SQL;
- **STANDARD**. Содержит описания стандартных типов PL/SQL. В отличие от остальных пакетов, для обращения к объектам этого пакета его имя не указывается.

Полный список стандартных пакетов и документацию по реализованным в них функциям можно найти в [12].

## Управление транзакциями

Как и при выполнении операторов SQL, при выполнении блока PL/SQL Oracle автоматически открывает транзакцию. Для управления транзакциями существуют следующие операторы:

- `commit [work] [comment '<текст комментария>'];`

фиксирует все изменения, сделанные в текущей транзакции, и делает их доступными для других транзакций. Команда `commit` освобождает все сегменты отката и уничтожает все блокировки. Так, например, если курсор был открыт с предложением `for update`, то после выполнения команды `commit` попытка выборки данных из этого курсора оператором `fetch` приведет к ошибке.

Промежуточные вызовы `commit` могут использоваться, например, для выполнения транзакций, изменяющих большое количество данных. Так, например, если оператор

```
update salary set amount=amount/1000 where datesal<'1-Jan-1998'
```

завершится с ошибкой

```
ORA-1555: snapshot too old: rollback segment number 1 with name "RBS_DEFAULT" too small
```

попробуйте выполнить его так:

```
begin
  loop
    update salary set amount=amount/1000
      where datesal<'1-Jan-1998' and rownum<3000;
    exit when SQL%ROWCOUNT=0;
    commit;
  end loop;
end;
```

При такой записи после изменения каждых 3000 строк изменение сохраняется в БД.

- `savepoint <имя>;`

устанавливает точку сохранения. Oracle запоминает, какие изменения были сделаны к моменту установки точки сохранения, но не пишет эти изменения в БД и не освобождает занятые транзакцией ресурсы.

- `rollback [to [savepoint] <имя точки сохранения>];`

откатывает все изменения, сделанные транзакцией. При этом также освобождаются ресурсы, занятые транзакцией. Если в команде `rollback` указана точка сохранения, то уничтожаются результаты только тех команд, которые выполнены с момента установки точки до вызова `rollback`. Обратите внимание, что откатываются только изменения в данных таблиц. Значения переменных пакетов, текущие значения последовательностей и т. п. остаются без изменений.

```
declare
  d number;
begin
  d := 0;
  update employee -- автоматически открывается транзакция
  set salary=10000 where isn=3861;
  savepoint a1;
  delete from employee where isn=3864;      -- 2
  d := 1;
  savepoint a2;
  delete from salary where amount<83.40;    -- 3
  d := 2;
  rollback to a2; -- аннулируются результаты команды 3
                  -- точка сохранения a2 остается
                  -- значение переменной d по-прежнему 2
  delete from salary where amount<0;        -- 4
  rollback to a1; -- аннулируются результаты команд 4 и 2
                  -- точка сохранения a1 остается
                  -- точка сохранения a2 уничтожается
                  -- значение переменной d по-прежнему 2
  rollback to a2; -- ошибка; точка a2 уничтожена предыдущей командой
  commit;        -- фиксируется результат команды 1;
                  -- уничтожаются все точки сохранения
end;
```

# Триггеры

*Триггером называется последовательность команд, выполняемая автоматически каждый раз при возникновении определенного события. Стандарт SQL99 предусматривает наличие триггеров, но никак не описывает их синтаксис, поэтому каждый производитель СУБД реализует триггеры по-своему. В частности, в Oracle триггеры пишутся на языке PL/SQL, что позволяет выполнять внутри триггеров достаточно сложные операции.*

## Создание триггеров

Триггер срабатывает при возникновении некоторого события. В Oracle 7.x таким событием является изменение (insert, update или delete) данных в таблице. Oracle 8 позволяет также определить триггеры, срабатывающие при изменении схемы базы данных (create, alter или drop), при запуске или останове сервера, при начале или завершении сессии, однако эти дополнительные возможности здесь не рассматриваются.

Триггер создается командой create trigger. Ее синтаксис приведен ниже:

```
create [or replace] trigger <имя>
(before | after | instead of)
(insert | delete | update [of <столбец> {,<столбец>}])
{or (insert | delete | update [of <столбец> {,<столбец>}])}
on <имя таблицы или представления>
[[referencing [old [as] <имя>] [new [as] <имя>]] for each row [when <условие>]]
<PL/SQL блок>
```

Имя триггера должно быть уникальным среди триггеров схемы, но оно может совпадать с именем таблицы или хранимой процедуры. Таблица может иметь сколь угодно много триггеров. Каждый триггер срабатывает на определенное событие — insert, update или delete, но есть возможность обрабатывать одним триггером несколько событий, перечислив их при создании триггера через or.

В триггере на изменение (update) можно явно перечислить столбцы, при изменении которых срабатывает триггер; если столбцы не указаны явно, триггер срабатывает при любом изменении данных.

## Выполнение триггеров

Oracle позволяет создавать два типа триггеров: триггер на уровне команды (statement trigger) и триггер на уровне строки (row trigger). Первые выполняются всякий раз при выполнении команды, независимо от того, сколько строк затрагивает команда, пусть даже и ни одной. Вторые выполняются для каждой затронутой изменением строки. Чтобы указать, что триггер должен срабатывать для каждой строки, используется фраза for each row. В этой же фразе после ключевого слова when можно указать дополнительные условия срабатывания триггера.

По времени срабатывания триггеры делятся на две категории: до выполнения команды (before), и после (after). При выдаче команды на изменение данных Oracle выполняет следующую последовательность действий:

1. Выполняются триггеры before на уровне команды (before statement);
2. Для каждой строки, на которую действует команда
  - 2.1. Выполняются триггеры before на уровне строки (before each row);
  - 2.2. Выполняется изменение данных. При этом строка блокируется, и блокировка не отпускается до конца транзакции;
  - 2.3. Выполняется проверка правил целостности для измененной строки;
  - 2.4. Выполняются триггеры after на уровне строки (after each row).
3. Завершается проверка правил целостности;
4. Выполняются триггеры after на уровне команды (after statement).

Определение этой модели рекурсивно, так как команды внутри триггеров могут изменять данные в других таблицах, вызывая тем самым выполнение триггеров для них. Если для таблицы создано несколько одинаковых (срабатывающих на одно и то же событие) триггеров, то порядок их выполнения непредсказуем.

Триггеры before и after можно создавать только для физических таблиц. Кроме того, в Oracle 8 появилась возможность создавать триггеры для представлений. С их помощью можно организовать изменение представлений в случаях, когда это не поддерживается автома-

тически. Такие триггеры составляют третью категорию по времени срабатывания — «вместо» (instead of), и всегда работают на уровне строки. При выполнении команды, изменяющей данные представления, на самом деле вызывается триггер.

## Особенности кода триггеров

### Дополнительные переменные и предикаты

Как говорилось выше, один и тот же триггер может обрабатывать несколько событий. Для того, чтобы определить, какое именно событие произошло, используются предикаты `inserting`, `updating` и `deleting`:

```
create or replace trigger employee_biuds
before insert or update or delete on employee
begin
  if inserting then
    -- выполнится только при вызове команды insert
  end if;
  -- выполнится всегда
end;
```

С предикатом `updating` можно использовать имя конкретного столбца:

```
if updating ('salary') then -- условие 1
...
end if;
```

В триггерах на уровне строки можно обращаться к старым (до выполнения команды, вызвавшей срабатывание триггера) и новым (которые должны быть получены после выполнения команды) значениям столбцов через переменные `old` и `new`. Они объявляются неявно как `<отношение>%ROWTYPE`. При выполнении команды `delete` обращение к любому полю `new` дает `null`; при выполнении `insert` обращение к любому полю `old` дает `null`; лишь при выполнении `update` доступны как `old`, так и `new`. Если слова `old` или `new` имеют в вашей программе особое значение (например, существует таблица `old`), то можно заменить их на другие имена во фразе `referencing`:

```
create or replace trigger salary_aui
after insert or update on salary
referencing old as obsolete for each row when (new.amount>10000)
begin
  insert into salary_log (isn, amount, datechange)
  values (:obsolete.isn, :obsolete.amount, sysdate);
end;
```

Обратите внимание, что перед переменными `old` и `new` (или их заменяющими) в теле триггера ставится двоеточие. Однако при использовании этих переменных во фразе `when` двоеточие не ставится.

Не всегда сравнением старых и новых значений можно заменить предикат `updating`. Например, условие

```
if :new.salary != :old.salary then -- условие 2
...
end if;.
```

не эквивалентно условию 1, т. к. при выдаче команды

```
update employee set salary=200 where salary=200
```

оно будет ложным, в то время, как первое условие будет истинно. Кроме того, в таком виде второе условие не выполнится, если старое или новое значение поля `salary` — `null`.

Хорошей практикой является такое именование триггеров, при котором понятно, к какой таблице они относятся и при каких действиях срабатывают. Например, здесь и далее используется следующее правило: сначала идет имя таблицы, затем подчеркивание, затем символ `b`, `a` или `i`, показывающий, когда сработает триггер (`before`, `after` или `instead of`), затем перечисляются действия (`i`, `u` и `d` — `insert`, `update` и `delete`) и, если триггер срабатывает на уровне команды, добавляется `s`.

### Триггеры и транзакции

В теле триггера запрещено использовать команды управления транзакциями `commit`, `rollback` и `savepoint`. Если в самом триггере либо в процедуре или функции, вызываемой из триггера, встречаются команды управления транзакциями, то такой триггер будет скомпилирован, но вызовет ошибку при выполнении.

Любая ошибка при выполнении триггера (команда управления транзакцией, необработанное исключение и т. п.) влечет за собой откат изменений, сделанных текущей командой,

причем неважно, произошла ли эта ошибка в триггере, относящемся непосредственно к изменяемой таблице, или в триггере таблицы, изменения в которой произошли вследствие работы первичного триггера. Неважно также, произошла эта ошибка в триггере before или after (или instead of). В частности, если в триггере содержится ошибка, и он не может быть откомпилирован, попытка его выполнения приводит к ошибке, и изменения в таблицу не могут быть внесены.

Триггер может находиться в выключенном состоянии — в этом случае он не влияет на выполнение SQL-команд, даже если в триггере содержится ошибка. Для включения/выключения триггеров используется команда

```
alter trigger <имя> (enable | disable)
```

Если планируется загрузка в таблицу большого объема данных, то в целях увеличения производительности можно выключить (и потом включить) все триггеры для таблицы. Делается это командами

```
alter table <имя> (enable | disable) all triggers
```

Имейте в виду, что включение триггера не приводит к его автоматическому выполнению для ранее измененных строк таблицы, в отличие от правил целостности. Таким образом, проверка данных при помощи правил целостности более надежна.

При выполнении команды триггеры before на уровне команды могут быть выполнены повторно. Если в первый раз были сделаны какие-то изменения в БД, то они не будут видны при втором исполнении, но если менялись переменные пакетов, то их значения не будут восстановлены. Обращайте на это внимание при проектировании пакетов, используемых в триггерах.

### Изменение данных в триггерах

Триггеры могут вносить изменения в данные других таблиц, вызывая тем самым выполнение других триггеров, которые, в свою очередь, могут изменять данные третьих таблиц... Уровень вложенности этих триггеров ограничен и регулируется переменной MAX\_OPEN\_CURSORS.

Триггеры на уровне строки не могут читать или изменять данные в изменяемой таблице — таким образом Oracle обеспечивает целостность чтения для триггеров:

```
create trigger employee_bu
before update on employee for each row
declare
    cur_salary number;
begin
    select salary into cur_salary from employee where isn=133;
    ...
end;
```

Предположим, что выдана команда

```
update employee set salary=salary+10 where isn between 100 and 300
```

Таблица employee, как и в предыдущих примерах, содержит данные

| isn | name        | hiredate    | salary | mgr | isn |
|-----|-------------|-------------|--------|-----|-----|
| 233 | Иванов И.И. | 28-Sep-2001 | 200    |     |     |
| 133 | Петров П.П. | 19-Sep-2001 | 150    | 233 |     |

Поскольку порядок обработки строк произволен, строка, описывающая Петрова, может быть обработана как до, так и после строки, описывающей Иванова. При этом переменная cur\_salary может получить значение как 150, так и 160. Следовательно, выполнение такого триггера недопустимо. Он будет скомпилирован, но при попытке выполнения Oracle вернет ошибку:

```
ORA-04091: table employee is mutating, trigger/function may not see it.
```

То же самое относится к хранимым процедурам, вызываемым из триггера, и другим триггерам, которые могут быть выполнены в результате выполнения команд текущего триггера. Для того чтобы триггер вносил изменения в другие строки таблицы, необходимо написать три триггера: триггер before на уровне команды инициализирует PL/SQL-таблицу в пакете, триггер after на уровне строки записывает в таблицу необходимые изменения, а триггер after на уровне команды переносит эти изменения из таблицы PL/SQL в таблицу БД.

Триггер, однако, может вносить изменения непосредственно в изменяемую строку. Делается это путем присвоения значений полям переменной new (или ее заменяющей) в триггере before each row. Изменения выполняются до проверок правил целостности, что дает триггеру возможность привести данные в соответствие с ними (см. пример 1).

## Примеры триггеров

### 1. Запись уникального значения в поле, являющееся первичным ключом

```
create or replace trigger employee_biu
before insert or update on employee
for each row
begin
    if updating('isn') then
        /*
            первый аргумент процедуры raise_application_error -
            код ошибки - произвольное число
            в диапазоне от -20000 до -20999
        */
        raise_application_error(-20600, 'Нельзя изменять первичный ключ!');
    end if;
    if :new.isn is null then -- если ключ не задан явно, генерируем уникальное значение
        select seq_employee.nextval into :new.isn from dual;
    end if;
end;
```

### 2. Запрет изменения таблицы в выходные и праздники

```
create or replace trigger employee_biuds
before insert or update or delete on employee
declare
    c number;
begin
    /*
        территория по умолчанию AMERICA;
        для территории CIS надо проверять 6 или 7
    */
    if to_char(sysdate, 'D')='1' or to_char(sysdate, 'D')='7' then
        raise_application_error(-20500, 'Выходной день');
    end if;
    /*
        таблица holidays содержит информацию о праздниках
    */
    select count(*) into c from holidays
    where dateholiday=trunc(sysdate);
    if c>0 then
        raise_application_error(-20500, 'Праздник');
    end if;
end;
```

### 3. Протоколирование изменений

```
create or replace trigger employee_aiud
after insert or update or delete on employee
for each row
declare
    action char(1);
    ctime date;
    csn number;
begin
    select sysdate into ctime from dual;
    /*
        :new.last_modified := ctime;
        ошибка! нельзя изменять данные в триггере after
        если необходимо сохранять время модификации так,
        чтобы оно совпадала с временем в
        employee_log, надо записывать его в триггере before
        и сохранять значение в переменной пакета
    */
    csn := :new.isn;
    if inserting then
        action := 'I';
    elsif
        updating then action := 'U';
    else
        action := 'D';
        csn := :old.isn;
    end if;
    insert into employee_log (isn, action, timemod)
    values(csn, action, ctime);
end;
```

# Оптимизация запросов

*Язык SQL – не алгоритмический, а декларативный, т. е. программист говорит лишь, что ему нужно, но не указывает, как это сделать. Тем не менее, знание того, как именно выполняются запросы, весьма полезно и в некоторых случаях позволяет существенно ускорить работу. Более того, Oracle, в отличие от других СУБД, позволяет в явном виде указать алгоритм выполнения запроса. В этой главе рассказывается о том, как из знания применяемых алгоритмов и структур данных извлечь практическую пользу. Не следует, однако, забывать, что грамотное проектирование приложения и структуры данных может оказать значительно большее влияние на производительность, чем все технические ухищрения вместе взятые. Кроме того, огромную роль играют личный опыт и интуиция.*

## Оптимизатор

### Основные понятия

Выполнение SQL-запроса состоит из двух шагов — определение алгоритма выполнения и преобразование алгоритма в исполняемый код. Часть транслятора, преобразующая SQL-предложение в алгоритм, называется оптимизатором. Оптимизатор вычисляет значения выражений, которые можно вычислить на этапе компиляции. Если запрос содержит обращения к представлениям, оптимизатор может включить в запрос полный текст представления и оптимизировать полученный текст целиком. Затем из всех возможных алгоритмов выполнения запроса выбирается наилучший.

Алгоритм (план) выполнения запроса представляет собой дерево, в листьях которого находятся операции чтения данных, а узлы являются операциями над этими данными. Каждый узел использует данные, полученные от дочерних узлов. Некоторые операции, такие как сортировка и группировка, требуют получения всех данных, другим, например, некоторым алгоритмам соединения, достаточно получить одну строку для начала работы. Узнать алгоритм выполнения запроса можно командой `explain plan`, которая записывает план в специальную таблицу. Формат таблицы описан в файле `utlxplan.sql`. Для визуализации плана запроса существует множество средств, например, SQL Navigator компании Quest Software.

Оптимизатор Oracle может работать в двух режимах — синтаксическом и стоимостном. Синтаксический оптимизатор исходит из predetermined стоимости (ранга) операций — из всех возможных алгоритмов выбирается алгоритм с наименьшим рангом. Стоимостной оптимизатор учитывает кроме «качества» алгоритмов еще и свойства данных, которые хранятся в таблицах. Например, стоимостной оптимизатор может отказаться от использования индекса, в котором много повторяющихся значений. Для сбора статистики по данным служит команда `analyze`, синтаксис которой можно найти в [8].

Целью работы оптимизатора может быть скорейшее получение первых результатов запроса (для интерактивных приложений) или получение полного результата (для пакетной обработки, например, построения отчетов).

Режим работы оптимизатора задается при помощи переменной `OPTIMIZER_MODE`. Значение переменной может быть изменено для экземпляра в целом или для конкретной сессии при помощи команды `alter session`. Переменная может принимать следующие значения:

- `CHOOSE` (по умолчанию) — позволяет оптимизатору самостоятельно выбрать режим работы, исходя из наличия статистики по используемым данным;
- `ALL_ROWS` — стоимостной оптимизатор, цель которого как можно быстрее получить результат целиком;
- `FIRST_ROWS` — стоимостной оптимизатор, цель которого как можно быстрее получить первые строки;
- `RULE` — включает синтаксический оптимизатор.

Кроме того, при помощи подсказок (хитов — `hints`) режим работы оптимизатора можно указывать индивидуально для каждого запроса.

### Методы доступа к данным

Данные из таблиц могут быть получены либо полным чтением таблицы (`table access full`), либо через обращение к индексу. Выборка по индексу эффективна в случае, когда считывается небольшое количество данных (в разных приложениях — от 2 до 25% строк). Неболь-

шие (до 5000 строк, в зависимости от объема оперативной памяти и количества одновременно работающих пользователей) таблицы всегда выгоднее читать целиком. При больших выборках следует учитывать, что доступ с использованием индекса увеличивает количество операций чтения, т. к. читается сначала индекс, а потом уже сама таблица, причем не подряд, а вразнобой. Обратите внимание, что имеет значение не количество строк в результирующем отношении, а количество считанных блоков данных.

Существует несколько вариантов чтения традиционных (построенных на B-деревьях) индексов — `unique scan`, `range scan`, `full scan` и `fast full scan`. `Unique scan` используется для доступа по первичному или уникальному ключу; он возвращает не более одной ссылки на строку (`rowid`). `Range scan` используется при доступе по неуникальному индексу. И, наконец, при `full scan` индекс читается целиком. `Full scan` может применяться тогда, когда хотя бы один из индексируемых столбцов имеет ограничение `not null`, т. е. в индекс входят все строки таблицы. Если все столбцы, требуемые в запросе, содержатся в индексе, то Oracle берет их значения прямо из индекса, чем существенно уменьшает количество считанных блоков. `Fast full scan` отличается от `full scan` тем, что индекс читается несколькими параллельными процессами. `Fast full scan` не гарантирует, что данные поступят в порядке возрастания (убывания) индексируемого значения и, следовательно, не может быть использован для исключения операции сортировки.

При использовании стоимостного оптимизатора возможна такая операция, как `index join` — в этом случае вместо чтения таблицы Oracle читает несколько индексов. `Index join` позволяет сократить количество прочитанных блоков и избежать сортировки. Операция `index join` не имеет ничего общего с соединением (`join`) отношений.

Для ускорения чтения данных фирменная документация Oracle дает несколько советов:

1. Для хранения строк используйте `varchar2` вместо `char` — этим вы сэкономите место за счет переменной длины `varchar2`;
2. В Oracle 7.x не храните большие объекты (`long` и `long raw`) в часто используемых таблицах — лучше вынести их в отдельную таблицу. Oracle 8 поддерживает `long` и `long raw` только для совместимости с Oracle 7 — в новых приложениях рекомендуется использовать типы `clob` и `blob`;
3. Создавайте индексы после заполнения таблиц — это позволит ускорить чтение индексов. Если таблица интенсивно растет, имеет смысл время от времени перестраивать индексы.

### Методы соединения отношений

Oracle использует три алгоритма соединения отношений: `nested loops`, `sort-merge` и `hash join`.

`Nested loops` — простейший алгоритм, при котором для каждого кортежа внешнего (`outer`) отношения просматриваются все кортежи внутреннего (`inner`) отношения и из них выбираются те, которые удовлетворяют условию соединения. Поскольку результат первого же сравнения может быть положительным, у `nested loops` нет задержки перед выдачей первого кортежа. Кроме того, этот метод не накладывает никаких ограничений на условие соединения и не требует дополнительной памяти.

Алгоритм `sort-merge` применим только для эквисоединений. При выполнении этого алгоритма оба отношения сортируются по ключу соединения, а затем пары кортежей с одинаковыми значениями ключа «сливаются». Этот алгоритм выгодно использовать, когда отношения уже отсортированы — например, данные считаны из индекса или отношение получено в результате выполнения операторов `union`, `intersect` и `minus`.

`Hash join` также пригоден только для эквисоединений. При использовании этого алгоритма подходящие пары составляются на основе значения хэш-функции. `Hash join` хорош для пакетной обработки, т. к. при готовой хэш-таблице соединение выполняется достаточно быстро. К недостаткам этого метода можно отнести высокие накладные расходы на построение хэш-таблицы и невозможность использования индексов, поскольку хэш-функция немонотонна.

При любых режимах работы оптимизатор старается изменить порядок соединения таблиц так, чтобы первыми в списке стояли отношения, из которых выбирается одна строка — такие случаи определяются по наличию первичного или уникального ключа. В случае открытого соединения в последнюю очередь читаются таблицы, со стороны которых соединение открыто. Затем оптимизатор выбирает наилучшие методы для чтения данных и соединения таблиц. Подробно алгоритм работы оптимизатора описан в [9].

## Подсказки оптимизатору (хинты)

В отличие от других СУБД, Oracle позволяет явно указать в запросе, как его выполнять. Предназначенная для этого конструкция называется `hint`. Русский перевод этого слова — «подсказка» — не используется в разговорной речи.

Хинт представляет собой комментарий особого вида. Этот комментарий должен располагаться на той же строке, что и команда SQL (`select`, `update`, `insert` или `delete`) и начинаться со знака `+`. Например:

```
select --+ ordered use_nl(e)
  d.name as deptname, e.name as empname
from department d, employee e
where d.isn=e.deptisn and d.name like 'Б%'
```

или

```
select /*+ ordered use_nl(e) */
  d.name as deptname, e.name as empname
from department d, employee e
where d.isn=e.deptisn and d.name like 'Б%'
```

Хинт состоит из зарезервированного слова и, возможно, аргументов. Если в комментарии-хинте встречается слово, не являющееся хинтом, то оно и все дальнейшие символы считаются обыкновенным комментарием. Хинты нечувствительны к регистру букв. Обратите внимание, что если у хинта есть аргументы, то, во-первых, между хинтом и скобкой не должно быть пробелов, а во-вторых, аргументы разделяются не запятыми, а пробелами. Если в запросе указываются псевдонимы к таблицам (как `d` и `e` в приведенном запросе), то аргументами хинта будут не имена таблиц, а именно псевдонимы.

Oracle не дает гарантии, что будет в точности следовать указанным хинтам. В частности, если несколько хинтов противоречат друг другу или хинты ошибочны (например, предписывают пользоваться несуществующим индексом), то они игнорируются.

Как говорилось раньше, можно выбирать режим работы оптимизатора для каждого запроса. Для этого служат хинты `choose`, `rule`, `all_rows` и `first_rows`, каждый из которых включает соответствующий режим:

```
select --+ all_rows
  d.name as deptname, e.name as empname
from department d, employee e
where d.isn=e.deptisn and d.name like 'Б%'
```

Для выбора метода доступа существует 20 различных хинтов. Здесь рассмотрены лишь некоторые из них.

- `full(<имя таблицы>)` — заставляет оптимизатор читать таблицу целиком, не используя индексы;
- `index(<имя таблицы> {<имя индекса>})` — сообщает оптимизатору, что к таблице надо обращаться, используя индекс. Если конкретный индекс не указан, то оптимизатор выберет наилучший. Возможно, что он будет использовать сразу несколько индексов. Если указать конкретный индекс, то оптимизатор будет использовать его. Если указать несколько индексов, то оптимизатор выберет наилучший из указанных индексов или, возможно, какую-то их комбинацию;
- `index_asc(<имя таблицы> {<имя индекса>})` и `index_desc(<имя таблицы> {<имя индекса>})` — то же, что и `index`, но задается еще и порядок сканирования индекса;

Заметьте, что с помощью хинта `index` легко решается задача о хит-параде: пусть есть таблица `sales`, содержащая информацию о продажах, и есть индекс по полю `amount`, содержащему количество проданных экземпляров. В таком случае следующий запрос вернет 10 наиболее продаваемых товаров:

```
select --+ index_desc(s x_sales_amount)
  name, amount
from sales s
where rownum<10
```

Этот запрос хорош при интерактивной работе с сервером, но в приложениях же лучше использовать более универсальный и надежный способ — курсор с фразой `order by` в операторе `select`.

- `no_index(<имя таблицы> {<имя индекса>})` — запрещает при выполнении запроса использовать перечисленные индексы для доступа к данным таблицы. Если ни один индекс не указан, то хинт запрещает использование всех индексов для данной таблицы. Если одновременно с этим хинтом указан один из хинтов, предписывающий использовать индекс, то оптимизатор игнорирует все хинты и выбирает индексы сам на основе имею-

щейся у него информации.

В некоторых случаях программист знает, из какой таблицы будет извлечено меньше строк. Например, в приведенном выше примере «отделы – сотрудники» строк, выбранных из таблицы `department`, заведомо меньше, чем строк из таблицы `employee`. В таком случае выгодно выбрать таблицу `department` в качестве ведущей (*outer*) при соединении. Для того, чтобы указать оптимизатору такой порядок соединения, существует хинт `ordered` – в этом случае отношения соединяются в том порядке, в котором они перечислены в фразе `from` оператора `select`. Если соединение открытое, то та таблица, со стороны которой оно открыто, участвует в соединении последней, независимо от ее положения в списке `from`.

Программист может выбрать алгоритмы, используемые для соединения отношений. Для этого предназначены хинты `use_nl`, `use_merge`, `use_hash` и некоторые другие. Аргументами перечисленных хинтов являются имена (псевдонимы) отношений, участвующих в соединении. `Use_nl` предписывает соединять отношения, перечисленные в качестве аргументов, при помощи алгоритма `nested loops`, используя эти отношения в качестве внутренних. Этот хинт эффективен в сочетании с хинтом `ordered`. `Use_merge` предписывает использовать для перечисленных отношений алгоритм `sort-merge`, а `use_hash` – `hash join`.

Подробнее об использовании хинтов и оптимизации приложений можно прочитать в [9].

## Литература

1. Кузнецов С. Д. Основы современных баз данных.  
[<http://www.citforum.ru/database/osbd/contents.shtml>]
2. Дейт К. Введение в системы баз данных. — К.; М.; СПб.: Издательский дом «Вильямс», 1999.
3. Грабер М. SQL92. Справочное руководство. — М.: ЛОРИ, 1998.
4. Дейт К. Руководство по реляционной СУБД DB2. — М.: Финансы и статистика, 1988.
5. Oracle 8i Documentation. *Oracle8i Concepts*
6. Oracle 8i Documentation. *Oracle8i National Language Support Guide*
7. Oracle 8i Documentation. *Oracle8i Reference*
8. Oracle 8i Documentation. *Oracle8i SQL Reference*
9. Oracle 8i Documentation. *Oracle8i Tuning*
10. Oracle 8i Documentation. *Oracle8i Application Developer's Guide – Fundamentals*
11. Oracle 8i Documentation. *PL/SQL User's Guide and Reference*
12. Oracle 8i Documentation. *Oracle8i Supplied Packages Reference*