МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
Московский физико-технический институт
(государственный университет)

Кафедра инфокоммуникационных систем и сетей

ORACLE SQL
В ЗАДАЧАХ

Учебно-методическое пособие

Составители: *А.В. Васильев*
Я.Ф. Ходаковский

МОСКВА
МФТИ
2011

УДК 004.655

Oracle SQL в задачах: Учебно – методическое пособие / Сост. А.В. Васильев, Я.Ф. Ходаковский — М.: МФТИ, 2011. — 47 с.

Задачник полностью соответствует курсу по выбору МФТИ «Практикум по Oracle SQL» и содержит весь необходимый материал для полного освоения возможностей Oracle SQL.

К каждой главе имеется введение, где рассматриваются теоретические аспекты, необходимые для успешного решения задач.

Издание будет полезно как начинающим изучать SQL, так и желающим повысить свой уровень: задачник содержит задачи разного уровня сложности.

УДК 004.655

Содержание

Введение....5
1. Выборка информации из таблиц....6
Задачи к главе....9
2. Использование однострочных функций, хранение и обработка дат и времени, неявное преобразование данных....12
Задачи к главе....18
3. Группировка данных....21
Задачи к главе....23
4. Подзапросы....25
Задачи к главе....26
5. Операторы DML....27
Задачи к главе....31
6. Операторы работы с множествами....32
Задачи к главе....33
7. Иерархические запросы....34
Задачи к главе....35
8. Аналитические функции....36
Задачи к главе....39
9. Транзакции....40
Задачи к главе....41
10. Создание таблиц....43
Задачи к главе....46
Литература....48

Введение

Задачник создан на основе курса Oracle SQL, который ведется в Московском физико-техническом институте, поэтому рекомендуется пользоваться задачником параллельно с посещением этого курса.

При изучении Oracle-диалекта языка SQL авторы часто сталкивались с тем, что книги по Oracle SQL либо рассчитаны на профессионалов (требуют предварительной подготовки), либо содержат только небольшую часть материала, необходимого для реальной работы с Oracle SQL. Этот факт заставляет читать большое количество разнородного материала, чтобы освоить данную область достаточно полно. Данный задачник создавался с целью экономии времени при изучении Oracle SQL, поэтому он сочетает сжатый объем с покрытием всех тем, часто встречающихся в реальной работе. Задачи составлялись таким образом, чтобы они образовывали некий «базис» знаний, т. е. не повторяли друг друга и охватывали очень широкий круг вопросов по теме. Конечно, невозможно уместить в сжатое пособие абсолютно все аспекты SQL, так что насколько удалась эта идея — судить читателю.

В основном все задачи составлены по одной стандартной схеме HR («сотрудники»), предоставляемой компанией Oracle в стандартном пакете. Скрипт для создания таблиц этой схемы также доступен для скачивания на сайте http://ipccenter.ru. Задачи, не использующие схему HR, не требуют каких-либо предварительных установок.

При решении задач необходимо пользоваться инструментами той главы, где они расположены. Если к задаче написано *Указание*, то решение надо основывать на этой рекомендации.

Авторы надеются, что задачник окажется полезным, и будут рады любым отзывам, предложениям по содержанию, замеченным опечаткам и др., направленным по адресу zolenenok@gmail.com.

1. Выборка информации из таблиц

Для выборки информации из одной или нескольких таблиц используется оператор SELECT. Все синтаксические конструкции не являются чувствительными к регистру, поэтому записи `select`, `Select`, `SeLeCt` — эквивалентны. Синтаксис оператора SELECT следующий:

```
SELECT [[ALL] | [DISTINCT | UNIQUE]]
{*, column [alias],...} FROM {table1 t1, ...}
[WHERE predicat]
```

Ключевое слово ALL является значением по умолчанию и означает выборку всех строк из множества. Для получения уникальной выборки необходимо указать ключевое слово UNIQUE или DISTINCT. Каждой таблице можно присвоить алиас для того, чтобы не писать полное имя таблицы каждый раз, когда требуется указать таблицу, из которой необходимо сделать выборку. Алиасы для столбцов указываются в самом конце, например, `e.employee_name AS emp_name`. Ключевое слово AS опционально. Если алиас для столбца указан без двойных кавычек, то название столбца приводится верхнему регистру. Допустимыми символами являются все буквы, цифры и символы `_, #, $`, название должно начинаться с буквы, нельзя использовать для названия столбца зарезервированные слова. Если столбцу необходимо присвоить чувствительное к регистру название или неразрешенное название (что не рекомендуется), то алиас надо взять в двойные кавычки, например, `... AS "Employee Name"`. Длина столбца не может превышать 30 символов. Для выборки всех столбцов необходимо написать * после слова SELECT (далее это место будем называть select-списком).

В select-списке возможно использовать:

- ✓ арифметические операции `+, -, *, /`, над столбцами — порядок выполнения операций стандартный,
- ✓ оператор конкатенирования `||`, строковые литералы заключаются в одинарные кавычки, литерал одинарная кавычка обозначается как `''''`,
- ✓ функции для работы со столбцами (групповые и одиночные),
- ✓ псевдостолбец ROWNUM — порядковый номер выбранной строки (начиная с 1), ROWID — уникальный идентификатор

(указатель) на строку в конкретной таблице.

Значение NULL означает "не определено" и имеет по умолчанию тип varchar2 (является пустой строкой, оно же литерал ''). Для арифметических операций и операций работы с датами один из аргументов – NULL, то и значение всего выражения будет равно NULL.

Для упорядочения выборки используется конструкция ORDER BY. Ее синтаксис следующий:

```
SELECT .. FROM .. [ORDER BY {column, expr, position}
[ASC | DESC][ NULLS FIRST | NULLS LAST]];
```

Так как упорядочивание выборки делается в самом конце, то допустимо в операторе использовать названия столбцов из таблицы, выражения или позицию столбца в выборки (начиная с 1). Если упорядочивание необходимо в обратном порядке, то необходимо указывать ключевое слово DESC, по умолчанию сортировка по возрастанию (ASC). Для специального расположения null-значений указывается конструкция NULLS FIRST или NULLS LAST (null-значения сначала или в конце). Они остаются в конце для сортировки ASC и в начале для сортировки DESC.

Для фильтрации используются предикаты — логические конструкции, которые возвращают TRUE или FALSE. Строки, для которых предикаты ложны, исключаются из выборки.

Примеры предикатов:

- ✓ сравнения (>, <, =, <=, >=, {!= | <> | ^=}), например, salary >= 1000, оператор сравнения со значением NULL всегда ложен,
- ✓ логические (NOT, AND, OR: указаны в порядке приоритета),
- ✓ с использованием оператора LIKE, например, employee_name like 'A%', символ % означает любую последовательность символов, символ _ означает любой символ; для использования специальных символов для поиска в строке необходимо экранировать их, например, employee_name like 'A_%' escape '\',
- ✓ с использованием оператора IN (…) — проверяет, что выражение совпадает хотя бы с одним из списка,
- ✓ сравнения на множестве ({>, <, =, <=, >=, {!= |

<> | ^=}}{ANY | SOME | ALL}()), где ANY (эквивалентно SOME) означает "хотя бы один из", ALL означает "каждый из",

- ✓ оператор .. BETWEEN A AND B — проверяет выражение .. >= A and .. <= B.

Предикаты используются для фильтрации результирующих значений и соединения таблиц.

Существует три типа соединения таблиц: декартовы (cartesian), внутренние (inner), внешние (outer).

Декартовы (cartesian). Используются очень редко и возвращают всевозможные комбинации из двух множеств (таблиц). Следующие записи эквивалентны:

```
SELECT ... FROM employees, departments
SELECT ... FROM employees CROSS JOIN departments
```

Внутренние (inner). Внутренние соединения могут быть реализованы через предикаты в WHERE, а также согласно синтаксису ANSI. Следующие три записи эквивалентны:

```
SELECT ... FROM employees e, departments d
   WHERE e.department_name = d.department_name,
SELECT ... FROM employees e
   [INNER] JOIN departments d
   on (e.department_name = d.department_name),
SELECT ... FROM employees e
   [INNER] JOIN departments d
   using(department_name).
```

Соединение возвращает записи, которые ТОЧНО удовлетворяют условиям. Слово INNER опционально; можно использовать конструкцию USING, если названия столбцов соединения совпадают.

Еще один вариант соединения — NATURAL, который внутренним образом соединяет таблицы по столбцам, имеющим одинаковое имя. Например, эквивалентные записи:

```
SELECT ... FROM employees e
   NATURAL JOIN departments d
   /* два столбца имеют одинаковые имена */,
SELECT ... FROM employees e
   [INNER] JOIN departments d
   on (e.department_name = d.department_name),
```

Внешние (outer). Внешние соединения вводят понятия основной

таблицы и дополнительной. Указанные условия должны быть выполнены, а если они не выполнены, то в столбцах дополнительно таблицы будут значения NULL. Например, следующие запросы эквивалентны:

```
SELECT ... FROM departments d, employees e
   WHERE d.department_name = e.department_name(+)
   /* Oracle: таблица e дополнительная */,
SELECT ... FROM departments d
   LEFT [OUTER] JOIN employees e
   ON (e.department_name = d.department_name)
   /* Таблица e дополнительная */,
SELECT ... FROM departments d
   LEFT [OUTER] JOIN employees e
   USING (department_name)
   /* Таблица e дополнительная */.
```

Существуют LEFT и RIGHT внешние соединения — они полностью эквивалентны, просто меняются положения основной и дополнительной таблиц.

Еще один вариант внешних соединений — FULL [OUTER] JOIN: возвращает все строки из обоих таблиц, дополненные NULL, если они не удовлетворяют условию соединения.

Во внешних соединениях запрещено создавать циклические внешние соединения, таблица может быть дополнительной только по отношению к одной другой таблице.

Порядок выполнения запроса следующий: сначала делаются соединения таблиц (joins), потом фильтрация (where), потом составляется select-список, потом делается упорядочивание. Надо понимать, что предикаты, указанные в WHERE, не обязательно являются фильтрами, а могут быть и условиями соединения.

Задачи к главе

1. Выведите имя сотрудника колонкой "Emp First Name", фамилию колонкой "Emp Last Name", номер сотрудника колонкой "Emp No.", зарплату этого сотрудника при увеличении ее на 10% и затем все остальные столбы таблицы employees.

2. Почему ошибочен запрос select employee_name "Full Em-

ployee Name in the company" from employees; ?

3. Выведите столбцом summ сумму зарплаты и комиссионных сотрудника, потом все строки таблицы employees. Результат должен быть отсортирован по summ, потом по четвертому столбцу в итоговой выборке.

4. Выведите все записи таблицы employees, отсортированные сначала по id отдела в порядке убывания, потом по году приема на работу в порядке возрастания, потом по комиссионным (если есть пустые значения, то они должны выводиться в начале).

5. По таблице employees в первом столбце person выведите имя, фамилию сотрудника и id его должности в формате: [<имя> <фамилия> (<id должности>)]. Во втором столбце mgr соедините строку: "Mgr's id: " и id менеджера этого сотрудника. Для двух этих столбцов используйте различные способы конкатенирования строк.
Указание: конкатенация строк оператором || и функцией concat.

6. Выведите уникальный набор из id должностей сотрудников. Предложите два варианта написания запроса.
Указание: использование ключевых слов distinct/unique.

7. По таблице employees выведите первым столбцом порядковый номер извлеченной строки, вторым — фамилию сотрудника.
Указание: использование псевдостолбца rownum.

8. Выведите всех сотрудников из таблицы employees, у кого нет комиссионных и есть менеджер.

9. Выведите всех сотрудников с зарплатой не менее 3000, кроме сотрудника с фамилией King.

10. Выведите всех сотрудников, которые находятся в подчинении у менеджеров со следующими id: 100, 101, 102.

11. Какие еще существуют операторы (назовите два), полностью эквивалентные оператору IN?

12. Выведите всех сотрудников, которые *не* находятся в подчинении у менеджеров с id: 101, 102.

13. Решите предыдущую задачу с помощью оператора `ALL`.

14. Выведите всех сотрудников, зарплата которых не менее 4200 и не более 6000.
Указание: использование оператора `between`.

15. Выведите всех сотрудников, фамилия которых записана в следующем формате: [M<любой символ>R_<любая последовательность символов>].
Указание: использование оператора `like`.

16. Выведите всех сотрудников, у которых id должности либо `"IT_PROG"`, либо `"FI_ACCOUNT"`, зарплата которых больше 4200.
Указание: запрос должен быть написан без использования скобок.

17. Выведите всевозможные комбинации id сотрудника и названия отдела. Решите задачу также с использованием ANSI синтаксиса.

18. Выведите все `id` сотрудников и название отдела, в котором этот сотрудник работает. Предложите вариант записи запроса с соединением в WHERE, а также два варианта синтаксиса ANSI. Какое слово во внутренних соединениях ANSI является опциональным?
Указание: использование `INNER JOIN`.

19. Выведите все колонки из таблиц `employees` и `departments`, соединяя таблицы внутренним образом по столбцам, имеющих одинаковое название.
Указание: использование `NATURAL JOIN`.

20. Выведите id всех отделов первой колонкой и сотрудников, в них состоящих, второй колонкой. Если в отделе нет ни одного сотрудника, то вторая колонка должна содержать NULL для такого отдела. Предложите решение, используя синтаксис Oracle, а также два варианта синтаксиса ANSI. Какое слово в записи ANSI является необязательным?

Указание: использование `LEFT OUTER JOIN`.

21. Решите предыдущую задачу, используя `RIGHT OUTER JOIN`.

22. Приведите пример запроса, когда Oracle возбуждает исключение о том, что таблица не может быть дополнительной сразу к нескольким основным (ORA-01417: a table may be outer joined to at most one other table).

23. Приведите пример запроса, когда Oracle возбуждает исключение о том, что таблицы не могут быть дополнительными друг к другу; другими словами, таблицы не образуют «циклов» по средствам внешних соединений (ORA-01416: two tables cannot be outer-joined to each other).

24. Где и почему с точки зрения производительности более разумно размещать предикаты, которые могут быть размещены как части `WHERE` запроса, так и части `ON` условия соединения таблиц запроса? Приведите пример таких запросов.

2. Использование однострочных функций, хранение и обработка дат и времени, неявное преобразование данных

Описание некоторых универсальных функций

NVL(*expr1*, *expr2*) — возвращает *expr1*, если он не `NULL`, и *expr2* – иначе, все значения вычисляются до передачи в функцию.

NVL2(*expr1*, *expr2*, *expr3*) — если *expr1* не является `NULL`, то возвращает *expr2*, иначе – *expr3*. Все значения вычисляются до передачи в функцию.

COALESCE(*expr1*, *expr2*, ...) — возвращает первый не `NULL` элемент, значения вычисляются по мере надобности (short-circuit evaluation).

DECODE(*expr*, *search*, *result*[, *search*, *result*]...[, *default*]) — сравнивает *expr* с *search* один по одному; если *expr* равен *search*, то возвращается соответствующий *result*; если совпадений не найдено, возвращается *default* или `NULL` (если *default* опущено). Использует short-circuit evaluation.

CASE *expr* WHEN *comparison_expr* THEN *return_expr* ... [ELSE *else_expr*] END — сравнивает *expr* последовательно с *comparison_expr* и возвращает *return_expr*, если они равны; если не было найдено ни одного совпадения, то возвращается *else_expr* (или `NULL`, если ELSE опущен).
CASE WHEN *condition* THEN *return_expr* ... [ELSE *else_expr*] END — проверяет последовательно условия *condition* и возвращает *return_expr*, если условие правдиво. Возвращает `NULL`, если ELSE опущен и ни одно условие не совпало.
NULLIF(*expr1*, *expr2*) — сравнивает *expr1* и *expr2*: если они равны, то возвращает `NULL`, иначе – *expr1*.
CAST(*expr* AS *type_name*) — приводит *expr* к типу данных *type_name*.
LNNVL(*expr*) — для использования в WHERE, позволяет оценить выражение, когда один или оба операнда условия могут быть NULL, возвращает: LNNVL(TRUE) => FALSE, LNNVL(FALSE) => TRUE, LNNVL(UNKNOWN) => TRUE.

Некоторые функции работы со строками

ASCII(*expr*) — возвращает числовое значение (в коде ASCII) символа *expr*.
CHR(*expr*) — возвращает символ, код ASCII которого *expr*.
CONCAT(*expr1*, *expr2*) — дописывает *expr2* в конец *expr1* и возвращает полученную строку.
INITCAP(*expr*) — конвертирует первую букву каждого слова в строке *expr* в верхний регистр.
INSTR(*string* , *substring* [, *position* [, *occurrence*]]) — ищет вхождение *substring* в *string* и возвращает позицию, в которой она была найдена. *position* — номер позиции, с которой следует начать поиск, *occurrence* — какое по счету вхождение надо возвращать.
LENGTH(*expr*) — возвращает количество символов в строке.
LOWER(*expr*) — приводит строку *expr* к нижнему регистру.
LPAD/RPAD(*expr1*, *n* [, *expr2*]) — возвращает *expr1*, дополненную слева/справа до длины *n* символами из *expr2*.
LTRIM/RTRIM/TRIM(*text_exp* [, *trim_exp*]) — функция удаляет символы из левой/правой/обоих частей *text_exp*, *text_exp* — удаляемые символы (по умолчанию пробел).
REPLACE(*expr*, *search_string* [, *replacement_string*]) — ищет вхождение *search_string* в *expr* и заменяет его строкой *replacement_string* (по умолчанию `NULL`).

SUBSTR(*expr*, *position* [, *substring_length*]) — возвращает подстроку *expr*, начинающуюся с позиции position; можно ограничить длину подстроки *substring_length*.
TRANSLATE(*expr*, *from_string*, *to_string*) — возвращает *expr*, где все символы из *from_string* заменены соответствующими символами из *to_string*.
UPPER(*expr*) — возвращает строку *expr*, все символы которой приведены к верхнему регистру.

Некоторые функции работы с числами

BITAND(*expr1*, *expr2*) — возвращает результат побитовой операции AND для чисел *expr1* и *expr2*.
CEIL(*expr*) — возвращает наименьшее целое число, большее или равное *expr*.
EXP(*expr*) — возвращает число *e*, возведенное в степень *expr*.
FLOOR(*expr*) — возвращает наибольшее целое число, меньшее или равное *expr*.
LN(*expr*) — возвращает натуральный логарифм числа *expr*.
MOD(*expr1*, *expr2*) — возвращает остаток от деления *expr1* на *expr2*.
POWER(*expr1*, *expr2*) — возвращает число *expr1*, возведенное в степень *expr2*.
ROUND(*n* [, *integer*]) — возвращает *n* округленное до *integer* знаков после запятой (по умолчанию 0); если *integer* отрицательный, то округление происходит влево от десятичной точки.
TRUNC(*number*, [*decimal_places*]) — усекает number до *decimal_places* знаков (по умолчанию 0); если *decimal_places* отрицательный, то округление происходит влево от десятичной точки.

Oracle хранит даты как числа, поэтому над датами можно выполнять операцию вычитания, и 1 будет соответствовать одному григорианскому дню. Таким образом, `SYSDATE + 1` будет равен следующему дню.

Некоторые функции работы с датами

ADD_MONTHS(*date*, *integer*) — возвращает date плюс integer месяцев.
EXTRACT({ YEAR | MONTH | DAY | HOUR | MINUTE | SECOND } FROM *date*) — выделяет год | месяц | .. из *date*.
LAST_DAY(*date*) — возвращает значение последнего дня месяца,

содержащегося в *date*.
MONTHS_BETWEEN(*date1*, *date2*) — возвращает количество месяцев между *date1* и *date2*.
ROUND(*date* [, *fmt*]) — округляет *date* по умолчанию до ближайших суток. Можно указать необязательный параметр *fmt* для указания единиц округления.
SYSDATE — возвращает текущую дату и время, установленную операционной системой, на которой установлена база данных. Возвращаемый тип DATE.
SYSTIMESTAMP — возвращает тип TIMESTAMP WITH TIMEZONE, содержащий текущее время базы наряду с ее временным поясом.
TRUNC(*date* [, *fmt*]) — усекает date по умолчанию до начала дня. Можно указать необязательный *fmt* для указания единиц округления.

Некоторые функции для работы с регулярными выражениями
REGEXP_LIKE(*source_string*, *pattern*[, *match_parameter*]) — возвращает TRUE, когда исходная строка *source_string* соответствует регулярному выражению *pattern*; *match_parameter* регулирует опции совпадения, такие как учитывать/не учитывать регистр ('c'/'i').
REGEXP_INSTR(*source_char*, *pattern* [, *position* [, *occurrence* [, *return_option* [, *match_parameter*]]]]) — ищет вхождение pattern в *source_char* и возвращает индекс этого вхождения; опционально указывается позиция, с которой следует вести поиск (*position*), номер вхождения (*occurrence*), опции возврата (*return_option*), опции совпадения (*match_parameter*).
REGEXP_REPLACE(*source_char*, *pattern* [, *replace_string* [, *position* [, *occurrence* [, *match_parameter*]]]]) — ищет вхождения шаблона с строку *source_char* и заменяет их на *replace_string*, остальные опции имеют то же значение, что и в предыдущем примере.
REGEXP_SUBSTR(source_char,pattern[,position[,occurrence[,match_param[,subexpr]]]]) — возвращает подстроку строки *source_char*, соответствующей шаблону, которая начинается с позиции, заданной параметром *position*, остальные опции имеют то же значение, что и в предыдущем примере.

Некоторые функции преобразования типов данных
TO_NUMBER(*expr* [, *fmt* [, 'nlsparam']]) — конвертирует *expr* в тип данных NUMBER.

TO_CHAR(*expr* [, *fmt* [, 'nlsparam']]) — конвертирует *expr* в тип данных VARCHAR2.
TO_DATE(*expr* [, *fmt* [, 'nlsparam']]) — конвертирует *expr* в тип данных DATA.
TO_TIMESTAMP(*expr* [, *fmt* ['nlsparam']]) — конвертирует *expr* в тип данных TIMESTAMP.

Описание некоторых форматов NUMBER

Параметр	Пример формата	Описание
9	999	Возвращает в указанных позициях цифры с ведущим знаком минус, если число отрицательное
0	0999	Возвращает число с ведущими (левыми) нулями
.	999.99	Возвращает в указанной позиции десятичную точку
,	999,99	Возвращает в указанной позиции запятую
$	$999	Возвращает ведущий знак доллара
EEEE	9.9EEEE	Возвращает число, используя научную нотацию
FM	FM90.9	Удаляет из числа ведущие и хвостовые пробелы
TM	TM	Возвращает число, используя минимальное количество символов

Описание некоторых форматов DATE

Параметр	Пример формата	Описание
YYYY	2011	Все четыре цифры года
YY	11	Последние две цифры года
MM	01	Двузначный номер месяца года
MONTH/ Month	JANUARY/ January	Полное название месяца в верхнем регистре/с первой заглавной буквой (тип char(9))*
MON/ Mon	JAN/Jan	Первые три буквы названия месяца*
DD	31	Двузначный номер дня месяца
DAY/	SATURDAY/	Полное название дня в верхнем

Day	Saturday	регистре/с первой заглавной буквой*
DY/Dy	SAT/Sat	Первые три буквы названия дня в верхнем регистре/с первой заглавной буквой*
J	2455720	Юлианский день — число дней, прошедшее с 1 января 4713 г. до н.э.
HH24	23	Номер часа в 24-часовом формате
HH	11	Номер часа в 12-часовом формате
MI	57	Двузначное число минут
SS	59	Двузначное число секунд
SSSSS	46748	Число секунд, отсчитываемое от полуночи

* Язык определяется параметром NLS_DATE_LANGUAGE в системном представлении V$NLS_PARAMETERS.

Неявное предобразование типов данных происходит каждый раз, когда сравниваемые типы данных не соответствуют. Примеры неявных преобразований:

Преамбула	**Пример**	**Явный эквивалент**
hire_date имеет тип Date	`WHERE hire_date = '17.06.1987' ...`	`WHERE hire_date = TO_DATE( '17.06.1987') ... *`
hire_date имеет тип Date	`SELECT '='|| hire_date ...`	`SELECT '='||TO_CHAR( hire_date) ... *`
code имеет тип VARCHAR2	`WHERE code = 10934 ...`	`WHERE TO_NUMBER(code) = 10934 ...`
salary имеет тип NUMBER	`SELECT salary + '100' ...`	`SELECT salary + TO_NUMBER('100') ...`
salary имеет тип NUMBER	`SELECT '$'|| salary ...`	`SELECT '$'|| TO_CHAR(salary) ...`
-	`WHERE rowid = 'AAAAECAABAAAAgi AAA' ...`	`WHERE rowid = CHARTOROWID( 'AAAAEC AABAAAAgiAAA') ...`

* Формат даты определяется параметром `NLS_DATE_FORMAT` в

системном представлении V$NLS_PARAMETERS.

Задачи к главе

1. Выведите числовое значение (в коде ASCII) символа 'a', символа 'A' и третьей колонкой символ ASCII, код которого равен 42.
Указание: использование ASCII и CHR.

2. Выведите по таблице locations первым столбцом уникальный идентификатор местоположения, вторым столбцом street — адрес, образованный усечением слева цифр и пробелов у колонки street_address.
Указание: использование LTRIM.

3. По таблице locations первой колонкой выведите адрес, второй — вывести тот же адрес, предварительно конвертировав каждую первую букву нового слова в верхний регистр.
Указание: использование INITCAP.

4. По таблице jobs выведите должность и второй колонкой последнее слово в названии этой должности.
Указание: использование SUBSTR, INSTR.

5. По таблице employees выведите строку следующего типа "<Имя сотрудника>'s email is <Электронная почта сотрудника, приведенная к нижнему регистру>@myCompany.com" (получившийся адрес электронной почты обернуть в угловые скобки); пример правильного синтаксиса строки для сотрудника Yaroslav: Yaroslav's email is <khodakovskiy@myCompany.com>.
Указание: использование LOWER.

6. Выведите зарплату сотрудников как строку длины 5 символов; в случае необходимости дополнить строку слева нулями до требуемой длины.
Указание: использование LPAD.

7. Выведите имя, фамилию сотрудника и третьей колонкой должность, заменив в ней встречающиеся символы '_' на '-'.

Отсортировать полученные результаты по суммарной длине имени и фамилии, затем по первому и второму столбцам выборки.
Указание: использование `LENGTH`, `REPLACE`.

8. Выведите имя, фамилию сотрудника, его `email` и четвертой колонкой — email сотрудника, заменив в нем символы по следующему соответствию: A-a, E-e, I-i, O-o, U-u, Y-y.
Указание: использование `TRANSLATE`.

9. Выведите зарплату сотрудников компании в предположении, что из нее вычли налог 12.5% в следующих колонках:

- `low`: округленная до целого снизу,
- `up`: округленная до целого сверху,
- `round_k`: округленная до сотен, в тысячах,
- `trunk_k`: усеченная до сотен, в тысячах.

Указание: использование `ROUND`, `TRUNC`, `CEIL`, `FLOOR`.

10. Выведите 'Yes', если в разложении числа 123456789 по степеням двойки существует двойка в степени 10, и 'No' – иначе.
Указание: использование `BITAND`, `POWER`.

11. Выведите 'Yes', если число 3607 является делителем числа 123456789, и 'No' – иначе.
Указание: использование `MOD`.

12. По таблице `regions` выведите первой колонкой с названием `num` значение 'First', если идентификатор региона 1, 'Second', если 2, и 'Other' – во всех остальных случаях; далее вывести остальные столбцы таблицы.

13. Напишите запрос, выводящий все простые числа до $N = 1000$, сложность алгоритма должна быть не больше, чем $O(N \cdot sqrt(N))$.

14. Предположим, что строка '$0,125' определяет одну единицу зарплаты в долларах. Выведите зарплату в упомянутых единицах, комиссионные (например, как '0.12'), зарплату (например, как '$3,456'). Ведущих и хвостовых пробелов быть не должно. Для каждой колонки используйте только функцию преобразования типов.

15. Выведите сотрудников, фамилии которых начинаются с H или K, содержат 2 одинаковые буквы подряд, оставшаяся после повторяющихся букв часть слова не заканчивается на букву s.
Указание: использование `REGEXP_LIKE`.

16. Выведите адрес и позицию второго буквенного символа, входящего в эту строку.
Указание: использование `REGEXP_INSTR`.

17. Выведите страну и вторым столбцом ту же строку с исключенными гласными буквами, кроме первой.

18. Выведите адрес и подстроку из этого адреса, начинающуюся и заканчивающуюся цифрой.
Решение: использование `REGEXP_SUBSTR`.

19. Выведите текущую дату, время и часовой пояс в формате:
<ГОД-МЕС-ДЕНЬ ЧАС24:МИН:СЕК.милиСЕК(3 разряда) ПОЯС>.
Указание: использование `SYSTIMESTAMP`.

20. В чем отличие системных функций `SYSDATE` и `SYSTIMESTAMP`?

21. Выведите дату приема на работу сотрудника, его имя, фамилию и текущий стаж. Пример вывода стажа: 3 Years, 10 Months, 18 Days.
Указание: использование `EXTRACT`.

22. Предположим, что в компании зарплата начисляется пятого числа каждого месяца, зарплату сотрудник начинает получать, начиная со следующего месяца после месяца приема на работу. Выведите дату приема на работу и дату, когда сотруднику была выплачена первая зарплата.

23. Объясните механизм неявного преобразования типа данных в запросе: `SELECT * FROM employees WHERE hire_date = '17.06.1987';`

24. Почему при исполнении запроса `SELECT * FROM locations WHERE postal_code = 10934;` могла возникнуть ошибка ORA-

01722: invalid number?

25. Каким образом происходит неявное преобразование данных при арифметических операциях над числовым и строковым столбцами?

26. Каким образом происходит неявное преобразование типов в запросе:
`SELECT * FROM employees WHERE employee_id = '100'`?

27. Как преобразуются числа при арифметических операциях с датами? Покажите это на примере единицы.

28. Выведите сотрудников, у которых комиссионные меньше 20% или не указаны. Задачу решите с применением только одного предиката.
Указание: использование `LNNVL`.

29. Выведите размер комиссионных (`commission_pct * salary`) и 0, если они отсутствуют с помощью функций `NVL`, `COALESCE`, `NVL2`, `DECODE`, `CASE`.

30*. Какое максимальное количество компонентов или аргументов возможно в функциях `DECODE`, `CASE`, `COALESCE`?

3. Группировка данных

В этой части рассматриваются групповые (агрегатные) функции, создание групп данных в выборке и наложение ограничений на них.

Групповые функции применяются в группе, оперируют целыми группами строк и возвращают всего одну строку. Общий синтаксис применения групповых функций на примере функции `COUNT`:
`SELECT COUNT({[ALL]|{UNIQUE | DISTINCT}} expr) ...`
Слово `ALL` (по умолчанию) необязательно, слово `UNIQUE | DISTINCT` необходимо, если групповая функция должна быть посчитана для уникального набора данных из группы. Групповые функции "выкидывают" из рассмотрения `NULL`-значения.

Общий синтаксис запросов:

```
SELECT group_columns, group_functions
FROM tables
```

```
[WHERE condition]
[GROUP BY group_by_expression]
[HAVING group_condition]
[ORDER BY column];
```

Фраза `GROUP BY` группирует строки в блоки с одним и тем же значением, если фраза опущена, то группой считаются все строки из выборки (и запрос подразумевает группировку),

фраза `HAVING` позволяет накладывать ограничения на группу, тут можно использовать столбцы из `GROUP BY` и также групповые функции,

фраза `ORDER BY` должна применяться на столбцах из `GROUP BY` и/или групповых функциях (выполняется самой последней в запросе), в `SELECT` разрешено использовать столбцы из `GROUP BY` и групповые функции, а также однострочные функции над ними.

Группировка выполняется уже над построенным множеством, т.е. после обработки соединений таблиц и условия `WHERE`.

Фильтрация групп (`HAVING`) выполняется над уже построенными группами, то есть после `GROUP BY`.

Примеры некоторых групповых функций (*expr* — это или столбец, или однострочная функция над столбцом/столбцами).

Функция	Описание
`AVG(`*expr*`)`	Возвращает среднее арифметическое значение *expr*
`COUNT(`*expr*`)`	Возвращает количество строк, возвращаемых запросом, в котором участвует *expr**
`MAX(`*expr*`)`	Возвращает максимальное значение *expr*
`MIN(`*expr*`)`	Возвращает минимальное значение *expr*
`SUM(`*expr*`)`	Возвращает сумму значений *expr*

* также допустима конструкция `COUNT(*)` — возвращает количество строковых кортежей (даже если они являются `NULL`-кортежами).

Допускается группировка по группам, например:
`SELECT MAX(AVG(salary)) .. GROUP BY ..`
выводит максимальное среди средних по группам.

Задачи к главе

1. Выведите по всем сотрудникам максимальную, среднюю и минимальную зарплату, суммарную зарплату всех сотрудников, количество сотрудников и количество отделов, в которых состоят сотрудники.

2. Посчитайте количество сотрудников, у кого есть комиссионные, средние комиссионные среди всех сотрудников и средние комиссионные среди получающих их сотрудников.

3. Дан запрос:
```
SELECT COUNT(*) c1, COUNT(employee_id) c2,
COUNT(manager_id) c3, COUNT(1) c4 FROM employees;.
```
Какие из его столбцов будут содержать одинаковые значения, а какие могут быть отличными и почему?

4. Выведите все номера отделов, в которых состоят сотрудники, и максимальную зарплату по каждому и них.

5. Выведите номер отдела, должность и максимальную зарплату для этой должности для отдела с `id = 50`. Выборку отсортировать по максимальной зарплате для этой должности.

6. Выведите номер отдела, должность, максимальную и минимальную зарплату для этой должности в отделах с номером, не большим 50. В выборку включить только те отделы, где минимальная зарплата по должности больше 5000. Результат отсортировать по максимальной зарплате.

7. Выведите максимальное значение средней зарплаты по отделам. Результат округлите до ближайшего целого.

8. Дан запрос:

```
SELECT e.job_id, MAX(e.salary) max_sal
FROM employees e, departments d
WHERE e.manager_id LIKE '%'||d.manager_id||'%'
AND e.department_id = d.department_id
GROUP BY d.department_id, e.job_id
HAVING MIN(e.salary) > 5000
ORDER BY MAX(e.salary);
```

В каком порядке выполняются его части?

9. Дан запрос:

```
SELECT
    COUNT(department_id) cnt,
    COUNT(ALL department_id) cnt_all,
    COUNT(UNIQUE department_id) cnt_unique,
    COUNT(DISTINCT department_id) cnt_distinct
FROM employees;
```

Какие из столбцов будут возвращать одинаковые результаты?

4. Подзапросы

Подзапросы предназначены для получения и передачи результата во внешний оператор. Подзапросы используются во фразе FROM, в SELECT-списке, во фразе WHERE, в DML – запросах и др.

Подзапросы разделяют на несколько групп: однострочные/многострочные, одностолбцовые/многостолбцовые, коррелированные/некоррелированные.

Однострочные подзапросы возвращают во внешний оператор ноль или одну строку.

Многострочные подзапросы могут возвращать больше, чем одну строку.

Одостолбцовые подзапросы возвращают только один столбец (однострочные и одностолбцовые подзапросы называют скалярными).

Многостолбцовые подзапросы могут возвращать несколько колонок.

Некореллированый подзапрос не зависит от того контекста, где он исполняется (т.о. его можно исполнить отдельно).

Кореллированный запрос зависит от контекста, т.е. в него передаются параметры из внешнего запроса, глубина передачи параметров в такие запросы — один уровень (за редким исключением с DUAL).

Если подзапрос используется в SELECT-списке и возвращает ноль строк, то значение колонки будет иметь значение NULL.

Подзапрос может быть использован во фразе FROM: SELECT .. FROM .., (<subquery>) alias WHERE ... Здесь результат подзапроса можно использовать как "отдельную таблицу", коррелированные подзапросы тут не допустимы.

Во фразе WHERE подзапросы могут использоваться для ограничения множества следующими способами:

Оператор	Пример	Описание
EXISTS	WHERE EXISTS(SELECT 1 FROM ..)	Проверяет наличие хотя бы одной строки в подзапросе

IN	WHERE (col1, col2) IN (SELECT val1, val2 FROM ..)	Проверяет, что кортеж перед IN (одно- или много-столбцовый) является элементом подзапроса, сравнение идет «через» оператор =, поэтому предикат NULL IN NULL ложен
ANY/ SOME	WHERE salary > ANY(SELECT .. FROM ..)	Проверяет, что выражение верно хотя бы для одного значения из подзапроса, ANY и SOME эквивалентны
ALL	WHERE salary > ALL(SELECT .. FROM ..)	Проверяет верность выражения для каждого значения из подзапроса
Логический	WHERE salary = (SELECT .. FROM ..)	Проверяет верность выражения (только для скалярного подзапроса)

Подзапросы могут быть вложены друг в друга не больше чем на 255 уровней.

Задачи к главе

1. Выведите информацию о сотрудниках с максимальной зарплатой внутри компании.

2. Выведите информацию о сотрудниках с максимальной зарплатой внутри своего отдела.
Указание: решите задачу с использованием коррелированного подзапроса.

3. Выведите информацию о сотрудниках с максимальной зарплатой внутри своего отдела.
Указание: решите задачу с использованием некоррелированного подзапроса во FROM.

4. Выведите информацию о сотрудниках с максимальной зарплатой внутри своего отдела.
Указание: решите задачу с использованием некоррелированного

подзапроса и оператора IN.

5. Рассмотрите запрос:

```
SELECT e.* FROM employees e WHERE e.salary =
    (SELECT MAX(salary) m FROM employees n
    WHERE n.department_id = e.department_id);
```

Перепишите запрос, исключив алиасы там, где это возможно.

6. Выведите информацию о тех сотрудниках, которые являются менеджерами для других сотрудников.

7. Выведите информацию о тех сотрудниках, которые не являются менеджерами для других сотрудников.
Указание: использование конструкции NOT IN.

8. Выведите информацию о тех сотрудниках, зарплаты которых больше, чем средняя зарплата в каждом из отделов.
Указание: использование оператора сравнения с оператором ALL.

9. Выведите информацию о тех сотрудниках, зарплаты которых больше, чем средняя зарплата хотя бы в одном из отделов.
Указание: использование операторов сравнения с операторами SOME/ANY.

10. Выведите тех сотрудников, у которых в трудовой истории (таблица job_history) есть запись о работе в должности ST_CLERK.
Указание: использование оператора EXISTS.

11*. С помощью EXPLAIN PLAN и DBMS_XPLAN покажите, что предикат в запросе

```
SELECT * FROM dual WHERE dummy NOT IN ('X', NULL);
```

преобразуется к виду dummy <> 'X' AND dummy <> NULL.

5. Операторы DML

Операторы DML позволяют добавлять, изменять, удалять данные в таблицах. Такие операторы являются транзакционными, т.о., для того, чтобы все сделанные изменения были зафиксированы,

необходимо выполнить команду COMMIT (или ROLLBACK, чтобы откатить все изменения).

INSERT

Оператор INSERT позволяет добавлять строки в таблицу или представление и имеет несколько вариантов записи, некоторые из них приведены ниже.

Для добавления одной строки в таблицу dml_table_expression_clause используется следующий синтаксис:

```
INSERT INTO dml_table_expression_clause
[ (column [, column ]...) ]
VALUES ({ expr | DEFAULT } [, { expr |
DEFAULT } ]... );
```

column — имена колонок, в которые необходимо произвести вставку (если колонки не указаны, то в VALUES следует указать значения для всех столбцов таблицы в правильном порядке; однако, не рекомендуется полагаться на такое поведение, так как при изменении структуры таблицы запросы становятся некорректными).
expr — выражение, значение которого будет вставлено в соответствующую колонку (вместо значения допустимо использовать ключевое слово DEFAULT — тогда в колонку будет вставлено значение по умолчанию, предусмотренное при создании таблицы).

Возможна вставка всего набора данных с помощью одного оператора:

```
INSERT INTO dml_table_expression_clause
[ (column [, column ]...) ]
SELECT (expr [, expr ]... ) FROM ...;
```

все значения из SELECT будут помещены в соответствующие строки таблицы dml_table_expression_clause.

Для вставки строк в несколько таблиц одновременно возможно использование оператора INSERT со следующим синтаксисом:

```
INSERT ALL insert_into_clause
[ values_clause ]
[ insert_into_clause
    [ values_clause ]
]...
```

subquery

каждая запись, возвращаемая subquery, помещается в каждую insert_into_clause.

Для вставки строк из подзапроса по условию используется следующая конструкция:

```
INSERT [ ALL | FIRST ]
WHEN condition
THEN insert_into_clause
    [ values_clause ]
    [ insert_into_clause
       [ values_clause ]
    ]...
...
[ ELSE insert_into_clause
    [ values_clause ]
    [ insert_into_clause
       [ values_clause ]
    ]...
] subquery;
```

Слова ALL | FIRST определяют, будут ли вставлены все строки, которые проходят условия WHEN, или только первая. В остальном синтаксис аналогичен предыдущему описанию. Если и одно из условий не удовлетворено, можно определить таблицу по умолчанию посредством условия ELSE.

UPDATE

Оператор UPDATE позволяет обновлять строки в таблице или представлении. Его синтаксис следующий:

```
UPDATE table_reference
SET { column_name = {sql_expression | (subquery2)}
|     (column_name     [,     column_name]...)     =
(subquery3)}...
[WHERE search_condition];
```

Оператор позволяет обновить строки таблицы table_reference, удовлетворяющих условию search_condition. Часть SET указывает, какие столбцы чем должны быть обновлены.

sql_expression — любое SQL-выражение (возможно использование текущих столбцов таблицы), subquery2 должна быть

скалярным подзапросом, а subquery3 – однострочным подзапросом.

В качестве table_reference, кроме таблицы, может выступать представление VIEW, созданное как явно, так и в виде подзапроса. Такие представления должны быть "key-preserved table", а обновления разрешены только над одной таблицей. Также возможность обновлений можно посмотреть в системном представлении USER_UPDATABLE_COLUMNS.

DELETE

Оператор DELETE удаляет строки из таблицы или представления. Его синтаксис следующий:

```
DELETE [FROM] table_reference
[WHERE search_condition];
```

Происходит удаление строк из таблицы table_reference, для которых search_condition истинно.

MERGE

Оператор MERGE позволяет обновить один источник данных на основе данных из другого. Его синтаксис следующий:

```
MERGE [ hint ]
   INTO { table | view } [ into_t_alias ]
   USING { table | view | subquery }
                              [ using_t_alias ]
   ON ( condition )
   [ merge_update_clause ]
   [ merge_insert_clause ]
merge_insert_clause::
   WHEN NOT MATCHED THEN
   INSERT [ (column [, column ]...) ]
   VALUES ({ expr [, expr ]... | DEFAULT })
   [ where_clause ]
merge_update_clause::
   WHEN MATCHED THEN
   UPDATE SET column = { expr | DEFAULT }
      [, column = { expr | DEFAULT } ]...
   [ where_clause ]
```

```
[ DELETE where_clause ]
```

Ищет строки в `into_t_alias` из набора `using_t_alias`, используя условие `condition`. Если строки были найдены, то выполняется `merge_update_clause`, если не было найдено, то делается `merge_insert_clause`.
`merge_insert_clause` имеет такой же синтаксис, как и у оператора `INSERT`.
`merge_update_clause` имеет такой же синтаксис, как и у операторов `UPDATE`; дополнительно можно указать условие `where_clause`: если условие будет истинно (только для обновленных строк), то строки будут удалены.

Задачи к главе

1. В таблицу `employees` добавьте следующего сотрудника: Alexey Vasiliev, с электронной почтой: AVASILIEV, на должность IT_PROG, `id` сотрудника — 42, `id` менеджера — 100, принят на работу в 00:00 часов текущей даты.

2. Создайте пустую таблицу `employees2` со структурой, полностью аналогичной таблице `employees` (используйте `CREATE TABLE .. AS SELECT ..`). Вставить в эту таблицу все записи о сотрудниках из таблицы `employees`, работающих в 20 отделе.

3. Для таблицы `employees2` измените поле `salary`, установив значение по умолчанию, равное 1000.
Указание: использование `ALTER TABLE`.

4. Вставьте в таблицу `employees2` одним запросом информацию о следующих сотрудниках: Alexey Vasiliev (id = 42), e-mail: AVASILIEV, должность: IT_PROG, менеджер: 100, зарплата: 1000; Yaroslav Khodakovskiy (id = 43), e-mail: YKHODAKOVSKIY, должность: IT_PROG, менеджер: 100, зарплата имеет значение по умолчанию. Оба сотрудника были приняты на работу в 00 часов текущего дня.
Указание: использование `INSERT ALL`.

5. Создайте три таблицы `employees20`, `employees30` и `employees_other` со структурой, полностью аналогичной таблице `employees`. Далее одним запросом «разложите» все записи из таблицы `employees` по таблицам согласно правилам: в таблицу `employees20` – сотрудников из 20 отдела, в `employees30` – сотрудников из 30 отдела, и в таблицу `employees_other` – всех остальных.
Указание: использование `INSERT FIRST`.

6. Произведите обновление таблицы `employees`, подняв всем сотрудникам из 100 отдела зарплату на 10% и установив комиссионные в размере 0.05.

7. Всем сотрудникам из 30 отдела установите зарплату, равную средней зарплате по этому отделу.

8. Осуществите следующие операции (двумя запросами): обновите колонку `employees.salary` значениями из колонки `employees2.salary`; после этого поместите в таблицу `employees2` отсутствующих там сотрудников из таблицы `employees`.
Указание: использование оператора `MERGE`.

9. Осуществите одним запросом следующие операции: вставьте в таблицу `employees2` недостающие значения из `employees`, обновите имеющиеся значения столбца `employees2.salary` значениями из `employees.salary`, если обновленное значение `employees2.salary` оказалось меньше, чем 3000, то строку необходимо удалить.

10. Удалите из таблицы `employees` сотрудника с `id` = 42.

6. Операторы работы с множествами

Возможно комбинирование нескольких запросов, используя операторы работы со множествами: `UNION`, `UNION ALL`, `INTERSECT`, `MINUS`.

Все операторы имеют одинаковый приоритет, в случае наличия нескольких операторов Oracle обрабатывает их слева направо.

Сравниваемые выражения в select-списке должны иметь одинаковое количество элементов и соответственно одинаковые типы данных (или неявно приводимые). Операторы работы со множествами недействительны для типов данных `BLOB`, `CLOB`, частично с `LONG`.

Общий синтаксис: subquery1 <SET OPERATOR> subquery2.
Оператор `UNION` комбинирует результаты двух запросов с удалением дублирующихся элементов.
Оператор `UNION ALL` объединяет два набора данных (без удаления дубликатов).
Оператор `INTERSECT` возвращает только те элементы, которые присутствуют в обоих запросах.
Оператор `MINUS` возвращает недублирующиеся элементы из первого подзапроса, которые отсутствуют во втором запросе.

Задачи к главе

1. Выведите номера департаментов, в которых не состоит ни один сотрудник.

2. Предположим, что таблица `employees2` имеет аналогичную `employees` структуру. Выведите 'Yes' в колонке с названием `equals`, если таблицы `employees` и `employees2` как множества совпадают, и 'No' – иначе.

3. Предположим, что таблица `employees2` имеет аналогичную `employees` структуру. Выведите тех сотрудников, которые присутствуют одновременно и в таблице `employees`, и в таблице `employees2`.

4. Как применяется оператор ORDER BY при появлении его в конце запроса, в котором имеется оператор работы с множествами, — к итоговому множеству или к ближайшему SELECT?

5. Покажите, что операторы работы с множествами выполняются

последовательно слева направо, если скобки явно не указывают другого порядка.

7. Иерархические запросы

Если выборка содержит иерархические данные, то можно выбирать строки в иерархическом порядке с обходом данных в глубину, используя условие:

```
... [ START WITH condition ]
CONNECT BY [ NOCYCLE ] condition,
```

где

- `START WITH` condition определяет строки, которые будут являться корневыми;
- `CONNECT BY` condition указывает на отношения между родительскими и дочерними строками в иерархии. Одно из выражений в условии может быть квалифицировано оператором `PRIOR` для ссылки на родительскую строку (полученную на предыдущем шаге);
- параметр `NOCYCLE` сигнализирует о том, что строки должны быть возвращены, даже если в выборке присутствуют циклы.

Условия `WHERE` обрабатываются после полного построения выборки посредством `CONNECT BY`, поэтому условия, отсеивающие данные, рекомендуется помещать по возможности в `CONNECT BY`. Например, условие `LEVEL < 3 в CONNECT BY` работает эффективнее, чем в `WHERE`.

Возможные псевдостолбцы (операторы):
`CONNECT_BY_ISLEAF` — равен 1, если выбранная строка является листовым элементом, и 0 – иначе;
`CONNECT_BY_ISCYCLE` — равен 1, если выбранная строка содержит дочерние строки, являющиеся ее прародителем, и 0 – иначе (по сути, столбец равен 1 в строках, которые приводят к петлям);
`LEVEL` — равен 1 для корневой строки, 2 – для ее детей и так далее;
`CONNECT_BY_ROOT` — когда указана колонка с этим оператором, то Oracle возвращает значение колонки, используя для этого данные из корневой строки.
Только в иерархических запросах может быть использована функция

`SYS_CONNECT_BY_PATH(column, char)` — возвращает путь из значений `column` из корневого элемента до текущей строки, используя разделитель `char` для каждой строки, возвращаемой условием `CONNECT BY`.

Указание условий `ORDER BY` или `GROUP BY` может изменить порядок, в котором данные были возвращены. Для сортировки данных внутри каждого родительского элемента (сортировка среди элементов того же уровня под одним родителем) необходимо использовать условие `ORDER SIBLINGS BY`.

Задачи к главе

1. Выведите натуральные числа от 1 до 100.

2. По таблице `employees` выведите сотрудника с `id = 110` и иерархию всех его менеджеров.

3. Выведите директора (`manager_id IS NULL`) и его непосредственных подчиненных (`level <= 2`). Расположите ограничения оптимальным образом.

4. Выведите из иерархии подчиненных сотрудника с `id = 100` тех сотрудников, у которых никого нет в подчинении в формате: `last_name` сотрудника с `id = 100` как `up_manager_last_name`, `last_name` как `last_name`.

5. Предположим, что есть множество, созданное запросом:

```
select 1object_id, 2parent_id from dual union all
select 2object_id, 3parent_id from dual union all
select 3object_id, 1parent_id from dual union all
select 4object_id, 1parent_id from dual;
```

выведите иерархию «вниз» от строки с `object_id = 1`. Отдельным столбцом вывести 1, если строка приводит к циклам в запросе и 0 иначе.
Указание: использование CONNECT_BY_ISCYCLE.

6. Какой алгоритм использует Oracle для обхода деревьев в

иерархических запросах: breadth-first (в ширину) или depth-first (в глубину)? Покажите это на примере.

7. Для сотрудника с `employee_id` = 110 выведите полную строку его подчинения в формате: <`first_name> <last_name`>. В качестве разделителя сотрудников необходимо использовать символ '#'. Пример такой иерархии: `#object#parent#grandparent`.
Указание: использование `SYS_CONNECT_BY_PATH`.

8. Для сотрудника с `employee_id` = 109 выведите информацию о подчинении в следующем виде:

```
EMP_NAME    MANAGERS                 MANAGERS_COUNT
--------    ----------------------   --------------
Faviet      King>Kochhar>Greenberg   3
```

9. Выведите иерархию подчинения в компании, отступая каждый уровень подчинения от предыдущего на четыре пробела. На каждом уровне сотрудники должны быть отсортированы по `last_name`.
Указание: использование сортировки `SIBLINGS BY`.

10*. Опишите, какие части в каком порядке выполняются в запросе

```
SELECT e.job_id, SUM (e.salary)
FROM employees e, departments d
WHERE e.department_id = d.department_id
AND d.location_id = 1700
AND CONNECT_BY_ISLEAF = 1
GROUP BY e.job_id
HAVING COUNT(1) > 1
START WITH e.employee_id = 100
CONNECT BY PRIOR e.employee_id = e.manager_id
ORDER BY e.job_id;
```

11*. Для каждого из чисел от 1 до 15 посчитайте факториал; пример вывода результата: "3! = 6".
Указание: используйте тот факт, что $n_1 \cdot n_2 \cdot \ldots = exp(ln(n_1) + ln(n_2) + ...)$.

8. Аналитические функции

В базе данных имеются встроенные аналитические функции,

позволяющие выполнять трудные вычисления.

Среди этих функций выделяют следующие несколько типов: функции ранжирования, функции для плавающего окна, функции с запаздывающим/опережающим аргументом, статистические функции.

Общий синтаксис записи следующий:

```
analytic_function([ arguments ])
OVER([PARTITION BY {value_expr[, value_expr]... }]
       [ order_by_clause [ windowing_clause ] ])
windowing_clause ::=
{ ROWS | RANGE }
{ BETWEEN
  { UNBOUNDED PRECEDING
  | CURRENT ROW
  | value_expr { PRECEDING | FOLLOWING }
  }
  AND
  { UNBOUNDED FOLLOWING
  | CURRENT ROW
  | value_expr { PRECEDING | FOLLOWING }
  }
| { UNBOUNDED PRECEDING
  | CURRENT ROW
  | value_expr PRECEDING
  }
}
```

windowing_clause определяет окно, внутри которого будет считаться значение функции. Значением по умолчанию является `RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW`. Если опущено слово `BETWEEN`, то Oracle выставляет указанное значение в start point, a end point равно current row.

Групповые функции, выполняемые над окном

`SUM, MIN, MAX, COUNT, AVG` — см. описание в разделе групповых функций.

Некоторые основные оконные функции

`RANK``()` `OVER([ query_partition_clause ] order_by_clause)` или **`RANK`**`(expr [, expr ]...) WITHIN`

GROUP (order_by_clause) — возвращает ранг элементов в группе. Оставляет промежуток в последовательности ранжирования при совпадении рангов нескольких элементов.
DENSE_RANK() OVER([query_partition_clause] order_by_clause) или **DENSE_RANK**(expr [, expr]...) WITHIN GROUP (order_by_clause) – возвращает ранг элементов в группе, не оставляя промежутка в последовательности ранжирования при совпадении рангов нескольких элементов.
ROW_NUMBER() OVER ([query_partition_clause] order_by_clause) — возвращает номер каждой строки в группе.
FIRST_VALUE (expr [IGNORE NULLS]) OVER (analytic_clause) — возвращает значение первой строки окна.
LAST_VALUE (expr [IGNORE NULLS]) OVER (analytic_clause) — возвращает значение последней строки окна.
LAG(value_expr [, offset] [, default]) OVER ([query_partition_clause] order_by_clause) — возвращает значение в строке, отстающей от текущей строки на offset (по умолчанию 1), значение по умолчанию определяется параметром default.
LEAD(value_expr [, offset] [, default]) OVER ([query_partition_clause] order_by_clause) — аналогична функции LAG, но возвращает опережающие строки.

Использование функций FIRST и LAST

aggregate_function KEEP (DENSE_RANK {**FIRST | LAST**} order_by_clause)[OVER query_partition_clause]
используются для получения первого и последнего значения в упорядоченных группах.
aggregate_function — одна из MIN, MAX, SUM, AVG, COUNT, VARIANCE, STDDEV.
ключевое слово KEEP — только для придания смысла: только первое или последнее значение агрегирующей функции будет возвращено.
DENSE_RANK {FIRST | LAST} — говорит о том, что агрегировать нужно только строки, имеющие минимальное/максимальное значение уплотненного ранга (olympic rank).

Аналитические функции могут использоваться совместно со

стандартными функциями.

Задачи к главе

1. Выведите из таблицы `employees` колонки `department_id`, `job_id`, `first_name`, `last_name`, `salary`, а также следующие колонки, подчиняющиеся правилам:
`company_max` – максимальная зарплата внутри всей компании,
`per_job_max` – максимальная зарплата для этой должности,
`raiting_in_dept` – рейтинг сотрудника внутри своего департамента (1 — самая высокая зарплата),
`min_salary_in_dept` – минимальная зарплата внутри своего департамента,
`max_salary_in_dept` – максимальная зарплата внутри своего департамента,
`delta_from_lower` – отличие в зарплате от предыдущего по рейтингу сотрудника.

Выборку упорядочьте по `department_id`, `first_name`, `last_name`, использование подзапросов запрещено.

2. Приведенная ниже таблица состоит из двух столбцов, являющихся выражениями из `select`-списка. Правая колонка — эквивалент левой колонки, записанный в другой форме, который необходимо закончить, дописав запрос на месте, где стоит `/*?*/`.

`ROW_NUMBER() OVER(PARTITION BY job_id ORDER BY salary)`	`COUNT(1) OVER(/*?*/)`
`MAX(salary) OVER()`	`MAX(salary) OVER(ORDER BY /*?*/)`
`COUNT(1) OVER(ORDER BY 1)`	`COUNT(1) OVER(ORDER /*?*/ AND CURRENT ROW)`
`RANK() OVER(ORDER BY department_id, job_id)`	`COUNT(1) OVER(/*?*/)`
`LEAD(salary) OVER(PARTITION BY job_id ORDER BY salary)`	`SUM(salary) OVER(/*?*/)`

`MIN(salary) OVER(ORDER BY salary RANGE UNBOUNDED PRECEDING)`	`MIN(salary) OVER(ORDER BY salary RANGE BETWEEN /*?*/)`
`SUM(salary) OVER(ORDER BY salary ROWS UNBOUNDED FOLLOWING)`	`SUM(salary) OVER(ORDER BY salary ROWS BETWEEN /*?*/)`

3. Согласно таблице `employees` для каждого отдела выведите количество сотрудников, получающих зарплату не меньшую, чем 3000. Считать, что зарплата — целое число, использование подзапросов запрещено.
Указание: использование функции `RANK (..) WITHIN GROUP`.

4. Для каждого департамента на основе таблицы `employees` выведите: минимальную зарплату среди сотрудников, имеющих наименьшие комиссионные, колонкой `worst`, максимальную зарплату среди сотрудников, получающих максимальные комиссионные, колонкой `best`.
Указание: использование функций `FIRST`, `LAST`.

5*. Для каждого сотрудника на основе таблицы `employees` выведите: минимальную зарплату среди сотрудников, имеющих наименьшие комиссионные внутри своего отдела, колонкой `worst_in_dept`, максимальную зарплату среди сотрудников, получающих максимальные комиссионные для своей должности, колонкой `best_for_job`.

6. По таблице `employees` выведите отчетность по суммарной зарплате для следующих групп: сначала для департамента, потом по должности, потом по всей компании, а также отдельную колонку, отражающую рейтинг суммы внутри всей выборки (1 — самая высокая сумма).
Указание: использование функции `RANK`.

9. Транзакции

Транзакцией базы данных называется один или группа

операторов SQL, образующих логическую единицу работы, которая может быть зафиксирована или отменена как единое целое. Фиксация происходит при выполнении оператора `COMMIT [WORK]`, а откат – при выполнении оператора `ROLLBACK [WORK]`. Транзакция начинается при:

- выполнении любого оператора DML.

Транзакция заканчивается при:

- выполнении оператора `COMMIT` или `ROLLBACK`,
- при выполнении оператора DDL (автоматически порождает фиксацию),
- при выполнении оператора DCL (автоматически порождает фиксацию),
- при выходе из SQL*Plus (фиксация при exit и откат при аварийном отключении).

Контрольные точки (`SAVEPOINT`) позволяют откатывать изменения именно до этого места. Откат до контрольной точки (`ROLLBACK TO SAVEPOINT <point_name>`) не заканчивает транзакцию.

Oracle поддерживает уровни изоляций транзакций `READ COMMITTED` и `SERIALIZABLE`. Изменить уровень изоляции транзакции можно командой:

```
SET TRANSACTION ISOLATION LEVEL {SERIALIZABLE |
READ COMMITTED}.
```

Задачи к главе

1. Для чего используется команда `SET TRANSACTION READ ONLY`?

2. Пользователь ввел для фиксации транзакции команду `COMMIT WORK WRITE IMMEDIATE WAIT`. Как можно записать эту команду более просто?

3. Какой уровень изоляции транзакций используется Oracle по умолчанию? Как установить/восстановить этот уровень?

4. Посмотрите временно*е* развитие двух параллельно работающих пользователей. Какая команда была введена вторым пользователем,

что получилась сложившаяся ситуация (транзакция не была помечена как READ ONLY).

```
---------------------------+----------------------
         Session 1         |       Session 2
---------------------------+----------------------
SQL> create table a as     |
  2  select 1 id from dual;|
Table created.             |
                           | SQL> ?????
SQL> delete a;             |
1 row deleted.             |
SQL> commit;               |
Commit completed.          |
                           | SQL> select * from a;
                           |          ID
                           |  ----------
                           |           1
                           | SQL> commit;
                           |  Commit completed.
                           | SQL> select * from a;
                           |  no rows selected
```

5*. Покажите, что `ROLLBACK TO SAVEPOINT` не заканчивает текущую транзакцию, даже если все изменения откачены.

6. Заполните две команды в сессии в правой колонке для того, чтобы получить тот же самый результат, как и посредством команд из левой колонки.

```
> update alva              |> update alva
  set id=30 where id=3;    |  set id=30 where id=3;
                           |
                           |> /* ???? */
                           |
> delete alva;             |> delete alva;
                           |
> rollback;                |> /* ???? */
                           |
> update alva              |
  set id=30 where id=3;    |
                           |
```

```
> delete alva where id=4;|>delete alva where id=4;
                         |
> commit;                |>commit;
```

7. Покажите, что даже неудачно завершившийся DDL-оператор завершает текущую транзакцию, т.е. что поведение DDL-оператора можно описать в виде

```
BEGIN
    COMMIT;
    do the ddl;
    COMMIT;
EXCEPTION
    WHEN OTHERS THEN
        ROLLBACK; RAISE;
END;
```

8*. Как известно, транзакцию фиксируют только те DDL-команды, которые затрагивают изменения словаря данных. Приведите пример DDL-команды, не фиксирующей изменения текущей транзакции, т.е. не затрагивающей словарь данных.

10. Создание таблиц

Таблицы базы данных (или отношения) — это объекты, содержащие столбцы с данными. По умолчанию таблицы создаются в текущей схеме. Существует несколько типов таблиц, общий синтаксис которых таков (полный синтаксис доступен на сайте Oracle):

```
CREATE TABLE [ schema.] table
[ (column_definition, ...) ]
[TABLESPACE tablespace]
column_definition::=
column datatype [ DEFAULT expr ]
[ ( { inline_constraint }... ) |
inline_ref_constraint ]
```

Информацию об имеющихся таблицах можно найти в системном представлении `USER_TABLES`.

Переименовать таблицу можно с помощью оператора `RENAME`

TABLE.

Комментарии к таблице и столбцам создаются операторами COMMENT ON TABLE, COMMENT ON COLUMN.

Основные типы данных

```
character_datatypes::=
{ CHAR [ (size) ] | VARCHAR2 (size)...}
number_datatypes::=
{ NUMBER [ (precision [, scale ]) ]...}
datetime_datatypes::=
{            DATE            |            TIMESTAMP
[    (fractional_seconds_precision)    ]    [    WITH
[ LOCAL ] TIME ZONE ] ...}
long_and_raw_datatypes::=
{ LONG | RAW (size) ... }
large_object_datatypes::=
{ BLOB | CLOB ... }
rowid_datatypes::=
{ ROWID ...}
```

Также поддерживаются типы данных ANSI.

VARCHAR2(size)	Хранит строки переменной длины, максимальный размер до 4000 символов.
CHAR(size)	Хранит строки фиксированной длины. Максимальный размер 2000 символов. Недостающие символы дополняются пробелами справа.
NUMBER(p,s)	Числа, имеющие точность *p* и масштаб *s*. Значения для *p* от 1 до 38. Значения для мантиссы – от 84 до 127.
LONG	Большие текстовые массивы переменной длины размером до 2 Gb.
DATE	Позволяет хранить дату и время от January 1, 4712 BC до December 31, 9999 AD.
TIMESTAMP (fractional_seconds_precision)	Позволяет хранить дату и время, дополнительно сотые доли секунд.

TIMESTAMP WITH TIME ZONE	Аналогично предыдущему, дополнительно хранится часовой пояс.
TIMESTAMP WITH LOCAL TIME ZONE	Аналогично предыдущему, с тем исключением, что дата нормализуется к зоне, установленной в базе данных.
RAW(size)	Бинарный массив размера size байт. Максимальный размер 2000 байт.
ROWID	Base-64 – представление уникального адреса строки в таблице. Возвращается в псевдостолбце ROWID.
CLOB	Большие текстовые массивы переменной длины размером до ~ 4Gb * (database block size). Отличие от LONG в физическом устройстве хранения.
BLOB	Аналогично CLOB, хранятся бинарные объекты.

Возможны различные действия над колонками таблицы: добавление, изменение, удаление командами:
ALTER TABLE: ADD COLUMN, MODIFY COLUMN, DROP COLUMN.

Информацию об имеющихся столбцах таблиц можно найти в системном представлении USER_TAB_COLUMNS.

Разрядность и длина столбцов может быть уменьшена, только если в столбце содержатся NULL-значения или строки отсутствуют.

Существуют различные типы таблиц как по структуре, так и по способу хранения информации, например GLOBAL TEMPORARY, ORGANIZATION INDEX, ORGANIZATION EXTERNAL.

Для изменения таблицы необходимо использовать оператор ALTER TABLE:

```
ALTER TABLE table
[ ADD ( column_definition, ...) | MODIFY
( modify_col_properties, ...) | DROP
(column, ...) | constraint_clauses ]
```

Полный синтаксис оператора доступен на сайте Oracle.
Оператор ANALYZE TABLE используется для сбора статистики по таблице.

Список табличных ограничений

CHECK (C) указывает, что значение столбца или группы столбцов должно удовлетворять определенному условию.

NO NULL (C) указывает, что в этом столбце нельзя хранить NULL-значения.

PRIMARY KEY (P) определяет первичный ключ таблицы, который состоит из одного или нескольких столбцов.

FOREIGN KEY (R) определяет для таблицы внешний ключ.

UNIQUE (U) определяет столбец или группу столбцов, где могут храниться только уникальные значения.

Задачи к главе

1. Создайте таблицу `users` для хранения данных о пользователях:
`employee_id` — числового типа,
`first_name` — строка переменной длины, до 200 символов,
`post_index` — всегда будет храниться 6 символов,
`hire_date` — для хранения даты рождения пользователя,
`cv` — текстовое представление его резюме,
`photo` — для хранения его фотографии.

2. Измените таблицу, созданную в предыдущей задаче, увеличив размер `first_name` до 300, удалив колонку `post_index`, добавив новую колонку `phone_number` — точно 10 символов.

3. Создайте комментарий к таблице `users` «Contains users' information» и к столбцу `photo` «Use .jpg format.».

4. Для созданной таблицы `users` выведите названия колонок и их тип в порядке, как они расположены в таблице.

5*. Для сегментов (таблиц в частности) вашей схемы выведите информацию по объему занимаемой ими пространства, отделяя каждые три разряда запятой.

6. Таблицы `t1` и `t2` были созданы следующими запросами:

```
create table t1 (id1 int primary key);
create table t2 (id1 int references t1);
```

Необходимо удалить таблицу `t1` одним запросом (со всеми внешними ключами). Какой запрос позволит это сделать?

7. К каким колонкам применимы, а к каким не применимы SET-операторы (UNION, MINUS,...)?

8*. Покажите, что по умолчанию `rowid` у строки в таблице не меняется при ее обновлении (если в блоке, где хранится строка, для нее не хватает места, то вместо строки хранится указатель на действительное местоположение — это так называемые сцепленные строки).

9*. Как оценить количество строк и количество сцепленных строк в таблице? Какие команды используются для точной и приблизительной оценки?

10*. Приведите два способа, как избавиться от большого количества сцепленных строк в таблице. Покажите, что они работают.
Указание: используйте `MOVE` и `SHRINK SPACE`.

11. Необходимо создать таблицу `t1` с колонкой `id`, являющейся первичным ключом. Какими двумя способами (одной командой) можно это сделать? Покажите, какие столбцы имеют ограничения, какие имена они имеют?

12. Создайте таблицу `users` со следующими полями:

`object_id int` — не допускает `null`-значений,
`parent_id int`,
`nickname varchar2(20)`,
`name varchar2(200)` — по умолчанию `'New user'`

и ограничениями со следующими именами:

`pk` — `object_id` – является первичным ключом,
`uk` — `nickname` – уникален в рамках таблицы,
`chk` — `name` – не пустое и не состоит из пробелов,
`pi_fk` — внешний ключ на таблицу `users`; при удалении должна удаляться вся иерархия объектов (object–parent).

Литература

1. *Кайт Том.* Oracle для профессионалов. – М.: Вильямс, 2011. – 848 с.
2. *Прайс Джейсон.* SQL для Oracle 10g. – М.: Лори, 2007. – 566 с.
3. Oracle Documentation: www.oracle.com/pls/xe102/homepage.
4. *Швинн Каучмэн.* Oracle Certified Professional. Подготовка администраторов баз данных. – М.: Лори, 2009. – 868 с.

Учебное издание

ORACLE SQL
В ЗАДАЧАХ

Учебно-методическое пособие

Составители: *А.В. Васильев*
Я.Ф. Ходаковский

Редактор *Л.В. Себова*. Корректор *О.П. Котова*

Подписано в печать 10.03.2009. Формат 60 × 84 $^{1}/_{16}$. Печать офсетная. Усл. печ. л. 1,75. Уч.-изд. л. 1,5. Тираж 100 экз.
Заказ № ф-026.

Государственное образовательное учреждение
высшего профессионального образования
«Московский физико-технический институт (государственный университет)»
141700, Московская обл., г. Долгопрудный, Институтский пер., 9

Отдел автоматизированных издательских систем "ФИЗТЕХ-ПОЛИГРАФ"
141700, Московская обл., г. Долгопрудный, Институтский пер., 9