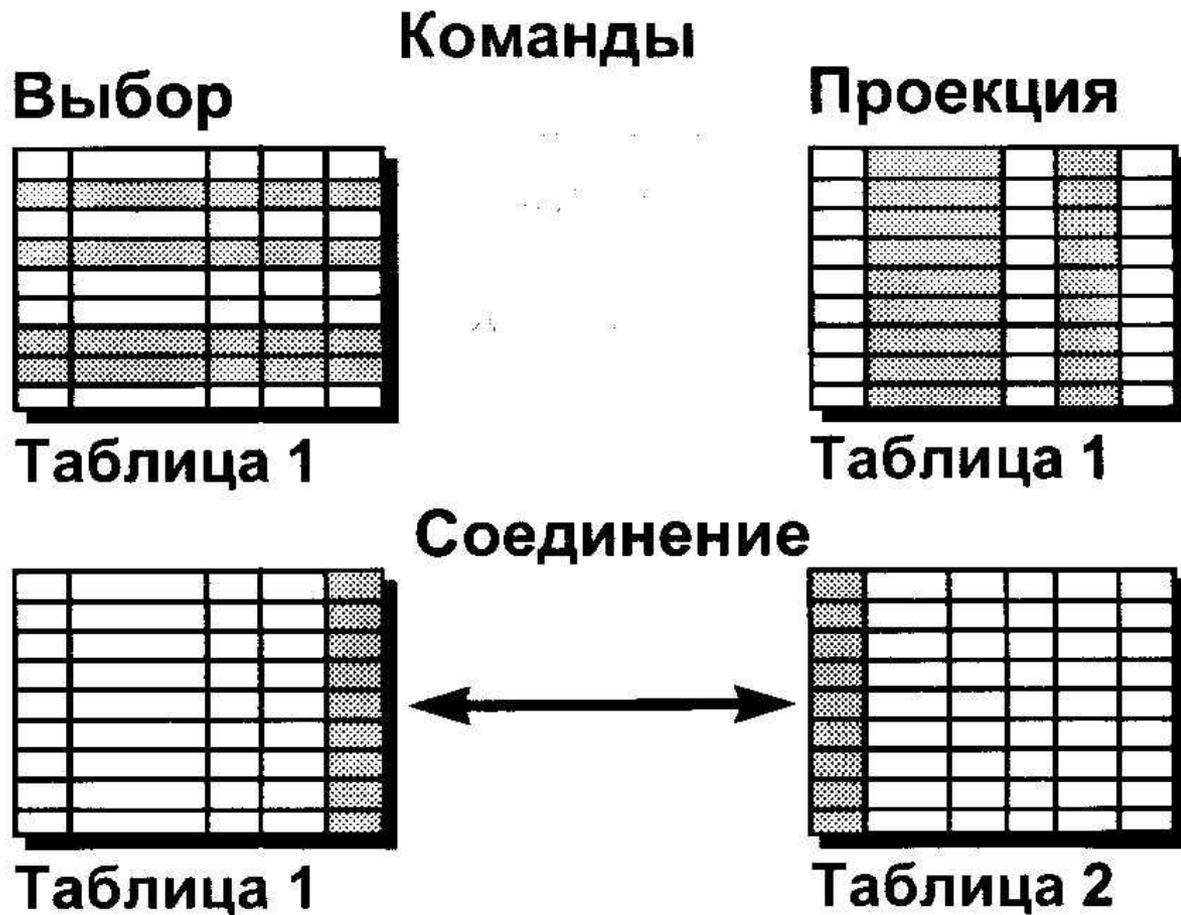


# Язык SQL

Реализация Oracle 10g

# Возможности команды SELECT языка SQL



# Основная команда SELECT

- SELECT [DISTINCT] {\*, column [*alias*],...)
- FROM *table*;
- SELECT указывает, *какие* столбцы
- FROM указывает, из *какой* таблицы

# 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

# Выбор конкретных столбцов

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

DEPTNO	LOC
10	NEW YORK
20	DALLAS
30	CHICAGO
40	BOSTON

- **Арифметические выражения**
- **Создаются из данных типа NUMBER и DATE с помощью арифметических операторов**

# Использование арифметических операторов

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

ENAME	SAL	SAL+300
KING	5000	5300
BLAKE	2850	3150
CLARK	2450	2750
JONES	2975	3275
MARTIN	1250	1550
ALLEN	1600	1900

...

14 rows selected.

# Неопределенное значение (NULL)

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

```
SQL> select ename NAME, 12*sal+comm  
2  from    emp  
3  WHERE   ename='KING';
```

NAME	12*SAL+COMM
-----	-----
KING	

# Псевдоним (алиас) столбца

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

# Использование псевдонимов столбцов

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

NAME	SALARY
-----	-----
...	

```
SQL> SELECT ename "Name",
2          sal*12 "Annual Salary"
3 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

# Дублирование строк

- По умолчанию выдаются все строки, включая дубликаты.

```
SQL> SELECT deptno  
2 FROM emp;
```

```
DEPTNO  
-----  
10  
30  
10  
20  
...  
14 rows selected.
```

# Устранение строк-дубликатов

- Дубликаты устраняются с помощью ключевого слова **DISTINCT** в команде **SELECT**.

```
SQL> SELECT DISTINCT deptno  
2 FROM emp;
```

DEPTNO
10
20
30

# Ограничение количества выбираемых строк путем отбора

**EMP**

EMPNO	ENAME	JOB	...	DEPTNO
7839	KING	PRESIDENT		10
7698	BLAKE	MANAGER		30
7782	CLARK	MANAGER		10
7566	JONES	MANAGER		20
...				

"...выбрать всех  
служащих отдела  
10"



**EMP**

EMPNO	ENAME	JOB	...	DEPTNO
7839	KING	PRESIDENT		10
7782	CLARK	MANAGER		10
7934	MILLER	CLERK		10

# Ограничение количества выбираемых строк

- Количество возвращаемых строк можно ограничить с помощью предложения **WHERE**.

```
SELECT          [DISTINCT] {*, column [alias], ...}  
FROM           table  
[WHERE        condition(s)];
```

- Предложение **WHERE** следует за предложением **FROM**.

# Использование предложения WHERE

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

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

# Прочие операторы сравнения

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

# Использование оператора BETWEEN

- Оператор BETWEEN используется для вывода строк по диапазону значений.

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

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

↑  
Нижняя  
граница

↑  
Верхняя  
граница

# Использование оператора IN

- Оператор IN используется для проверки на входжение значений в список.

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

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

# Использование оператора LIKE

- Оператор LIKE используется для поиска символьных значений по шаблону с метасимволами.
- Условия поиска могут включать алфавитные и цифровые символы.
- % обозначает ноль или много символов
- \_ обозначает один символ

```
SQL> SELECT   ename  
2 FROM      emp  
3 WHERE     ename LIKE 'S%';
```

# Использование оператора LIKE

- Метасимволы можно комбинировать.

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

```
ENAME
```

```
-----  
JAMES
```

```
WARD
```

# 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.

# Предложение ORDER BY

- Предложение ORDER BY используется для сортировки строк
  - ASC: сортировка по возрастанию (используется по умолчанию)
  - DESC: сортировка по убыванию
- В команде SELECT предложение ORDER BY указывается последним.

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

ENAME	JOB	DEPTNO	HIREDATE
SMITH	CLERK	20	17-DEC-80
ALLEN	SALESMAN	30	20-FEB-81

...

14 rows selected.

# Сортировка по нескольким столбцам

- Последовательность сортировки определяется порядком столбцов в предложении ORDER BY.

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

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

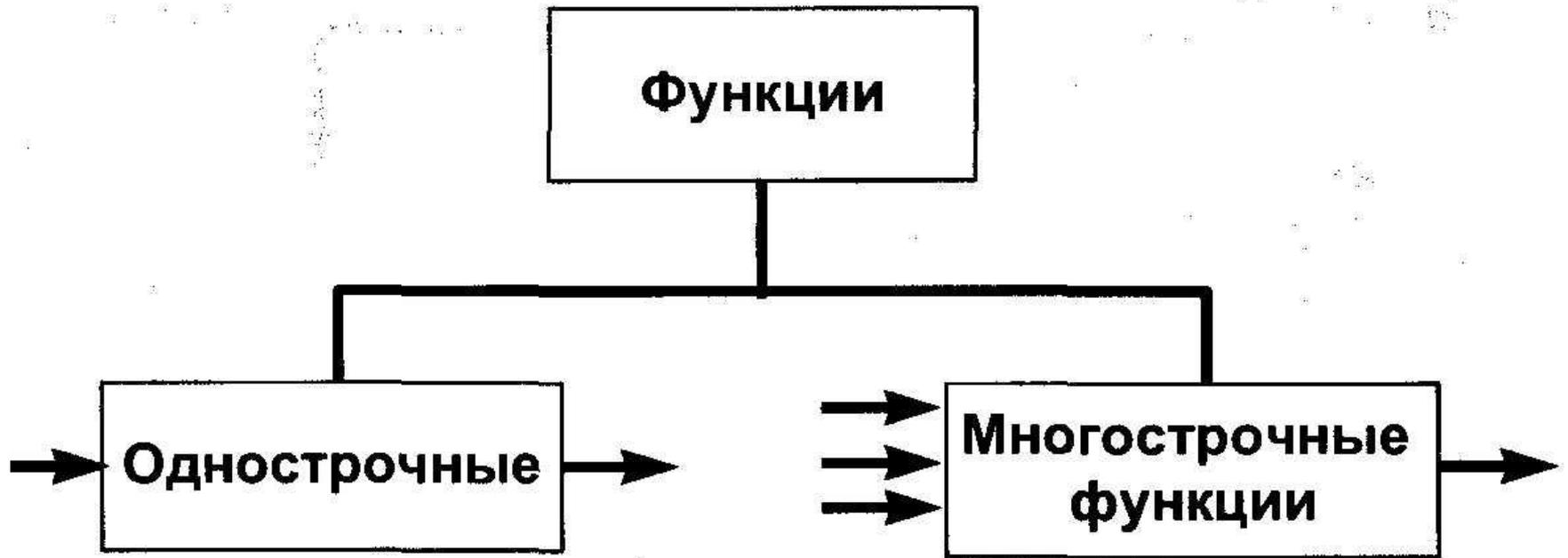
14 rows selected.

- Можно сортировать по столбцу, не входящему в список SELECT.

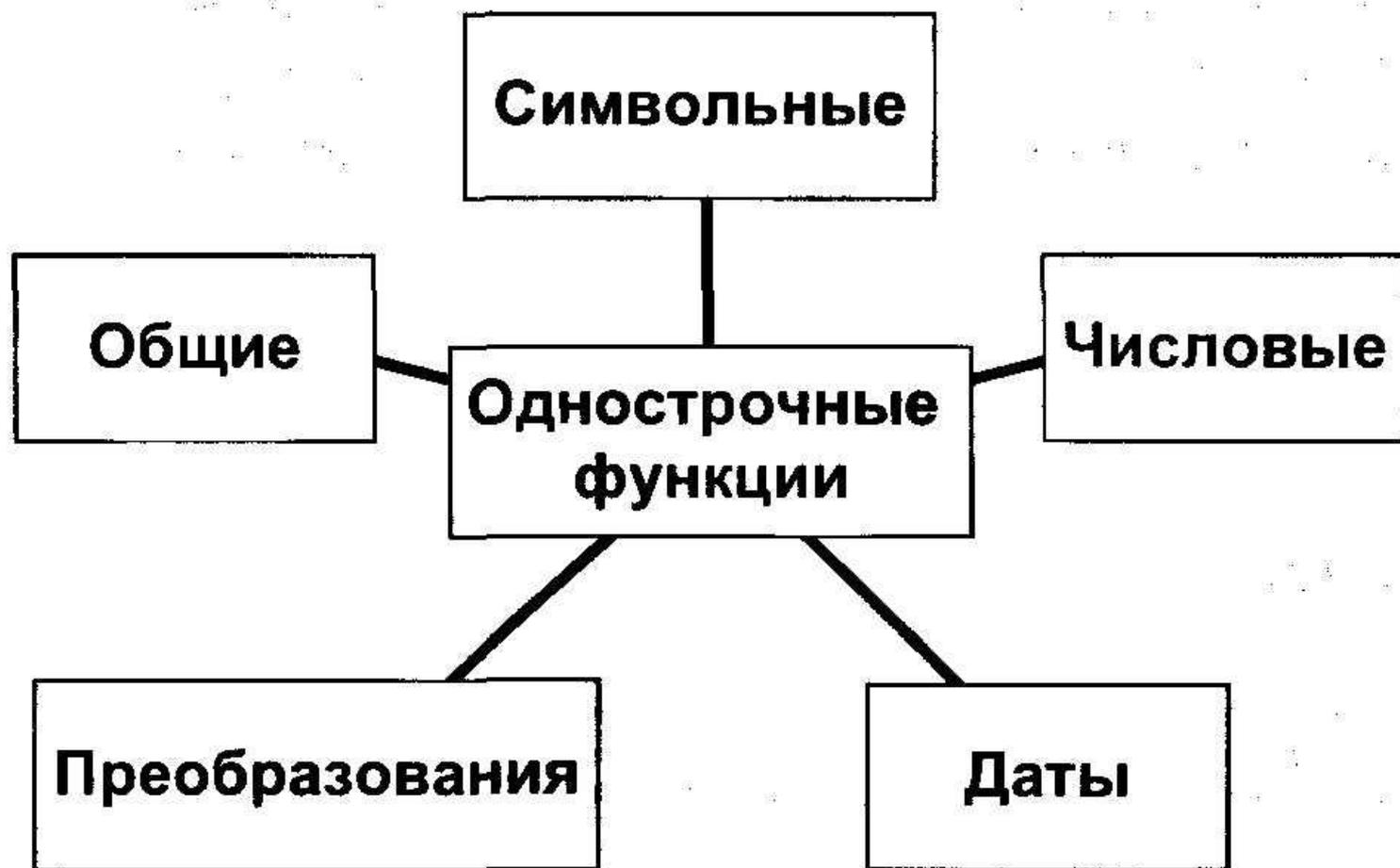
# Однострочные функции

- **Различные типы функций в SQL.**
- **Включение в команды SELECT функций различных типов — символьных, числовых и типа "дата".**
- **Функции преобразования данных и их использование.**

# Два типа функций SQL



# Однострочные функции



# Символьные функции

Символьные  
функции

Функции преобразования  
регистра символов

LOWER

UPPER

INITCAP

Функции манипулирования  
символами

CONCAT

SUBSTR

LENGTH

INSTR

LPAD

# Использование функций преобразования регистра

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

```
SQL> SELECT empno, ename, deptno  
2 FROM emp  
3 WHERE ename = 'blake';  
no rows selected
```

```
SQL> SELECT empno, ename, deptno  
2 FROM emp  
3 WHERE LOWER(ename) = 'blake';
```

EMPNO	ENAME	DEPTNO
7698	BLAKE	30

# Использование функций манипулирования символами

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

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

# Числовые функции

- **ROUND:** Округляет 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

**MOD(1600, 300) → 100**

# Использование функции ROUND

```
SQL> SELECT ROUND (45.923,2) , ROUND (45.923,0) ,  
2         ROUND (45.923,-1)  
3 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),  
2          TRUNC(45.923,-1)  
3 FROM DUAL;
```

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

# Работа с датами

- • Oracle хранит данные во внутреннем цифровом формате.
- - Век, год, месяц, число, часы, минуты, секунды
- По умолчанию дата выдается в формате DD-MON-YY (число- месяц-год)
- Функция SYSDATE возвращает текущие дату и время
- DUAL - это фиктивная таблица, используемая для просмотра SYSDATE

# 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.

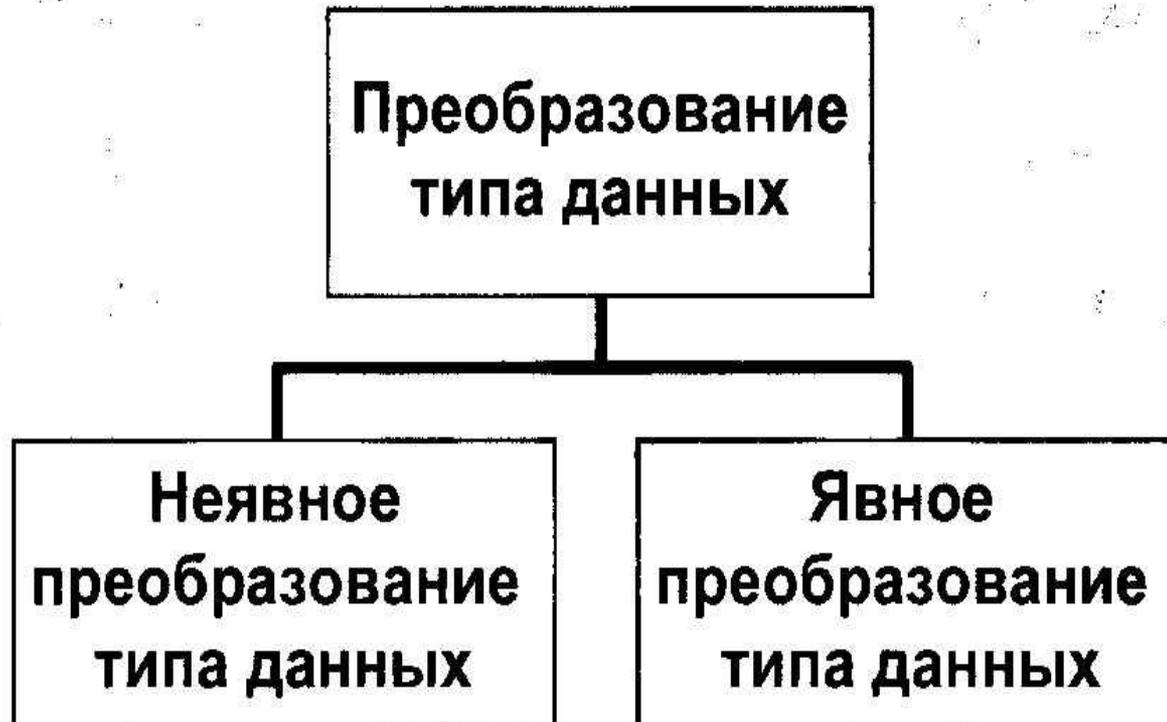
# Использование арифметических операторов с датами

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

ENAME	WEEKS
KING	830.93709
CLARK	853.93709
MILLER	821.36566

- **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'**

# Функции преобразования

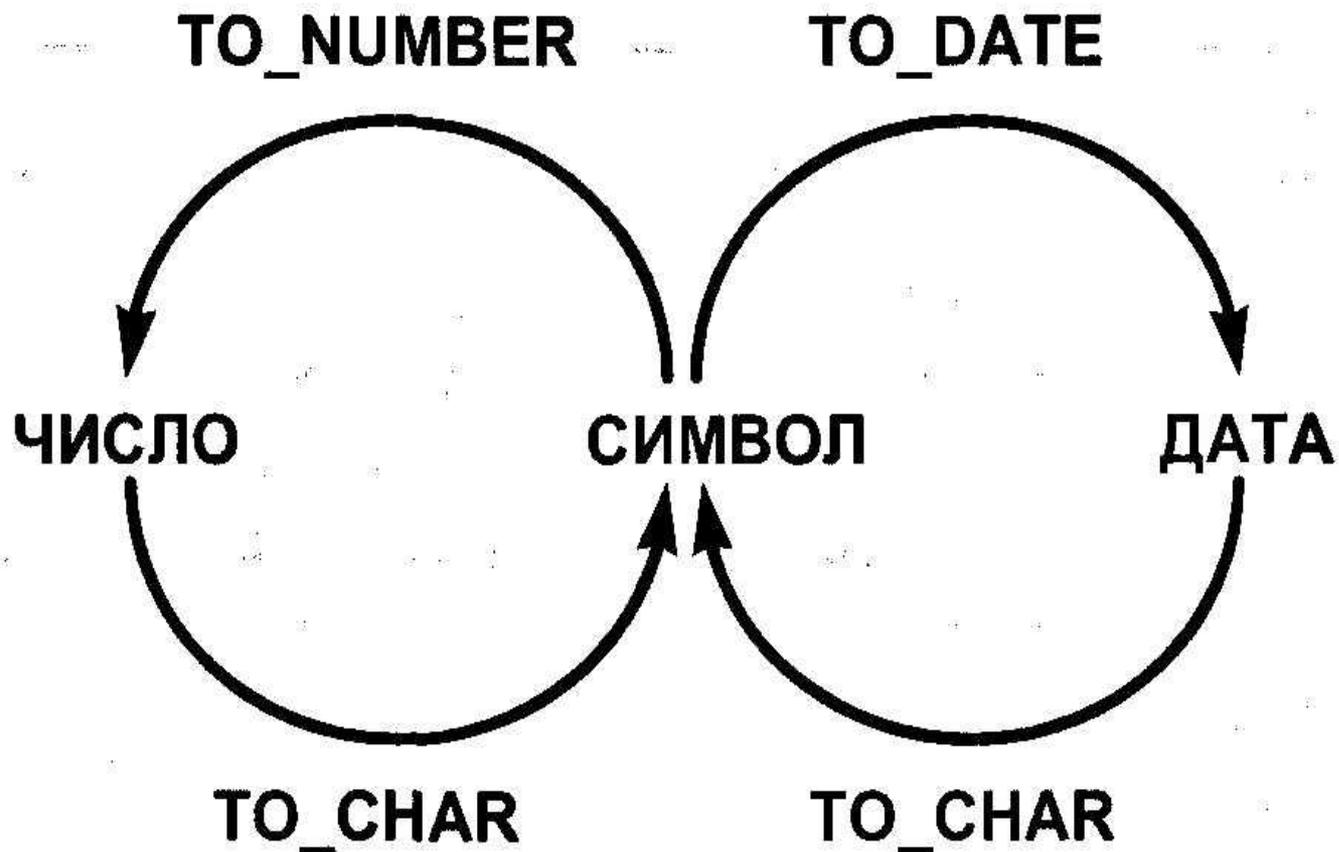


# Неявное преобразование ТИПОВ ДАННЫХ

- Для операций присваивания Oracle может автоматически выполнять следующие преобразования:

Исходный формат	Новый формат
VARCHAR2 или CHAR	NUMBER
VARCHAR2 или CHAR	DATE
NUMBER	VARCHAR2
DATE	VARCHAR2

# Явное преобразование типов данных



# Функция TO\_CHAR с датами

```
TO_CHAR(date, 'fmt')
```

**Модель формата:**

- Должна быть заключена в апострофы. Различает символы верхнего и нижнего регистров. • Может включать любые разрешенные элементы формата даты. • Использует элемент *fm* для удаления конечных пробелов и ведущих нулей.
- Отделяется от значения даты запятой.

# Элементы формата даты

<b>YYYY</b>	Полный год цифрами
<b>YEAR</b>	Год прописью
<b>MM</b>	Двузначное цифровое обозначение месяца
<b>MONTH</b>	Полное название месяца
<b>DY</b>	Трехзначное алфавитное сокращенное название дня недели
<b>DAY</b>	Полное название дня недели

# Элементы модели формата даты

- Элементы, которые задают формат части даты, обозначающей время.

<b>HH24:MI:SS AM</b>
----------------------

<b>15:45:32 PM</b>
--------------------

- Символьные строки добавляются в кавычках.

<b>DD "of" MONTH</b>
----------------------

<b>12 of OCTOBER</b>
----------------------

- Числовые суффиксы используются для вывода числительных прописью.

<b>ddspth</b>
---------------

<b>fourteenth</b>
-------------------

# Функция TO\_CHAR с числами

```
TO_CHAR(number, 'fmt')
```

Форматы, используемые с функцией TO\_CHAR для вывода символьного значения в виде числа

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

# Функция TO\_CHAR с числами

```
SQL> SELECT TO_CHAR(sal, '$99,999') SALARY  
2 FROM emp  
3 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 NVL

Преобразует неопределенное значение в действительное

- Используемые типы данных - DATE, символьные (CHARACTER) и числовые (NUMBER).
- Типы данных должны совпадать
  - NVL(comm,0)
  - NVL(hiredate,'01-JAN-97')
  - NVL(job,'No Job Yet')

# Использование функции NVL

```
SQL> SELECT ename, sal, comm, (sal*12)+NVL(comm,0)
2 FROM emp;
```

ENAME	SAL	COMM	(SAL*12)+NVL(COMM,0)
KING	5000		60000
BLAKE	2850		34200
CLARK	2450		29400
JONES	2975		35700
MARTIN	1250	1400	16400
ALLEN	1600	300	19500

...  
14 rows selected.

# Функция DECODE

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

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

# Использование функции DECODE

```
SQL> SELECT job, sal,  
2          DECODE(job, 'ANALYST', SAL*1.1,  
3                'CLERK', SAL*1.15,  
4                'MANAGER', SAL*1.20,  
5                SAL)  
6          REVISED_SALARY  
7 FROM emp;
```

JOB	SAL	REVISED_SALARY
PRESIDENT	5000	5000
MANAGER	2850	3420
MANAGER	2450	2940

...  
14 rows selected.

# Вложенные функции

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

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

- **Выборка данных из нескольких таблиц**

# Темы

- Команды **SELECT** для выборки данных из более, чем одной таблицы с помощью эквисоединений и прочих видов соединений.
- Использование внешних соединений для просмотра данных, не удовлетворяющих обычным условиям соединения
- Соединение таблицы с собой

# Выборка данных из нескольких таблиц

EMP

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

DEPT

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

↓ ↓

EMPNO	DEPTNO	LOC
7839	10	NEW YORK
7698	30	CHICAGO
7782	10	NEW YORK
7566	20	DALLAS
7654	30	CHICAGO
7499	30	CHICAGO
...	...	...

14 rows selected.

# Что такое соединение?

- Соединение используется для выборки данных из более, чем одной таблицы.
- Условие соединения указывается в предложении **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 произведения образуется , если:
- Опущено условие соединения.
- Условие соединения недействительно.
- Все строки первой таблицы соединяются со всеми строками второй таблицы.
- Во избежание получения декартова произведения предложение **WHERE** всегда должно включать допустимое условие соединения.

# Получение декартова произведения

EMP (14 rows)

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

DEPT (4 rows)

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

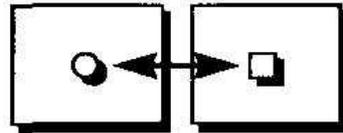
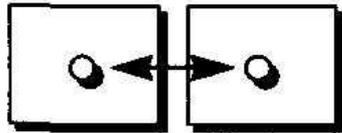
“Декартово  
произведение:  
14\*4=56 строк”



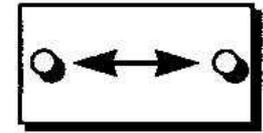
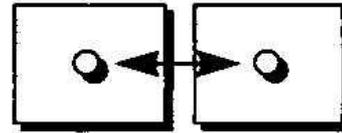
ENAME	DNAME
KING	ACCOUNTING
BLAKE	ACCOUNTING
...	
KING	RESEARCH
BLAKE	RESEARCH
...	
56 rows selected.	

# Виды соединений

Эквисоединение



Внешнее  
соединение



Не-эквисоединение



Соединение  
таблицы с  
собой





# Выборка записей с помощью ЭКВИСОЕДИНЕНИЙ

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

EMPNO	ENAME	DEPTNO	DEPTNO	LOC
7839	KING	10	10	NEW YORK
7698	BLAKE	30	30	CHICAGO
7782	CLARK	10	10	NEW YORK
7566	JONES	20	20	DALLAS

...  
14 rows selected.

# 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.

# Дополнительные условия поиска с оператором AND

EMP

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

...

14 rows selected.

DEPT

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

...

14 rows selected.

# Псевдонимы таблиц

Использования псевдонимов таблиц упрощает запросы.

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

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

# Соединение более, чем двух таблиц

## CUSTOMER

NAME	CUSTID
-----	-----
JOCKSPORTS	100
TKB SPORT SHOP	101
VOLLYRITE	102
JUST TENNIS	103
K+T SPORTS	105
SHAPE UP	106
WOMENS SPORTS	107
...	
9 rows selected.	

## ORD

CUSTID	ORDID
-----	-----
101	610
102	611
104	612
106	601
102	602
106	
106	
21 rows	

## ITEM

ORDID	ITEMID
-----	-----
610	3
611	1
612	1
601	1
602	1
64 rows selected.	

# Не-эквисоединения

**EMP**

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

...

14 rows selected.

**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 низким и высоким окладами в таблице SALGRADE”

# Выборка записей с помощью НЕ-ЭКВИСОЕДИНЕНИЙ

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

ENAME	SAL	GRADE
JAMES	950	1
SMITH	800	1
ADAMS	1100	1

....

14 rows selected.

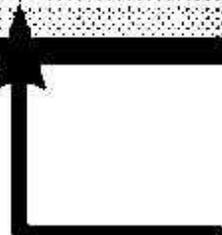
# Внешние соединения

**EMP**

ENAME	DEPTNO
-----	-----
KING	10
BLAKE	30
CLARK	10
JONES	20
...	

**DEPT**

DEPTNO	DNAME
-----	-----
10	ACCOUNTING
30	SALES
10	ACCOUNTING
20	RESEARCH
...	
40	OPERATIONS



**В отделе OPERATIONS  
служащих нет**

# 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
  2  FROM      emp e, dept d
  3  WHERE     e.deptno(+) = d.deptno
  4  ORDER BY e.deptno;
```

ENAME	DEPTNO	DNAME
-----	-----	-----
KING	10	ACCOUNTING
CLARK	10	ACCOUNTING
...		
	40	OPERATIONS

15 rows selected.

# Соединение таблицы с собой

**EMP (WORKER)**

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

**EMP (MANAGER)**

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



**"MGR в таблице WORKER равен EMPNO в таблице  
MANAGER"**

# Соединение таблицы с собой

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

```
WORKER. ENAME | | 'WORKSFOR' | | MANAG  
-----
```

```
BLAKE works for KING  
CLARK works for KING  
JONES works for KING  
MARTIN works for BLAKE  
...  
13 rows selected.
```

- **Агрегирование данных с помощью групповых функций**

# Темы

- Общие сведения об имеющихся групповых функциях
- Использование групповых функций
- Вывод данных по группам с помощью предложения GROUP BY
- Включение и исключение групп с помощью предложения HAVING

# Что такое групповые функции?

Групповые функции работают с множеством строк и возвращают один результат на группу.

EMP

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

“максимальный  
оклад в  
таблице EMP”

MAX (SAL)

5000

# Типы групповых функций

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

# Использование групповых функций

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

# Использование функций AVG и SUM

Функции AVG и SUM применяются к столбцам с числовыми данными

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

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

# Использование функций MIN и MAX

Функции MAX и MIN применяются к данным любого типа

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

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

# Использование функции COUNT

**COUNT(\*)** возвращает количество строк в таблице

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

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

```
6
```

# Использование функции COUNT

**COUNT(expr)** возвращает количество строк с определенными значениями (не NULL)

```
SQL> SELECT COUNT(comm)
2 FROM emp
3 WHERE deptno = 30;
```

```
COUNT (COMM)
```

```
-----
```

```
4
```

# Создание групп данных

EMP

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

2916.6667

2175

1566.6667

“средний  
оклад  
в таблице  
EMP  
по каждому  
отделу”

DEPTNO	AVG (SAL)
10	2916.6667
20	2175
30	1566.6667

# Создание групп данных: предложение 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 список SELECT и к которым не применяются групповые функции, должны быть указаны в предложении GROUP BY.

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

DEPTNO	AVG(SAL)
10	2916.6667
20	2175
30	1566.6667

# Использование предложения GROUP BY

Столбец, указанный в GROUP BY, может отсутствовать в списке SELECT.

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

```
AVG (SAL)
-----
2916.6667
      2175
1566.6667
```

# Группировка по нескольким столбцам

EMP

DEPTNO	JOB	SAL
10	MANAGER	2450
10	PRESIDENT	5000
10	CLERK	1300
20	CLERK	800
20	CLERK	1100
20	ANALYST	3000
20	ANALYST	3000
20	MANAGER	2975
30	SALESMAN	1600
30	MANAGER	2850
30	SALESMAN	1250
30	CLERK	950
30	SALESMAN	1500
30	SALESMAN	1250

“просуммировать  
оклады в  
таблице EMP  
по каждой  
должности  
внутри каждого  
отдела”

DEPTNO	JOB	SUM(SAL)
10	CLERK	1300
10	MANAGER	2450
10	PRESIDENT	5000
20	ANALYST	6000
20	CLERK	1900
20	MANAGER	2975
30	CLERK	950
30	MANAGER	2850
30	SALESMAN	5600

# Использование предложения GROUP BY

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

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

9 rows selected.

# Недействительные запросы с групповыми функциями

Все столбцы и выражения из списка SELECT, не являющиеся групповой функцией, должны быть включены в предложение GROUP BY.

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

```
SELECT deptno, COUNT(ename)
```

```
*
```

```
ERROR at line 1:
```

```
ORA-00937: not a single-group group function
```

В предложении GROUP BY недостает столбца

# Недействительные запросы с групповыми функциями

- Предложение WHERE для исключения групп не используется.
- Для исключения некоторых групп следует пользоваться предложением HAVING

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

```
WHERE AVG(sal) > 2000
*
ERROR at line 3:
ORA-00934: group function is not allowed here
```

# Исключение групп

EMP

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

5000

3000

2850

“максимальный  
оклад  
в отделе  
превышает  
\$2900”

DEPTNO	MAX(SAL)
10	5000
20	3000

# Исключение групп: предложение HAVING

Для исключения групп пользуйтесь предложением  
HAVING

- Строки группируются.
- Применяется групповая функция.
- Выводятся группы, удовлетворяющие условию в предложении HAVING.

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

# Использование предложения HAVING

```
SQL> SELECT deptno, max(sal)
2 FROM emp
3 GROUP BY deptno
4 HAVING max(sal)>2900;
```

DEPTNO	MAX(SAL)
10	5000
20	3000

# Использование предложения HAVING

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

JOB	PAYROLL
ANALYST	6000
MANAGER	8275

# Вложенные групповые функции

Вывод максимального среднего оклада.

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

```
MAX(AVG(SAL))  
-----  
      2916.6667
```

# Заключение

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

**Последовательность оценки предложений:**

- **WHERE**
- **GROUP BY**
- **HAVING**

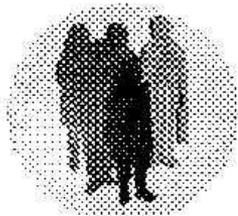
# Подзапросы

- **Типы проблем, решаемых с помощью подзапросов**
- **Определение подзапросов**
- **Типы подзапросов**
- **Написание однострочных и многострочных подзапросов**

# Использование подзапроса для решения проблемы

“У кого оклад больше, чем у Джонса?”

Главный запрос



“У кого из служащих оклад больше, чем у Джонса?”

Подзапрос



“Каков оклад Джонса?”



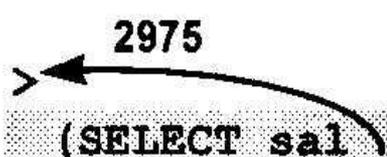
# Подзапросы

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

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

# Использование подзапроса

```
SQL> SELECT  ename
2 FROM      emp
3 WHERE     sal >
4           (SELECT sal
5            FROM   emp
6            WHERE  empno=7566) ;
```

A diagram consisting of a curved arrow pointing from the value '2975' to the '>' operator in the WHERE clause of the main query. The subquery text '(SELECT sal FROM emp WHERE empno=7566)' is highlighted with a grey background.

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



# Однострочные подзапросы

- Возвращают только одну строку
- Используют однострочные операторы сравнения

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

# Выполнение однострочных подзапросов

```
SQL> SELECT  ename, job
2 FROM      emp
3 WHERE     job =
4           (SELECT  job
5            FROM    emp
6            WHERE   empno = 7369)
7 AND      sal >
8           (SELECT  sal
9            FROM    emp
10           WHERE   empno = 7876);
```

CLERK

1100

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

# Использование групповых функций в подзапросах

```
SQL> SELECT  ename, job, sal
2 FROM      emp
3 WHERE     sal =
4           (SELECT  MIN(sal)
5            FROM      emp);
```

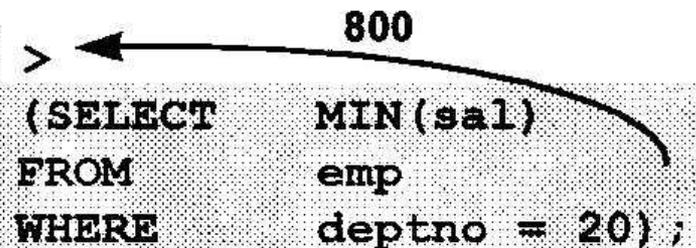


ENAME	JOB	SAL
-----	-----	-----
SMITH	CLERK	800

# Предложение HAVING с подзапросами

- Сервер Oracle сначала выполняет подзапрос.
- Сервер Oracle возвращает результаты в предложение HAVING главного запроса.

```
SQL> SELECT      deptno, MIN(sal)
2  FROM          emp
3  GROUP BY     deptno
4  HAVING       MIN(sal) >
5              (SELECT  MIN(sal)
6              FROM    emp
7              WHERE   deptno = 20);
```



The diagram illustrates the execution of the HAVING clause. A curved arrow points from the value '800' to the comparison '>' in the HAVING clause, indicating that the subquery result is used to evaluate the condition.

# Что неправильно в этой команде?

```
SQL> SELECT empno, ename
2 FROM emp
3 WHERE sal =
4         (SELECT MIN(sal)
5 FROM emp
6 GROUP BY deptno);
```

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

no rows selected
```

# Будет ли выполнена эта команда?

```
SQL> SELECT ename, job
2 FROM emp
3 WHERE job =
4           (SELECT job
5            FROM emp
6            WHERE ename='SMYTHE');
```

```
no rows selected
```

Подзапрос не возвращает никаких значений

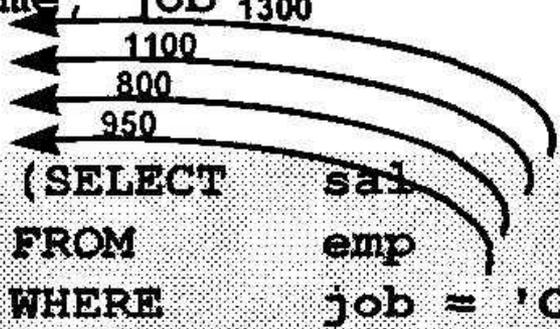
# Многострочные запросы

- 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

# Использование оператора ANY в многострочных подзапросах

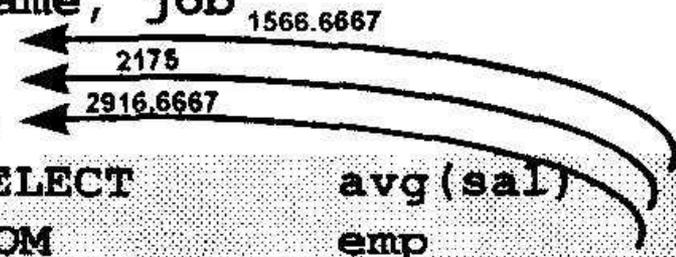
```
SQL> SELECT empno, ename, job 1300
2 FROM emp 1100
3 WHERE sal < ANY 800
4 (SELECT sal 950
5 FROM emp
6 WHERE job = 'CLERK')
7 AND job <> 'CLERK';
```



EMPNO	ENAME	JOB
7654	MARTIN	SALESMAN
7521	WARD	SALESMAN

# Использование оператора ALL в многострочных подзапросах

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



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

# Заключение

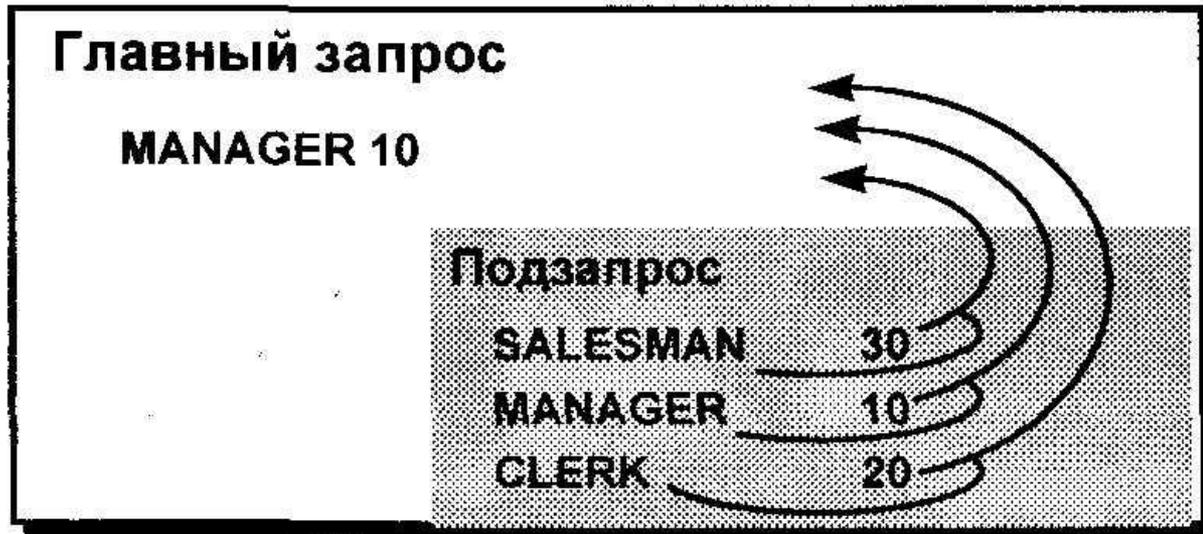
Подзапросы полезны, когда запрос основан на неизвестных значениях.

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

# Многостолбцовые подзапросы

- **Создание многостолбцовых подзапросов**
- **Поведение подзапросов при выборке неопределенных значений**
- **Включение подзапроса в предложение FROM**

# Многостолбцовые подзапросы



Главный запрос  
производит  
сравнение

с

значениями из многострочного  
и многостолбцового  
подзапроса

---

MANAGER 10

SALESMAN 30  
MANAGER 10  
CLERK 20

# Использование многостолбцовых подзапросов

Вывод фамилии, номера отдела, оклада и комиссионных всех служащих, оклад и комиссионные которых совпадают как с окладом, так и с комиссионными одного и того же служащего в отделе 30.

```
SQL> SELECT  ename, deptno, sal, comm
 2  FROM      emp
 3  WHERE     (sal, NVL(comm, -1)) IN
 4            (SELECT sal, NVL(comm, -1)
 5             FROM emp
 6             WHERE deptno = 30);
```

# Сравнения столбцов

## Парное

SAL		COMM
1600	↔	300
1250	↔	500
1250	↔	1400
2850		
1500	↔	0
950		

## Непарное

SAL		COMM
1600	↔	300
1250	↔	500
1250	↔	1400
2850	↔	
1500	↔	0
950		

## Подзапрос с непарным сравнением

Вывод фамилии, номера отдела, оклада и комиссионных всех служащих, оклад и комиссионные которых совпадают с комиссионными и окладом любого служащего в отделе 30.

```
SQL> SELECT  ename, deptno, sal, comm
2 FROM      emp
3 WHERE     sal IN      (SELECT sal
4                               FROM   emp
5                               WHERE  deptno = 30)
6 AND
7           NVL(comm, -1) IN (SELECT NVL(comm, -1)
8                               FROM   emp
9                               WHERE  deptno = 30);
```

## Изменение таблицы EMP

- Предположим, что изменяются оклад и комиссионные Кларка (Clark).
- Новый оклад - до 1500 долларов, а новые комиссионные - до 300 долларов.

ENAME	SAL	COMM
...		
CLARK	1500	300
...		
ALLEN	1600	300
TURNER	1500	0
...		

14 rows selected.

The diagram shows a table with columns ENAME, SAL, and COMM. The rows are CLARK, ALLEN, and TURNER. The original values for CLARK are SAL=1600 and COMM=300. The new values are SAL=1500 and COMM=300. Arrows indicate the change from 1600 to 1500 in SAL and from 300 to 300 in COMM.

## Подзапрос с парным сравнением

```
SQL> SELECT  ename, deptno, sal, comm
2  FROM      emp
3  WHERE     (sal, NVL(comm, -1)) IN
4           (SELECT sal, NVL(comm, -1)
5            FROM   emp
6            WHERE  deptno = 30);
```

ENAME	DEPTNO	SAL	COMM
JAMES	30	950	
WARD	30	1250	500
MARTIN	30	1250	1400
TURNER	30	1500	0
ALLEN	30	1600	300
BLAKE	30	2850	

6 rows selected.

# Подзапрос с непарным сравнением

```
SQL> SELECT   ename,deptno, sal, comm
  2  FROM      emp
  3  WHERE     sal IN      (SELECT sal
  4                                     FROM      emp
  5                                     WHERE     deptno = 30)
  6  AND
  7           NVL(comm,-1) IN (SELECT NVL(comm,-1)
  8                                     FROM      emp
  9                                     WHERE     deptno = 30);
```

ENAME	DEPTNO	SAL	COMM
JAMES	30	950	
BLAKE	30	2850	
TURNER	30	1500	0
CLARK	10	1500	300
...			

7 rows selected.

# Неопределенные значения в подзапросе

```
SQL> SELECT  employee.ename
  2  FROM    emp employee
  3  WHERE   employee.empno NOT IN
  4          (SELECT manager.mgr
  5           FROM   emp manager);
no rows selected.
```

## Использование подзапроса в предложении FROM

```
SQL> SELECT a.ename, a.sal, a.deptno, b.salavg
2 FROM emp a, (SELECT deptno, avg(sal) salavg
3 FROM emp
4 GROUP BY deptno) b
5 WHERE a.deptno = b.deptno
6 AND a.sal > b.salavg;
```

ENAME	SAL	DEPTNO	SALAVG
KING	5000	10	2916.6667
JONES	2975	20	2175
SCOTT	3000	20	2175

...

6 rows selected.

# Заключение

- **Многостолбцовый подзапрос возвращает значения нескольких столбцов.**
- **Сравнение столбцов в многостолбцовых подзапросах может быть парным и непарным.**
- **Многостолбцовый подзапрос может также использоваться в предложении FROM команды SELECT.**

# Манипулирование данными

- **Описание команд DML**
- **Вставка строк в таблицы**
- **Обновление строк в таблице**
- **Удаление строк из таблицы**
- **Управление транзакциями**

# Язык манипулирования данными (DML)

- • Команды DML выполняются при следующих операциях:
- Вставка новых строк в таблицу
- Изменение существующих строк в таблице
- Удаление существующих строк из таблицы
- \* *Транзакция* - это совокупность команд DML, образующих логическую единицу работы.

# Вставка новой строки в таблицу

50	DEVELOPMENT	DETROIT
----	-------------	---------

Новая строка

DEPT

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

“...вставка новой строки  
в таблицу DEPT ...”



DEPT

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

# Вставка новых строк в таблицу: пример

- Вставка новой строки, содержащей значения для каждого из столбцов.
- Значения указываются в стандартном порядке столбцов таблицы (используемом по умолчанию).
- Перечисление столбцов в предложении INSERT необязательно.

```
SQL> INSERT INTO dept (deptno, dname, loc)
      2 VALUES (50, 'DEVELOPMENT', 'DETROIT');
1 row created.
```

- Символьные значения и даты заключаются в апострофы.

# Вставка строк с неопределенными значениями

- Неявный метод: столбец не указывается в списке столбцов.

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

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

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

# Вставка специальных значений

Функция **SYSDATE** записывает текущие дату и время.

```
SQL> INSERT INTO emp (empno, ename, job,  
2 mgr, hiredate, sal, comm,  
3 deptno)  
4 VALUES (7196, 'GREEN', 'SALESMAN',  
5 7782, SYSDATE, 2000, NULL,  
6 10);  
1 row created.
```

# Вставка конкретных значений даты и времени

- Добавление нового служащего

```
SQL> INSERT INTO emp
  2  VALUES      (2296, 'AROMANO', 'SALESMAN', 7782,
  3                TO_DATE('FEB 3, 97', 'MON DD, YY'),
  4                1300, NULL, 10);
1 row created.
```

- Проверка вставки.

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

## Вставка значений с помощью переменных подстановки

Создайте интерактивный скрипт-файл с помощью параметров подстановки SQL\*Plus.

```
SQL> INSERT INTO      dept (deptno, dname, loc)
  2  VALUES          (&department_id,
  3                  '&department_name', '&location');
```

```
Enter value for department_id: 80
Enter value for department_name: EDUCATION
Enter value for location: ATLANTA

1 row created.
```

# Команда UPDATE

- Для обновления существующих строк используется команда UPDATE.

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

- В случае необходимости можно одновременно обновлять несколько строк

# Обновление строк в таблице: пример

- Предложение **WHERE** позволяет изменить конкретную строку или строки.

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

- Если предложение **WHERE** отсутствует, обновляются все строки таблицы.

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

## Обновление с помощью многостолбцового подзапроса

Изменение должности и номера отдела служащего под номером 7698, чтобы они стали такими же, как у служащего под номером 7499.

```
SQL> UPDATE emp
  2 SET      (job, deptno) =
  3          (SELECT job, deptno
  4          FROM emp
  5          WHERE empno = 7499)
  6 WHERE empno = 7698;
1 row updated.
```

## Обновление строк на основе значений из другой таблицы

Для изменения строк таблицы на основе значений из другой таблицы используйте подзапросы в командах UPDATE.

```
SQL> UPDATE   employee
  2  SET      deptno = (SELECT   deptno
  3                               FROM     emp
  4                               WHERE    empno = 7788)
  5  WHERE    job      = (SELECT   job
  6                               FROM     emp
  7                               WHERE    empno = 7788);
2 rows updated.
```

# Обновление строк: нарушение правила целостности данных

```
SQL> UPDATE emp
      2 SET deptno = 55
      3 WHERE deptno = 10;
```

```
UPDATE emp
```

```
*
```

```
ERROR at line 1:
```

```
ORA-02291: integrity constraint (USR.EMP_DEPTNO_FK)
violated - parent key not found
```

Отдела номер 55 не существует

# Удаление строки из таблицы

DEPT

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON
50	DEVELOPMENT	DETROIT
60	MIS	
...		

“...удаление строки  
из таблицы DEPT ...”

DEPT



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

# Команда DELETE

Для удаления строк используется команда DELETE.

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

# Удаление строк из таблицы: пример

- Конкретная строка или строки удаляются с помощью предложения **WHERE**.

```
SQL> DELETE FROM      department
      2  WHERE          dname = 'DEVELOPMENT';
1 row deleted.
```

- Если предложение **WHERE** отсутствует, удаляются все строки таблицы.

```
SQL> DELETE FROM      department;
4 rows deleted.
```

## Удаление строк на основе значений из другой таблицы

Для удаления строк на основе значений из другой таблицы используйте подзапросы в командах DELETE.

```
SQL> DELETE FROM      employee
  2  WHERE              deptno =
  3                    (SELECT  deptno
  4                      FROM    dept
  5                      WHERE   dname = 'SALES');
  6 rows deleted.
```

# Удаление строк: нарушение правила целостности

```
SQL> DELETE FROM dept
      2 WHERE deptno = 10;
```

```
DELETE FROM dept
```

```
*
```

```
ERROR at line 1:
```

```
ORA-02292: integrity constraint (USR.EMP_DEPTNO_FK)
violated - no record found
```

Нельзя удалить строку, содержащую  
первичный ключ, который используется в  
качестве внешнего ключа в другой таблице.

# Транзакции базы данных

- Сервер Oracle обеспечивает согласованность данных на основе транзакций.
- Транзакции обеспечивают большую гибкость, более широкий спектр средств управления при изменении данных, а также согласованность данных в случае ошибки в пользовательском процессе или сбоя системы.

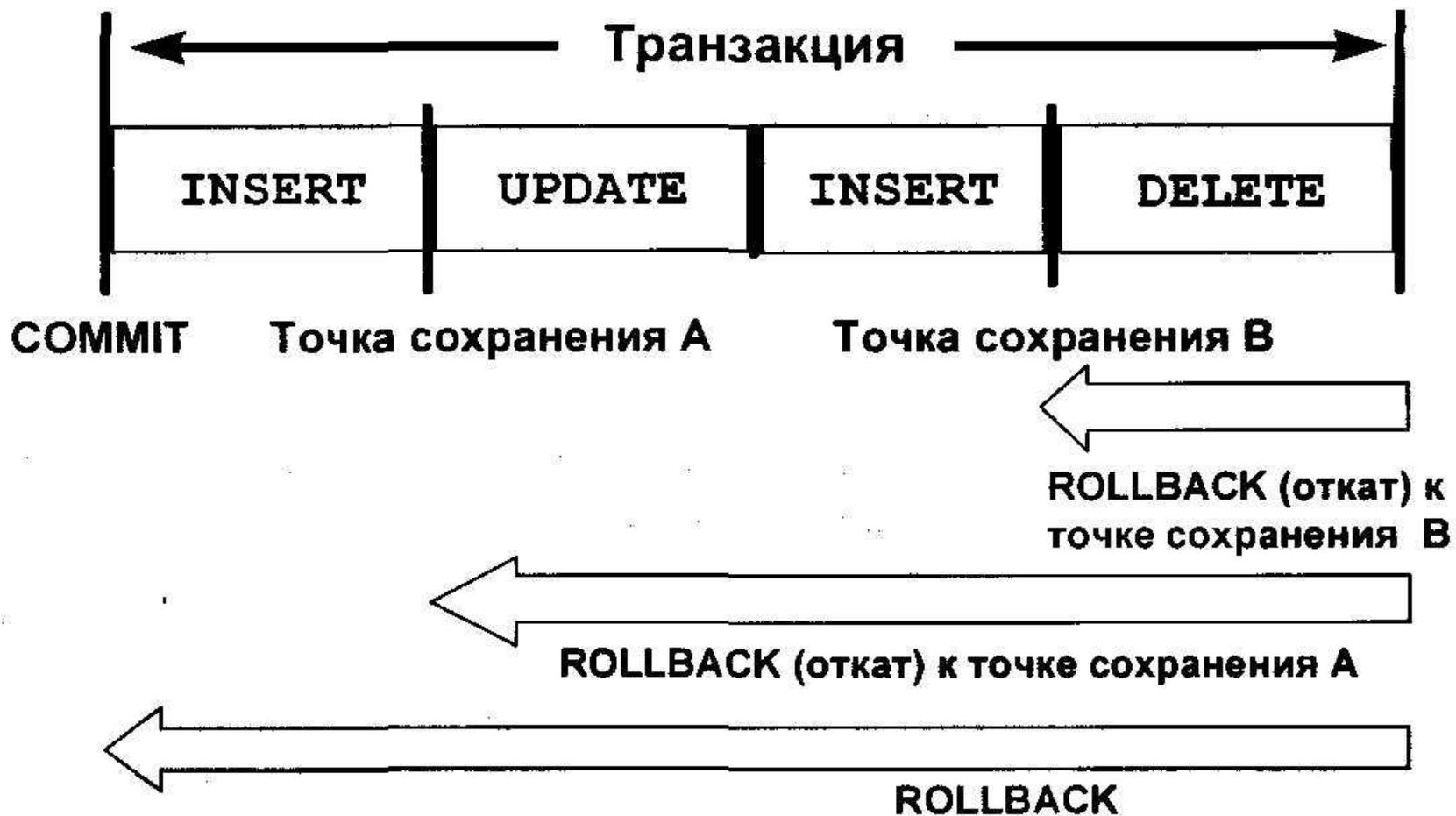
# Транзакции базы данных

- **Начинаются с выполнения первой исполняемой команды SQL.**
- **Заканчиваются одним из следующих событий:**
  - **Выполнение команды COMMIT или ROLLBACK**
  - **Выполнение команды DDL или DCL (автоматическая фиксация транзакции)**
  - **Выходы в подпрограммы пользователя**
  - **Отказы системы**

# **Преимущества команд COMMIT и ROLLBACK**

- **Обеспечивают согласованность данных.**
- **Позволяют проверить изменения в данных прежде, чем сделать их постоянными.**
- **Логически группируют взаимосвязанные операции.**
-

# Управление транзакциями



# Неявная обработка транзакций

- **Автоматическая фиксация изменений (COMMIT) происходит в следующих случаях:**
  - **Выполнение команды DDL**
  - **Выполнение команды DCL**
  - **Нормальный выход из SQL\*Plus без явной посылки команды COMMIT или ROLLBACK**
- **Автоматический откат (ROLLBACK) выполняется в случае аварийного прекращения сеанса работы в SQL\*Plus или отказа системы**

# Состояние данных до выполнения команды COMMIT или ROLLBACK

- Предыдущее состояние данных может быть восстановлено, т.к. изменения производятся в буфере базы данных.
- Текущий пользователь может просмотреть результаты своих операций DML с помощью команды SELECT.
- Другие пользователи не *могут* видеть , результаты команд DML, выполняемых текущим пользователем.
- Изменяемые строки *блокируются*, и другие пользователи не могут обновлять их содержимое.

# **Состояние данных после выполнения команды COMMIT**

- **Измененные данные записываются в базу данных.**
- **Предшествующее состояние данных теряется.**
- **Все пользователи могут видеть результаты.**
- **Измененные строки разблокируются, и другие пользователи получают доступ к ним для обработки данных.**
- **Все точки сохранения стираются.**

# Фиксация изменений в данных

- **Внесение изменений.**

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

- **Фиксация изменений.**

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

# Состояние данных после выполнения команды ROLLBACK

Команда ROLLBACK отменяет все незавершенные изменения.

- Изменения в данных отменяются.
- Данные возвращаются в прежнее состояние.
- Блокировка строк, над которыми выполнялись операции, отменяется.

```
SQL> DELETE FROM      employee;  
14 rows deleted.  
SQL> ROLLBACK;  
Rollback complete.
```

# Откат к точке сохранения

- Маркеры (точки сохранения) для отката в текущей транзакции создаются с помощью команды **SAVEPOINT**.
- Откат до такого маркера выполняется с помощью команды **ROLLBACK TO SAVEPOINT**.

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

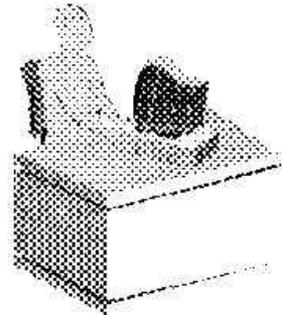
# **Откат на уровне команды**

- **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**

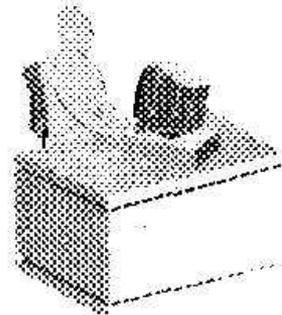
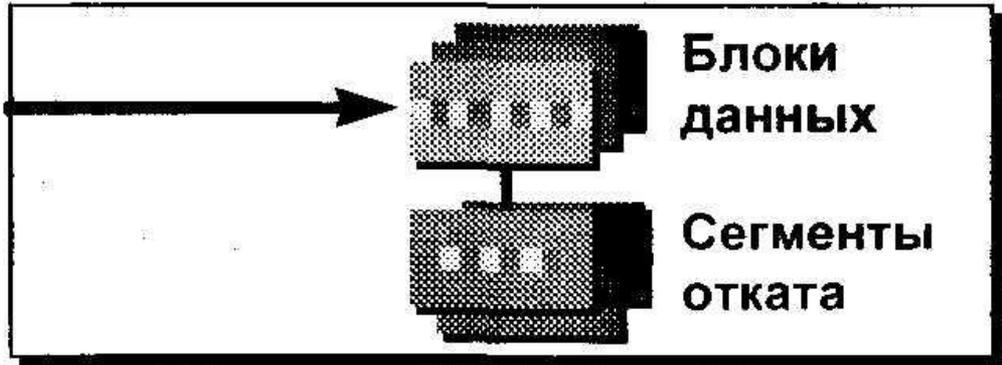
- **Согласованность чтения гарантирует непротиворечивое представление данных в любой момент времени.**
- **Изменения, сделанные одним пользователем, не вступают в противоречие с изменениями, сделанными другим пользователем.**
- **Гарантируется, что для одних и тех же данных:**
  - **“Читатели” никогда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

# Реализация согласованности чтения



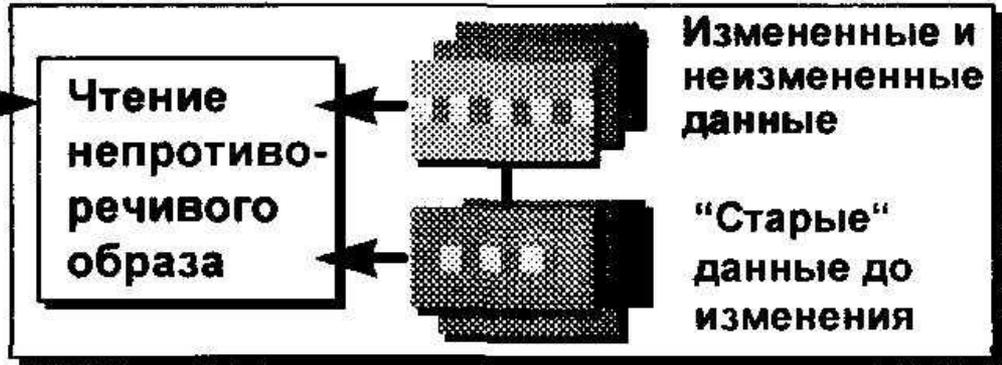
```
UPDATE emp  
SET sal = 2000  
WHERE ename =  
  'SCOTT';
```

Пользователь А



```
SELECT *  
FROM emp;
```

Пользователь В



# Блокировка данных

## Блокировка данных в Oracle:

- Предотвращает деструктивное взаимодействие между одновременными транзакциями
- Не требует действий со стороны пользователя
- Автоматически использует ограничение самого низкого уровня
- Сохраняется до конца транзакции
- Имеет два основных режима:
  - Исключительная блокировка (*Exclusive*)
  - Разделяемая блокировка (*Share*)

# Заключение

<b>Команда</b>	<b>Описание</b>
<b>INSERT</b>	<b>Вставляет новую строку в таблицу.</b>
<b>UPDATE</b>	<b>Изменяет существующие строки таблицы.</b>
<b>DELETE</b>	<b>Удаляет существующие строки из таблицы.</b>
<b>COMMIT</b>	<b>Делает все незафиксированные изменения постоянными.</b>
<b>SAVEPOINT</b>	<b>Позволяет произвести откат до определенной точки сохранения.</b>
<b>ROLLBACK</b>	<b>Отменяет все незафиксированные изменения данных.</b>

# Создание таблиц и управление ими

- Главные объекты базы данных
- Создание таблиц
- Типы данных, которые могут использоваться в определениях столбцов
- Изменение определений таблиц
- Удаление, переименование и усечение таблиц

# Объекты базы данных

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

# Правила присвоения имен

- **Имя должно начинаться 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 этому же самому 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]);
```

- Вы задаете:
  - Имя таблицы
  - Имя столбца, тип данных столбца и размер столбца, значение по умолчанию

# **Ссылки на таблицы других пользователей**

- Таблицы, принадлежащие другим пользователям, не входят в схему пользователя.**
- В качестве префикса в имени таблицы следует указать имя владельца.**

# Опция DEFAULT

- **Задаёт значение по умолчанию, если при вставке данных значение не указано явно.**

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

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

# Создание таблиц

- Создание таблицы.

```
SQL> CREATE TABLE dept
2      (deptno NUMBER(2) ,
3      dname  VARCHAR2(14) ,
4      loc    VARCHAR2(13)) ;
Table created.
```

- Проверка создания таблицы.

```
SQL> DESCRIBE dept
```

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

# Запрос к словарю данных

- Вывод описаний таблиц, принадлежащих пользователю.

```
SQL> SELECT *  
2 FROM user_tables;
```

- Просмотр типов объектов, принадлежащих пользователю.

```
SQL> SELECT DISTINCT object_type  
2 FROM user_objects;
```

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

```
SQL> SELECT *  
2 FROM user_catalog;
```

# Типы данных

Тип данных	Описание
<b>VARCHAR2(size)</b>	Символьные данные переменной длины
<b>CHAR(size)</b>	Символьные данные постоянной длины
<b>NUMBER(p,s)</b>	Числовые данные переменной длины
<b>DATE</b>	Значения даты и времени
<b>LONG</b>	Символьные данные переменной длины до 2 гигабайтов
<b>CLOB</b>	Однобайтовые символьные данные до 4 гигабайтов
<b>RAW и LONG RAW</b>	Необработанные ("сырые") двоичные данные
<b>BLOB</b>	Двоичные данные до 4 гигабайтов
<b>BFILE</b>	Двоичные данные, которые хранятся во внешнем файле; длина до 4 гигабайтов

## Создание таблицы с использованием подзапроса

- Создание таблицы и вставка строк путем совместного использования команды **CREATE TABLE** и опции **“AS subquery”**.

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

- Количество заданных столбцов должно совпадать с количеством столбцов в подзапросе.
- Столбцы могут быть определены с именами и значениями, используемыми по умолчанию.

# Команда ALTER TABLE

Команда ALTER TABLE используется для следующих операций:

- Добавление столбца
- Изменение существующего столбца
- Задание значения по умолчанию для нового столбца

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

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

# Добавление столбца

**DEPT30**

**Новый столбец**

EMPNO	ENAME	ANNSAL	HIREDATE	JOB
7698	BLAKE	34200	01-MAY-81	
7654	MARTIN	15000	28-SEP-81	
7499	ALLEN	19200	20-FEB-81	
7844	TURNER	18000	08-SEP-81	
...				

“...вставка столбца в таблицу DEPT30...”



**DEPT30**

EMPNO	ENAME	ANNSAL	HIREDATE	JOB
7698	BLAKE	34200	01-MAY-81	
7654	MARTIN	15000	28-SEP-81	
7499	ALLEN	19200	20-FEB-81	
7844	TURNER	18000	08-SEP-81	
...				

## Добавление столбца

- Столбцы добавляются с помощью предложения ADD.

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

- Новый столбец становится в таблице последним.

EMPNO	ENAME	ANNSAL	HIREDATE	JOB
7698	BLAKE	34200	01-MAY-81	
7654	MARTIN	15000	28-SEP-81	
7499	ALLEN	19200	20-FEB-81	
7844	TURNER	18000	08-SEP-81	

....  
6 rows selected.

# Изменение столбца

- Вы можете изменить тип данных столбца, размер и значение, используемое по умолчанию.

```
ALTER TABLE dept30  
MODIFY      (ename VARCHAR2(15));  
Table altered.
```

- Новое значение по умолчанию влияет только на последующие вставки в таблицу.

# Удаление таблицы с помощью команды DROP

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

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

# Принципы поддержки целостности в реляционной модели данных

- поддержка *структурной целостности*
- поддержка *языковой целостности*
- поддержка *ссылочной целостности*
- поддержка *семантической целостности.*
-

# поддержка *структурной целостности*

- реляционная СУБД должна допускать работу только с однородными структурами данных типа «реляционное отношение» т.е.
- отсутствие дубликатов кортежей,
- соответственно обязательное наличие первичного ключа,
- отсутствие понятия упорядоченности кортежей.

# поддержка языковой целостности

Реляционная СУБД должна обеспечивать языки описания и манипулирования данными не ниже стандарта SQL.

- Не должны быть доступны иные низкоуровневые средства манипулирования данными, не соответствующие стандарту.

# поддержка *ссылочной целостности*

- кортежи подчиненного отношения уничтожаются при удалении кортежа основного отношения, связанного с ними.
- кортежи основного отношения модифицируются при удалении кортежа основного отношения, связанного с ними, при этом на месте ключа родительского отношений ставится неопределенное Null значение.

# *Семантическая поддержка целостности.*

- Семантическая поддержка может быть обеспечена двумя путями:
- Декларативным и
- процедурным путем.

# **Включение ограничений**

- **Ограничения обеспечивают декларативную поддержку целостности.**
- **Что такое ограничения?**
- **Создание и сопровождение ограничений**

# Что такое ограничения?

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

## **Указания по ограничениям**

- **Присваивайте имена ограничениям сами; в противном случае сервер Oracle присвоит имя в формате `SYS_Cn`.**
- **Создавайте ограничения:**
  - **При создании таблицы**
  - **После создания таблицы**
- **Устанавливайте ограничения на уровне столбца или таблицы.**
- **Просматривайте ограничения в словаре данных.**

# 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

EMP

EMPNO	ENAME	JOB	...	COMM	DEPTNO
7839	KING	PRESIDENT			10
7698	BLAKE	MANAGER			30
7782	CLARK	MANAGER			10
7566	JONES	MANAGER			20
...					

↑  
Ограничение NOT NULL  
(ни одна строка не может  
содержать неопределенное  
значение в этом столбце)

↑  
Отсутствие ограничения  
NOT NULL (любая строка  
может содержать  
неопределенное значение  
в этом столбце)

↑  
Ограничение  
NOT NULL

# Ограничение NOT NULL

Может быть задано только для столбца

```
SQL> CREATE TABLE emp (  
2     empno      NUMBER(4) ,  
3     ename      VARCHAR2(10) NOT NULL,  
4     job        VARCHAR2(9) ,  
5     mgr        NUMBER(4) ,  
6     hiredate   DATE ,  
7     sal        NUMBER(7,2) ,  
8     comm       NUMBER(7,2) ,  
9     deptno     NUMBER(7,2) NOT NULL);
```

# Ограничение UNIQUE

ограничение UNIQUE

DEPT

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

Insert into

50	SALES	DETROIT	← Не разрешается (DNAME=SALES уже существует)
60		BOSTON	← Разрешается

# Ограничение UNIQUE

Может быть задано на уровне столбца или таблицы

```
SQL> CREATE TABLE dept(  
2     deptno      NUMBER(2),  
3     dname       VARCHAR2(14),  
4     loc         VARCHAR2(13),  
5     CONSTRAINT dept_dname_uk UNIQUE(dname));
```

# Ограничение PRIMARY KEY

главный ключ

DEPT

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

Insert into

20	MARKETING	DALLAS
	FINANCE	NEW YORK

← Не разрешается  
(DEPTNO=20 уже  
существует)

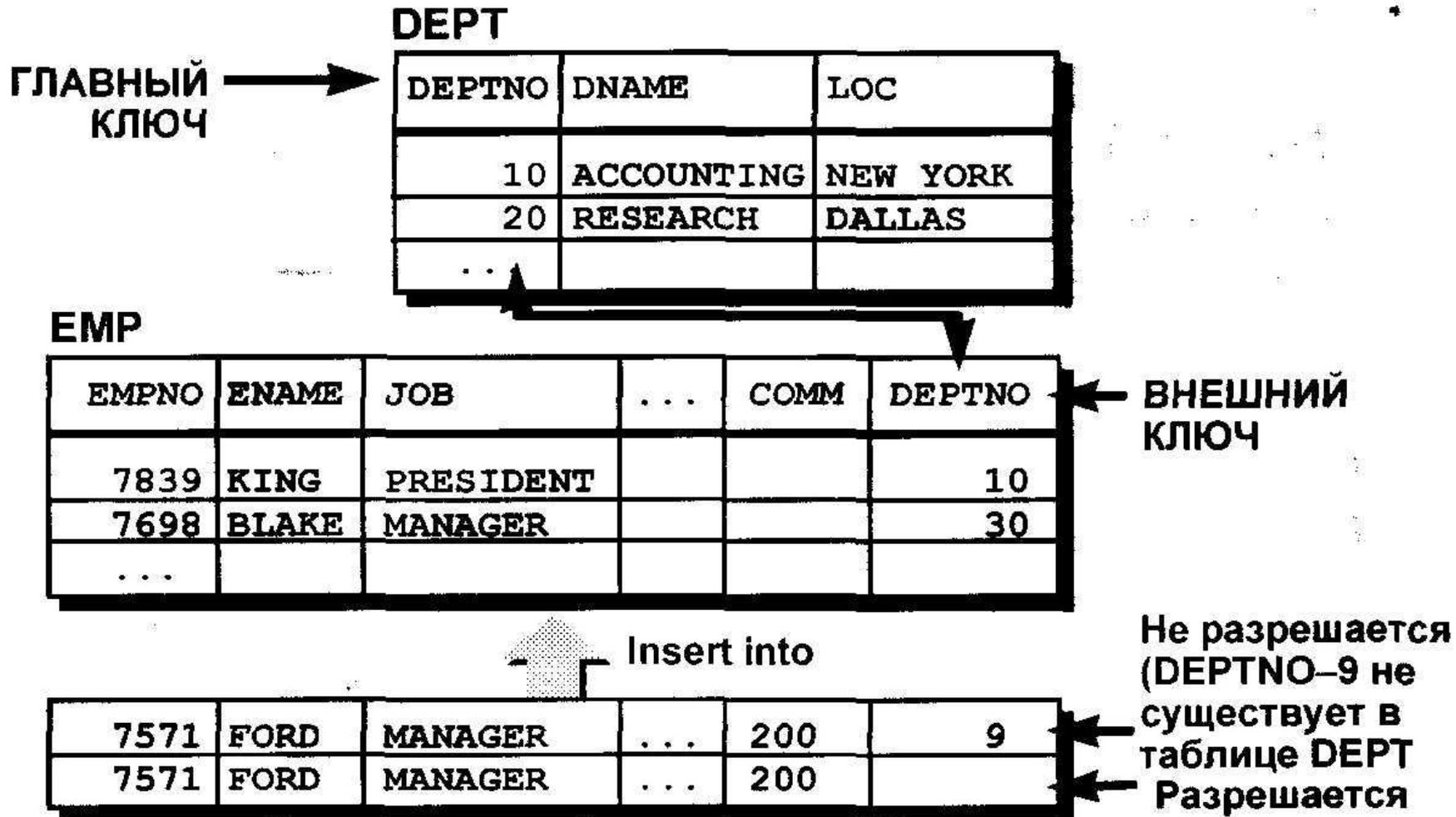
← Не разрешается  
(DEPTNO имеет  
неопределенное  
значение)

# Ограничение PRIMARY KEY

Может быть задано на уровне таблицы или столбца

```
SQL> CREATE TABLE dept(  
2     deptno      NUMBER(2),  
3     dname       VARCHAR2(14),  
4     loc         VARCHAR2(13),  
5     CONSTRAINT dept_dname_uk UNIQUE (dname),  
6     CONSTRAINT dept_deptno_pk PRIMARY KEY(deptno));
```

# Ограничение FOREIGN KEY



# Ограничение FOREIGN KEY

Может быть задано на уровне таблицы или столбца

```
SQL> CREATE TABLE emp (  
 2      empno      NUMBER(4) ,  
 3      ename      VARCHAR2(10) NOT NULL,  
 4      job        VARCHAR2(9) ,  
 5      mgr        NUMBER(4) ,  
 6      hiredate   DATE ,  
 7      sal        NUMBER(7,2) ,  
 8      comm       NUMBER(7,2) ,  
 9      deptno     NUMBER(7,2) NOT NULL,  
10      CONSTRAINT emp_deptno_fk FOREIGN KEY (deptno)  
11      REFERENCES dept (deptno));
```

# Ключевые слова ограничения **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 в родительской таблице с одновременным удалением зависимых строк в подчиненной таблице

# Пример

- Оператор создания таблицы BOOKS из базы данных «Библиотека».
- Бизнес-правила:
- *Шифр книги* — последовательность символов длиной не более 14, однозначно определяющая книгу, значит, это — фактически первичный ключ таблицы BOOKS.
- *Название книги* — последовательность символов, не более 120. Обязательно должно быть задано.
- *Автор* — последовательность символов, не более 30, может быть не задан.
- *Соавтор* — последовательность символов, не более 30, может быть не задан.
- *Год издания* — целое число, не менее 1960 и не более текущего года. По умолчанию ставится текущий год.
- *Издательство* — последовательность символов, не более 20, может отсутствовать.
- *Количество страниц* — целое число не менее 5 и не более 1000.

# Оператор

```
CREATE TABLE BOOKS
```

```
ISBN      varchar(14) NOT NULL PRIMARY KEY,
```

```
TITLE     varchar(120) NOT NULL,
```

```
AUTOR     varchar (30) NULL,
```

```
COAUTOR   varchar(30) NULL,
```

```
YEAR_PUBL smallint DEFAULT Year(GetDate()) CHECK(YEAR_PUBL >= 1960 AND  
YEAR_PUBL <= YEAR(GetDate()))),
```

```
PUBLICH   varchar(20) NULL,
```

```
PAGES     smallint CHECK(PAGES > = 5 AND PAGES <= 1000)
```

```
);
```

# Дополнительное ограничение для таблицы

```
CREATE TABLE BOOKS
(
    ISBN        varchar(14)    NOT NULL PRIMARY KEY,
    TITLE       varchar(120)   NOT NULL,
    AUTHOR      varchar (30)    NULL,
    COAUTHOR    varchar(30)    NULL,
    YEAR_PUBL   smallint DEFAULT Year(GetDate()) CHECK(YEAR_PUBL >= 1960 AND
        YEAR_PUBL <= YEAR(GetDate()))),
    PUBLISHER   varchar(20)    NULL,
    PAGES       smallint       CHECK(PAGES >= 5 AND PAGES <= 1000),
    CHECK (NOT (AUTHOR IS NULL AND COAUTHOR IS NOT NULL))
);
```

# Именованные ограничения

- Для анализа ошибок целесообразно именовать все ограничения, особенно если таблица содержит несколько ограничений одного типа.
- Для именования ограничений используется ключевое слово **CONSTRAINT**

# Создание BOOKS с именованными ограничениями

```
CREATE TABLE BOOKS
(
  ISBN ,      varchar(14)      NOT NULL,
  TITLE      varchar(120)     NOT NULL,
  AUTOR      varchar (30)      NULL,
  COAUTOR    varchar(30)      NULL,
  YEAR_PUBL  smallint         NOT NULL,
  PUBLICH    varchar(20)      NULL,
  PAGES      smallint         NOT NULL,
  CONSTRAINT PK_BOOKS PRIMARY KEY (ISBN),
  CONSTRAINT DF_ YEAR_PUBL DEFAULT (Year(GetDate())),
  CONSTRAINT CK_ YEAR_PUBL CHECK (YEAR_PUBL >= 1960 AND
    YEAR_PUBL <= YEAR(GetDate())),
  CONSTRANT CK_PAGES CHECK (PAGES > = 5 AND PAGES <= 1000),
  CONSTRAINT CK_BOOKS CHECK (NOT (AUTOR IS NULL AND COAUTOR IS NOT NULL))
);
```

# Таблица READERS:

- *Номер читательского билета* - это целое число в пределах 32 000 и он уникально определяет читателя.
- *Имя, фамилия* читателя — это последовательность символов, не более 30.
- *Адрес* — это последовательность символов, не более 50.
- *Номера телефонов рабочего и домашнего* — последовательность символов, не более 12.
- *Дата рождения* — календарная дата. В библиотеку принимаются читатели не младше 17 лет.

# Оператор

```
CREATE TABLE READERS
(
  READER_ID      Smallint(4) PRIMARY KEY,
  FIRST_NAME     char(30)    NOT NULL,
  LAST_NAME      char(30)    NOT NULL,
  ADRES          char(50),
  HOME_PHON      char(12),
  WORK_PHON      char(12),
  BIRTH_DAY date CHECK(DateDiff(year, GetDate(), BIRTH_DAY) >=17),
):
```

# Таблица Exemplar

```
CREATE TABLE EXEMPLAR
(
  EXEMPLAR_ID INT          IDENTITY PRIMARY KEY,
  ISBN         varchar(14) NOT NULL FOREIGN KEY references BOOKS(ISBN),
  READER_ID   Smallint(4)  NULL FOREIGN KEY references READERS (READER_ID),
  DATA_IN    date,
  DATA_OUT   date,
  EXIST       Logical,
);
```

# Порядок создания таблиц

- В нашем примере с библиотекой порядок описания таблиц следующий:
  1. Таблица BOOKS
  2. Таблица READERS
  3. Таблица CATALOG (системный каталог)
  4. Таблица EXEMPLAR
  5. Таблица RELATION\_1 (дополнительная связующая таблица между книгами и системным каталогом).

# Средства определения схемы базы данных

- В СУБД ORACLE база данных создается в ходе установки программного обеспечения собственно СУБД. Все таблицы пользователей помещаются в единую базу данных.
- Однако они могут быть разделены на группы, объединенные в подсхемы.
- Понятие подсхемы не стандартизировано в SQL и не используется в других СУБД.

# Семантическое обеспечение целостности данных

Процедуры и триггеры

# Хранимые процедуры

- Хранимые процедуры пишутся на специальном встроенном языке программирования, они могут включать любые операторы SQL, а также включают некоторый набор операторов, управляющих ходом выполнения программ

# Синтаксис

- CREATE [ OR REPLACE]
- ( “аргумент” IN | OUT | IN OUT “Тип данных” [,..])
- IS | AS
- “Тело процедуры PL/SQL”

# Функция получения ip-адреса

- **create or replace function**

**client\_ip\_address**

**return varchar2 is**

**begin**

**return dbms\_standard.client\_ip\_address;**

**end;**

# Пример процедуры

- create or replace procedure update\_debts is
- Begin
- update computation c set n\_pay=(select sum(n\_sum)
- from payment
- where n\_client=clients.n\_client  
and d\_pay between dates.d\_computation and  
add\_months(dates.d\_computation,1))
- end update\_debts;

# Триггеры

- Фактически триггер — это специальный вид хранимой процедуры, которую SQL Server вызывает при выполнении операций модификации соответствующих таблиц.
- Триггер автоматически активизируется при выполнении операции, с которой он связан.
- Триггеры связываются с одной или несколькими операциями модификации над одной таблицей.

# два типа триггеров

- В СУБД Oracle определены два типа триггеров:
- триггеры, которые могут быть запущены перед реализацией операции модификации, они называются BEFORE-триггерами,
- и триггеры, которые активизируются после выполнения соответствующей модификации, аналогично триггерам MS SQL Server, — они называются AFTER-триггерами.

# Синтаксис

- CREATE [ OR REPLACE] TRIGGER  
<имя\_триггера> BEFORE | AFTER
- ON <имя\_таблицы>
- FOR { [INSERT] [,UPDATE] [, DELETE] }
- FOR EACH ROW
- WHEN (условие)
- AS
- SQL-операторы (Тело триггера)

# Пример1

- **create or replace trigger** add\_author **AFTER**  
**INSERT OR UPDATE OF C\_AUTHOR**
- **ON T CLAUSES**  
**FOR EACH ROW**
- **DECLARE**  
id\_cl int;  
aut varchar2(500);  
res int;
- **BEGIN**  
id\_cl := :new.N\_ID\_CL;  
aut := :new.C\_AUTHOR;  
res := ANALIZ\_AUT(id\_cl, aut);  
**END;**

# Пример2

- create or replace trigger "BI\_COMPUTATION"
- before insert on "COMPUTATION"
- for each row
- begin
- select "COMPUTATION\_SEQ".nextval into  
:NEW.N\_COMPUTATION
- from dual;
- :NEW.D\_COMPUTATION := SYSDATE();
- end;

# Ограничения

- Нельзя использовать в теле триггера операции создания объектов БД (новой БД, новой таблицы, нового индекса, новой хранимой процедуры, нового триггера, новых индексов, новых представлений),
- Нельзя использовать в триггере команду удаления объектов DROP для всех типов базовых объектов БД.
- Нельзя использовать в теле триггера команды изменения базовых объектов ALTER TABLE, ALTER DATABASE.
- Нельзя изменять права доступа к объектам БД, то есть выполнять команду GRANT или REVOKE.
- Нельзя создать триггер для представления (VIEW).
- В отличие от хранимых процедур, триггер не может возвращать никаких значений, он запускается автоматически сервером и не может связаться самостоятельно ни с одним клиентом.