История реляционной модели

- Плоские файлы
- Модели данных:
  - иерархическая
  - сетевая
- Реляционная модель данных

Реляционная модель данных

- Явное представление данных
- Гарантированный доступ к данным
- Полная обработка неопределённых значений
- Доступ к описанию БД в терминах реляционной модели
- Полнота подмножества языка
- Возможность обновления представлений
- Наличие высокоуровневых операций управления данными
- Физическая независимость данных
- Логическая независимость данных
- Независимость контроля целостности
- Дистрибутивная независимость
- Согласование языковых уровней

Свойства реляционной БД

- Базовые порции данных представляют собой отношения (relations) или таблицы
- Операции над таблицами затрагивают только отношения

Терминология реляционной БД

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7369	SMITH	CLERK	7902	17-DEC-80	800		20
7499	ALLEN	SALESMAN	7698	20-FEB-81	1600	300	30
7521	WARD	SALESMAN	7698	22-FEB-81	1250	500	30
7566	JONES	MANAGER	7839	02-APR-81	2975		20
7654	MARTIN	SALESMAN	7698	28-SEP-81	1250	1400	30
7698	BLAKE	MANAGER	7839	01-MAY-81	2850		30
7782	CLARK	MANAGER	7839	09-JUN-81	2450		10
7839	KING	PRESIDENT		17-NOV-81	5000		10
7844	TURNER	SALESMAN	7698	08-SEP-81	1500	0	30
7900	JAMES	CLERK	7698	03-DEC-81	950		30
7902	FORD	ANALYST	7566	03-DEC-81	3000		20
7876	ADAMS	CLERK	7788	12-JAN-83	1100		20
7934	MILLER	CLERK	7782	23-JAN-82	1300		10
7788	SCOTT	ANALYST	7566	09-DEC-82	3000		20

Требования к таблице

- Данные в ячейках структурно неделимы
- Данные в столбце одного типа
- Столбцы размещаются в произвольном порядке
- Строки размещаются в произвольном порядке
- Столбцы имеют уникальные имена
- Каждая строка должна быть уникальной

Общие понятия

- **Первичный ключ** - столбец или подмножество столбцов, которые уникально, т.е. единственным образом, определяют строки.

  Первичный ключ, который содержит более одного столбца, называется множественным, комбинированным или составным.

- **Внешний ключ** - столбец (или подмножество столбцов одной таблицы), который может служить в качестве первичного ключа для другой таблицы. Используется для обеспечения отношений между таблицами.

Правило целостности обьектов

- ✔ Первичный ключ не может быть полностью или частично пустым.

Правило ссылочной целостности

- ✔ Внешний ключ может быть пустым или соответствовать значению первичного ключа.

Oracle: Система управления объектно-реляционными базами данных

- ✔ Типы данных и объекты, заданные пользователем.
- ✔ Полная совместимость с реляционными базами данных.
- ✔ Поддержка мультимедийных и больших объектов.
- ✔ Высококачественное оснащение сервера баз данных.

Команды SQL

- SELECT: выборка данных
- INSERT, UPDATE, DELETE: язык манипулирования данными (DML)
- CREATE, ALTER, DROP, RENAME, TRUNCATE: язык определения данных (DDL)
- COMMIT, ROLLBACK, SAVEPOINT: управление транзакциями
- GRANT, REVOKE: язык управления данными (DCL)

Идентификаторы записей ROWID

- Каждой строке таблицы в БД ORACLE назначается уникальный ROWID.

Пользовательские объекты

- Файлы данных
- Расширения
- Табличные пространства
- Сегменты отката
- Временные сегменты
- Таблицы
- Индексы
- Словарь данных
- Обзоры
- Последовательности
- Синонимы
- Триггеры
- Связи БД
- Пакеты, процедуры и функции

Написание команд SQL

- Команды SQL не различают регистры символов.
- Команды SQL могут занимать одну или несколько строк.
- Ключевые слова нельзя сокращать и размещать на двух строках.
- Предложения обычно пишутся на отдельных строках.
- Для облегчения чтения используются табуляция и отступы.

Выборка данных – оператор SELECT

- Выбор информации из БД (*selection*)
- Проекция (*projection*)
- Соединение (*join*)

Основной синтаксис SELECT

SELECT *column1,* column2,…., columnN
FROM *table*

Основной синтаксис SELECT

```
SQL> SELECT * FROM dept;

    DEPTNO DNAME          LOC
---------- -------------- -------------
        10 ACCOUNTING     NEW YORK
        20 RESEARCH       DALLAS
        30 SALES          CHICAGO
        40 OPERATIONS     BOSTON
```

Основной синтаксис SELECT

```
SQL> SELECT deptno, loc FROM
  dept;

    DEPTNO LOC
---------- -------------
        10 NEW YORK
        20 DALLAS
        30 CHICAGO
        40 BOSTON
```

Арифметические выражения

Для типов данных *NUMBER* и *DATE* можно использовать арифметические выражения:

Оператор	Описание
+	Сложение
-	Вычитание
*	Умножение
/	Деление

Использование арифм. операторов

```
SQL> SELECT ename, sal, sal+300
  FROM emp;
```

ENAME	SAL	SAL+300
SMITH	800	1100
ALLEN	1600	1900
WARD	1250	1550
JONES	2975	3275
MARTIN	1250	1550

…

Приоритеты операторов

* / + –

- Умножение и деление имеют приоритет над сложением и вычитанием.
- Операторы с одинаковым приоритетом выполняются слева направо.
- Для выполнения операторов в определенном порядке и упрощения их чтения используются скобки.

Приоритеты операторов

```
SQL> SELECT ename, sal, 12*sal+100
  FROM emp;
```

ENAME	SAL	12*SAL+100
SMITH	800	9700
ALLEN	1600	19300
WARD	1250	15100
JONES	2975	35800
MARTIN	1250	15100
BLAKE	2850	34300

…

Приоритеты операторов

```
SQL> SELECT ename, sal, 12*(sal+100)
  FROM emp;
```

ENAME	SAL	12*(SAL+100)
SMITH	800	10800
ALLEN	1600	20400
WARD	1250	16200
JONES	2975	36900
MARTIN	1250	16200
BLAKE	2850	35400

…

Неопределенное значение NULL

- Неопределенное значение (NULL) – значение, которое недоступно, не присвоено, неизвестно или неприменимо.
- Это не ноль и не пробел.

```
SQL> SELECT ename, job, comm FROM emp;

ENAME      JOB          COMM
---------- --------- ---------
SMITH      CLERK
ALLEN      SALESMAN       300
WARD       SALESMAN       500
JONES      MANAGER
MARTIN     SALESMAN      1400
…
```

NULL в арифметических выражениях

Результат вычисления выражения, содержащего неопределенное значение, также будет неопределенным

```
SQL> SELECT ename 12*sal+comm
  FROM emp;
```

```
ENAME      12*SAL+COMM
---------- -----------
SMITH
ALLEN            19500
WARD             15500
JONES
MARTIN           16400
BLAKE
...
```

Псевдоним (алиас) столбца

- Альтернативный заголовок столбца.
- Удобен при вычислениях.
- Следует сразу за именем столбца; ключевое слово AS между именем столбца и псевдонимом необязательно.
- Заключается в двойные кавычки, если содержит пробелы, спец. символы или различает регистры символов.

Псевдоним (алиас) столбца

```
SQL>  SELECT ename AS name, sal salary
  FROM emp;

NAME          SALARY
---------- ---------

SQL>  SELECT ename 'Name', sal*12
  'Annual Salary'                FROM
  emp;

Name       Annual Salary
---------- -------------
```

Оператор конкатенации

- Соединяет столбцы или символьные строки с другими столбцами.
- Изображается двумя вертикальными линиями (||).
- Создает столбец с результатом, представляющий символьное выражение.

Использование оператора конкатенации

```
SQL> SELECT ename || job AS
  'Employees'  FROM emp;


Employees
--------------------
SMITHCLERK
ALLENSALESMAN
WARDSALESMAN
JONESMANAGER
MARTINSALESMAN
BLAKEMANAGER

...
```

Литералы

- Литерал – символ, выражение или число, включенные в SELECT список.
- Даты и символьные литералы должны быть заключены в апострофы.
- Каждая символьная строка выводится один раз для каждой возвращаемой строки таблицы.

Использование литералов

```
SQL>  SELECT ename ||' '||'is a'||'
  '|| job      AS "Employee details"
  FROM emp;
Employee details
------------------------------
SMITH is a CLERK
ALLEN is a SALESMAN
WARD is a SALESMAN
JONES is a MANAGER
MARTIN is a SALESMAN
BLAKE is a MANAGER
…
```

Дублирование строк

По умолчанию выдаются все строки, включая дубликаты.

```
SQL> SELECT deptno FROM emp;

 DEPTNO
---------
        10
        30
        10
        20
...
```

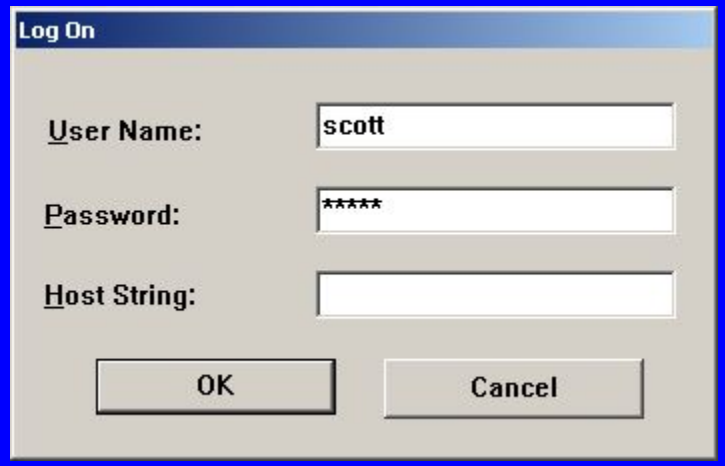
Дублирование строк

Дубликаты устраняются при помощи ключевого слова DISTINCT в команде SELECT.

```
SQL> SELECT DISTINCT deptno
  FROM emp;

 DEPTNO
---------
        10
        20
        30
```

Соединение с базой данных



Соединение с БД из командной строки:

sqlplus [username [/password [@database]]]

Отображение структуры таблицы в SQL*Plus

SQL> DESCRIBE *tablename*

```
SQL> DESCRIBE dept

 Name                      Null?    Type
 ------------------------- -------- -------------
 DEPTNO                    NOT NULL NUMBER(2)
 DNAME                              VARCHAR2(14)
 LOC                                VARCHAR2(13)
```

Редактирование SQL команд и PL/SQL блоков

- A[PPEND] text
- C[HANGE] / old / new /
- C[HANGE] / text /
- CL[EAR] BUFF[ER]
- DEL
- DEL n
- DEL m n

Редактирование SQL команд и PL/SQL блоков

- **I[NPUT]**
- **I[NPUT] text**
- **L[IST]**
- **L[IST] n**
- **L[IST] m n**
- **R[UN]**
- **n**
- **n text**
- **0 text**

Команды SQL*Plus для работы с файлами

- SAV[E] filename [CREATE | REPLACE | APPEND]
- GET filename
- STA[RT] filename
- @filename
- ED[IT] filename
- SPO[OL] [filename | OFF|OUT]
- EXIT

Ограничение колич. выбираемых строк

SELECT ***column1,*** **column2,…., columnN**
FROM ***table***
WHERE condition (s)

- **Количество возвращаемых строк можно ограничить с помощью предложения WHERE.**
- **Предложение WHERE следует за предложением FROM.**

Использование предложения WHERE

```
SQL> SELECT ename, job, deptno
     FROM emp
     WHERE job = 'CLERK';
```

ENAME	JOB	DEPTNO
SMITH	CLERK	20
JAMES	CLERK	30
MILLER	CLERK	10
ADAMS	CLERK	20

Операторы сравнения

Оператор	Значение
=	Равно
>	Больше, чем
>=	Больше или равно
<	Меньше, чем
<=	Меньше или равно
<>	Не равно

Использование операторов сравнения

```
SQL> SELECT ename, sal, comm
     FROM emp
     WHERE sal <=comm;
```

ENAME	SAL	COMM
MARTIN	1250	1400

Другие операторы сравнения

Оператор	Значение
BETWEEN..AND ..	Находится в диапазоне от одного значения до другого (включительно)
IN (list)	Совпадает с каким-либо значением из списка
LIKE	Соответствует символьному шаблону
IS NULL	Является неопределённым значением

Использование оператора BETWEEN

```
SQL> SELECT ename, sal FROM emp
    WHERE sal BETWEEN 1000 AND
  1500;
```

```
ENAME                 SAL
---------- ----------
WARD                 1250
MARTIN               1250
TURNER               1500
MILLER               1300
ADAMS          1100
```

Использование оператора IN

```
SQL> SELECT empno, ename, sal, mgr FROM
   emp
      WHERE mgr IN (7902, 7566, 7788);
```

EMPNO	ENAME	SAL	MGR
7369	SMITH	800	7902
7902	FORD	3000	7566
7876	ADAMS	1100	7788
7788	SCOTT	3000	7566

Использование оператора LIKE

- Оператор LIKE используется для поиска символьных значений по шаблону с метасимволами
- Условия поиска могут включать алфавитные и цифровые символы

```
SELECT ename
FROM   emp
WHERE  ename LIKE 'S%';
```

Использование оператора LIKE

- Метасимволы можно комбинировать

```
SQL> SELECT ename
     FROM   emp
     WHERE  ename LIKE '_A%';

ENAME
----------
WARD
MARTIN
JAMES
```

Использование оператора IS NULL

С помощью оператора IS NULL производится проверка на неопределенные значения

```
SQL> SELECT ename, mgr
       FROM emp
         WHERE mgr IS NULL;

ENAME                MGR
---------- ----------
KING
```

Логические операторы

Оператор	Значение
AND	Возвращает результат ИСТИННО, если выполняются оба условия
OR	Возвращает результат ИСТИННО, если выполняется любое из условий
NOT	Возвращает результат ИСТИННО, если следующее условие не выполняется

Использование оператора AND

Оператор AND (‘И’) требует выполнения обоих условий.

```
SQL> SELECT empno, ename, job, sal
     FROM emp
     WHERE sal>=1100 AND job='CLERK';

     EMPNO ENAME      JOB              SAL
---------- ---------- --------- ----------
      7876 ADAMS      CLERK            1100
      7934 MILLER     CLERK            1300
```

Использование оператора OR

Оператор OR (‘ИЛИ’) требует выполнения любого из условий.

```
SQL> SELECT empno, ename, job, sal
     FROM emp
     WHERE sal>=1100 OR job='CLERK';
```

EMPNO	ENAME	JOB	SAL
7369	SMITH	CLERK	800
7499	ALLEN	SALESMAN	1600
7521	WARD	SALESMAN	1250

…

Использование оператора NOT

Оператор NOT - логическое отрицание условия.

```
SQL> SELECT ename, job
     FROM emp
     WHERE job NOT IN
   ('CLERK','MANAGER','ANALYST');
ENAME      JOB
---------- ---------
MARTIN     SALESMAN
KING       PRESIDENT
TURNER     SALESMAN
...
```

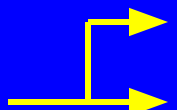
Приоритеты операторов

Порядок вычисления	Оператор
1	Все операторы сравнения
2	NOT
3	AND
4	OR

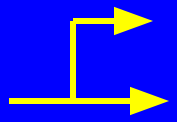
Изменить стандартную последовательность вычислений можно при помощи круглых скобок.

Приоритеты операторов

```
SQL> SELECT    ename, job, sal
     FROM      emp
     WHERE     job='SALESMAN'
      OR        job = 'PRESIDENT'
     AND       sal>1500;
```



```
SQL> SELECT    ename, job, sal
     FROM      emp
     WHERE     (job='SALESMAN'
      OR        job = 'PRESIDENT')
     AND       sal>1500;
```



Сортировка данных

- Предложение ORDER BY используется для сортировки строк:

  ASC: сортировка по возрастанию (по умолчанию)

  DESC: сортировка по убыванию

- В команде SELECT предложение ORDER BY указывается последним.
- Возможна сортировка по псевдониму столбца
- Возможна сортировка по нескольким столбцам, в т. ч. можно сортировать по столбцу, не входящему в SELECT список.

Сортировка данных

```
SQL> SELECT ename, job, deptno, hiredate
     FROM emp ORDER BY hiredate;
```

ENAME	JOB	DEPTNO	HIREDATE
SMITH	CLERK	20	17-DEC-80
ALLEN	SALESMAN	30	20-FEB-81
WARD	SALESMAN	30	22-FEB-81
JONES	MANAGER	20	02-APR-81

…

Сортировка данных

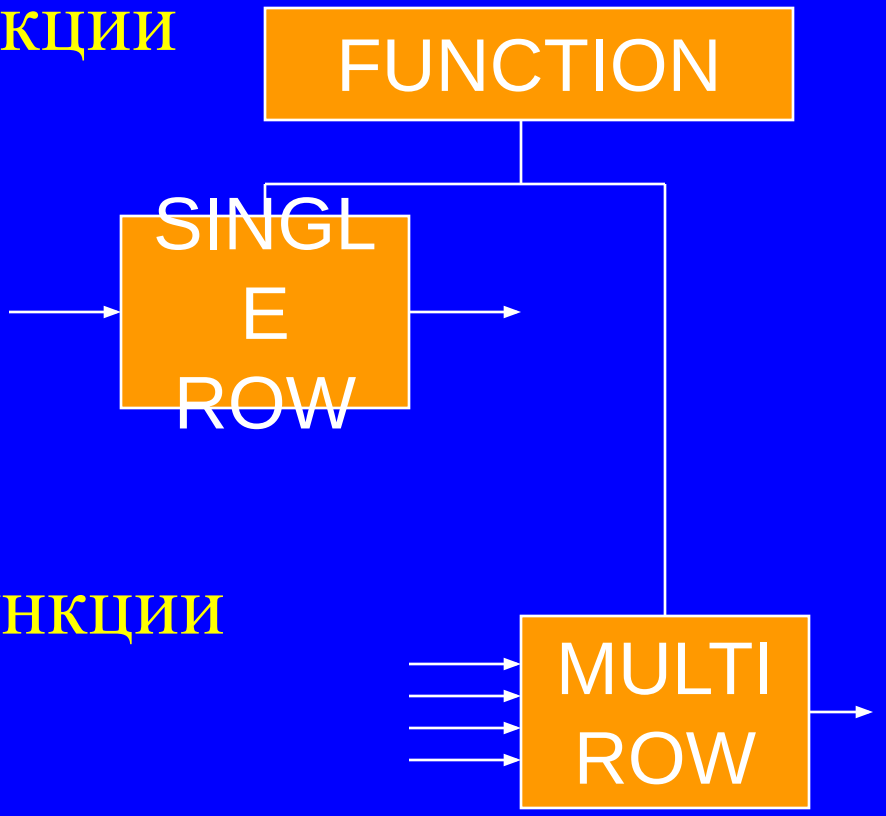
```
SQL> SELECT ename, deptno, sal
     FROM emp ORDER BY deptno ASC, sal
   DESC;
```

ENAME	DEPTNO	SAL
KING	10	5000
CLARK	10	2450
MILLER	10	1300
FORD	20	3000

…

2 типа SQL функций

- Однострочные функции
  - символьные
  - числовые
  - над датами
  - преобразования
- Многострочные функции
  - групповые



Однострочные функции

- Обрабатывают каждую полученную по запросу строку.
- Выдают результат отдельно для каждой строки.
- Могут иметь вложенность.
- Могут задаваться везде, где допускаются переменные, имена столбцов или выражения.
- Могут изменять тип данных.

Однострочные функции

FUNCTION_NAME (column | expression,
[arg1, arg2, ...])

Символьные функции

LOWER

UPPER

INITCAP

CONCAT

SUBSTR

LENGTH

INSTR

LPAD

TRIM

Функции преобразования регистра

- **LOWER :** Переводит все символы строки на нижний регистр
- **UPPER :** Переводит все символы строки на верхний регистр
- **INITCAP :** Делает первую букву всех слов строки прописной, остальные буквы - строчными

Функции преобразования регистра

Преобразование регистра для символьных строк.

Функция	Результат
LOWER (‘SQL Course’)	sql course
UPPER (‘SQL Course’)	SQL COURSE
INITCAP (‘SQL course’)	Sql Course

Использование функций преобразования

Вывод номера служащего, фамилии и номера отдела для служащего по фамилии Blake

```
SQL> SELECT empno, ename, deptno FROM emp
  WHERE
    ename = 'blake';
      no rows selected
SQL> SELECT empno, ename, deptno FROM emp
  WHERE
    LOWER(ename) = 'blake';
     EMPNO  ENAME                DEPTNO
  ---------- ---------- ----------
       7698 BLAKE                30
```

Функции манипулирования символами

Манипулирование символьными строками.

Функция	Результат
CONCAT (‘Good’,’String’)	GoodString
SUBSTR(‘String’,1,3)	Str
LENGTH(‘String’)	6
INSTR(‘String’,’r’)	3
LPAD(sal,10,’*’)	******5000

Использование символьных функций

```
SQL> SELECT ename, CONCAT(ename, job),
     LENGTH(ename),INSTR(ename, 'A')
      FROM emp
     WHERE SUBSTR(job,1,5) = 'SALES';
```

ENAME	CONCAT(ENAME,JOB)	LENGTH(ENAME)	INSTR(ENAME,'A')
ALLEN	ALLENSALESMAN	5	1
WARD	WARDSALESMAN	4	2
MARTIN	MARTINSALESMAN	6	2
TURNER	TURNERSALESMAN	6	0

Числовые функции

- ROUND : Округляет числовое значение до заданной точности.

  ROUND(45.926, 2) → 45.93

- TRUNC : Усекает значение до заданного количества десятичных знаков.

  TRUNC(45.926, 2) → 45.92

- MOD : Возвращает остаток от деления первого на второе.

  MOD(1600,300) → 100

Использование функции ROUND

```
SQL> SELECT
  ROUND(45.923,2),ROUND(45.923,0),
  ROUND(45.923,-1)
    FROM DUAL;

ROUND(45.923,2) ROUND(45.923,0) ROUND(45.923,-1)
--------------- --------------- ---------------
          45.92              46              50
```

Использование функции TRUNC

```
SQL> SELECT
  TRUNC(45.923,2),TRUNC(45.923),
  TRUNC(45.923,-1)
    FROM DUAL;

TRUNC(45.923,2) TRUNC(45.923) TRUNC(45.923,-1)
--------------- ------------- ----------------
          45.92            45               40
```

Использование функции MOD

```
SQL> SELECT ename, sal, comm,
  MOD(sal, comm)
    FROM EMP
    WHERE job='SALESMAN';
```

ENAME	SAL	COMM	MOD(SAL,COMM)
ALLEN	1600	300	100
WARD	1250	500	250
MARTIN	1250	1400	1250
TURNER	1500	0	1500

Работа с датами

- Oracle хранит данные во внутреннем цифровом формате.

  век, год, месяц, число, часы, минуты, секунды

- По умолчанию дата выдаётся в формате DD-MON-YY.
- Функция SYSDATE возвращает текущие дату и время.
- DUAL – фиктивная таблица, используемая для просмотра SYSDATE.

Просмотр текущей даты

```
SQL> SELECT SYSDATE FROM DUAL;

SYSDATE
---------
01-MAY-02
```

Арифметические операции с датами

- Результатом прибавления числа к дате и вычитания числа из даты является дата.
- Результатом вычитания одной даты из другой является количество дней, разделяющих эти даты.
- Прибавление часов к дате производится путем деления количества часов на 24.

Использование операций с датами

```
SQL> SELECT ename, (SYSDATE-hiredate)/7
  WEEKS
   FROM EMP
   WHERE deptno=10;
```

```
ENAME                 WEEKS
---------- -----------
CLARK        1090.26313
KING         1067.26313
MILLER        1057.6917
```

Функции для работы с датами

Функция	Описание
MONTHS_BETWEEN	Число месяцев, разделяющих 2 даты
ADD_MONTHS	Добавление календарных месяцев к дате
NEXT_DAY	Ближайшая дата, когда наступит заданный день недели
LAST_DAY	Последняя дата текущего месяца
ROUND	Округление до целых суток
TRUNC	Отсечение части даты, обозначающей время

Функции для работы с датами

- MONTHS_BETWEEN(’01-SEP-95’,’11-JAN-94’) 19.6774194
- ADD_MONTHS (’11-JAN-94’,6) ’11-JUL-94’
- NEXT_DAY (’01-SEP-95’,’FRIDAY’) ’08-SEP-95’
- LAST_DAY (’01-SEP-95’) ’30-SEP-95’
- ROUND (’25-JUL-95’,’MONTH’) ’01-AUG-95’
- ROUND (’25-JUL-95’,’YEAR’) ’01-JAN-96’
- TRUNC (’25-JUL-95’,’MONTH’) ’01-JUL-95’
- TRUNC (’25-JUL-95’,’YEAR’) ’01-JAN-95’

Функции для работы с датами

```
SELECT empno, hiredate,
ROUND(hiredate,'MONTH'),TRUNC(hiredate,
  'MONTH')
FROM emp
WHERE hiredate like'%82‘;
```

EMPNO	HIREDATE	ROUND(HIR	TRUNC(HIR
7934	23-JAN-82	01-FEB-82	01-JAN-82
7788	09-NOV-82	01-NOV-82	01-NOV-82

Функции преобразования

Явное преобразование типов данных

Из какого	В какой формат				
	CHAR	NUMBER	DATE	RAW	ROWID
CHAR		TO_NUMBER	TO_DATE	HEXTORAW	CHARTOROWID
NUMBER	TO_CHAR		TO_DATE		
DATE	TO_CHAR	TO_CHAR			
RAW	RAWTOHEX				
ROWID	ROWIDTOCHAR				

Функция TO_CHAR с датами

TO_CHAR (DATE, ‘fmt’)

Модель формата:

- Заключается в апострофы. Различает символы верхнего и нижнего регистров.
- Может включать любые разрешенные элементы формата даты.
- Отделяется от значение даты запятой.

Элементы формата даты

Формат	Описание
YYYY	Полный год цифрами
YEAR	Год прописью
MM	Двузначное цифровое обозначение месяца
MONTH	Полное название месяца
DY	Трехзначное алфавитное сокращенное название дня недели
DAY	Полное название дня недели

Функция TO_CHAR с датами

```
SQL> SELECT empno, TO_CHAR(hiredate,
  'MM/YY')  Month_Hired FROM emp WHERE
  ename = 'BLAKE';

     EMPNO MONTH
---------- -----
      7698 05/81
SQL> SELECT
  TO_CHAR(SYSDATE,'DD-MONTH-YEAR') FROM
  DUAL;

TO_CHAR(SYSDATE,'DD-MONTH-YEAR')
-----------------------------------------------

01-MAY-TWO THOUSAND TWO
```

Функция TO_CHAR с числами

TO_CHAR (number, ‘fmt’)

Форматы, используемые с функцией TO_CHAR для вывода числового значения в виде символьной строки.

9	Цифра
0	Вывод нуля
$	Плавающий знак доллара
L	Плавающий символ местной валюты
.	Вывод десятичной точки
,	Вывод разделителя троек цифр

Функция TO_CHAR с числами

```
SQL>  SELECT TO_CHAR(sal,'$99,999')
  SALARY
   FROM emp WHERE ename = 'SCOTT';

SALARY
--------
  $3,000
```

Функции TO_NUMBER и TO_DATE

- Преобразование строки символов в числовой формат с помощью функции TO_NUMBER.

  TO_NUMBER (char)

- Преобразование строки символов в формат даты с помощью функции TO_DATE.

  TO_DATE (char [, ‘fmt’])

Функция DECODE

Упрощает условные запросы, выполняя работу команды CASE или IF-THEN-ELSE:

```
DECODE (col/expression, search1, result1
          [, search2, result2,…,]
          [, default])
```

Функция DECODE

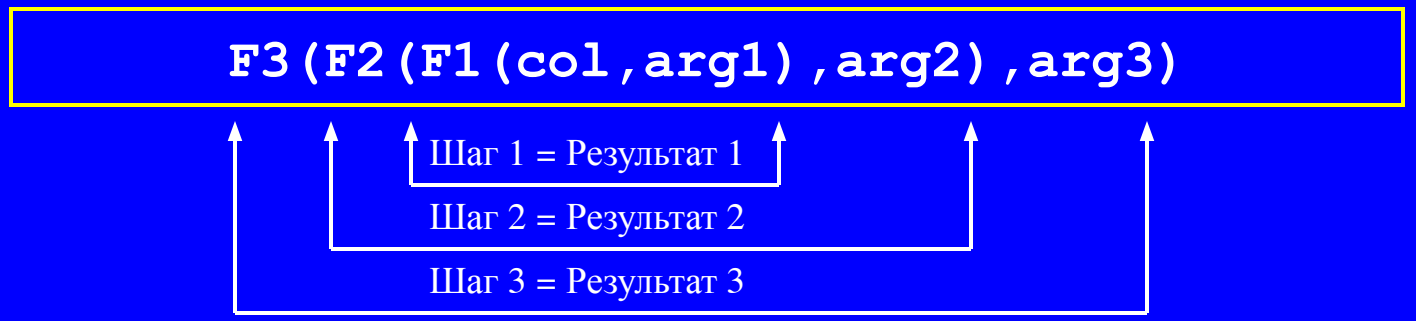
```
SQL> SELECT job, sal,
     DECODE (job, 'ANALYST', SAL*1.1,
                  'CLERK', SAL*1.15,
                 'MANAGER', SAL*1.20, SAL)
     REVISED_SALARY FROM EMP;
```

JOB	SAL	REVISED_SALARY
CLERK	800	920
SALESMAN	1600	1600
SALESMAN	1250	1250

…

Вложенные функции

- Однострочные функции могут быть вложены на любую глубину
- Вложенные функции вычисляются от самого глубокого уровня к внешнему

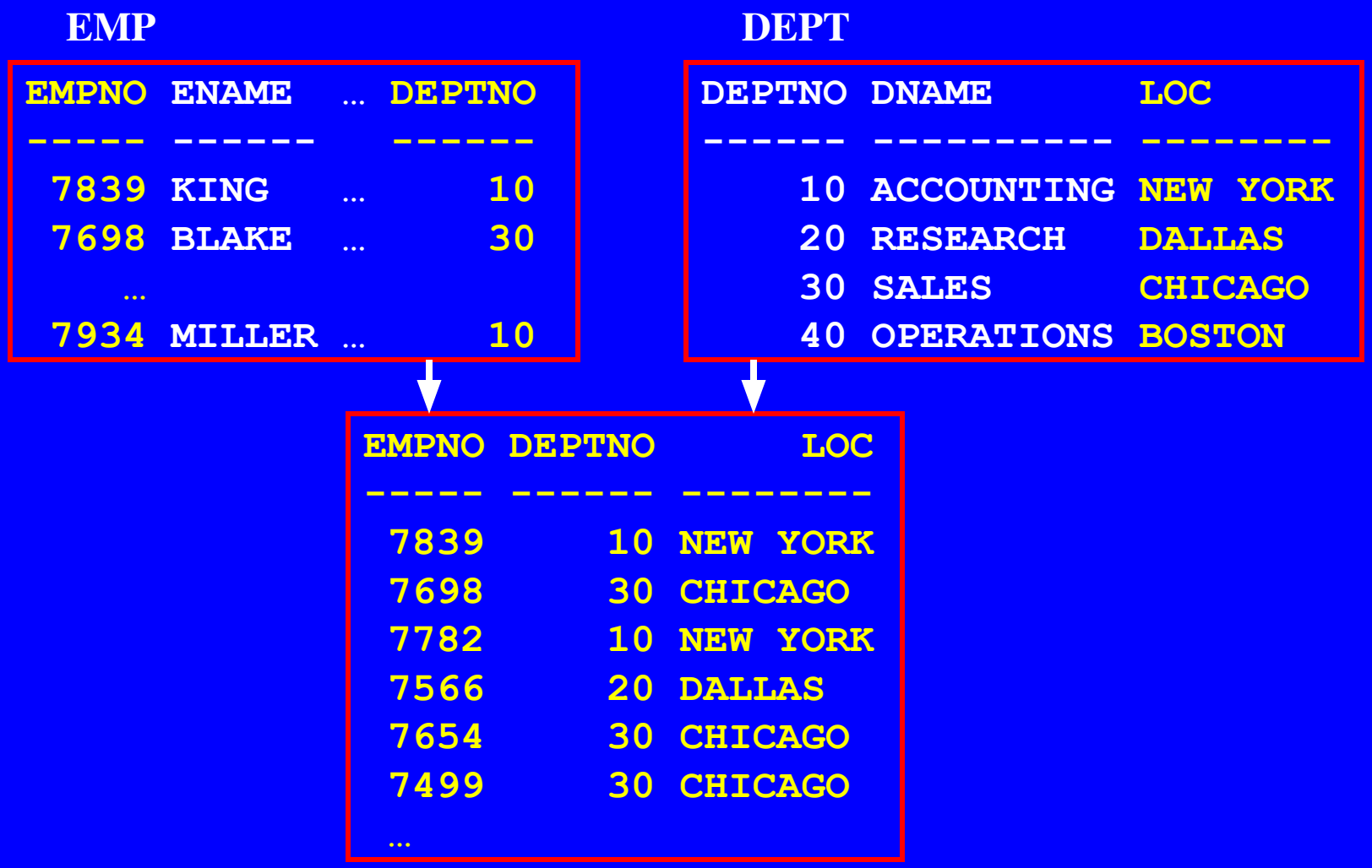


Вложенные функции

```
SQL> SELECT ename,
     NVL(TO_CHAR(mgr), 'No Manager')
     FROM emp
     WHERE mgr IS NULL;


ENAME      NVL(TO_CHAR(MGR),'NOMANAGER')
---------- -----------------------------
KING       No Manager
```

Выборка данных из нескольких таблиц



Понятие соединения

Соединение используется для выборки данных из нескольких таблиц.

```
SELECT table1.column,
   table2.column
FROM table1, table2
WHERE table1.column1 =
   table2.column2;
```

Условие соединения указывается во фразе WHERE.

- Если одно и то же имя столбца присутствует более, чем в одной таблице, к имени столбца добавляется имя таблицы в виде префикса.

Декартово произведение

- Декартово произведение образуется, если:
  - Опущено условие соединения.
  - Условие соединения недействительно.
  - Все строки первой таблицы соединяются со всеми
    строками второй таблицы
- Во избежании получения декартова произведения предложение WHERE всегда должно включать правильное условие соединения.

Получение декартова произведения

```
SQL> SELECT ename,dname FROM emp,
dept;
```

EMP (14 записей)

EMPNO	ENAME	…	DEPTNO
7839	KING	…	10
7698	BLAKE	…	30
…			
7934	MILLER	…	10

DEPT (4 записи)

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

Декартово произведение: 14*4=56 строк

ENAME	DNAME
KING	ACCOUNTING
BLAKE	ACCOUNTING
…	
KING	RESEARCH
BLAKE	RESEARCH

Виды соединений

Основные типы соединений

- Эквисоединения
- Неэквисоединения

Дополнительные типы соединений

- Внешние соединения
- Соединения таблицы с собой

Эквисоединение

EMP

EMPNO	ENAME	DEPTNO
7369	SMITH	20
7499	ALLEN	30
7521	WARD	30
7566	JONES	20
7654	MARTIN	30
7698	BLAKE	30
7782	CLARK	10
7839	KING	10
7844	TURNER	30
7900	JAMES	30
7902	FORD	20
...		

Внешний ключ

DEPT

DEPTNO	DNAME	LOC
20	RESEARCH	DALLAS
30	SALES	CHICAGO
30	SALES	CHICAGO
20	RESEARCH	DALLAS
30	SALES	CHICAGO
30	SALES	CHICAGO
10	ACCOUNTING	NEW YORK
10	ACCOUNTING	NEW YORK
30	SALES	CHICAGO
30	SALES	CHICAGO
20	RESEARCH	DALLAS
	...	

Главный ключ

Выборка при помощи эквисоединений

```
SQL> SELECT emp.empno, emp.ename,
emp.deptno, dept.deptno,dept.loc
   FROM   emp, dept
   WHERE  emp.deptno = dept.deptno;
```

EMPNO	ENAME	DEPTNO	DEPTNO	LOC
7369	SMITH	20	20	DALLAS
7499	ALLEN	30	30	CHICAGO
7521	WARD	30	30	CHICAGO
7566	JONES	20	20	DALLAS

...

Различение столбцов с одинаковыми именами

- Для различения одноименных столбцов из разных таблиц используются префиксы в виде имен таблиц.
- Использование префиксов в виде имен таблиц увеличивает производительность.
- Одноименные столбцы из разных таблиц можно различать по их псевдонимам.

Дополнит. условия при эквисоединении

```
SQL> SELECT emp.empno, emp.ename,
emp.deptno, dept.deptno,dept.loc
   FROM   emp, dept
   WHERE  emp.deptno = dept.deptno
   AND INITCAP(ename) = 'King';
```

```
     EMPNO ENAME          DEPTNO     DEPTNO LOC
---------- --------- ---------- ---------- -------------
      7839 KING               10         10 NEW YORK
```

Псевдонимы таблиц

Использование псевдонимов таблиц упрощает запросы.

```
SQL> SELECT emp.empno, emp.ename,
emp.deptno, dept.deptno,dept.loc
   FROM   emp, dept
   WHERE  emp.deptno = dept.deptno;
```

```
SQL> SELECT   e.empno, e.ename,
e.deptno, d.deptno,d.loc
   FROM    emp e, dept d
   WHERE   e.deptno = d.deptno;
```

Не-эквисоединение

EMP

EMPNO	ENAME	SAL
7499	ALLEN	1600
7566	JONES	2975
7654	MARTIN	1250
7698	BLAKE	2850
7782	CLARK	2450
7839	KING	5000
7844	TURNER	1500
7900	JAMES	950
...		

SALGRADE

GRADE	LOSAL	HISAL
1	700	1200
2	1201	1400
3	1401	2000
4	2001	3000
5	3001	9999

Оклад в таблице EMP находится между нижней и верхней границами окладов в таблице SALGRADE

Выборка при помощи не-эквисоединений

```
SQL> SELECT  e.ename, e.sal, s.grade        FROM
     emp e, salgrade s
   WHERE   e.sal BETWEEN s.losal AND s.hisal;
```

ENAME	SAL	GRADE
SMITH	800	1
JAMES	950	1
ADAMS	1100	1
WARD	1250	2

...

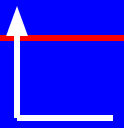
Внешние соединения

EMP

EMPNO	ENAME	…	DEPTNO
7839	KING	…	10
7698	BLAKE	…	30
7934	MILLER	…	20
...			

DEPT

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON



В отделе OPERATIONS служащих нет

EMP.deptno = DEPT.deptno

Внешние соединения

- Внешнее соединение используется для выборки строк, не удовлетворяющих обычным условиям соединения.
- Оператором внешнего соединения является знак плюс (+).

```
SELECT table.column, table.column
FROM   table1, table2
WHERE  table1.column(+) = table2.column;
SELECT table.column, table.column
FROM   table1, table2
WHERE  table1.column = table2.column(+);
```

Использование внешних соединений

```
SQL> SELECT e.ename, d.deptno, d.dname
     FROM  emp e, dept d
     WHERE e.deptno(+)=d.deptno
     ORDER BY e.deptno;

ENAME          DEPTNO DNAME
---------- ---------- --------------
BLAKE              30 SALES
MARTIN             30 SALES
WARD               30 SALES
                   40 OPERATIONS
```

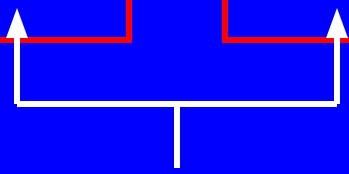
Соединение таблицы с собой

EMP (WORKER)

EMPNO	ENAME	MGR
7499	ALLEN	7698
7654	MARTIN	7698
7566	JONES	7839
7698	BLAKE	7839
7782	CLARK	7839
7839	KING	

EMP (MANAGER)

EMPNO	ENAME
7698	BLAKE
7698	BLAKE
7839	KING
7839	KING
7839	KING



MGR в таблице WORKER равен EMPNO в таблице MANAGER

Соединение таблицы с собой

```
SQL> SELECT worker.ename ||' works for '||
           manager.ename
     FROM emp worker, emp manager
     WHERE worker.mgr = manager.empno;

WORKER.ENAME||'WORKSFOR'||MANAG
------------------------------
SMITH works for FORD
ALLEN works for BLAKE
WARD works for BLAKE
JONES works for KING
…
```

Групповые функции

EMP

DEPTNO	SAL
20	800
30	1600
30	1250
20	2975
30	1250
30	2850
10	2450
10	5000
30	1500
30	950
20	3000
20	1100
...	

Максимальный оклад в таблице EMP

MAX(SAL)
5000

Типы групповых функций

- AVG
- COUNT
- MAX
- MIN
- STDDEV
- SUM
- VARIANCE

Использование функций AVG и SUM

AVG и SUM применяются к столбцам с числовыми данными.

```
SQL> SELECT AVG(sal), MAX(sal), MIN(sal),
   SUM(sal)
      FROM emp
       WHERE job LIKE 'SALES%';
```

AVG(SAL)	MAX(SAL)	MIN(SAL)	SUM(SAL)
1400	1600	1250	5600

Использование функций MIN и MAX

MIN и MAX применяются к данным любого типа.

```
SQL> SELECT MIN(hiredate), MAX(hiredate) FROM emp;

MIN(HIRED MAX(HIRED
--------- ---------
17-DEC-80 12-JAN-83
```

Использование функции COUNT

COUNT(*) возвращает количество строк в таблице.

```
SQL> SELECT COUNT(*) FROM emp WHERE deptno = 30;

  COUNT(*)
----------
         6
```

Использование функции COUNT

COUNT(expr) возвращает количество строк с определёнными значениями (не NULL).

```
SQL> SELECT COUNT(COMM) FROM emp WHERE deptno = 30;

 COUNT(COMM)
-----------
           4
```

Групповые функции

Групповые функции игнорируют неопределенные значения в столбцах.

```
SQL> SELECT AVG(comm) FROM emp;

 AVG(COMM)
----------
       550
```

Использование NVL с гр. функциями

Функция NVL заставляет групповые функции включать

неопределенные значения.

```
SQL> SELECT AVG(NVL(comm,0)) FROM emp;

 AVG(NVL(COMM,0))
----------------
      157.142857
```

Создание групп данных

EMP

DEPTNO	SAL
10	2450
10	5000
10	1300
20	800
20	1100
20	3000
20	2975
20	3000
30	1600
30	950
30	1500
30	2850
30	1250
30	1250

Средний оклад в таблице EMP по каждому отделу

DEPTNO	AVG(SAL)
10	2916.6667
20	2175
30	1566.6667

Создание групп данных: GROUP BY

```
SELECT column, group_function(column)
FROM table
[WHERE condition]
[GROUP BY group_by_expression]
[ORDER BY column];
```

Предложение GROUP BY разбивает строки таблицы на группы.

Использование предложения GROUP BY

Все столбцы, которые входят в SELECT список и к которым не применяются групповые функции, должны быть указаны в GROUP BY.

```
SQL> SELECT deptno, AVG(sal) FROM emp GROUP BY
   deptno;

    DEPTNO   AVG(SAL)
---------- ----------
        10 2916.66667
        20       2175
        30 1566.66667
```

Группировка по нескольким столбцам

```
SQL> SELECT deptno, job, sum(sal) FROM emp
     GROUP BY deptno, job;

    DEPTNO JOB         SUM(SAL)
---------- --------- ----------
        10 CLERK           1300
        10 MANAGER         2450
        10 PRESIDENT       5000
        20 ANALYST         6000
        20 CLERK           1900
        ...
```

Ошибки при использовании гр. функций

```
SQL> SELECT deptno, COUNT(ename) FROM emp;
```

```
SQL> SELECT deptno, AVG(sal)
     FROM emp
     WHERE AVG(sal) > 2000
     GROUP BY deptno;
```

Исключение групп

EMP

DEPTNO	SAL
10	2450
10	5000
10	1300
20	800
20	1100
20	3000
20	2975
20	3000
30	1600
30	950
30	1500
30	2850
30	1250
30	1250

Максимальный оклад в отделе превышает $2900

DEPTNO	MAX(SAL)
10	5000
20	3000

Исключение групп: HAVING

```
SELECT column, group_function(column)
FROM table
[WHERE condition]
[GROUP BY group_by_expression]
[HAVING group_condition]
[ORDER BY column];
```

Выводятся группы, удовлетворяющие условию в предложении HAVING.

Использование предложения HAVING

```
SQL> SELECT job, SUM(sal) PAYROLL
     FROM emp
     WHERE job NOT LIKE 'SALES%'
     GROUP BY job
     HAVING SUM(sal)>5000
     ORDER BY SUM(sal);

JOB              PAYROLL
---------- ----------
ANALYST            6000
MANAGER            8275
```

Вложенные групповые функции

```
SQL> SELECT max(avg(sal))
     FROM emp
     GROUP BY deptno;


MAX(AVG(SAL))
-------------
    2916.66667
```

Подзапросы

```
SELECT select_list
FROM   table
WHERE  expr operator
          (SELECT select_list
           FROM table);
```

- Вложенный запрос выполняется один раз до главного запроса.
- Результат подзапроса используется главным запросом (внешним запросом).

Использование подзапроса

```
SELECT ename
FROM   emp
WHERE  sal > (SELECT sal
              FROM emp
              WHERE empno = 7566);
```

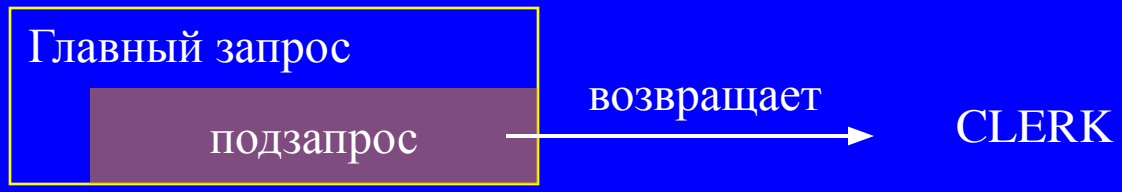

```
ENAME
----------
KING
FORD
SCOTT
```

Указания по использованию подзапросов

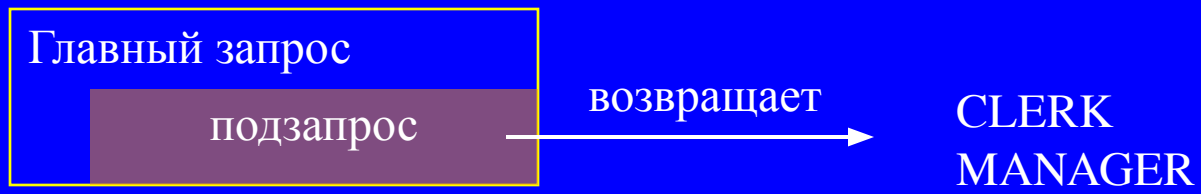
- Подзапрос должен быть заключен в скобки.
- Подзапрос должен находиться справа от оператора сравнения.
- Подзапрос не может содержать предложение ORDER BY.
- В однострочных подзапросах используются однострочные операторы.
- В многострочных подзапросах используются многострочные операторы.

Типы подзапросов

- Однострочный подзапрос.



- Многострочный подзапрос.

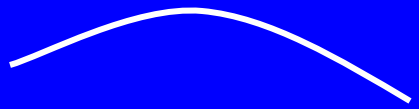


- Многостолбцовый подзапрос.

Выполнение однострочных подзапросов

```
SELECT ename, job
FROM   emp
WHERE  job =
(SELECT job FROM emp WHERE empno = 7369)

AND    sal >
(SELECT sal FROM emp WHERE empno = 7876);
```




```
ENAME      JOB
---------- ---------
MILLER     CLERK
```

Использование гр. функций в подзапросах

```
SELECT ename, job, sal
FROM   emp
WHERE  sal =
       (SELECT MIN(sal)
        FROM emp);
```

800

```
ENAME      JOB              SAL
---------- --------- ----------
SMITH      CLERK            800
```

HAVING с подзапросами

```
SELECT    deptno, MIN(sal)
FROM      emp
GROUP BY deptno
HAVING    MIN(sal) >
          (SELECT MIN(sal)
           FROM   emp
           WHERE  deptno = 20);
```

800

```
    DEPTNO   MIN(SAL)
---------- ----------
        10       1300
        30        950
```

Ошибки при использовании подзапросов

```
SQL> SELECT  empno, ename
     FROM    emp
     WHERE   sal = (SELECT MIN(sal)
                    FROM emp
                 GROUP BY deptno);

SQL> SELECT  ename, job
     FROM    emp
     WHERE   job = (SELECT job
                    FROM emp
                 WHERE ename = 'SMYTHE');
```

Многострочные запросы

- Возвращают более одной строки
- Используют многострочные операторы сравнения

Оператор	Значение
IN	Равно любому члену списка
ANY	Сравнение значения с любым значением, возвращаемым подзапросом.
ALL	Сравнение значения с каждым значением, возвращаемым подзапросом.

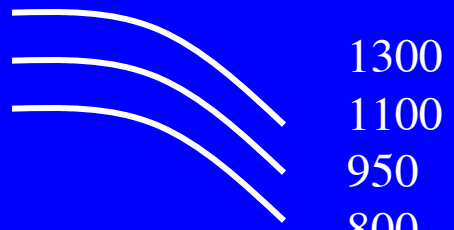
Оператор IN

```
SQL>  SELECT ename, sal, deptno
      FROM emp
    WHERE sal IN (800, 950, 1300);
```

ENAME	SAL	DEPTNO
SMITH	800	20
JAMES	950	30
MILLER	1300	10

Оператор ANY

```
SQL>  SELECT empno, ename, job
      FROM   emp
      WHERE  sal < ANY
            (SELECT sal
             FROM emp
             WHERE job = 'CLERK')
      AND job <> 'CLERK';
```

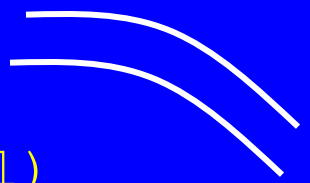


1300
1100
950
800

```
     EMPNO ENAME      JOB
---------- ---------- ---------
      7521 WARD       SALESMAN
      7654 MARTIN     SALESMAN
```

Оператор ALL

```
SQL>  SELECT empno, ename, job
      FROM   emp
      WHERE  sal > ALL
             (SELECT avg(sal)
              FROM emp
              GROUP BY deptno);
```



1566.6667
2175
2916.6667

```
     EMPNO ENAME      JOB
---------- ---------- ---------
      7566 JONES      MANAGER
      7839 KING       PRESIDENT
      7902 FORD       ANALYST
      7788 SCOTT      ANALYST
```

Многостолбцовые подзапросы

```
SELECT column, column, ...
FROM table
WHERE (column, column, ...) IN
        (SELECT column, column, ...
            FROM table
            WHERE condition);
```

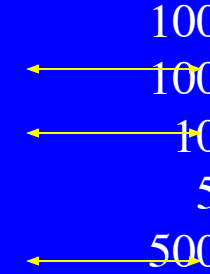
Использование многостолбц. подзапросов

```
SELECT ordid, prodid, qty
FROM   item
WHERE (prodid, qty) IN
          (SELECT prodid, qty
           FROM item
           WHERE ordid = 605)
AND ordid <> 605;
```

Сравнения столбцов

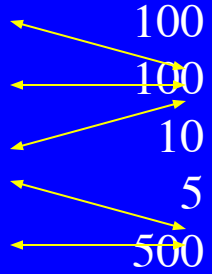
Парное

PRODID	QTY
101863	100
100861	100
102130	10
100890	5
100870	500
101860	50



Непарное

PRODID	QTY
101863	100
100861	100
102130	10
100890	5
100870	500
101860	50



Подзапрос с непарным сравнением

```
SELECT ordid, prodid, qty
FROM   item
WHERE  prodid IN
           (SELECT prodid
            FROM   item
            WHERE  ordid = 605)
AND    qty IN
           (SELECT qty
            FROM   item
            WHERE  ordid = 605)
AND    ordid <> 605;
```

Неопределенные значения в подзапросе

```
SQL> SELECT employee.ename
     FROM emp employee
      WHERE employee.empno NOT IN
        (SELECT manager.mgr FROM emp
   manager);


no rows selected
```

NOT IN эквивалентно !=ALL

Подзапрос в предложении FROM

```
SQL> SELECT a.ename, a.sal, a.deptno, b.salavg
    FROM emp a, (SELECT deptno, avg(sal) salavg
  FROM          emp GROUP BY deptno) b
    WHERE a.deptno = b.deptno AND a.sal >
  b.salavg
```

ENAME	SAL	DEPTNO	SALAVG
KING	5000	10	2916.66667
FORD	3000	20	2175
JONES	2975	20	2175
SCOTT	3000	20	2175

...

Переменные подстановки

- Использование переменных подстановки SQL*Plus для временного хранения значений.
  - Одиночный амперсанд(&)
  - Двойной амперсанд (&&)
  - Команды DEFINE и ACCEPT
- Передача значений переменных из одной команды SQL в другую.
- Динамическое изменение верхних и нижних колонтитулов.

Переменная подстановки с одним &

Данная переменная позволяет запросить значение у пользователя.

```
SQL> SELECT empno, ename, sal, deptno
     FROM emp
     WHERE empno = &employee_num;

Enter value for employee_num: 7369

    EMPNO ENAME                SAL     DEPTNO
--------- ---------- ---------- ----------
     7369 SMITH                 800         20
```

Использование команды SET VERIFY

Если задан режим SET VERIFY ON, SQL*Plus выводит текст команды до и

после замены переменных подстановки значениями.

```
SQL> SET VERIFY ON
SQL> SELECT empno, ename, sal, deptno
     FROM emp
     WHERE empno = &employee_num;


Enter value for employee_num: 7369
old   1: WHERE empno = &employee_num
new   1: WHERE empno = 7369
```

Символьные значения и даты с &

Даты и символьные значения заключаются в апострофы.

```
SQL> SELECT ename, deptno, sal*12
     FROM emp
     WHERE job = '&job_title';

Enter value for job_title: ANALYST

ENAME          DEPTNO     SAL*12
---------- ---------- ----------
FORD               20      36000
SCOTT              20      36000
```

Задание параметров во время выполнения

Переменные подстановки могут замещать:

- Условие WHERE
- Предложение ORDER BY
- Выражение со столбцами
- Имя таблицы
- Целую команду SELECT

Задание параметров во время выполнения

```
SQL> SELECT empno, ename, job, &column_name
   FROM emp
      WHERE  &condition ORDER BY &order_column;

Enter value for column_name:   sal
Enter value for condition:     sal>=3000
Enter value for order_column: ename

EMPNO ENAME      JOB                SAL
---------- ---------- --------- ----------
      7902 FORD       ANALYST          3000
      7839 KING       PRESIDENT        5000
      7788 SCOTT      ANALYST          3000
```

Переменная подстановки с двумя &&

Переменная подстановки с двумя амперсандами (&&) позволяет многократно использовать значение переменной, не запрашивая его повторно у пользователя.

```
SQL> SELECT empno, ename, job,
  &&column_name
    FROM emp ORDER BY &column_name;

Enter value for column_name: deptno
EMPNO ENAME      JOB              DEPTNO
---------- ---------- --------- ----------
      7782 CLARK      MANAGER             10
      7839 KING       PRESIDENT           10
      ...
```

Задание пользовательских переменных

- Задать переменную можно с помощью одной из двух команд SQL*Plus:

  - DEFINE: создает пользовательскую переменную с типом данных CHAR
  - ACCEPT: считывает входные данные пользователя и сохраняет их в переменной

- Если в команде DEFINE требуется одиночный пробел, этот пробел должен быть заключен в апострофы

Команда ACCEPT

- Создает более удобное приглашение пользователю ввести данные.
- Явно задает переменную типа NUMBER или DATE.
- Скрывает вводимые пользователем данные в целях защиты.

```
ACCEPT variable [datatype]
  [FORMAT format]
   [PROMPT text] {HIDE}
```

Использование команды ACCEPT

```
SQL> ACCEPT dept PROMPT 'Provide the
  department name:'
     SELECT *
     FROM dept
     WHERE dname = UPPER('&dept');


Provide the department name: Sales

    DEPTNO DNAME          LOC
---------- -------------- -------------
        30 SALES          CHICAGO
```

Команды DEFINE и UNDEFINE

- Переменная остается заданной:

  - До её удаления командой UNDEFINE
  - До выхода из SQL*Plus

- Проверить изменения можно с помощью команды DEFINE.

- Чтобы задать переменные для всех сеансов, необходимо изменить файл login.sql, чтобы переменные создавались при запуске системы.

Использование команды DEFINE

- Переменная для хранения названия отдела:

```
SQL> DEFINE deptname = sales

SQL> DEFINE deptname
DEFINE DEPTNAME          = "sales" (CHAR)
```

- Использование переменной:

```
SELECT *
FROM dept
WHERE dname = UPPER('&deptname');
```

Настройка среды SQL*Plus

- Для управления текущим сеансом пользуйтесь командой SET.

```
SET system_variable value
```

- Проверка заданных значений с помощью SHOW:

```
SQL> SET ECHO ON

SQL> SHOW ECHO
echo ON
```

Переменные команды SET

- ARRAYSIZE {20 | n}
- COLSEP {_ | text}
- FEEDBACK {6 | n | OFF | ON}
- HEADING {OFF | ON}
- LINESIZE {80 | n}
- LONG {80 | n}
- PAGESIZE {24 | n}
- PAUSE {OFF | ON | text}
- TERMOUT {OFF | ON}

Команды форматирования среды SQL*Plus

- `COLUMN [column option]`
- `TTITLE [text | OFF | ON]`
- `BTITLE [text | OFF | ON]`
- `BREAK  [ON report_element]`

Команда COLUMN

Управляет форматом вывода столбца

```
COLUMN [column option]
```

- **CLE[AR]:** сбрасывает все установки для столбца.
- **FOR[MAT] format:** изменяет вывод столбца с помощью форматной модели.
- **HEA[DING] text:** задает заголовок столбца.
- **JUS[TIFY] {align}:** выравнивает заголовок столбца слева, по центру или справа.

Использование команды COLUMN

- Создание заголовков столбцов:

```
COLUMN ename HEADING 'Employee|Name' FORMAT A15
COLUMN sal JUSTIFY LEFT FORMAT $99,990.00
COLUMN mgr FORMAT 999999999 NULL 'No manager'
```

- Вывод на экран текущих установок для ENAME:

```
COLUMN ename
```

- Сброс установок для ENAME:

```
COLUMN ename CLEAR
```

Модели формата в команде COLUMN

Элемент	Описание	Пример	Результат
An	Задает ширину выходного столбца n	A3	N/A
9	Один цифровой разряд	999999	1234
0	Ведущий ноль	099999	01234
$	Плавающий знак доллара	$9999	$1234
L	Местная валюта	L9999	$1234
.	Позиция десятичной точки	9999.99	1234.00
,	Разделитель тысяч	9,999	1,234

Использование команды BREAK

Устраняет дубликаты и группирует строки

- Для устранения дубликатов

```
SQL> BREAK ON ename ON job
```

- Для вычисления общих сумм

```
SQL> BREAK ON report
```

- Для группировки строк по заданным значениям

```
SQL> BREAK ON ename SKIP 4 ON job
  SKIP2
```

Использование команд TTITLE и BTITLE

Вывод заголовков и нижних колонтитулов

```
TTI[TLE] [text|OFF|ON]
```

- Задание заголовка отчета

```
SQL> TTITLE 'Salary|Report'
```

- Задание нижнего колонтитула отчета

```
SQL> BTITLE 'Confidental'
```

Язык манипулирования данными (DML)

- Команды DML выполняются при следующих операциях:

  - Добавление новых строк в таблицу
  - Изменение существующих строк в таблице
  - Удаление существующих строк из таблицы

- Транзакция – совокупность команд DML, образующих логическую единицу работы.

Команда INSERT

- Для добавления новых строк в таблицу используется команда INSERT.

```
 INSERT INTO table [(column [, column…])]
VALUES (value [, value…]);
```

- Данный синтаксис позволяет заносить в таблицу только по одной строке.

Вставка новых строк в таблицу

- Символьные значения и даты заключаются в апострофы.

```
SQL> INSERT INTO dept (deptno, dname, loc)
     VALUES (50, 'DEVELOPMENT', 'DETROIT');

1 row created.
```

Добавление строк с NULL значениями

- Неявный метод: столбец не указывается в списке столбцов.

```
SQL> INSERT INTO dept (deptno, dname)
      VALUES (60, 'MIS');
1 row created.
```

- Явный метод: использование ключевого слова NULL или пустой строки(‘’) в списке VALUES.

```
SQL> INSERT INTO dept (deptno, dname)
      VALUES (70, 'FINANCE', NULL);
1 row created.
```

Вставка специальных значений

Функция SYSDATE записывает текущие дату и время.

```
SQL> INSERT INTO emp (empno, ename, job,
mgr, hiredate, sal, comm, deptno)
VALUES (7196, 'GREEN','SALESMAN', 7782,
        SYSDATE, 2000, NULL, 10);
1 row created.
```

Задание конкретных значений даты

- Добавление нового служащего.

```
SQL> INSERT INTO emp
     VALUES (2296, 'AROMANO','SALESMAN',
7782,          TO_DATE('FEB,3,97','MON DD,
YY'),                          1300, NULL,
10);
1 row created.
```

- Проверка добавления нового служащего.

```
EMPNO ENAME  JOB       MGR  HIREDATE  SAL  COMM DEPTNO
----- ------ -------- ---- --------- ---- ---- ------
  2296 AROMANO SALESMAN 7782 03-FEB-97 1300       10
```

Задание значений с переменными

```
SQL> INSERT INTO dept (deptno,
dname, loc)
     VALUES (&department_id,
'&department_name',
             '&location');
Enter value for department_id: 80
Enter value for department_name: EDUCATION
Enter value for location: ATLANTA

1 row created.
```

Копирование строк из другой таблицы

- Команда INSERT включает подзапрос.

```
SQL> INSERT INTO managers (id, name, salary,
  hiredate)
        SELECT empno, ename, sal, hiredate
     FROM emp
     WHERE job = 'MANAGER';

3 rows created.
```

- Предложение VALUES не используется.
- Количество столбцов, указанных в предложении INSERT, должно совпадать с количеством столбцов в подзапросе.

Команда UPDATE

- Для обновления существующих строк в таблице используется команда UPDATE.

```
UPDATE table
SET column = value [, column = value]
[WHERE condition];
```

- В случае необходимости можно обновлять несколько строк.

Обновление строк в таблице

- Предложение WHERE позволяет изменить конкретную строку или строки.

```
SQL> UPDATE emp
       SET deptno = 20
       WHERE empno = 7782;
1 row updated.
```

- Если предложение WHERE отсутствует, обновляются все строки таблицы.

```
SQL> UPDATE employee
       SET deptno = 20;
14 rows updated.
```

UPDATE с помощью многостолб. подзапр.

- Изменение должности и номера отдела служащего под номером 7698 на такие же значения, как у служащего под номером 7499.

```
SQL> UPDATE emp
       SET (job, deptno) =
     (SELECT job, deptno
      FROM emp
      WHERE empno = 7499)
       WHERE empno = 7782;

1 row updated.
```

UPDATE на основе другой таблицы.

- Для изменения строк таблицы на основе значений из другой таблицы следует использовать подзапросы в командах UPDATE.

```
SQL> UPDATE employee
     SET deptno =   (SELECT deptno
                     FROM   emp
                     WHERE  empno = 7788)
     WHERE job =    (SELECT job
                     FROM   emp
                     WHERE  empno = 7788);

2 rows updated.
```

Команда DELETE

- Для удаления существующих строк используется команда DELETE.

```
DELETE [FROM] table [WHERE condition];
```

- В случае необходимости можно удалять несколько строк.

Удаление строк из таблицы

- Конкретная строка или строки удаляются с помощью предложения WHERE.

```
SQL> DELETE FROM department
       WHERE dname = 'DEVELOPMENT';

1 row deleted.
```

- Если предложение WHERE отсутствует, удаляются все строки таблицы.

```
SQL> DELETE FROM department;

4 rows deleted.
```

DELETE на основе другой таблицы.

- Для изменения строк таблицы на основе значений из другой таблицы следует использовать подзапросы в командах UPDATE.

```
SQL> DELETE FROM employee
      WHERE deptno =    (SELECT deptno
                         FROM   dept
                         WHERE  dname =
'SALES');

6 rows deleted.
```

Транзакции базы данных

Содержат что-либо из следующего:

- Команды DML, выполняющие единое согласованное изменение данных.
- Одну команду DDL.
- Одну команду DCL.

Транзакции базы данных

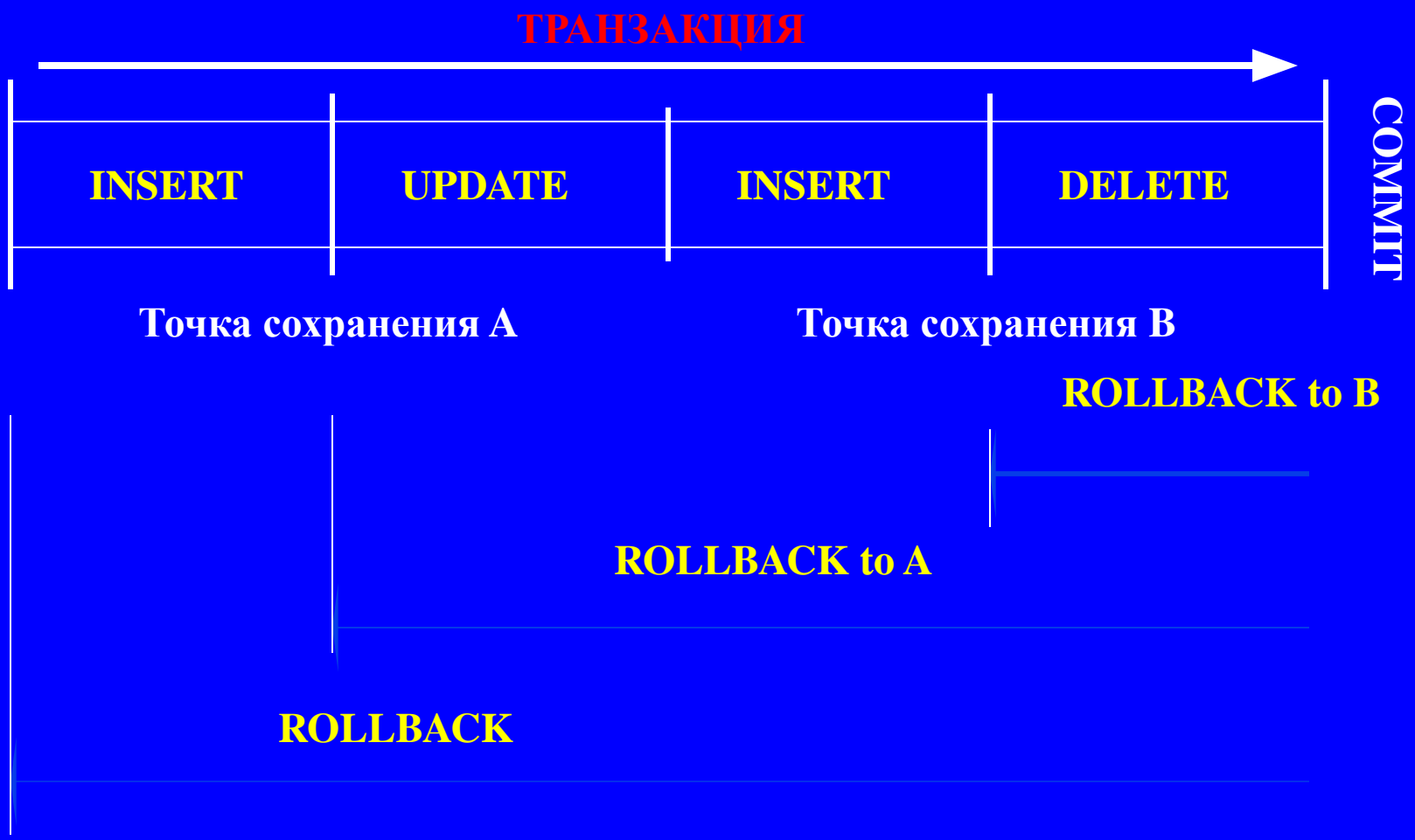
- Начинаются с выполнения первой исполняемой команды SQL.
- Заканчиваются одним из сл. событий:

  - Выполнение команды COMMIT или ROLLBACK
  - Выполнение команды DDL или DCL (автоматическая фиксация транзакции)
  - Выходы в подпрограммы пользователя
  - Отказы системы

Управление транзакциями

- COMMIT
- SAVEPOINT name
- ROLLBACK [TO SAVEPOINT name]

Управление транзакциями



Неявная обработка транзакций

- Автоматическая фиксация изменений (COMMIT) происходит в сл. случаях:

  - Выполнение команды DDL
  - Выполнение команды DCL
  - Нормальный выход из SQL*Plus без явного выполнения команды COMMIT или ROLLBACK

- Автоматический откат (ROLLBACK) происходит в случае аварийного прекращения сеанса работы в SQL*Plus или отказа системы.

Состояние данных

- Состояние данных может быть восстановлено.
- Текущий пользователь может просмотреть результаты своих операций DML с помощью команды SELECT.
- Другие пользователи не могут видеть результаты команд DML текущего пользователя.
- Изменяемые строки блокируются, и другие пользователи не могут обновлять их содержимое.

Фиксация изменений в данных

- Внесение изменений.

```
SQL> UPDATE emp
       SET deptno = 10
 WHERE empno = 7782;
1 row updated.
```

- Фиксация изменений.

```
SQL> COMMIT;
Commit complete.
```

Откат изменений в данных

- Внесение изменений.

```
SQL> DELETE FROM employee;
14 rows deleted.
```

- Откат изменений.

```
SQL> ROLLBACK;
Rollback complete.
```

Откат к точке сохранения

```
SQL> UPDATE...
SQL> SAVEPOINT update_done;
Savepoint created.
SQL> INSERT...
SQL> ROLLBACK TO update_done;
Rollback complete.
```

Откат на уровне команды

- Если ошибка возникла при выполнении одной конкретной команды DML, отменяются только результаты этой команды.
- Сервер Oracle использует неявную точку сохранения.
- Все прочие изменения сохраняются.
- Пользователю следует завершать транзакции явно командой COMMIT или ROLLBACK.

Согласованность чтения

- Согласованность чтения гарантирует непротиворечивое представление данных в любой момент времени.
- Изменения, сделанные одним пользователем, не вступают в противоречие с изменениями, сделанными другим пользователем.
- Гарантируется, что для одних и тех же данных:
  - “Читатели” никогда не блокируют ”Писателей”.
  - ”Писатели” никогда не блокируют ”Читателей”.

Блокировка данных

- Exclusive (исключительный)
- Share (разделяемый)

Объекты базы данных

Объект	Описание
Таблица	Основная единица хранения; состоит из строк и столбцов
Представление	Логически представляет подмножество данных из одной или нескольких таблиц
Последовательность	Генерирует значения первичных ключей
Индекс	Увеличивает производительность некоторых запросов
Синоним	Задает альтернативные имена для некоторых объектов

Правила задания имен

- Имя начинается с буквы
- Может быть длиной до 30 символов
- Должно содержать только символы A-Z, a-z, 0-9, _, $ и #
- Не должно совпадать с именем другого объекта, принадлежащего тому же пользователю
- Не должно совпадать со словом, зарезервированным сервером Oracle

Команда CREATE TABLE

- Необходимо иметь:
  - привилегию CREATE TABLE
  - область хранения

```
CREATE TABLE [schema.] table
   (column datatype [DEFAULT expr]);
```

Опция DEFAULT

- Задает значение по умолчанию, если при добавлении данных значение не указывается явно.

```
...hiredate DATE DEFAULT SYSDATE, ...
```

- В качестве значения допускается литерал, выражение или функция SQL.
- Не может использоваться имя другого столбца или псевдостолбца.
- Тип данных, используемый по умолчанию, должен совпадать с типом данных столбца.

Создание таблиц

```
SQL> CREATE TABLE ABC
       (DEPTNO NUMBER (2),
        DNAME  VARCHAR2 (14),
        LOC    VARCHAR2 (13));
Table created.

SQL> DESCRIBE abc
Name                          Null?    Type
----------------------------- -------- ------------
DEPTNO                                 NUMBER(2)
DNAME                                  VARCHAR2(14)
LOC                                    VARCHAR2(13)
```

Таблицы в базе данных Oracle

- Таблицы пользователей

  - таблицы, создаваемые и поддерживаемые пользователями
  - содержат информацию пользователей

- Словарь базы данных

  - таблицы, создаваемые и поддерживаемые сервером Oracle
  - содержат служебную информацию о базе данных

Запрос к словарю данных

- Вывод определений таблиц, принадлежащих пользователю.

```
SELECT * FROM user_tables;
```

- Просмотр типов объектов, принадлежащих пользователю.

```
SELECT DISTINCT object_type
FROM   user_objects;
```

- Просмотр таблиц, представлений, синонимов и последовательностей, принадлежащих пользователю.

```
SELECT * FROM user_catalog;
```

Типы данных

Тип данных	Описание
VARCHAR2(size)	Символьные, переменной длины
CHAR(size)	Символьные, постоянной длины
NUMBER(p,s)	Числовые, переменной длины
DATE	Значения даты и времени
LONG	Символьные, переменной длины (до 2 Гб)
CLOB	Однобайтовые, символьные (до 4 Гб)
RAW и LONGRAW	Необработанные (“сырые”) двоичные
BLOB	Двоичные (до 4 Гб)
BFILE	Двоичные, во внешнем файле (до 4 Гб)

Создание таблицы, используя подзапрос

- Количество заданных столбцов должно совпадать с количеством столбцов в подзапросе

```
CREATE TABLE table
        [column (, column...)]
AS subquery;
```

Создание таблицы, используя подзапрос

```
CREATE TABLE dept30
AS
  SELECT empno, ename, sal*12 ANNSAL, hiredate
  FROM emp
  WHERE deptno = 30;

Table created.
```

Команда ALTER TABLE

- Используется для следующих операций:
  - Добавления / Удаления столбца
  - Изменение существующего столбца
  - Задание значения по умолчанию для столбца

```
ALTER TABLE table
ADD         (column datatype [DEFAULT expr]
         [, column datatype]...);

ALTER TABLE table
MODIFY      (column datatype [DEFAULT expr]
         [, column datatype]...);
```

Добавление столбца

- Столбцы добавляются с помощью предложения ADD.

```
ALTER TABLE dept30
ADD           (job VARCHAR2(9));
Table altered.
```

```
 EMPNO      ENAME      ANNSAL HIREDATE  JOB
 ---------- ---------- ------ --------- ---
      7499 ALLEN       19200 20-FEB-81
      7521 WARD        15000 22-FEB-81
      7654 MARTIN      15000 28-SEP-81
      ...
```

Изменение столбца

- Столбцы изменяются с помощью предложения MODIFY.

```
ALTER TABLE dept30
MODIFY      (ename VARCHAR2(15));

Table altered.
```

Удаление столбца

- Столбцы удаляются с помощью предложения DROP COLUMN.

```
ALTER TABLE dept30
DROP COLUMN job;

Table altered.
```

Опция SET UNUSED

- Столбцы помечаются, как неиспользуемые с помощью предложения SET UNUSED.

```
ALTER TABLE table
SET UNUSED (column);
```

- Удаление помеченных столбцов – DROP UNUSED COLUMNS.

```
ALTER TABLE table
DROP UNUSED COLUMNS;
```

Удаление таблицы

- Удаляются все данные и структура таблицы.
- Все незафиксированные транзакции фиксируются.
- Все индексы удаляются.
- Откат этой команды невозможен.

```
SQL> DROP TABLE dept30;
Table dropped.
```

Переименование объекта

Для переименования таблицы, представления, последовательности или синонима используется команда RENAME.

```
SQL> RENAME dept TO department;
Table renamed.
```

Усечение таблицы

- Команда TRUNCATE TABLE:
  - Удаляет все строки из таблицы
  - Освобождает пространство, используемое таблицей

```
SQL> TRUNCATE TABLE department;
Table truncated.
```

- Отмена удаления строк невозможна
- Альтернативная команда - DELETE

Добавление комментариев к таблице

- Добавление комментария к таблице или столбцу при помощи команды COMMENT.

```
SQL> COMMENT ON TABLE emp IS 'Employee info';
Comment created.
```

- Просмотр комментариев:
  - ALL_COL_COMMENTS
  - USER_COL_COMMENTS
  - ALL_TAB_COMMENTS
  - USER_TAB_COMMENTS

Понятие ограничения

- Ограничения обеспечивают выполнение правил на уровне таблицы.
- Ограничения предотвращают удаление таблицы при наличии подчиненных данных в других таблицах.
- В Oracle допускаются сл. виды ограничений:
  - NOT NULL
  - UNIQUE
  - PRIMARY KEY
  - FOREIGN KEY
  - CHECK

Определение ограничений

```
CREATE TABLE [schema.] table
            (column datatype [DEFAULT expr]
            [column_constraint],
        ...
            [table_constraint]);
CREATE TABLE emp
            empno NUMBER(4),
            ename VARCHAR2(10),
        ...
            deptno NUMBER(7,2) NOT NULL,
            CONSTRAINT emp_empno_pk
          PRIMARY KEY (empno));
```

Определение ограничений

- Ограничение на уровне столбца.

```
column [CONSTRAINT constraint_name]
 constraint_type
```

- Ограничение на уровне таблицы.

```
column,...
  [CONSTRAINT constraint_name]
 constraint_type
  (column,...),
```

Ограничение NOT NULL

- Предотвращает появление неопределенных значений в столбце

```
CREATE TABLE EMP (
   EMPNO      NUMBER(4),
   ENAME      VARCHAR2(10)  NOT NULL,
   JOB        VARCHAR2(9),
   MGR        NUMBER(4),
   HIREDATE   DATE,
   SAL        NUMBER(7,2),
   COMM       NUMBER(7,2),
   DEPTNO     NUMBER(2)     NOT NULL);
```

Ограничение UNIQUE

- Может быть задано на уровне столбца или таблицы

```
CREATE TABLE DEPT (
   DEPTNO   NUMBER(2),
   DNAME    VARCHAR2(14),
   LOC      VARCHAR2(13),
   CONSTRAINT dept_dname_uk UNIQUE(dname));
```

Ограничение PRIMARY KEY

- Может быть задано на уровне столбца или таблицы

```
CREATE TABLE DEPT (
   DEPTNO   NUMBER(2),
   DNAME    VARCHAR2(14),
   LOC      VARCHAR2(13),
   CONSTRAINT dept_deptno_pk PRIMARY
  KEY(deptno));
```

Ограничение FOREIGN KEY

- Может быть задано на уровне столбца или таблицы

```
CREATE TABLE EMP (
  EMPNO      NUMBER(4)     NOT NULL,
  ENAME      VARCHAR2(10),
  JOB        VARCHAR2(9),
  MGR        NUMBER(4),
  HIREDATE   DATE,
  SAL        NUMBER(7,2),
  COMM       NUMBER(7,2),
  DEPTNO     NUMBER(2),
  CONSTRAINT emp_deptno_fk FOREIGN KEY (deptno)
             REFERENCES dept (deptno));
```

Ключевые слова FOREIGN KEY

- FOREIGN KEY:

  определяет столбец в подчиненной таблице,

  используемый в качестве внешнего ключа.

- REFERENCES:

  определяет родительскую таблицу и столбец в ней.

- ON DELETE CASCADE:

  разрешает удаление строки в родительской таблице с одновременным удалением зависимых строк в подчиненной таблице.

Ограничение CHECK

- Задаёт условие, которому должна удовлетворять каждая строка таблицы
- Не допускаются:

  - Ссылки на псевдостолбцы CURRVAL, NEXTVAL, LEVEL и ROWNUM.
  - Вызовы функций SYSDATE, UID, USER и USERENV.
  - Запросы, ссылающиеся на другие значения в других строках.

```
..., deptno NUMBER(2),
 CONSTRAINT emp_deptno_ck
    CHECK (deptno BETWEEN 10 AND 99),...
```

Добавление ограничения

```
ALTER TABLE table
ADD [CONSTRAINT constraint] type (column);
```

- Ограничение можно добавить, удалить, но не изменить.
- Ограничение можно включать или выключать.
- Ограничение NOT NULL добавляется с помощью предложения MODIFY.

Добавление ограничения

- Добавление ограничения FOREIGN KEY для таблицы EMP. Это ограничение означает, что информация о менеджере уже должна существовать как о служащем в таблице EMP.

```
ALTER TABLE emp
ADD CONSTRAINT emp_mgr_fk
   FOREIGN KEY(mgr) REFERENCES emp(empno);
Table altered.
```

Удаление ограничения

- Удаление ограничения FOREIGN KEY из таблицы EMP.

```
ALTER TABLE emp
DROP CONSTRAINT emp_mgr_fk;

Table altered.
```

- Удаление ограничения PRIMARY KEY из таблицы DEPT и соответствующего ограничения FOREIGN KEY для столбца EMP.DEPTNO.

```
ALTER TABLE dept
DROP PRIMARY KEY CASCADE;

Table altered.
```

Отключение ограничений

- Для отключения ограничения используется команда ALTER TABLE с предложением DISABLE.
- Для отмены ограничения вместе с зависимыми ограничениями используется опция CASCADE.

```
ALTER TABLE emp
DISABLE CONSTRAINT emp_empno_pk CASCADE;

Table altered.
```

Включение ограничений

- Для включения ограничения используется команда ALTER TABLE с предложением ENABLE.
- При включении ограничения UNIQUE или PRIMARY KEY, автоматически создается уникальный индекс.

```
ALTER TABLE emp
ENABLE CONSTRAINT emp_empno_pk;

Table altered.
```

Каскадное удаление огранич. целостности

- Вместе с предложением DROP COLUMN можно использовать опцию CASCADE CONSTRAINTS.

- Опция CASCADE CONSTRAINTS удаляет все ссылочные ограничения целостности, которые зависят от первичного или уникального ключей, основанных на удаляемом столбце.

- Опция CASCADE CONSTRAINTS удаляет все составные ограничения целостности, которые созданы на основе удаляемого столбца.

Просмотр ограничений

- Просмотреть все определения и имена ограничений можно с помощью запроса к представлению USER_CONSTRAINTS.

```
SQL> SELECT constraint_name, constraint_type,
            search_condition
     FROM user_constraints WHERE table_name =
  'EMP';

CONSTRAINT_NAME C SEARCH_CONDITION
--------------- - ----------------
PK_EMP          P
FK_DEPTNO       R
```

Просмотр полей, участвующих в огран-ях

- Просмотреть столбцы, связанные с ограничениями можно с помощью запроса к представлению USER_CONS_COLUMNS.

```
SQL> SELECT constraint_name, column_name
     FROM user_cons_columns WHERE table_name
  = 'EMP';

CONSTRAINT_NAME COLUMN_NAME
--------------- -----------
FK_DEPTNO       DEPTNO
PK_EMP          EMPNO
```

Возможности представлений

- Ограничение доступа к базе данных.
- Упрощение сложных запросов.
- Обеспечение независимости от данных.
- Представление одних и тех же данных в разных видах.

Простые и сложные представления

Характеристика	Простые	Сложные
Количество таблиц	Одна	Одна или более
Содержат функции	Нет	Да
Содержат группы данных (предложе-ние DISTINCT или групповые ф-ии)	Нет	Да
Выполнение операций DML с представлениями	Да	Не всегда

Создание представления

- В команду CREATE VIEW включается подзапрос.

```
CREATE [OR REPLACE] [FORCE|NOFORCE] VIEW view
  [(alias[, alias]...)]
AS subquery
[WITH CHECK OPTION [CONSTRAINT constraint]]
[WITH READ ONLY];
```

- Подзапрос может содержать сложную команду SELECT.
- Подзапрос не может содержать предложение ORDER BY.

Создание представления

- Создание представления EMPVU10 с информацией о служащих
  отдела 10.

```
SQL> CREATE VIEW empvu10
     AS SELECT empno, ename, job
     FROM emp
     WHERE deptno = 10;
   View created.
```

- Вывод структуры представления с помощью команды DESCRIBE SQL*Plus.

```
SQL> DESCRIBE empvu10;
```

Создание представления

- Создание представления SALVU30 с использованием псевдонимов в подзапросе.

```
SQL> CREATE VIEW salvu30
     AS SELECT empno EMPLOYEE_NUMBER, ename NAME, sal
  SALARY
     FROM emp
     WHERE deptno = 30;
   View created.
```

- Вывод структуры представления с помощью команды DESCRIBE SQL*Plus.

```
SQL> DESCRIBE empvu10;
```

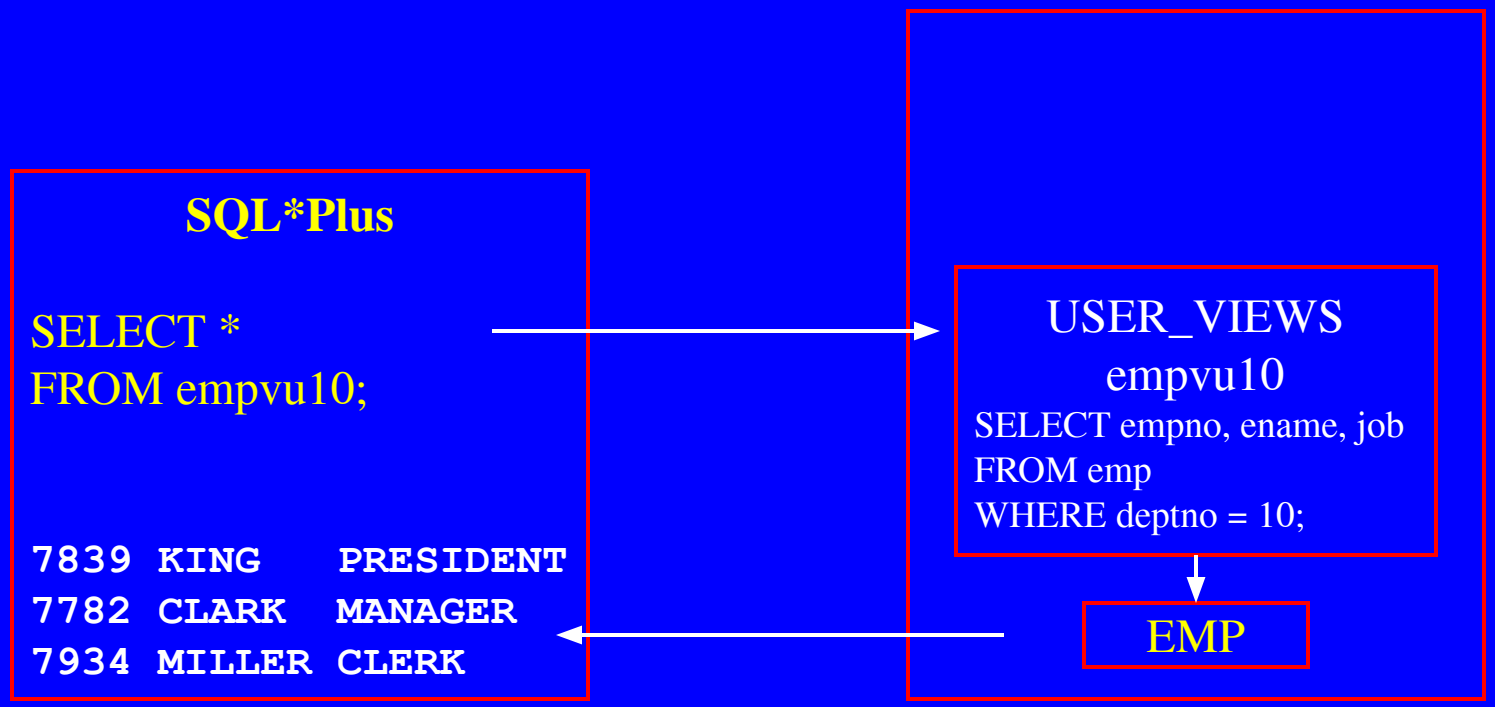
Выборка данных из представления

```
SQL> SELECT * FROM salvu30;

EMPLOYEE_NUMBER NAME                 SALARY
--------------- ---------- ----------
           7499 ALLEN              1600
           7521 WARD               1250
           7654 MARTIN             1250
           7698 BLAKE              2850
           7844 TURNER             1500
           7900 JAMES               950

6 rows selected.
```

Запрос к представлению



Изменение представления

- Изменение представления EMPVU10 с помощью предложения CREATE OR REPLACE VIEW. Добавление псевдонима для каждого столбца.

```
SQL> CREATE OR REPLACE VIEW empvu10
       (employee_number, employee_name,
  job_title)
     AS SELECT empno, ename, job
     FROM emp
     WHERE deptno = 10;
View created.
```

- Порядок псевдонимов в предложении CREATE VIEW должен быть таким же, как порядок столбцов в подзапросе.

Создание сложного представления

- Создание сложного представления с групповыми функциями для вывода данных из двух таблиц.

```
SQL> CREATE VIEW dept_sum_vu (name, minsal,
  maxsal, avgsal)
     AS SELECT d.dname, MIN(e.sal),
  MAX(e.sal), AVG(e.sal)
     FROM emp e, dept d
     WHERE e.deptno = d.deptno;
     GROUP BY d.dname;

View created.
```

Правила выполнения DML - операций

- Операции DML можно выполнять с простыми представлениями.
- Нельзя удалить строку, если представление содержит:
  - Групповые функции
  - Предложение GROUP BY
  - Ключевое слово DISTINCT
  - Ссылку на псевдостолбец ROWNUM
- Нельзя изменить строку, если представление содержит:
  - Столбцы, определяемые выражениями
- Невозможно добавить данные в представление, если:
  - В базовых таблицах имеются столбцы с ограничением NOT NULL, но они не входят в представление.

Использование WITH CHECK OPTION

- Необходимо следить за тем, чтобы результаты DML операций оставались в пределах домена представления.

```
SQL> CREATE OR REPLACE VIEW empvu20
     AS SELECT *
     FROM emp
     WHERE deptno = 20
     WITH CHECK OPTION CONSTRAINT empvu20_ck;
View created.
```

- Попытка изменить номер отдела для какой-либо строки в представлении закончится неудачей, т. к. при этом нарушится ограничение CHECK OPTION.

Запрет DML операций

- Использование опции WITH READ ONLY запрещает выполнять над представлением любые DML операции.

```
SQL> CREATE OR REPLACE VIEW empvu10
        (employee_number, employee_name,
    job_title)
        AS SELECT empno, ename, job
        FROM emp
        WHERE deptno = 10
        WITH READ ONLY;
View created.
```

Попытка выполнить команду DML для любой строки представления приведет к ошибке сервера Oracle (ORA-01752).

Удаление представления

- Удаление представления не вызывает потери данных, т. к. представление основано на реальных таблицах базы данных.

```
DROP VIEW view;
```

```
SQL> DROP VIEW empvu10;

View dropped.
```

- Порядок псевдонимов в предложении CREATE VIEW должен быть таким же, как порядок столбцов в подзапросе.

Понятие последовательности

- Автоматически генерирует уникальные числа.
- Является совместно используемым объектом.
- Обычно используется для генерации значений первичного ключа.
- Заменяет код в прикладной программе.
- Ускоряет доступ к числам последовательности, если они находятся в сверхоперативной (кэш) памяти.

Команда CREATE SEQUENCE

Определение последовательности для автоматической генерации чисел.

```
CREATE SEQUENCE sequence
      [INCREMENT BY n]
      [START WITH n]
      [{MAXVALUE n | NOMAXVALUE}]
      [{MIXVALUE n | NOMINVALUE}]
      [{CYCLE | NOCYCLE}]
      [{CACHE | NOCACHE}];
```

Создание последовательности

- Создание последовательности DEPT_DEPTNO для использования в качестве первичного ключа таблицы DEPT.
- При генерации значений первичных ключей не следует использовать опцию CYCLE.

```
CREATE SEQUENCE dept_deptno
      INCREMENT BY 1
      START WITH 91
      MAXVALUE 100
      NOCYCLE
      NOCACHE;
```

Проверка параметров последовательности

- Проверить значения последовательности можно в представлении USER_SEQUENCES словаря данных.

```
SELECT sequence_name, min_value,
max_value, increment_by, last_number
FROM   user_sequences;
```

SEQUENCE_NAME	MIN_VALUE	MAX_VALUE	INCREMENT_BY	LAST_NUMBER
CUSTID	1	1.0000E+27	1	109
DEPT_DEPTNO	1	100	1	91
ORDID	1	1.0000E+27	1	622
PRODI	1	1.0000E+27	1	200381
STUDENT_SEQUENS	1	1.0000E+27	1	10020

Псевдостолбцы NEXTVAL и CURRVAL

- NEXTVAL генерирует следующее свободное число в последовательности. Столбец возвращает уникальное значение при каждом обращении к нему – даже для разных пользователей.

- CURRVAL возвращает текущее значение последовательности. Чтобы данный столбец содержал значение, необходимо сначала обратиться к NEXTVAL для этой последовательности.

Использование последовательности

- Добавление нового отдела.

```
SQL> INSERT INTO dept (deptno, dname, loc)
     VALUES
    (dept_deptno.NEXTVAL, ' MARKETING',
  'MOSCOW');
1 row created.
```

- Вывод текущего значения в последовательности DEPT_DEPTNO.

```
SQL> SELECT dept_deptno.CURRVAL FROM dual;
 CURRVAL
---------
```

Изменение последовательности

Изменение шага приращения, макс. и мин. значений, режима циклической генерации значений и кэширования.

```
ALTER SEQUENCE dept_deptno
      INCREMENT BY 1
      MAXVALUE 999999
      NOCYCLE
      NOCACHE;
Sequence altered.
```

Изменение последовательности

- Для изменения последовательности необходимо быть её владельцем или иметь привилегию ALTER для неё.
- Команда влияет только на числа, генерируемые после изменения.
- Чтобы начать последовательность с другого числа, необходимо удалить её и создать заново.

Удаление последовательности

Для удаления последовательности используется команда DROP SEQUENCE.

```
SQL> DROP SEQUENCE dept_deptno;
Sequence dropped.
```

Понятие индекса

- Используется для ускорения выборки строк с помощью указателя.
- Уменьшает количество операций дискового ввода-вывода за счёт использования быстрого метода поиска данных.
- Независим от таблицы, для которой был создан.
- Автоматически используется и поддерживается сервером Oracle.

Создание индекса

- Автоматически

  -Уникальный индекс создаётся автоматически, если в

  определении таблицы задано ограничение

  PRIMARY KEY или UNIQUE.

- Вручную

  - Для ускорения доступа к строкам пользователи могут

  создать неуникальные индексы

Команда CREATE INDEX

- Создание индекса по одному или нескольким столбцам.

```
CREATE INDEX index
ON table (column[, column]...);
```

- Увеличение скорости доступа по столбцу ENAME таблицы EMP.

```
SQL> CREATE INDEX emp_ename_idx
     ON emp(ename);
Index created.
```

Проверка индексов

- Представление словаря данных USER_INDEXES содержит имя индекса и информацию о его уникальности.
- Представление словаря данных USER_IND_COLUMNS содержит имя индекса, имя таблицы и имя столбца.

```
SELECT ic.index_name, ic.column_name,
       ic.column_position col_pos,
  ix.uniqueness
FROM   user_indexes ix, user_ind_columns ic
WHERE  ic.index_name = ix.index_name
AND    ic.table_name = 'EMP';
```

Индексы, основанные на функции

- Это индексы, основанные на каком-либо выражении.
- Выражение может строиться на основе значений столбцов, констант, функций SQL или пользовательских функций.

```
SQL> CREATE TABLE test (col1 number);

SQL> CREATE INDEX test_index on test (col1,
  col1+10);

SQL> SELECT col1+10 FROM test;
```

Удаление индекса

- Удаление индекса

```
SQL> DROP INDEX index;
```

- Удаление индекса EMP_ENAME_IDX

```
SQL> DROP INDEX emp_ename_idx;
Index dropped.
```

- Для удаление индекса необходимо быть его владельцем или иметь привилегию DROP ANY INDEX.

Синонимы

Синонимы (альтернативные имена объектов) упрощают доступ к объектам:

- Позволяют обращаться к таблицам других пользователей.
- Устраняют необходимость использования длинных имен объектов.

```
CREATE [PUBLIC] SYNONYM synonym
FOR object;
```

Создание и удаление синонимов

- Создание более короткого имени для представления DEPT_SUM_VU.

```
SQL> CREATE SYNONYM d_sum
     FOR dept_sum_vu;
Synonym created.
```

- Удаление синонима.

```
SQL> DROP SYNONYM d_sum;
Synonym dropped.
```

Привилегии

- Безопасность базы данных
  - Безопасность системы
  - Безопасность данных
- Системные привилегии: получение доступа к базе данных.
- Объектные привилегии: манипулирование содержимым объектов базы данных.
- Схема: совокупность объектов, владельцем которых является пользователь

Создание пользователей

- Команда создания пользователя – CREATE USER:

```
CREATE USER user IDENTIFIED BY
  password;
```

```
SQL> CREATE USER scott IDENTIFIED BY
  tiger;
User created.
```

Системные привилегии пользователя

- Сразу после создания пользователя АБД может предоставить ему конкретные системные привилегии.

```
GRANT privilege [, privilege...]
 TO user [, user...];
```

- Разработчик приложения может иметь следующие системные привилегии:
  - CREATE SESSION
  - CREATE TABLE
  - CREATE SEQUENCE
  - CREATE VIEW
  - CREATE PROCEDURE

Предоставление системных привилегий

- Сразу после создания пользователя АБД может предоставить ему конкретные системные привилегии.

```
SQL> GRANT create table, create
  sequence, create view
      TO    scott;
Grant succeeded.
```

Создание роли

```
SQL> CREATE ROLE manager
Role created.

SQL> GRANT create table, create view
     TO    manager;
Grant succeeded.

SQL> GRANT manager to BLAKE, CLARK;
Grant succeeded.
```

Изменение пароля пользователя

Используется команда ALTER USER

```
SQL> ALTER USER scott IDENTIFIED BY
  lion;
User altered.
```

Объектные привилегии

Привилегии на объект	Таблица	Представление	Последов.	Процедура
ALTER	•		•	
DELETE	•	•		
EXECUTE				•
INDEX	•			
INSERT	•	•		
REFERENCES	•			
SELECT	•	•	•	
UPDATE	•	•		

Объектные привилегии

- Объектные привилегии разные для разных типов данных.
- Владелец объекта имеет все привилегии на этот объект.
- Владелец может предоставлять конкретные привилегии на принадлежащий ему объект.

```
GRANT object_priv [(columns)]
ON object
TO {user|role|PUBLIC}
[WITH GRANT OPTION];
```

Предоставление объектных привилегий

- Предоставление привилегии на выполнение запросов к таблице EMP.

```
SQL> GRANT select
     ON emp
     TO sue, rich;
Grant succeeded.
```

- Предоставление привилегий пользователю и роли на обновление конкретных столбцов.

```
SQL> GRANT update (dname, loc)
     ON dept
     TO scott, manager;
Grant succeeded.
```

WITH GRANT OPTION и PUBLIC

- Предоставление полномочий пользователю на передачу привилегий.

```
SQL> GRANT select, insert
     ON dept
     TO scott;
     WITH GRANT OPTION
```

- Предоставление разрешения всем пользователям БД на выборку данных из таблицы DEPT, принадлежащей пользователю Alice.

```
SQL> GRANT select
     ON alice.dept
     TO PUBLIC;
```

Проверка предоставленных привилегий

Таблица БД	Описание
ROLE_SYS_PRIVS	Системные привилегии, предоставленные ролям.
ROLE_TAB_PRIVS	Объектные привилегии, предоставленные ролям.
USER_ROLE_PRIVS	Роли, доступные пользователю.
USER_TAB_PRIVS_MADE	Объектные привилегии, предоставленные пользователем на его объекты
USER_TAB_PRIVS_RECD	Объектные привилегии, предоставленные пользователю.
USER_COL_PRIVS_MADE	Привилегии, предоставленные пользователем на столбцы его объектов.
USER_COL_PRIVS_RECD	Привилегии на столбцы чужих объектов, предоставленные пользователю

Отмена объектных привилегий

- Для отмены привилегий, прелоставленных другим пользователям, используется команда REVOKE.
- Одновременно отменяются привилегии, предоставленные другим пользователям посредством опции WITH GRANT OPTION.

```
REVOKE {privilege [, privilege...]|ALL}
ON object
FROM {user [, user...]|role|PUBLIC}
[CASCADE CONSTRAINTS];
```

Отмена объектных привилегий

- Отмена пользователем ALICE привилегий SELECT и INSERT, предоставленных пользователю SCOTT на таблицу DEPT.

```
SQL> REVOKE select, insert
     ON dept
     FROM scott;
Revoke succeeded.
```

Расширение SQL

Соединения SQL:1999

- Тип соединения указывается явно в классе FROM
- Предикаты соединения могут быть указаны в классе ON, выделенном из класса WHERE
- Типы соединений:

- Пересекающееся
- Естественное
- Эквивалентное (равное) и класс USING
- Внешнее (полное, левое, правое)

Пересекающееся соединение

Эквивалентно декартову произведению двух таблиц

```
SELECT c.country_name
,    r,region_name
From countries c
     CROSS JOIN
     regions r;
```

Естественное соединение

- Объединение по эквивалентности, основанное на всех столбцах, у которых одинаковые наименования
- Соединяющиеся столбцы должны иметь совместимые данные
- Для соединяющихся столбцов нельзя использовать префикс псевдонима (или имени таблицы): ORA-25155

Пример естественного соединения

```
SELECT department_id, location_id
,    city, country_id
From      departments NATURAL JOIN
     locations;
```

Эквивалентное соединение и класс USING

- Отдельно от естественного соединения можно создать правильное эквивалентное соединение с классом USING
- Не должно быть префикса на столбцы в классе USING: ORA-25154
- Ключевые слова NATURAL и USING взаимоисключающие

Пример класса USING

```
SELECT e.employee_id, e.last_name
,    d.location_id
From      employees e JOIN
     departments d USING (department_id)
Where rownum < 6
```

Предикаты соединения и класс ON

- Используются для разделения предикатов соединения от других предикатов
- Класс ON позволяет использовать любые предикаты, включая подзапросы и логические операторы

```
Select e.employee_id, e.last_name
,    d.department_id, d.location_id
From  employees e JOIN
     departments d ON
     (e.department_id = d.department_id)
Where     manager_id = 102;
```

Трехстороннее соединение с классом ON

```
SELECT d.department_name
,    l.city, c.country_name
From      departments d
JOIN      locations l ON
          (d.location_id = l.location_id)
JOIN      countries c ON
          (l.country_id = c.country_id)
Where     c.region_id = 1;
```

Внешние соединения

- Типы: LEFT, RIGHT, FULL
- Больше внушительности и наглядности, чем оператор (+)

TYPE DESCR

2	D2
3	D3
4	D4

PROD TYPE

P1	1
P2	2
P3	3
P4	3

Пример внешнего соединения

```
SELECT p.prod, p.type
,    t.type, t.descr
From      p {LEFT|RIGHT|FULL} OUTER JOIN
          t ON (p.type = t.type);
```

LEFT

P1	1		
P2	2	2	D2
P3	3	3	D3
P4	3	3	D3
		4	D4

RIGHT

Усовершенствованные CASE выражения

4 типа в SQL:1999:

- Простое
- Поисковое
- NULLIF
- COALESCE

Простое CASE выражение

- Аналогично функции DECODE
- Поиск и замена значений внутри выражения

```
Select e.last_name
,    (CASE extract (year from e.hire_date)
     WHEN 1996 THEN ‘ 5 years of service’
     WHEN 1991 THEN ‘10  years of service’
     WHEN 1986 THEN ‘15 years of service’
          ELSE ‘              may be next year!’
     END) as “Awards for 2004”
From employees e;
```

Поисковое CASE выражение

- Аналогично конструкции IF…THEN …ELSE
- Поиск по условию и замена значений внутри выражения

```
Select e.first_name, e.last_name, e.job_id
,    (CASE
     WHEN e.job_id LIKE ‘AD%’ THEN ’10%’
     WHEN e.job_id LIKE ‘IT%’ THEN ’15%’
     WHEN e.first_name = ‘Lex’ THEN ’18%’
                         ELSE ‘ 0%’
     END) as “Raise”
From employees e;
```

NULLIF и COALESCE

```
NULLIF (expr1, expr2) ⇔
CASE WHEN expr1 = expr2
    THEN NULL
    ELSE expr1
END
```

```
COALESCE (expr1, expr2, expr3, …) ⇔
CASE WHEN expr1 IS NOT NULL
    THEN expr1
    ELSE COALESCE(expr2, expr3, …)
END
```

Скалярные подзапросы

- Возвращают одну строку с одним значением столбца
- Ограниченная поддержка в Oracle8i
- В Oracle9i разрешены в любом месте, где может быть использовано скалярное выражение
- Тип данных возвращаемого значения должен совпадать со значением, выбранным в подзапросе

Пример скалярного подзапроса

```
Select d.department_name
,    (select count(*)
     from employees e
     where e.department_id =
          d.department_id)  as empcount
From departments d;
```

Явный DEFAULT

```
Insert into employees
(employee_id, first_name, department_id)
Values (1, 'Scott', DEFAULT);

Update employees
Set    department_id = DEFAULT
Where department_id = 10;
```

Команда MERGE

Известна также как “upsert”

- Производит обновление, если строки выполнены, в противном случае выполняется вставка
- Важна для приложений в хранилище данных
- Лучшая производительность, требуется меньше сканирований команд и таблиц-источников

```
MERGE INTO t1
    USING t2 ON (join_predicate)
WHEN MATCHED THEN UPDATE SET …
WHEN NOT MATCHED THEN INSERT (…) VALUES (…)
```

Фраза GROUPING SETS

- надмножество GROUP BY {ROLLUP|CUBE}
- Выдает единственный результат, который эквивалентен приблизительно UNION ALL

```
Select time_id, channel_id, prod_id
    sum(amount_sold) as amount
From sales
Group by GROUPING SETS
    ((time_id, channel_id, prod_id)
    , (time_id, channel_id)
    , (channel_id, prod_id)
    );
```

Составные столбцы

Обработка группы столбцов как одной единицы:

- GROUP BY ROLLUP (a, b ,c) возвращает 4 группы
- GROUP BY ROLLUP (a, (b ,c)) возвращает 3 группы

```
Select …
From sales
Where …
Group by rollup
    ((prod_id, (channel_id, time_id));
```

Связанные группы

- Перемножение многочисленных групп
- Задаются перечислением многочисленных GROUPNG SETS, CUBE, ROLLUP

```
Select prod_id, channel_id, time_id
    sum(amount_sold)
From sales
Where …
Group by prod_id
    Cube (channel_id)
    Rollup (time_id);
```

Класс WITH

- Наименование блока запроса в команде SELECT для ссылки, если несколько блоков в запросе
- Класс WITH может содержать многочисленные блоки в запросе, разделенные запятыми
- Реализованы как встроенные представления или временные таблицы

Пример класса WITH

```
WITH summary as (
Select d.department_name
,    sum(e.salary) as dept_total
From employees e, departments d
Where e.department_id = d.department_id
Group by department_name)
Select department_name, dept_total
From summary
Where dept_total >
    (select sum(dept_total) * 1/8
    from summary)
Order by dept_total desc;
```