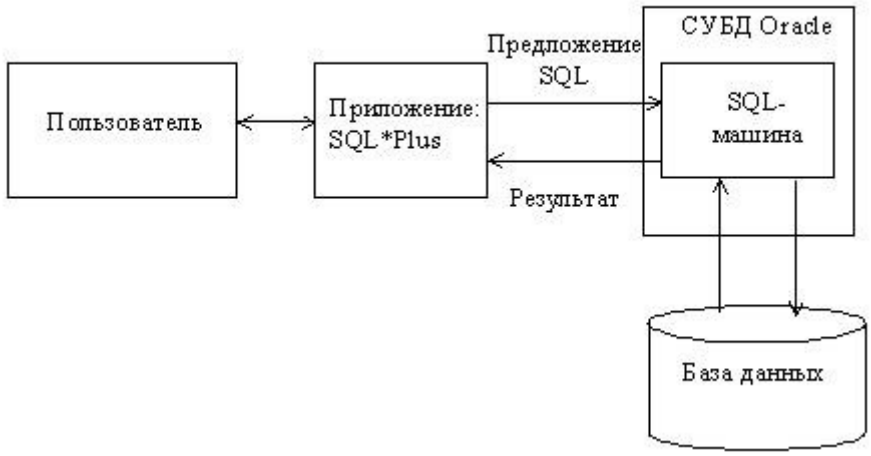
Инструмент для общения с базой данных

Фирма Oracle предоставляет два основных инструмента для общения с *БД* в диалоге посредством *SQL*: SQL*Plus и SQL Developer. Дальнейшие примеры в тексте, как правило, предполагаются для исполнения в SQL*Plus, однако с разной степенью корректировки исполняемы и в *SQL Developer*.

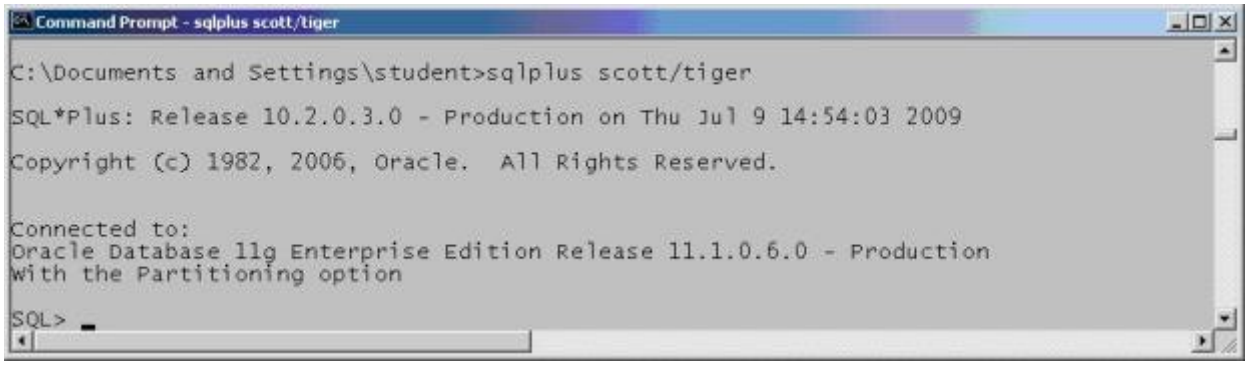
SQL*Plus — *программа* из обычного комплекта *ПО Oracle* для диалогового общения с *БД* путем ввода пользователем (или, возможно, из сценарного файла-"скрипта") текстов на *SQL* и *PL/SQL* и предъявления на экране компьютера результата, полученного от *СУБД*:



Запуск SQL*Plus может осуществляться:

- через меню ОС (в Windows);
- из командной строки (во всех ОС).

Пример запуска из командной строки:



SQL*Plus обрабатывает тексты на трех языках: *SQL*, *PL/SQL* и собственном. Во всех случаях *регистр* набора не имеет значения. Точнее, можно полагать, что, приняв на входе команду на любом из трех языков, "система" (выполняет ли эту работу SQL*Plus или *СУБД*, в данном случае неважно) повышает *регистр* всем буквам, кроме закавыченных символами """ и """, а потом уж производит обработку. Следующие две команды *SQL* содержательно равносильны:

```
SELECT * FROM emp;
select *   FRoM Emp ;
```

Исключение из правила автоматического повышения регистра перед обработкой команды касается значения пароля в версиях, начиная с 11. С этой версии выдача следующих двух команд *SQL* приведет к установке *разных* значений пароля пользователю SCOTT:

```
ALTER USER scott IDENTIFIED BY tiger;
ALTER USER scott IDENTIFIED BY Tiger;
```

Символ ";" используется в качестве признака окончания ввода команды *SQL* (они могут быть многострочными), но в некоторых случаях для этого используется / в *первой* позиции новой строки. Символ "-" используется как перенос продолжения набираемой команды SQL*Plus на новую строку (если эта *команда* чересчур длинна).

Собственные команды SQL*Plus служат для настроек работы этой программы, установления форматов и выполнения некоторых действий. Их несколько десятков, и полный перечень (в жизни избыточный) приведен в документации *по Oracle*.

Примеры:

- DESCRIBE — выдача на экран общего описания структуры таблиц, представлений данных, типов объектов или пакетов:
  ```
  DESCRIBE emp
  ```
- SET — установка (а SHOW — просмотр) режимов выдачи данных на экран, например установка длины внутреннего буфера формирования строк ответа и установка разбиения ответа на запрос на страницы:
  ```
  SET LINESIZE 200
  SET PAGESIZE 50>
  ```
- COLUMN — задание формата выдачи данных столбца на экран:
  ```
  COLUMN object_type FORMAT A20
  ```
- CONNECT/DISCONNECT — установление сеанса связи с СУБД, например:
  ```
  CONNECT scott/tiger
  ```

В отличие от *SQL* и *PL/SQL*, большинство ключевых слов в языке SQL*Plus имеют сокращенные формы, часто употребляемые в жизни и в литературе, например:

```
DESC emp
SET LINES 200
COL object_type FOR A20
CONN scott/tiger
```

Для обработки вводимых команд *SQL* (а заодно блоков на *PL/SQL*) в SQL*Plus применяется внутренний *буфер* команды. Он обновляется при каждом новом наборе текста на *SQL* (или блока на *PL/SQL*). *Команда* **SQL*Plus** LIST позволяет выдать на экран текущее содержимое буфера, *команда* RUN или же символ "/" — запустить содержимое на *исполнение*, а *команда* EDIT — редактировать.

SQL Developer тоже позволяет пользователю обращаться к *БД* на *SQL*, но имеет графический *интерфейс* и графические средства отладки. Для **SQL Developer** фирмой *Oracle* и третьими фирмами разработан ряд расширений: для администрирования

картографической информации в *БД* (*Oracle Spatial*), для графического администрирования средств анализа данных (*Oracle Data* Mining) и др.

С целью моделирования *БД* графическим образом можно использовать родственный продукт *Oracle* **SQL Developer Data Modeler**.

В качестве графической среды разработки и отладки запросов *SQL* и программ на *PL/SQL* имеется также несколько программных продуктов третьих фирм. Они появились раньше, чем **SQL Developer**, в общем обладают теми же возможностями, но часто более тщательно проработаны в деталях. Дальнейшие примеры можно отрабатывать и с их помощью.

Данные для дальнейших примеров

Таблицы

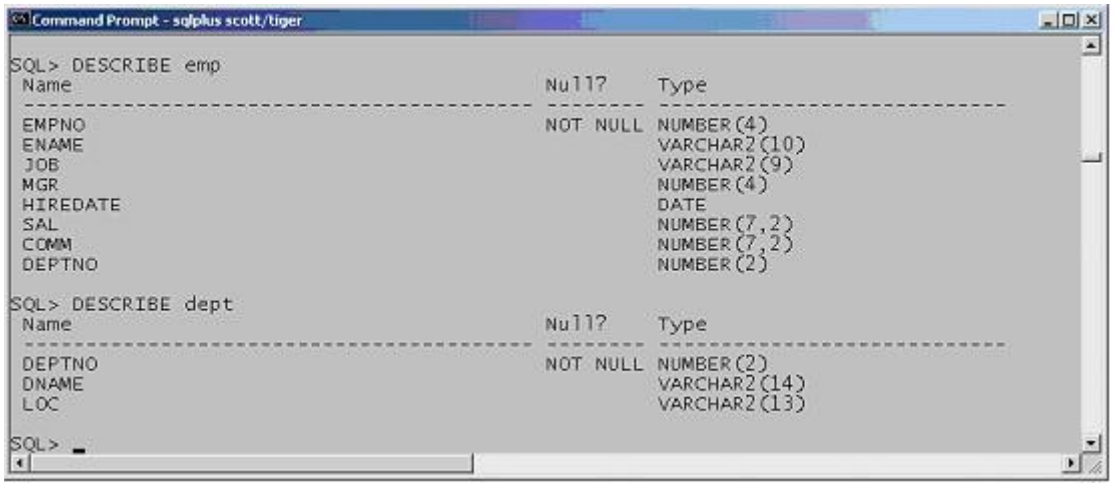
В дальнейших примерах чаще всего будут использоваться две таблицы, хранимые в "схеме данных" с именем SCOTT: это EMP и DEPT.

В Oracle и большинстве других подобных систем схемой данных называется специально оформляемое подмножество хранимых объектов, с которыми конкретные прикладные программы имеют право работать в данный момент. Понятие схемы данных задумано ради удобства работы с содержимым БД и защиты доступа. Объектами схемы в Oracle могут быть не только таблицы, но и прочие элементы, такие как индексы, процедуры и другие (в том числе так называемые "ограничения целостности"). Слово "схема" иногда употребляется и применительно к таблице. Под "схемой таблицы" понимают ее структуру вместе с некоторыми элементами описания.

В стандарте SQL множество рабочих объектов для программы устанавливается несколько сложнее, но тоже с участием понятия "схема". В реляционной модели это понятие не разработано, однако это сделано в общей теории баз данных.

Схема с именем SCOTT поставляется в Oracle с давних времен для демонстрационных целей.

Общее описание таблиц:



Данные в таблицах:

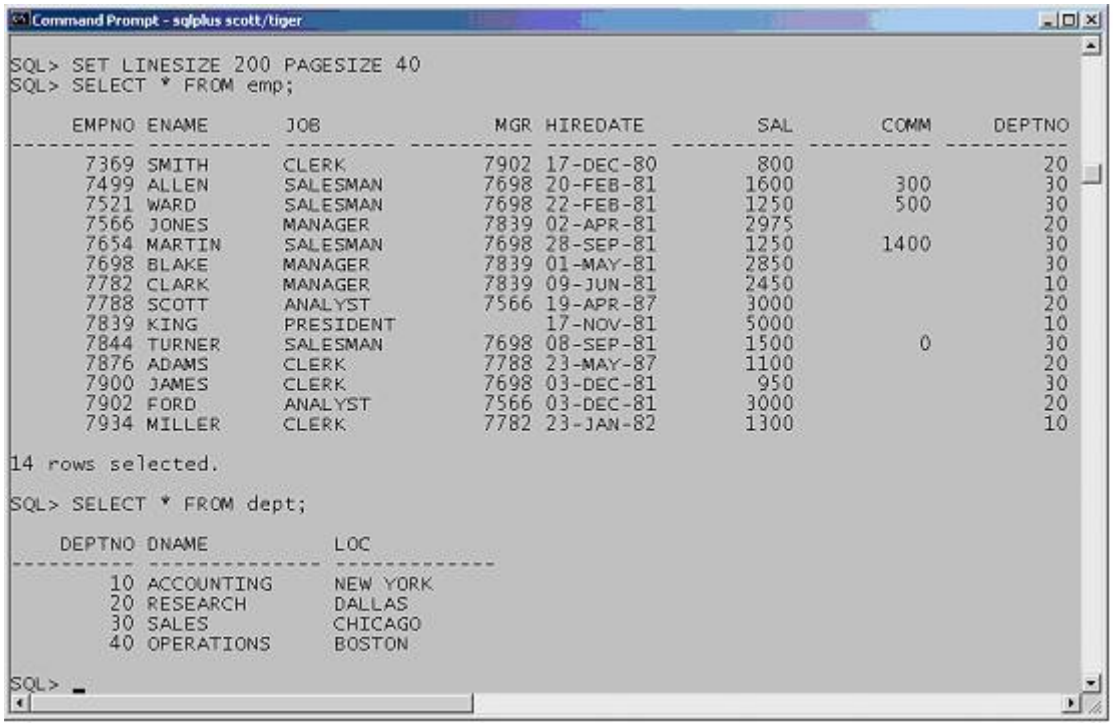
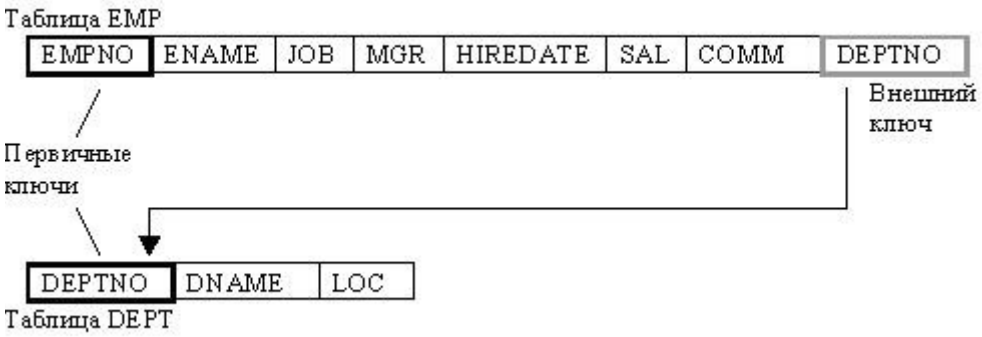


Таблица EMP имеет (первичный) ключ: столбец EMPNO; ключ в таблице DEPT — столбец DEPTNO. В таблице EMP столбец DEPTNO образует "внешний ключ", ссылающийся на значения из одноименного столбца таблицы DEPT:



Совпадение имен столбцов внешнего ключа и столбцов адресатов в SQL не обязательно, но в жизни приветствуется.

Таблица DUAL не принадлежит схеме SCOTT, а является "системной" (принадлежит схеме SYS) и доступна для выборки любому пользователю. Состоит из единственной строки и единственного столбца:

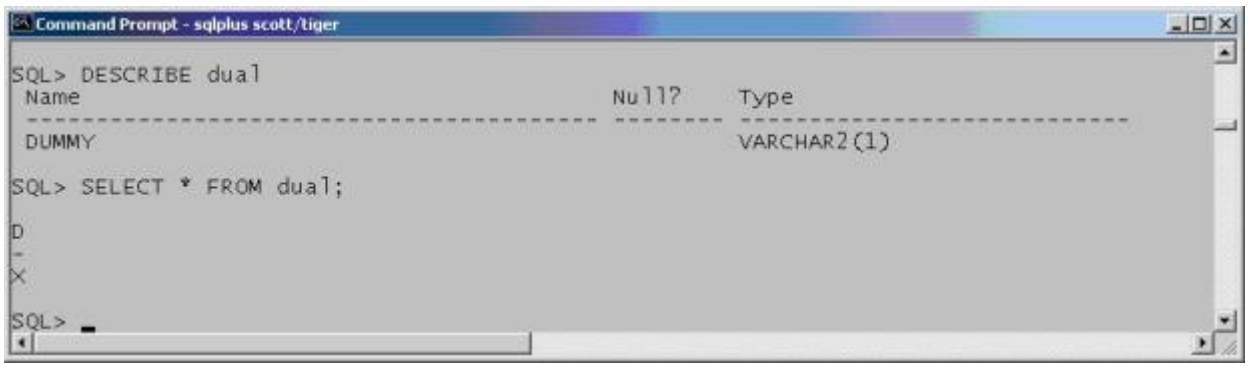


Таблица DUAL предназначена играть техническую роль и с этой целью не раз используется в примерах ниже.

Такой характер ее употребления сложился не сразу. Когда она создавалась в старые времена, то имела две, а не одну, как нынче, строки (отсюда название DUAL) и служила для "удвоения" строк системной таблицы выполнением соединения (join) в запросе о распределении памяти на диске в табличном пространстве БД. Впоследствии эта ее задача оказалась неактуальной, и она приобрела нынешнее значение.

В схеме SCOTT присутствует еще пара таблиц, BONUS и SALGRADE, гораздо менее интересных и далее почти не востребованых.

Упражнение. Выясните в SQL*Plus или **SQL Developer** структуру и содержимое таблиц BONUS и SALGRADE в схеме SCOTT.

Со временем фирма Oracle разработала более правдоподобно и сложно устроенный демонстрационный пример, нежели схема SCOTT, на этот раз из нескольких схем: HR, OE, PM, SH, IX, BI. С ним часто можно столкнуться в нынешних примерах из разных источников. Далее в тексте предпочтение отдано все-таки схеме SCOTT: она небольшого размера, и ее обычно достаточно для решаемых задач.

Пользователи и полномочия

В отличие от схемы, понятие пользователя в базах данных служит для моделирования возможностей прикладной программы совершать те или иные действия в БД. Что касается действий с объектами БД, то специального разрешения на такие действия Oracle не требует в двух случаях:

- когда объект доступа принадлежит самому пользователю,
- когда действие с объектом объявлено "публичным".

В остальных случаях для выполнения действия пользователь должен иметь специально выданное ему полномочие ("привилегию", по терминологии Oracle). Примерами таких полномочий могут служить полномочие создавать таблицу (в собственной схеме!), полномочие обращаться запросом SELECT к чужой таблице.

В БД Oracle самостоятельным является понятие пользователя, а понятие схемы подчинено ему. Так, по команде CREATE USER СУБД создаст в БД *пользователя* и одновременно автоматически — *схему* с тем же именем, к которой будет относить все объекты "этого

пользователя", и только их. Схема будет удалена из БД также автоматически при удалении пользователя.

Большинство объектов БД, например, таблицы, индексы, процедуры вида SYNONYM, обязаны находиться в одной и ровно одной схеме ("принадлежать одному пользователю"). Однако небольшая часть объектов, например, вида PUBLIC SYNONYM или DIRECTORY, хранятся в БД вне всякой схемы и "не принадлежат никакому конкретному пользователю".

Демонстрационный пользователь SCOTT изначально обладает некоторым набором полномочий, например, создания таблиц и подключения к СУБД. В то же время ему изначально не приданы, скажем, полномочия создавать представления данных или же синонимы. Придать ему эти полномочия способны, например, пользователи SYS и SYSTEM. Эти пользователи могут быть названы "администраторами"; они имеются в любой БД Oracle, и при этом SYS может быть неформально назван "суперпользователем" или "главным администратором", а SYSTEM несколько ограничен в правах по сравнению с SYS.

Полномочия (= привилегии) выдаются командой GRANT и изымаются командой REVOKE.

Пример. Подсоединение в SQL*Plus в качестве SYS, простое создание пользователя YARD вместе со своей схемой и с паролем pass, наделение пользователя YARD некоторыми общими привилегиями:

```
CONNECT / AS SYSDBA
CREATE USER yard IDENTIFIED BY pass;
GRANT connect, resource, create view TO yard;
Подключение к СУБД под именем YARD и попытка обратиться к таблице EMP:
SQL> CONNECT yard/pass
Connected.
SQL> DESCRIBE emp
ERROR:
ORA-04043: object emp does not exist
```

Ошибка возникла потому, что пользователь YARD (как и любой другой) работает в собственной схеме БД, а в схеме YARD таблицы EMP нет — она имеется в схеме SCOTT. Разрешить же пользователю YARD обращаться к данным таблицы EMP в схеме SCOTT можно специально выданной командой GRANT, например:

```
CONNECT scott/tiger
GRANT SELECT ON emp TO yard;
```

В Oracle понятие "пользователь" естественно связывать не с физическим лицом, обращающимся к БД, а с приложением.

Создание, удаление и изменение структуры таблиц

Таблицы представляют собой главный инструмент моделирования данных в *БД*, как построенных на основе *SQL* вообще, так и в *Oracle* в частности.

Предложение CREATE TABLE

Создание таблиц осуществляется предложением CREATE TABLE категории *DDL*.

Пример:

```
CREATE TABLE proj
(
   projno NUMBER   ( 4 )
 , pname  VARCHAR2 ( 14 )
 , bdate  DATE
 , budget NUMBER   ( 10, 2 )
);
```

Помимо указанных в этом примере предложение CREATE TABLE может включать чрезвычайно много других конструкций, большая часть которых связана с организацией хранения, а не с логикой данных. Полностью они приводятся в документации Oracle по SQL.

Типы данных в столбцах

Существующие в Oracle встроенные (предопределенные) типы позволяют указывать столбцам таблицы следующие виды данных:

- Числа:
  - типы NUMBER, NUMBER (n), NUMBER (n, m) (в частности, NUMBER (*, m))
  - типы FLOAT, REAL, NUMERIC, DECIMAL, INTEGER и другие совместимые с ANSI
  - типы BINARY_FLOAT(10.1-), BINARY_DOUBLE(10.1-)
- Строки текста:
  - типы VARCHAR2 (n), CHAR (n)
  - типы NVARCHAR2 (n), NCHAR (n)(9.0-)
  - типы *CLOB*(8-), NCLOB(8-)
  - типы STRING, *CHARACTER VARYING*, *NATIONAL CHARACTER* VARYING и другие
- Строки байтов:
  - RAW (n)
  - BLOB(8-)
  - LONG, LONG RAW
- Моменты времени:
  - DATE
  - TIMESTAMP(9.2-), TIMESTAMP (n)(9.2-)
  - TIMESTAMP WITH *TIME ZONE*(9.2-), TIMESTAMP WITH *LOCAL TIME* ZONE(9.2-)
- Интервалы времени:
  - INTERVAL YEAR TO MONTH(9.2-), INTERVAL DAY TO SECOND(9.2-)
- Прочие типы данных:
  - BFILE(8-)
  - ROWID, UROWID (физический и "логический физический" адреса строк или объектов в таблицах)

- XMLTYPE(9.2-), ANYDATA(9.2-), URITYPE с подтипами(9.2-); типы для Oracle *Spatial*(8-), для построения геоинформационных систем; прочие (встроенные объектные)

Помимо встроенных типов Oracle с версии 8 позволяет использовать в описании столбцов типы, создаваемые самостоятельно пользователями БД. Это объектные ("структурные") типы.

Типы BLOB, *CLOB*, NCLOB и BFILE иногда объединяют в единую категорию типов "больших неструктурированных объектов" (объекты LOB). Для работы с ними в SQL иногда требуется прибегать к функциям встроенного пакета DBMS_LOB, хотя с версии 9.2 поводов для этого стало меньше. Их употребление в SQL связано с определенными ограничениями. Например, столбцы этих типов не могут употребляться в формировании ключа таблицы и вообще индексироваться стандартным образом. Часть подобных ограничений употребления обязаны предположительно гигантским объемам значений, а часть — особенному способу хранения, отличному от принятого для "обычных" данных.

В коде СУБД Oracle сохранились попытки ввести в некоторых случаях более разумные встроенные типы, по ряду причин не доведенные до официального предоставления. Примером может служить тип TIME для обозначения времени суток. Разрешить его употребление в своем сеансе связи с СУБД можно следующей командой (http://citforum.ru/database/oracle/time/):

```
ALTER SESSION SET EVENTS '10407 trace name context forever, level 1';
```

Официально этого типа не существует, а следами его фактического наличия являются возможности указывать в обычных выражениях SQL значение времени суток, например, TIME '12:30:45', и некоторые функции, например, TO_TIME и EXTRACT.

Похожим путем в версии 8.1 можно активировать типы TIMESTAMP и INTERVAL (впоследствии ставшие штатными, в отличие от TIME):

```
ALTER SESSION SET EVENTS '10406 trace name context forever';
```

Типы данных ORACLE

Следующие секции описывают типы данных ORACLE, которые могут использоваться в определениях столбцов.

Символьные типы данных

Типы данных CHAR и VARCHAR2 хранят алфавитно-цифровые данные; в столбце одного из этих типов данных можно хранить любые символы. Символьные данные хранятся как строки символов, где байтовые значения соответствуют схеме кодирования символов (обычно называемой набором символов или кодовой страницей); набор символов базы данных устанавливается при создании базы данных и никогда не изменяется. Примерами наборов символов служат 7-битовый ASCII (американский стандартный код для обмена информацией), кодовая страница 500 набора символов EBCDIC (расширенный двоично-кодированный десятичный код обмена) или Japan Extended UNIX. ORACLE поддерживает как однобайтовые, так и мультибайтовые схемы кодирования. Обратитесь к приложению E для дополнительной информации о средствах

поддержки национальных языков (NLS) в ORACLE и о том, как поддерживаются различные схемы кодирования символов.

Так как ORACLE дополняет пробелами значения в столбцах CHAR и не дополняет пробелами значения в столбцах VARCHAR2, столбцы VARCHAR2 более экономны в смысле затрат памяти на хранение. По этой причине, при полном просмотре большой таблицы, содержащей столбцы VARCHAR2, необходимо прочитать меньше блоков, чем для аналогичной таблицы со столбцами CHAR. Если ваше приложение часто выполняет полные просмотры больших таблиц, содержащих символьные данные, вы можете улучшить производительность, выбрав для таких данных тип данных VARCHAR2 вместо CHAR.

Однако производительность - не единственный фактор, который необходимо рассматривать при выборе между этими типами данных. ORACLE применяет различные семантики при сравнении значений для каждого из этих типов данных. Вы можете предпочесть один тип другому, если ваше приложение чувствительно к различиям между этими семантиками. Например, если вы хотите, чтобы ORACLE игнорировал хвостовые пробелы при сравнении символьных значений, то вы должны хранить такие значения в столбцах CHAR. Для информации о семантиках сравнения для символьных типов данных обратитесь к документу ORACLE7 Server SQL Language Reference Manual.

Тип данных CHAR

Тип данных CHAR хранит строки ФИКСИРОВАННОЙ длины. При создании таблицы со столбцом CHAR для этого столбца задается длина (в байтах, а не в символах) от 1 до 255 (по умолчанию 1). Затем ORACLE гарантирует соблюдение следующих правил:

- Если входное значение короче, оно дополняется пробелами до фиксированной длины.
- Если входное значение слишком длинно, ORACLE возвращает ошибку.

ORACLE сравнивает значения CHAR, используя ДОПОЛНЯЮЩУЮ СЕМАНТИКУ сравнения. Если сравниваемые значения имеют разную длину, то ORACLE дополняет более короткое значение пробелами до равной длины. Если два значения отличаются лишь числом хвостовых пробелов, то они считаются равными.

Тип данных VARCHAR2

Тип данных VARCHAR2 хранит символьные строки ПЕРЕМЕННОЙ длины. При создании таблицы со столбцом VARCHAR2 для этого столбца задается максимальная длина (в байтах, а не в символах) от 1 до 2000. Для каждой строки, значение столбца VARCHAR2 записывается как поле переменной длины (если входное значение превышает максимальную длину столбца, ORACLE возвращает ошибку). Например, предположим, что столбец объявлен с типом VARCHAR2 и максимальной длиной 50 символов. Если входное значение для этого столбца имеет длину 10 символов, то (в однобайтовом наборе символов) значение столбца в строке будет иметь длину 10 символов (10 байт), а не 50.

ORACLE сравнивает значения VARCHAR2, используя НЕДОПОЛНЯЮЩУЮ СЕМАНТИКУ сравнения. Два значения считаются равными лишь тогда, когда они состоят из одних и тех же символов и имеют одинаковую длину.

Тип данных VARCHAR

Тип данных VARCHAR в настоящее время является синонимом типа данных VARCHAR2. Однако в будущей версии ORACLE тип данных VARCHAR будет хранить строки символов переменной длины с иной семантикой сравнения. Поэтому используйте тип данных VARCHAR2 для символьных строк переменной длины.

Длины столбцов для символьных типов данных

Длины для столбцов CHAR и VARCHAR2 специфицируются в байтах, а не в символах, и ограничения на их длины рассматриваются в байтах. Если набор символов базы данных использует однобайтовую схему кодирования символов, то число символов в столбец совпадает с числом байтов. В мультибайтовой схеме кодирования такого соответствия в общем случае нет. Например, некоторые символы могут занимать один байт, другие - два байта, и т.д. Это соображение должно рассматриваться при оценке памяти для таблиц, столбцы которых содержат символьные данные.

Тип данных NUMBER

Тип данных NUMBER используется для хранения нуля и положительных или отрицательных чисел с фиксированной и плавающей точкой. Для этого типа данных гарантируется переносимость между любыми операционными системами, которые поддерживает ORACLE, с точностью до 38 цифр. Вы можете хранить положительные и отрицательные числа в интервале от 1 x 10**-130 до 9.99..9 x 10**125 (с точностью до 38 значащих цифр), а также ноль.

Для числовых столбцов можно просто указать NUMBER, например:

имя_столбца NUMBER

или можно указать ТОЧНОСТЬ (общее число цифр) и МАСШТАБ (число цифр справа от десятичной точки):

имя_столбца NUMBER (точность, масштаб)

Если точность не указана, столбец хранит значения так, как они задаются. Если не указан масштаб, он считается нулевым. Масштаб может принимать значения от -84 до 127.

Хотя это и не обязательно, при задании числовых полей рекомендуется явно указывать точность и масштаб; это обеспечивает возможность дополнительной проверки данных на входе. На табл.3-1 приведены примеры хранения данных при использовании различных показателей масштаба.

Табл.3-1

Влияние показателя масштаба на хранение числовых данных

Входные данные	Тип столбца	Хранится как
7,456,123.89	NUMBER(*,1)	7456123.9
7,456,123.89	NUMBER(9)	7456123
7,456,123.89	NUMBER(9,2)	7456123.89
7,456,123.89	NUMBER(9,1)	7456123.9
7,456,123.89	NUMBER(6)	ошибка: превышена точность
7,456,123.89	NUMBER(7,-2)	7456100

Тип данных DATE

Тип данных DATE хранит значения в виде точек времени (т.е. дату и время). Тип данных DATE запоминает год (включая век), месяц, день, часы, минуты и секунды. ORACLE может хранить даты в диапазоне от 1 января 4712 года до н.э. до 31 декабря 4712 года нашей эры. Если в маске формата не указано BC (до н.э.), предполагается по умолчанию наша эра (AD).

ORACLE использует для хранения дат собственный внутренний формат. Данные дат хранятся в фиксированных полях длиной семь байт, соответствующих веку, году, месяцу, дню, часу, минуте и секунде.

Стандартный формат даты ORACLE для ввода и вывода имеет вид DD-MON-YY, например:

'13-NOV-92'

Этот умалчиваемый формат даты можно изменить для инстанции с помощью параметра NLS_DATE_FORMAT. Его можно также изменить на время сессии пользователя с помощью предложения ALTER SESSION. Для ввода дат в формате, отличном от текущего умалчиваемого формата даты, используйте функцию TO_DATE с маской формата, например:

TO_DATE ('November 13, 1992', 'Month DD, YYYY')

Если используется стандартный формат DD-MON-YY, то YY указывает год в 20-м веке (например, 31-DEC-92 означает 31 декабря 1992 г.). Если вы хотите задавать годы в другом веке, используйте другую маску формата, как показано выше.

Время хранится в 24-часовом формате HH:MM:SS. Если не введено значение времени, по умолчанию предполагается полночь (12:00:00 A.M.). Если вводится только порция, содержащая время, то за дату принимается первый день текущего месяца. Для того, чтобы ввести в дату время, необходимо это указать в маске формата функции TO_DATE, например:

```
INSERT INTO birthdays (bname, bday) VALUES
  ('ANNIE',TO_DATE('13-NOV-92 10:56 AM','DD-MON-YY HH:MI AM'));
```

Для сравнения дат, содержащих время, используйте функцию SQL TRUNC, если вы хотите проигнорировать компоненту времени. Используйте функцию SQL SYSDATE, чтобы получить текущие системные дату и время. С помощью параметра FIXED_DATE можно установить SYSDATE как константу; это может быть полезно при отладке.

Использование юлианских дат

Юлианские даты позволяют датировать события от общей точки. (Эта точка принимается за 01-01-4712 г. до н.э., так что сегодняшние даты лежат где-то в пределах 2.4 миллиона дней.) Юлианская дата по определению нецелая, ее дробная часть составляет часть дня. ORACLE использует упрощенный подход, в котором используются целые числа. Юлианские даты могут вычисляться и интерпретироваться по-разному; метод, используемый в ORACLE, представляет дату в виде семизначного числа (для наиболее

часто используемых дат); например, 8 апреля 1991 года будет представлено в виде 2448355.

Для преобразования дат в даты юлианского календаря в функциях преобразования даты (TO_DATE, TO_CHAR) может быть использована маска формата "J". Например следующий запрос возвращает все даты в юлианском формате:

SELECT TO_CHAR (hiredate, 'J') FROM emp;

Чтобы использовать юлианские даты в вычислениях, необходимо использовать также функцию TO_NUMBER. Для ввода юлианских дат можно использовать функцию TO_DATE:

INSERT INTO emp (hiredate) VALUES (TO_DATE(2448921, 'J'));

Арифметика дат

Арифметика дат ORACLE учитывает аномалии исторически применяемых календарей. Например, при переходе с юлианского календаря на грегорианский календарь, 15 октября 1582 года, были потеряны предыдущие 10 дней (с 05 по 14 октября). Кроме того, год 0 не существует.

Пропущенные даты могут быть введены в базу данных, но они игнорируются в арифметике дат и рассматриваются как следующая "реальная" дата. Например, следующим днем за 04 октября 1582 будет 15 октября 1582, а следующим днем за 05 октября 1582 будет 16 октября 1582.

Замечание: Это обсуждение арифметики дат применимо не ко всем национальным стандартам дат (например, некоторым в Азии).

Тип данных LONG

Столбец, описанный как LONG, может содержать символьную строку переменной длины до двух гигабайт. Столбцы типа LONG имеют многие характеристики столбцов VARCHAR2. Длина значений LONG может лимитироваться основной памятью, доступной на вашем компьютере.

Использование данных LONG

Тип данных LONG используется в словаре данных для хранения текста определений обзоров. Вы можете использовать столбцы, определенные как LONG, в списках SELECT, фразах SET предложений UPDATE и фразах VALUES предложений INSERT.

Ограничения на данные типа LONG и LONG RAW

Хотя столбцы типа LONG (и LONG RAW; см. ниже) находят много различных применений, на их использование накладываются некоторые ограничения:

- Только один столбец типа LONG допускается в таблице.
- Столбцы LONG нельзя индексировать.
- Столбцы LONG нельзя использовать в ограничениях целостности.

- Столбцы LONG нельзя использовать в фразах WHERE, GROUP BY, ORDER BY, CONNECT BY, а также с оператором DISTINCT в предложениях SELECT.
- Столбцы LONG нельзя использовать в функциях SQL (таких как SUBSTR или INSTR).
- Столбцы LONG нельзя использовать в списке SELECT подзапроса или запросов, объединяемых операторами множеств (UNION, UNION ALL, INTERSECT или MINUS).
- Столбцы LONG нельзя использовать в выражениях.
- Нельзя ссылаться на столбцы LONG при создании таблицы с помощью запроса (CREATE TABLE ... AS SELECT ...) или при вставке в таблицу (обзор) через запрос (INSERT INTO ... SELECT ...).
- Переменная или аргумент программной единицы PL/SQL не могут объявляться с типом данных LONG.

Проектируя таблицы, содержащие данные LONG или LONG RAW, помещайте каждый столбец LONG или LONG RAW в особую таблицу, отдельно от любых других связанным с ними данных, вместо того чтобы хранить столбец LONG или LONG RAW в общей таблице с другими данными. После этого вы можете связать обе таблицы ограничением ссылочной целостности. Такой проект позволит предложениям SQL, использующим лишь другие ассоциированные данные, избежать сканирования по данным LONG или LONG RAW.

Пример

Чтобы сохранить информацию о журнальных статьях, включая текст каждой статьи, создайте две таблицы:

```
CREATE TABLE article_header (id          NUMBER
                             PRIMARY KEY,
                        title       VARCHAR2(200),
                        first_author VARCHAR2(30),
                        journal     VARCHAR2(50),
                        pub_date    DATE)
CREATE TABLE article_text   (id          NUMBER
                             REFERENCES
                             article_header,
                        text        LONG)
```

Таблица ARTICLE_TEXT хранит только текст каждой статьи. Таблица ARTICLE_HEADER хранит всю остальную информацию о статье, включая ее заголовок, первого автора, название журнала и дату публикации. Обе таблицы связаны ограничением ссылочной целостности по столбцу ID в каждой таблице.

Этот проект позволяет предложениям SQL опрашивать данные, отличные от текстов статей, не читая самих текстов. Например, если вы хотите выбрать всех первых авторов, публиковавшихся в журнале "Nature" в течение июля месяца 1991 года, вы можете выдать следующий запрос по таблице ARTICLE_HEADER:

```
SELECT first_author
  FROM article_header
  WHERE journal = 'NATURE'
     AND TO_CHAR(pub_date,'MM YYYY') = '07 1991'
```

Если бы текст каждой статьи хранился в одной таблице с ее автором, названием журнала и датой публикации, ORACLE пришлось бы просматривать все эти тексты при выполнении данного запроса.

Типы данных RAW и LONG RAW

Типы данных RAW и LONG RAW используются для данных, которые не должны ни интерпретироваться ORACLE, ни преобразовываться при передаче данных между различными системами. Эти типы данных предназначены для двоичных данных или байтовых строк. Например, LONG RAW можно использовать для хранения графики, звука, документов или массивов двоичных данных; их интерпретация зависит от их использования.

RAW эквивалентен VARCHAR2, а LONG RAW эквивалентен LONG, с тем исключением, что SQL*Net (который соединяет пользовательские сессии с инстанцией) и утилиты экспорта и импорта не выполняют преобразований при передаче данных RAW или LONG RAW. Напротив, SQL*Net и импорт/экспорт автоматически конвертируют данные CHAR, VARCHAR2 и LONG между набором символов базы данных и набором символов сессии пользователя (установленным параметром NLS_LANGUAGE или командой ALTER SESSION), если эти наборы символов различны.

Когда ORACLE автоматически преобразует данные RAW или LONG RAW в тип данных CHAR или из него (как в случае, когда данные RAW вводятся как литерал в предложении INSERT), эти данные рассматриваются как шестнадцатеричные цифры, каждая из которых представляет полубайт (четыре бита). Например, один байт данных RAW с битовым представлением 11001011 вводится и отображается как 'CB'.

Данные LONG RAW не могут индексироваться, однако данные RAW можно индексировать.

ROWID'ы и тип данных ROWID

Каждой строке некластеризованной таблицы в базе данных ORACLE назначается уникальный ROWID, соответствующий физическому адресу данной строки (начального куска строки, если строка хранится как несколько кусков, связанных в цепочку). В случае кластеризованных таблиц, строки разных таблиц, если они хранятся в одном и том же блоке данных, могут иметь одинаковый ROWID.

Каждая таблица в базе данных ORACLE внутренне имеет ПСЕВДОСТОЛБЕЦ с именем ROWID; этот псевдостолбец не виден при выдаче структуры таблицы с помощью предложения SELECT * FROM ... или предложения DESCRIBE в SQL*Plus. Однако адрес каждой строки можно извлечь запросом SQL, используя ключевое слово ROWID как имя столбца, например:

SELECT ROWID, ename FROM emp;

ROWID'ы используют двоичное представление физического адреса для каждой выбираемой строки. При запросах из SQL*Plus или SQL*DBA это двоичное представление преобразуется в шестнадцатеричное представление VARCHAR2, и запрос, показанный выше, мог бы возвратить следующую информацию строк:

ROWID ENAME

------------------ ----------
00000DD5.0000.0001 SMITH
00000DD5.0001.0001 ALLEN
00000DD5.0002.0001 WARD

Как показано выше, VARCHAR2/шестнадцатеричное представление ROWID разделяется на три компоненты: блок.строка.файл.

- БЛОК ДАННЫХ, содержащий строку (блок DD5 в примере). Номера блоков относятся к их файлу данных, а НЕ к табличному пространству. Поэтому в двух разных файлах одного и того же табличного пространства могут храниться строки с одинаковыми номерами блоков.
- СТРОКА в блоке, содержащем строку (строки 0, 1 и 2 в примере). Номера строк в данном блоке всегда начинаются с 0.
- ФАЙЛ ДАННЫХ, содержащий строку (файл 1 в примере). Первый файл данных в каждой базе данных всегда имеет номер 1, и номера файлов уникальны внутри базы данных.

ROWID, назначенный строке, остается неизменным до тех пор, пока строка не будет экспортирована и вновь импортирована (с помощью утилит IMPORT и EXPORT). Когда строка удаляется (и соответствующая транзакция подтверждена), ROWID, ассоциированный с удаленной строкой, может быть назначен строке, вставляемой в последующей транзакции.

Нельзя устанавливать значение псевдостолбца ROWID в предложениях INSERT или UPDATE. Значения ROWID в псевдостолбце ROWID внутренне используются ORACLE в разнообразных операциях (см. следующую секцию). Хотя к значениям псевдостолбца ROWID можно обращаться как к другим столбцам таблицы (в списках SELECT и фразах WHERE), эти значения не хранятся в базе данных и не являются данными базы данных.

ROWID'ы и базы данных не-ORACLE

Приложения базы данных ORACLE можно выполнять на серверах баз данных, отличных от ORACLE, используя SQL*Connect или Oracle Open Gateway. В таких случаях двоичный формат значений ROWID изменяется в соответствии с характеристиками системы не-ORACLE. Более того, стандартная трансляция значений ROWID в формат VARCHAR2/шестнадцатеричный недоступна. Программы могут по-прежнему использовать тип данных ROWID; однако они должны применять нестандартную трансляцию в шестнадцатеричный формат, используя до 256 байт. Обратитесь к соответствующему

руководству по OCI или прекомпилятору за дополнительными подробностями об использовании значений ROWID в сочетании с системами, отличными от ORACLE.

Как используются ROWID'ы

ROWID'ы внутренне используются ORACLE в конструкциях индексов. Каждый ключ в индексе ассоциируется с ROWID'ом, указывающим на адрес соответствующей строки, для быстрого доступа.

Некоторые характеристики ROWID'ов могут также использоваться разработчиками приложений: * ROWID'ы дают самый быстрый доступ к конкретным строкам.

- ROWID'ы позволяют увидеть, как организована таблица.
- ROWID'ы уникально идентифицируют строки в таблице.

Прежде чем использовать ROWID'ы в предложениях DML, они должны быть проверены и гарантированы от изменений; иными словами, необходимые строки должны быть заблокированы, чтобы их нельзя было удалить. Попытка обращения к данным с некорректным значением ROWID приведет либо к тому, что строка не будет возвращена, либо к ошибке 1410 (неверный ROWID).

Вы можете также создавать таблицы со столбцами, определенными с типом данных ROWID; например, вы определяете таблицу исключений со столбцом типа данных ROWID, чтобы запоминать ROWID'ы тех строк базы данных, которые нарушают ограничения целостности. Столбцы, определенные с типом данных ROWID, ведут себя как обычные столбцы; их значения можно обновлять, и т.п. Все значения в столбце типа данных ROWID занимают шесть байт.

Примеры использования значений ROWID

Используя ROWID с некоторыми групповыми функциями, вы можете увидеть, как данные внутренне хранятся в базе данных ORACLE.

Функцию SUBSTR можно использовать, чтобы разбить значение ROWID на три его компоненты (файл, блок и строку). Например, запрос

```
SELECT ROWID, SUBSTR(ROWID,15,4) "FILE",
    SUBSTR(ROWID,1,8) "BLOCK",
    SUBSTR(ROWID,10,4) "ROW"
    FROM emp;
```

мог бы возвратить следующие данные:

```
ROWID              FILE BLOCK    ROW
------------------ ---- -------- ---
00000DD5.0000.0001 0001 00000DD5 0000
00000DD5.0001.0001 0001 00000DD5 0001
00000DD5.0002.0001 0001 00000DD5 0002
```

ROWID'ы могут быть полезны для получения информации о физическом хранении данных таблицы. Например, если вы хотите узнать о физическом размещении строк таблицы (скажем, принимая решение о разрезании таблицы), то следующий запрос сообщит вам, сколько файлов данных содержат строки заданной таблицы:

```
SELECT COUNT(DISTINCT(SUBSTR(ROWID,15,4))) "FILES" FROM таблица;
```

результаты этого запроса могут быть следующими:

```
FILES
-------
2
```

Типы BINARY_FLOAT и BINARY_DOUBLE

Эти два типа представляют собой второй после NUMBER используемый в Oracle формат хранения чисел, определяемый стандартом *IEEE 754* для 32- и 64- разрядного внутреннего представления. Стандарт *IEEE 754* реализован аппаратно во многих видах процессоров и программно в ряде языков программирования (Java).

Стандарт *IEEE 754* определяет больше, чем просто формат хранения чисел. Он, например, предусматривает особые значения "не число" (Not a Number) и +/? бесконечность. В Oracle точно воспроизведен формат хранения, но не все прочие подробности — стандарта *IEEE 754*. Возможность сослаться на особые "значения" дают следующие обозначения:

BINARY_FLOAT_NAN
BINARY_FLOAT_INFINITY
BINARY_DOUBLE_NAN
BINARY_DOUBLE_INFINITY

Например:

```
SQL> SELECT -BINARY_FLOAT_INFINITY FROM dual;
-BINARY_FLOAT_INFINITY
----------------------
                 -Inf
```

Из-за несовместимости форматов простая передача данных из вида NUMBER в BINARY_FLOAT/DOUBLE и обратно может приводить к потере точности. Зато при общении СУБД посредством этих двух форматов с внешними средами, работающими на основе стандарта *IEEE 754*, точность, наоборот, будет сохраняться.

Общие свойства типов

У типов данных в Oracle имеются общие свойства:

- у всех типов дополнительно ко множеству допустимых величин наличествует особый символ NULL;
- большинство типов допускает сравнение на равенство.

Пропущенные значения и NULL

NULL можно воспринимать формально (и механически следовать правилам выполнения операций с NULL), но пытаться интерпретировать эти обозначения в столбце таблицы можно по-разному: как "значение отсутствует" (например, "сотрудник не получал комиссионных") и как "неизвестно какое" из допустимых "значение" (например, "сотрудник получил какие-то комиссионные, неизвестные БД"). К сожалению это не одно и то же, равно как и наделения NULL этими двумя смыслами в жизни недостаточно (хотя и хватает для описания 99% возникающих ситуаций). По этим причинам, хотя на первый взгляд возможность опустить значение в столбце за его отсутствием может и показаться привлекательной, последующая работа с такими данными способна принести программисту значительно больше неприятностей из-за необходимости учета при составлении запросов неформализованных в схеме сведений о данных, усложнения запросов и возникающих рисков ошибиться при программировании. Примеры подобных неприятностей обозначены в разных местах текста ниже.

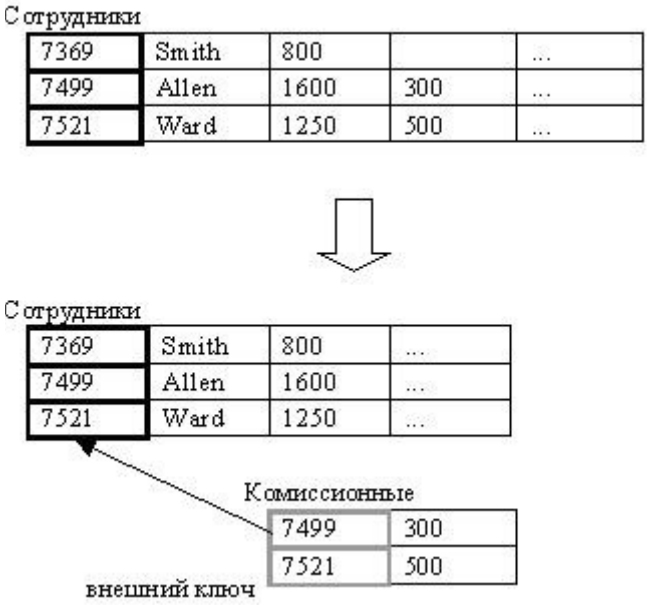
Стандарт SQL допускает NULL, но оговаривает, что во имя надежности обращения к данным программисту следует всячески избегать пропусков значений в столбцах либо уж употреблять их лишь в смысле "значение неизвестно" (unknown). Стандарт называет NULL особым *значением*, общим для всех типов, и обрекает себя тем самым на критику со стороны специалистов, полагающих, что правильнее назвать NULL символом[1], так как он не удовлетворяет признакам значения. Действительно, если x "имеет значение" NULL, то x = x не истина, что довольно необычно. Это уже проблема не интерпретации, а принятых правил употребления.

Oracle наследует все проблемы с NULL стандартного SQL, но, как показывает опыт, рассматривает NULL принадлежностью каждого типа в отдельности, что иногда дает о себе знать в попытках сформулировать некорректное с точки зрения Oracle выражение.

Что касается реляционной теории, то первые 10 лет своего существования она не рассматривала пропущенных значений вовсе и не имела дела с NULL, после чего мнения специалистов разошлись: одни (в их числе Э. Кодд) посчитали возможным (хотя и нежелательным на деле) использование пропущенных значений, а другие настаивали на их недопущении. То есть разрешение значениям в базе отсутствовать, вместе с вытекающей из этого необходимостью громоздкого аппарата их учета, можно считать одним из предлагаемых расширений, притом спорным, реляционной модели.

Крайнюю позицию занимает К. Дейт. По его мнению, пропущенным значениям в реляционной модели не место, а все задачи, которые предлагают решать с их помощью, можно решить, не прибегая к ним.

Например, если требуется учесть возможность отсутствия комиссионных у работника, атрибут "комиссионные" можно исключить из отношения "сотрудники" и перенести в дополнительное отношение:



То же в SQL можно сделать для таблиц, но там, к сожалению, это усложнит запрос к данным, а в реальных системах еще и замедлит вычисление.

В некоторых случаях удается обойтись и более простым решением. Например, если для хранения почтового индекса используется столбец типа CHAR (6), то отсутствие индекса можно обозначить пробелами. Недостаток в том, что обработка отсутствующих значений перестанет быть тогда универсальной.

Сравнение значений на равенство

Сравнение значений на равенство — более безобидное и очевидное свойство типов в Oracle. Однако для некоторых встроенных "сложных" типов, не говоря уже об объектных, созданных программистом, оно не выполняется. Например, Oracle не позволяет сравнить в запросе SQL два значения типов LOB или XMLTYPE. В приводимом ниже доказательстве команда VARIABLE предназначена для определения в SQL*Plus переменной указанного типа и не имеет отношения к SQL:

```
VARIABLE x CLOB
VARIABLE y CLOB
SELECT 'ok' FROM dual WHERE :x = :y;
-- ошибка !
```

Другая невоодушевляющая особенность сравнения на равенство лежит в области формулировки действия и связана как раз с отсутствующими значениями. Она упоминалась только что. Сравнения с NULL требуют самостоятельного оформления, пример коего последует позже.

Уточнения возможных значений в столбцах

Кроме необходимого в описании столбца типа, можно сообщить дополнительные правила для допустимых значений в полях добавляемых или обновляемых строк таблицы.

Пример:

```
CREATE TABLE projx
(
   projno NUMBER   (4)  NOT NULL
 , pname  VARCHAR2 (14) CHECK (SUBSTR(pname,1,1) BETWEEN 'A' AND 'Z')
 , bdate  DATE         DEFAULT TRUNC ( SYSDATE )
 , budget NUMBER   (10,2)
);
```

Такие правила делятся на две категории: значения данных по умолчанию и "ограничения" (или же "правила") "целостности".

Приписка DEFAULT выражение к определению столбца указывает на значение, которое будет заноситься СУБД в поле добавляемой строки, если программист в операции INSERT никакого значения для этого поля не привел.

Упражнение. Проверьте результаты изменений данных:

```
INSERT INTO projx ( projno, bdate ) VALUES ( 15, SYSDATE + 1 );
INSERT INTO projx ( projno )        VALUES ( 16 );
SELECT * FROM projx;
UPDATE projx SET bdate = DATE '2009-09-17' WHERE projno = 15;
```

```
SELECT * FROM projx;
```

К "ограничениям целостности" в примере выше относятся уточнения описаний столбцов, оформленные с помощью слов NOT NULL и CHECK. Более систематично они вместе с другими разрешенными ограничениями целостности будут описаны в соответствующем разделе ниже.

Свойства столбцов, не связанные со значениями

Шифрование при хранении

С версии 10.2 Enterprise Edition, (а) при установленном дополнении к СУБД Advanced Security Option и (б) при наличии предварительно созданного на сервере "бумажника" Oracle *Wallet* можно потребовать автоматического ("прозрачного") шифрования значений столбца при помещении их в БД и автоматической дешифровки при извлечении. Пример указания трех разных технических способов шифрования для трех разных столбцов:

```
  ...
, sal      NUMBER ( 7, 2 ) ENCRYPT
, comm     NUMBER ( 7, 2 ) ENCRYPT NO SALT
, hiredate DATE ENCRYPT USING 'AES256' IDENTIFIED BY 'SecretWord'
  ...
```

Свойство "хранить в зашифрованном виде" — нефиксированное: его можно добавлять или отменять для столбцов существующей таблицы, при том что эти действия будут сопровождаться автоматической правкой ранее заведенных в таблице данных.

"Прозрачному" шифрованию подлежат только основные встроенные типы (с версии 11 плюс LOB, хотя и отдельным способом), и на употребление этой возможности наложены некоторые ограничения.

Виртуальные столбцы

Версия 11 разрешила объявлять в таблице виртуальные (мнимые) столбцы. Значения в них не хранятся в БД самостоятельно, а вычисляются автоматически при запрашивании на основе действительных значений других полей строки (тем самым они не нарушают 3-ю нормальную форму). Например, в таблице EMP могло бы иметься такое определение:

```
  ...
, sal      NUMBER ( 7, 2 )
, comm     NUMBER ( 7, 2 )
, earnings AS ( sal + comm )
  ...
```

Дополнительные ключевые слова помогут при этом сделать запись яснее, не меняя сути формулировки, например:

```
... earnings AS GENERATED ALWAYS ( sal + comm ) ...
```

Виртуальные столбцы, подобно действительным, можно индексировать (индекс при этом получается с "функционально преобразованным значением ключа") и использовать в формулировании ограничений целостности.

Создание таблиц по результатам запроса к БД

Второй способ завести таблицы в SQL состоит не в явном перечислении столбцов и их свойств, а в ссылке на результат запроса к БД:

```
CREATE TABLE dept_copy AS SELECT * FROM dept;
```

Запрос следует после ключевого слова AS и выше указан чрезвычайно простым, но имеет право быть и сколь угодно сложным. Сначала он вычисляется СУБД, после чего в БД создается таблица со структурой полученного результата (с воспроизведением всех столбцов с их типами), и эта новая таблица тут же заполняется данными результата. Такой способ создания таблицы позволяет экономить усилия программиста и время, когда новую таблицу нужно создать на основе уже имеющихся данных.

Следующий пример показывает, как в таких случаях можно распоряжаться именами столбцов в новой таблице, не копируя слепо имена столбцов из результата запроса:

```
CREATE TABLE emps ( name, department )
AS
SELECT ename, dname FROM emp, dept WHERE emp.deptno = dept.deptno
;
```

Иногда, как возможно в первом примере, команду CREATE TABLE ... AS используют для "копирования" существующей таблицы, указав запрос в форме SELECT * FROM таблица. В таких случаях не следует забывать, что в новой таблице из структурных свойств старой окажутся вспроизведены только типы столбцов и свойства NULL/NOT NULL, но не ограничения целостности и не DEFAULT. При этом воспроизведение свойств NULL/NOT NULL столбцов довольно необычно, так как оно не имеет смысла для запросов с более общей формулировкой.

Именование таблиц и столбцов

Правила именования таблиц и столбцов:

- Две таблицы одной схемы (или принадлежащие одному владельцу, что в Oracle одно и то же) должны иметь разные имена.
- Два столбца одной таблицы должны иметь разные имена.
- Длина имени может достигать не более 30 знаков (в версии 11 словарь-справочник стал допускать имена длиною до 128 знаков, однако для обычных объектов внутренняя программная логика ориентируется на прежнее ограничение).
- "Обычное" имя может состоять только из букв, цифр и символов _, $, # и начинаться с буквы. Будучи заключены в двойные кавычки, имена могут состоять из любых символов.
- Имена не должны совпадать с зарезервированными в Oracle словами.
- Русские названия допускаются без ограничения употребления, если только для БД указана одна из "русских" кодировок.

Имя объекта в предложении SQL, не заключенное в двойные кавычки ("обычное"), попадает в словарь-справочник БД, будучи приведенным к верхнему регистру. Двойные кавычки предотвращают повышение регистра.

Упражнение. Выполните последовательно и сравните ответы СУБД:

```
CREATE TABLE t ( a NUMBER, a VARCHAR2 ( 1 ) );
CREATE TABLE t ( a NUMBER, "a" VARCHAR2 ( 1 ) );
CREATE TABLE t ( a NUMBER, "a" VARCHAR2 ( 1 ) );
CREATE TABLE "t" ( a NUMBER, "a" VARCHAR2 ( 1 ) );
```

Замечания

1. Заключение имени в двойные кавычки — обычно мера вынужденная, так как влечет обязательность указания двойных кавычек и в дальнейшем, после создания таблицы (иначе СУБД все время будет пытаться повысить регистр символов), то есть неудобства употребления.
2. Понятие "зарезервированное слово" в силу разных причин определено в Oracle нечетко. В Oracle SQL и в PL/SQL множества зарезервированных слов, большей частью совпадая, все же различаются. Например, слово TIMESTAMP можно использовать для именования столбца.
3. Использование русских букв в именах не влечет никаких неприятностей со стороны СУБД. Тем не менее внешние по отношению к СУБД программы не всегда умеют их правильно обрабатывать. В силу этого к русским именам таблиц и столбцов в Oracle следует относиться настороженно.

Oracle допускает в SQL ссылки не только на односоставные имена таблиц, но и на составные.

Так, имя таблицы в команде SQL может быть уточнено именем схемы, в которой числится эта таблица, например: SCOTT.DEPT, SYS.OBJ$. В действительности, если в запросе приведено односоставное имя, то при обработке Oracle самостоятельно дополнит его именем схемы. Обычно это будет имя схемы, с которой работает программа (что в Oracle равнозначно имени пользователя), но при желании такое подразумеваемое расширение имени таблицы можно заменить в пределах отдельного сеанса на имя любой другой *существующей* в данный момент в БД схемы, например:

```
ALTER SESSION SET CURRENT_SCHEMA = yard;
```

После этого обращения просто к EMP будут считаться обращениями к YARD.EMP, а не к SCOTT.EMP, как по умолчанию. Этим иногда пользуются при разработке приложения для придания ему гибкости. Подобная подмена умолчательного имени схемы не влечет несанкционированного доступа к объектам чужой схемы, так как будет означать только *попытку* обращения к чужому объекту. Окажется ли успешным обращение, определяется совсем другим механизмом полномочий доступа (прав).

Дополнительно в обращении к таблице можно указать имя заведенной предварительно ссылки на другую БД, с которой установлена связь, и тогда имя может в конечном итоге оказаться трехсоставным, например: SCOTT.EMP@PERSONNELDB. Этим обозначено обращение к таблице EMP в схеме SCOTT БД, именованной как PERSONNELDB.

Удаление таблиц

Удаление таблицы выполняется командой DROP TABLE имя_таблицы.

Сведения о таблице СУБД хранит в двух отдельных местах БД: в справочной части БД (в словаре-справочнике) — описание и в виде самостоятельной структуры (сегмента) в файлах *табличного пространства* — содержимое, данные.

До версии 10 единственным способом удаления по команде DROP TABLE было удаление из словаря-справочника сведений о таблице и удаление сегмента с данными из "рабочей части" БД. С версии 10 возможен (и действует автоматически по умолчанию) второй вариант, когда по команде DROP TABLE описание таблицы и сегмент с ее данными получают новое системное имя, после чего таблица под прежним именем перестает существовать, но фактически какое-то время может быть еще восстановлена. Этот второй способ удаления таблицы воплощает идею "мусорной корзины".

Простое удаление

Пример простой команды удаления таблицы:

```
DROP TABLE dept_copy2;
```

Если на столбцы таблицы определены ссылки внешними ключами других таблиц, СУБД не позволит выполнить DROP TABLE и вернет ошибку. (**Замечания**: (а) это связано не с наличием конкретных ссылающихся строк, а с наличием самого правила внешнего ключа; (б) внешний ключ, ссылающийся на значения столбцов "собственной" таблицы, не препятствует ее удалению). Конструкция CASCADE CONSTRAINTS в команде DROP TABLE позволит-таки удалить таблицу, но при этом СУБД удалит сначала "мешающее" правило внешнего ключа. Столбцы другой, оставшейся таблицы в результате сохранят свои значения, но они уже не будут обременены ограничением ссылочной целостности. Фактически использование CASCADE CONSTRAINTS равносильно последовательному удалению всех правил внешнего ключа, имеющих адресатом таблицу (таковых может быть несколько), и выполнению в завершение простой команды DROP TABLE.

Например, если на столбцы таблицы DEPT_COPY определены ссылки внешними ключами из других таблиц, настоять на удалении DEPT_COPY можно, выдав

```
DROP TABLE dept_copy2 CASCADE CONSTRAINTS;
```

Мусорная корзина

С версии 10 смысл команды DROP изменился. В основном случае после нее и описание, и данные таблицы продолжают храниться на своих местах, но под новыми, присвоенными системой автоматически именами. Для пользователя таблица, как и прежде, пропала, однако на деле все, что нужно для ее восстановления, если такая необходимость возникнет, продолжает храниться в БД. Тем самым для таблиц реализована техника мусорной корзины **(recycle bin)**, хорошо известная по файловым системам.

Список содержимого мусорной корзины можно получить из системной таблицы USER_RECYCLEBIN (публичный синоним — RECYCLEBIN):

```
SELECT object_name, original_name, droptime FROM user_recyclebin;
```

Восстановить таблицу по *исходному имени* (поле ORIGINAL_NAME из USER_RECYCLEBIN) можно, например, так:

```
FLASHBACK TABLE dept_copy2 TO BEFORE DROP;
```

Восстанавливается не все и не точно. У восстановленной подобным образом таблицы будут отсутствовать, возможно, имевшиеся до удаления ограничения целостности, а индексы, хотя и восстановятся, но с системными именами.

Для удаления из мусорной корзины нужно использовать команду *PURGE*, например:

```
PURGE TABLE dept_copy3;
PURGE RECYCLEBIN;
```

Чтобы таблица удалялись сразу безвозвратно, следует использовать конструкцию *PURGE* в команде DROP, например:

```
DROP TABLE dept_copy3 PURGE;
```

Умолчательное помещение таблицы в мусорную корзину можно отменить и вернуться к старой обработке команды DROP. Для этого нужно задать значение OFF параметру СУБД RECYCLEBIN. Последний допускает динамическую установку на уровне СУБД (ALTER SYSTEM …) и на уровне сеанса (ALTER SESSION …), например:

```
ALTER SESSION SET RECYCLEBIN = OFF;
```

Как и в файловых системах, в Oracle мусорная корзина хранит содержимое до поры до времени. В основном она — инструмент восстановления данных после ошибочных действий пользователя, возникших в результате проведения опытов с базой или по неосторожности.

Изменение структуры таблиц

Изменение структуры и других свойств существующей таблицы необязательно требует удаления ее и воссоздания в переделанном виде. Часто оно осуществимо "по живому", что менее хлопотно для программиста и быстрее. Oracle не чинит чрезмерных препятствий таким изменениям и, как правило, допустит их, если они:

- не вступают в логические противоречия с описанием таблицы в БД, и
- не противоречат уже занесенным в таблицу данным.

В общем допустимы следующие структурные изменения: добавление и удаление столбцов, изменение типа столбца, добавление и убирание *ограничений целостности данных*. Все они осуществляются разновидностями команды ALTER TABLE.

С версии 11.2 Oracle предлагает к тому же особую технику редакций объектов хранения для внесения изменений в схему данных. Эта техника не рассматривается непосредственно здесь, но в одном из разделов ниже.

Кроме того, с версии 9 Oracle дает возможность переопределять структуру существующих таблиц не средствами SQL (в принципе достаточными для этой цели), а программно с помощью встроенного системного пакета DBMS_REDEFINITION. Делается это исключительно по технологическим соображениям с тем, чтобы время недоступности данных при перестройке было по возможности малым (формально переопределение происходит online, то есть без прекращения доступности вовсе). Главным образом это актуально для таблиц с большими объемами данных при существующих жестких требованиях к доступности. Такая техника в настоящем тексте, посвященном SQL, естественно, не затрагивается.

Добавление столбца

Добавление столбца не связано ни с какими возможными логическими противоречиями и осуществимо практически всегда.

Пример:

```
ALTER TABLE projx ADD ( pcode VARCHAR2 ( 1 ) );
```

Единственным логическим препятствием к добавлению столбца служит достижение предельного количества столбцов в таблице Oracle — 1000 (значение взято из документации по Oracle).

Следующее замечание касается не виртуальных (с версии 11), а реальных добавляемых столбцов.

В типичном случае такой добавляемый столбец не будет иметь значений, и его добавление не потребует правки в БД существующих строк таблицы, то есть совершится со скоростью внесения изменений в описание таблицы в словаре-справочнике. Однако если для нового столбца указано свойство DEFAULT, потребуется внести значение в новое поле у всех имеющихся строк. Если таблица велика, на это может уйти много времени. С версии 11 действует оптимизация такого исправления данных для случаев, когда вместе с DEFAULT для столбца одновременно указано NOT NULL. В самом деле, наличие этих свойств обоих сразу позволяет отказаться от фактического добавления значения в каждую строку таблицы, а вместо этого единожды сохранить значение выражения, указанного во фразе DEFAULT, в описании таблицы, а при последующих запросах к строке имитировать наличие этого значения в добавленном поле. С версии 11 СУБД Oracle так и поступает, экономя вдобавок дисковое пространство.

Изменение типа столбца

Выполняется с использованием слова MODIFY, например:

```
ALTER TABLE projx MODIFY ( pcode NUMBER ( 6 ) );
```

Логическим препятствием к выполнению такого действия может оказаться вступление в противоречие с заданным ранее ограничением (правилом) целостности, в определении которого участвует столбец. Например наличие внешнего ключа требует *согласованности типов* ссылающихся столбцов и столбцов-адресатов. Сослаться столбцом типа NUMBER на столбец типа VARCHAR2 нельзя.

Техническим препятствием к изменению типа может служить необходимость переформатировать данные в столбце, что здесь не допускается. Как следствие:

- тип столбца можно поменять произвольно (TIMESTAMP на VARCHAR2 и так далее), когда столбец целиком пуст или строки в таблице отсутствуют;
- если данные в столбце есть, возможна замена только:
  - в пределах однородных типов (с одинаковым форматом хранения), например, всех разновидностей NUMBER друг на друга, VARCHAR2 на CHAR и обратно;
  - и если это допускают фактические данные:
    - всегда можно увеличить точность хранения в столбце, но

- уменьшить же ее можно только, когда все существующие значения укладываются в новую точность.

Тип DATE хранится как TIMESTAMP (0) и однороден с ним в указанном смысле, а вот TIMESTAMP WITH *TIME ZONE* не однороден не только с DATE, но и TIMESTAMP и допускает взаимные замены с ними только на пустом столбце.

К сказанному имеется оговорка. В силу особенностей хранения данных типов LOB менять тип столбца на них или из них в остальные нельзя даже на пустом столбце. При необходимости такой столбец придется удалить и воссоздать с другим типом.

Добавление и упразднение ограничений целостности

Выполняется с помощью ключевых слов ADD и DROP или же MODIFY (с отчасти различной областью применимости).

Пример употребления слова MODIFY для добавления и для снятия ограничения NOT NULL:

```
ALTER TABLE projx MODIFY ( pcode NOT NULL );
ALTER TABLE projx MODIFY ( pcode NULL );
```

Пример употребления слов ADD и DROP с целью добавления и снятия *ограничения первичного ключа*:

```
ALTER TABLE projx ADD PRIMARY KEY ( projno );
ALTER TABLE projx DROP PRIMARY KEY;
```

Другие примеры и пояснения последуют далее в самостоятельном разделе.

Со стороны данных препятствий к упразднению ранее существовавших ограничений целостности нет. Препятствие может возникнуть со стороны описания данных, когда имеется зависимое другое ограничение целостности, как это происходит в случае внешнего ключа (тогда нельзя упразднить уникальность или же правило первичного ключа, на которые ссылается какой-нибудь внешний ключ).

Препятствием к добавлению ограничения целостности может служить вступление в противоречие с имеющимися в таблице данными. Например, если в столбце обнаруживается отсутствие значения (хотя бы в поле одной строки), добавить к определению столбца правило NOT NULL не удастся; наличие повторяющихся значений в столбцах не позволит завести правило уникальности или первичного ключа, и так далее.

В некоторых случаях такое противоречие со сложившимися в таблице данными можно преодолевать, о чем будет сказано ниже.

Удаление столбца

Возможно с версии Oracle 8.

Примеры:

```
ALTER TABLE projx DROP ( pcode );
```

```
ALTER TABLE projx DROP COLUMN pcode;
```

Здесь указаны две формы выполнения одного и того же, разве что первая форма носит более общий характер и рассчитана на возможность разом удалить группу столбцов.

Логическим препятствием для удаления столбца может оказаться его участие в определении внешнего ключа в качестве адресата. Не будь этого, по удалению столбца в БД осталось бы некорректно определенное правило внешнего ключа. Вместо последовательного упразднения мешающего ограничения и удаления столбца программист может обойтись одной командой, что быстрее и короче:

```
ALTER TABLE projx DROP ( pcode ) CASCADE CONSTRAINTS;
```

Значения в столбцах бывшего внешнего ключа остаются, но уже не обремененные ссылочной целостностью.

Средства повышения эффективности удаления столбца

Следующие замечания не касаются удаления виртуальных столбцов (с версии 11), но только реальных. Удаление реального столбца влечет физическое переформатирование всех строк таблицы, что может составить технологическую проблему в случае больших объемов данных. Для нее предлагается два решения.

Следующее указание позволит ускорить удаление столбца за счет выполнения контрольной точки автоматически после правки каждых 1000 строк (и тогда сегмент отмены не будет расти чрезмерно):

```
ALTER TABLE projx DROP ( pcode ) CHECKPOINT 1000;
```

Если в процессе удаления произошел сбой, процедуру можно продолжить командой

```
ALTER TABLE projx DROP COLUMNS CONTINUE;
```

или командой

```
ALTER TABLE projx DROP COLUMNS CONTINUE CHECKPOINT 1000;
```

Другой способ — объявить сначала столбец "неиспользуемым", что *не* приведет к физическому перебору имеющихся записей:

```
ALTER TABLE projx SET UNUSED COLUMN pcode CASCADE CONSTRAINTS;
```

Для программы таблица мгновенно окажется без столбца, хотя в базе данные столбца никуда не денутся. СУБД сама их будет изымать из полей по мере обращения к строкам по текущим поводам, то есть заодно. В конечном итоге фактическое переформатирование строк таблицы может растянуться на очень долгое время. Однако при желании его можно будет целенаправленно завершить. Для этого нужно дождаться невысокой загруженности СУБД обычной работой и выдать по образцу следующего:

```
ALTER TABLE projx DROP UNUSED COLUMNS;
```

Переименования

Неудачно или несовременно названную таблицу можно переименовать, например:

```
ALTER TABLE emp RENAME TO employee;
```

Другой способ:

```
RENAME employee TO emp;
```

Команда RENAME по сравнению с более сфокусированной ALTER TABLE … RENAME является более общей, так как позволяет переименовать помимо таблиц объекты некоторых других типов: представление данных (view), генератор последовательности (sequence) и частный синоним. Ее название унаследовано языком SQL от реляционной модели, где имеется одноименная операция.

Пример переименования столбца:

```
ALTER TABLE projx RENAME COLUMN pcode TO project_code;
```

При переименовании объекта СУБД пометит свойства зависимых от данного объектов (например, хранимых процедур на PL/SQL, обращающихся к данной таблице) признаком INVALID, что вызовет потребность их перекомпиляции перед очередным обращением к ним. Ссылки же на таблицу из внешних программ находятся вне компетенции БД; изменение имен пройдет для таких программ незаметно, но в то же время они потеряют свою работоспособность. Это обстоятельство существенно уменьшает ценность операции переименования в реальной практике.

Переименовать таблицу можно и непосредственной правкой таблицы OBJ$ словаря-справочника (см. далее). Такой же подход позволяет переименовать и столбцы таблиц путем внесения правки в таблицу COL$. Однако применять его следует только опытным пользователям и с осторожностью. Он осуществим только для пользователей, имеющих доступ к этим таблицам схемы SYS, и к тому же не изменит состояния зависимых от таблицы подпрограмм на значение INVALID.

Использование синонимов для именования таблиц

Синонимы позволяют завести *дополнительные* имена для обращения к таблице, не обесценивая основного имени:

```
CREATE SYNONYM members FOR emp;
SELECT * FROM emp;
SELECT * FROM members;
```

Теперь, обнаружив обращение к MEMBERS, СУБД определит по своей справочной информации, что это синоним имени EMP, и обратится фактически к EMP. Возможность прямого обращения по имени EMP в тексте команды SQL при этом не теряется, и на работе старых программ появление у таблицы синонимов никак не скажется. Это, однако, не касается команд DDL ALTER/DROP TABLE, где ссылаться следует только на истинное имя таблицы (в полном соответствии с синтаксисом, так как в этих командах используется именно ключевое слово TABLE).

На практике синонимы заводятся с разными целями:

- присвоить таблице имя, больше подходящее ее содержанию;
- упростить имя таблицы в запросе, например, OBJECTS вместо SYS.OBJ$;
- замаскировать обращение к таблице в другой БД или в другой схеме для придания гибкости кода.

Удаление выполняется командой DROP *SYNONYM*:

```
DROP SYNONYM members;
```

Синоним, заведенный в схеме (например, SCOTT), как и таблица, сам становится объектом схемы, доступным изначально только пользователю — *хозяину схемы* или же администратору с соответствующим полномочием. Однако Oracle позволяет создавать еще и PUBLIC *SYNONYM*: "внесхемный", общедоступный синоним. Публичные синонимы активно используются в административной части БД Oracle, но нередко и обычными разработчиками в своих целях. В силу того, что пространства имен публичных и схемных синонимов разные, возможны "спорные" ситуации:

```
CREATE PUBLIC SYNONYM syn1 FOR dept;
-- публичный синоним
CREATE SYNONYM syn1 FOR emp;
-- собственный синоним схемы
SELECT * FROM syn1;
-- DEPT или EMP ?
```

По существующему правилу разрешения имен Oracle отдаст в последнем запросе предпочтение схемному ("частному") синониму. Получается, что появление частного синонима в некоторых случаях способно изменять смысл существовавшего в программе до этого запроса, о чем не следует забывать разработчику.

Создание синонимов в Oracle требует привилегий (полномочий) CREATE *SYNONYM* и CREATE PUBLIC *SYNONYM*, изначально отсутствующих у пользователя SCOTT. В жизни, чтобы обеспечить пользователя синонимами, не обязательно выдавать ему эти привилегии. Создать пользователю Oracle синоним способен, например, администратор.

Использование синонимов для таблиц — это частный случай. В общем синонимы можно создавать и для некоторых прочих хранимых в БД объектов, например, для процедур или функций.

Справочная информация о таблицах и прочих объектах в БД

Место хранения справочной информации об объектах, составляющих БД ("хранимых в БД"), называется в базах данных *словарем-справочником* (*data dictionary*). Русский термин-перевод пришел в эту область ИТ из более старой лексики, где обозначает "справочную книгу, которая содержит собрание слов (словосочетаний, идиом и т. д.), расположенных по определенному принципу, и дает сведения об их значениях, употреблении, происхождении; информацию о понятиях и предметах, ими обозначаемых и др.". В БД этот термин, не утратив самой общей сути, стал значительно более конкретен. Иногда вместо него пользуются термином *системный каталог*.

В Oracle словарь-справочник реализован набором таблиц в составе самой БД, составляющих "справочную часть" БД. Количество таких таблиц в любой БД Oracle — несколько сотен. Официальный их перечень приведен в документации по Oracle. Вот два примера таблиц словаря-справочника:

- USER_TABLES — перечень всех таблиц схемы пользователя и их одиночных (не множественных) свойств;
- USER_TAB_COLUMNS — перечень столбцов всех таблиц схемы пользователя и одиночных свойств столбцов.

Следующие два запроса к таблицам словаря-справочника предоставляют основные сведения о всех имеющихся в схеме пользователя таблицах и основные сведения о столбцах таблицы PROJ. Три команды COLUMN предназначены для SQL*Plus и задают приемлемый формат выдачи на экран:

```
COLUMN table_name FORMAT A30
SELECT
  table_name
, status
FROM
  user_tables
;
COLUMN column_name FORMAT A30
COLUMN data_type   FORMAT A15
SELECT
  column_name
, data_type
, data_length
, data_precision
FROM
  user_tab_columns
WHERE
  table_name = 'PROJ'
;
```

Возможности словаря-справочника дополняются способностью Oracle хранить в нем "комментарии", краткие пояснения к сведениям о таблицах и столбцах. Для заведения комментария в Oracle SQL имеется особая команда COMMENT:

```
COMMENT ON TABLE proj IS 'Проекты в фирме';
COMMENT ON COLUMN proj.budget IS 'Утвержденный бюджет проекта';
```

Наблюдать имеющиеся комментарии можно через таблицы словаря-справочника USER_COL_COMMENTS и USER_TAB_COMMENTS:

```
SELECT comments
FROM   user_tab_comments
WHERE  table_name = 'PROJ'
;
SELECT comments
FROM   user_col_comments
WHERE  table_name = 'PROJ'
```

```
 AND   column_name = 'BUDGET'
;
```

Задания по теме для аудиторной и самостоятельной работы

1) Создать базу данных с названием предметной области на английском языке (список предметных областей находится на странице 33, номер человека в таблице успеваемости подгруппы = его предметная область). В базе данных должно быть не менее 10 таблиц и в каждой из них не менее 500 записей, и 5000 записей в любой одной таблице. Количество атрибутов на одну таблицу от 5 до 10;

2) Вручную добавить по 10 записи в каждую таблицу, далее воспользоваться синтетическим наполнением базы данных с помощью SQL. Чистую заполненную базу данных сохранить в отдельный файл, дальнейшая работа в рамках курса будет проходить с ней;

3) Вручную изменить по 10 записи в каждой таблице, далее придумать правило изменения записей в рамках одной таблицы базы данных с помощью SQL;

4) Вручную удалить по 10 записи в каждой таблице, далее придумать правило удаление записей в рамках одной таблицы базы данных с помощью SQL;

5) Реализовать не менее 10 различных выборок, результат которых имеет смысл и не повторяется;

6) Написать отчёт, в котором:

- ERD диаграмма базы данных;
- Пошаговый процесс создания базы данных;
- Наполнения базы данных данными;
- Выполнение всех операций и вид БД после них.

Предметные области.

1. Электронная коммерция;
2. Финансы;
3. Здравоохранение;
4. Образование;
5. Транспорт;
6. Гостиничный бизнес;
7. Спорт;
8. Музыкальная индустрия;
9. Киноиндустрия;
10. Издательское дело;
11. Логистика;
12. Строительство;
13. Недвижимость;
14. Туризм;
15. Правоохранительные органы;
16. Социальные сети;
17. E-learning;
18. Медицинская диагностика;
19. Промышленность;
20. Энергетика;
21. Телекоммуникации;
22. Сфера услуг;
23. Автомобильная промышленность;
24. Сельское хозяйство;
25. Разработка программного обеспечения;
26. Научные исследования;
27. Биология;
28. История.