

**Министерство образования и науки Украины**  
**Харьковский национальный университет радиоэлектроники**

**Дисциплина:**  
**«Организация баз данных и знаний»**

для студентов дневной формы обучения  
по направлению  
6.050101 «Компьютерные науки»

# Базовые концепции языка SQL

Для выполнения запросов к реляционной базе данных используется структурированный язык запросов

**SQL (Structured Query Language).**

Основной структурной единицей реляционной БД является таблица (отношение).



**Выборка данных** – итоговая таблица результатов, полученных на основе запроса к таблицам БД.

# Свойства языка SQL

- основные конструкции языка интуитивно понятны, так как основываются на лексемах английского языка;
- SQL является непроцедурным языком;
- язык структурированных запросов SQL стандартизирован;
- конструкции языка работают с набором данных;
- SQL не зависит от регистра символов;
- SQL конструкции идентичны для конечных пользователей, программистов и администраторов БД.



# Структура запроса SQL

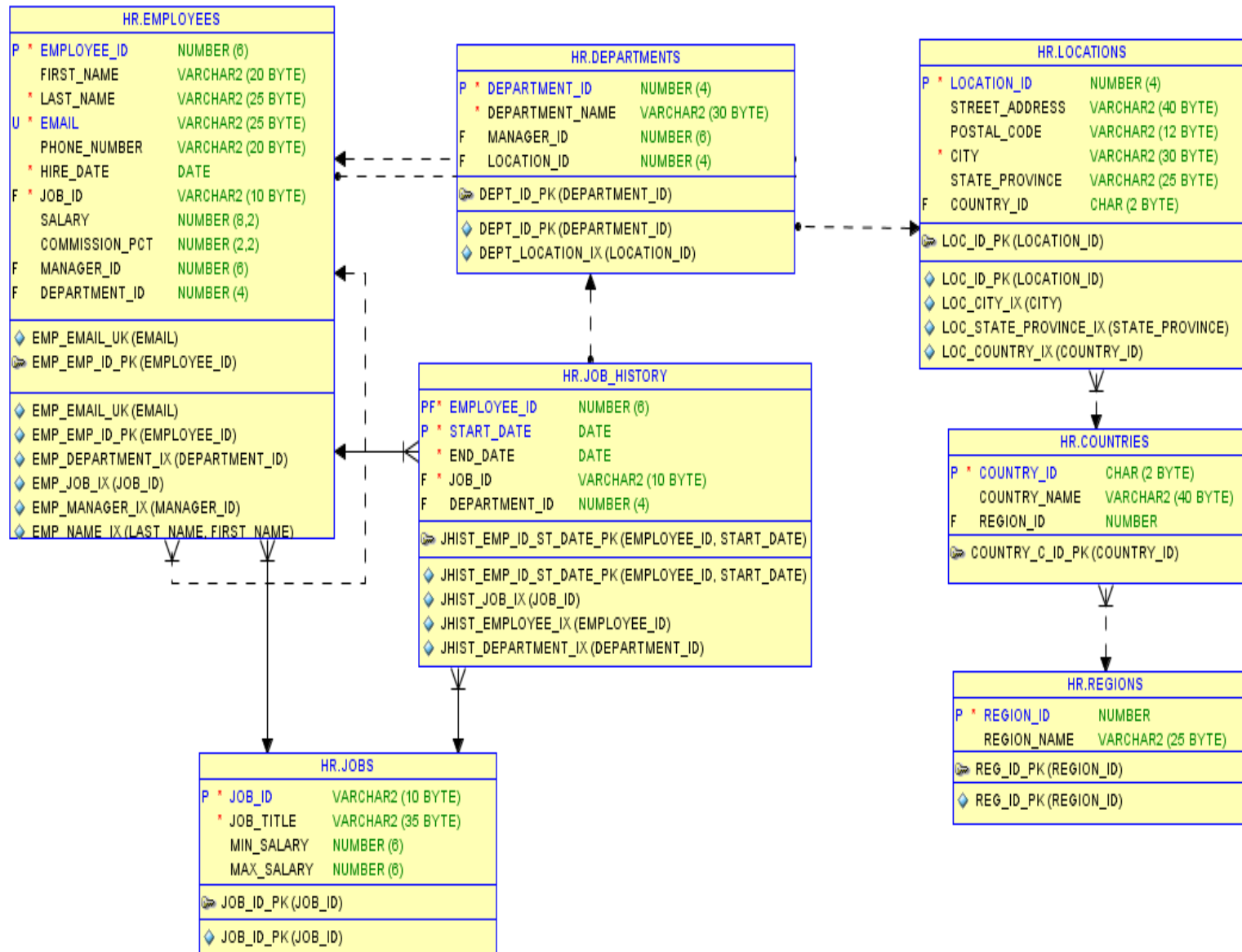
*SELECT* { \* | **DISTINCT** } *column1, columnN*]  
*FROM* *table1[,table2,...,tableN, query,view]*  
[ *WHERE* *conditions* ]  
[ *GROUP BY* *column1 [, columnN]* ]  
[ *HAVING* *conditions* ]  
[ *ORDER BY* *column1, column2, columnN*  
{ *ASC|DESC* } ];

# Предложение *SELECT* может содержать:

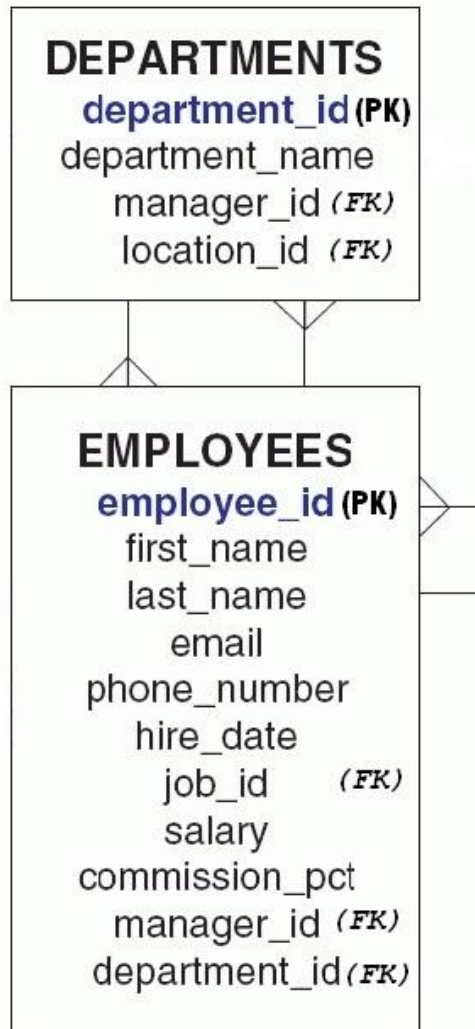
- имена столбцов и их псевдонимы;
- арифметические операции;
- комбинации числовых значений;
- оператор ();
- функции;
- конкатенированные столбцы;
- литералы.



# Схема БД Human Resource (HR)



# Фрагмент схемы БД (HR)



Все таблицы равноправны!

*Организация иерархии  
средствами задания  
внешнего ключа*

# Простые запросы

Запрос на выборку данных без дополнительных ограничений в общем виде:

***SELECT** column1[,column2, ...,column]*  
***FROM** table1[,table2,...,tableN];*

где

*column1[,column2, ...,column]* – имена атрибутов отношений (таблиц),

*table1[,table2,...,tableN]* – имена соответствующих таблиц.



## Выбор столбцов из таблиц

Для выбора всех столбцов из указанных таблиц, используется оператор \*

**SELECT \***

**FROM** table1[,table2,...,tableN];

*или*

**SELECT** table1.\*

**FROM** table1 [,table2,...,tableN];

# Выполнение запроса

SQL Navigator 6.7.0 - XPert Edition - [Project1] - [Code Editor - [HR@XE(1)];Untitled1]

File Edit Search View Session Object Tools Team Coding Window Help

HR@XE(1)

Untitled1

Execute to End (F9)

```
SELECT *  
FROM hr.departments;
```

Auto-Refresh every 20 sec(s)

Row #	DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
1	10	Administration	200	1700
2	20	Marketing	201	1800
3	30	Purchasing	114	1700
4	40	Human Resources	203	2400
5	50	Shipping	121	1500
6	60	IT	103	1400
7	70	Public Relations	204	2700
8	80	Sales	145	2500
9	90	Executive	100	1700
10	100	Finance	108	1700
11	110	Accounting	205	1700
12	120	Treasury		1700
13	130	Corporate Tax		1700
14	140	Control And Credit		1700
15	150	Shareholder Services		1700

Script Spool

1:1 lines:3 0 lines 0 characters Insert Modified Exec time: 0.031 sec Data Set is Read-Only: 27 row(s) fetched

DB Navigator [HR@XE(1)] HR@XE(1):Untitled1

Output

General HR@XE(1)

13:14:22 Start Script Execution ...

13:14:22 \*\*\*\*\* SCRIPT STARTED: 20-feb-2014 13:14:22 \*\*\*\*\*

13:14:22 SELECT \*

13:14:22 FROM hr.departments

13:14:22 Data Set is Read-Only: 27 row(s) fetched

# Литерал и ключевое слово *DISTINCT*

**Литералом** называется произвольный набор символов в предложении *SELECT*, который **не** содержит имен столбцов или их псевдонимов. Символьные литералы используются совместно с оператором ‘ ’.

```
SELECT column1 || '_' || columnN  
FROM table1[,table2,...,tableN];
```

Предложение *DISTINCT* – для подавления повторяющихся записей.

```
SELECT DISTINCT column1 || columnN  
FROM table1[,table2,...,tableN];
```

# Сортировка данных

Для сортировки результатов выборки используется предложение **ORDER BY**, по умолчанию используется сортировка по возрастанию (**ASC**). Если необходима сортировка по убыванию, следует использовать (**DESC**).

*SELECT* column1, columnN

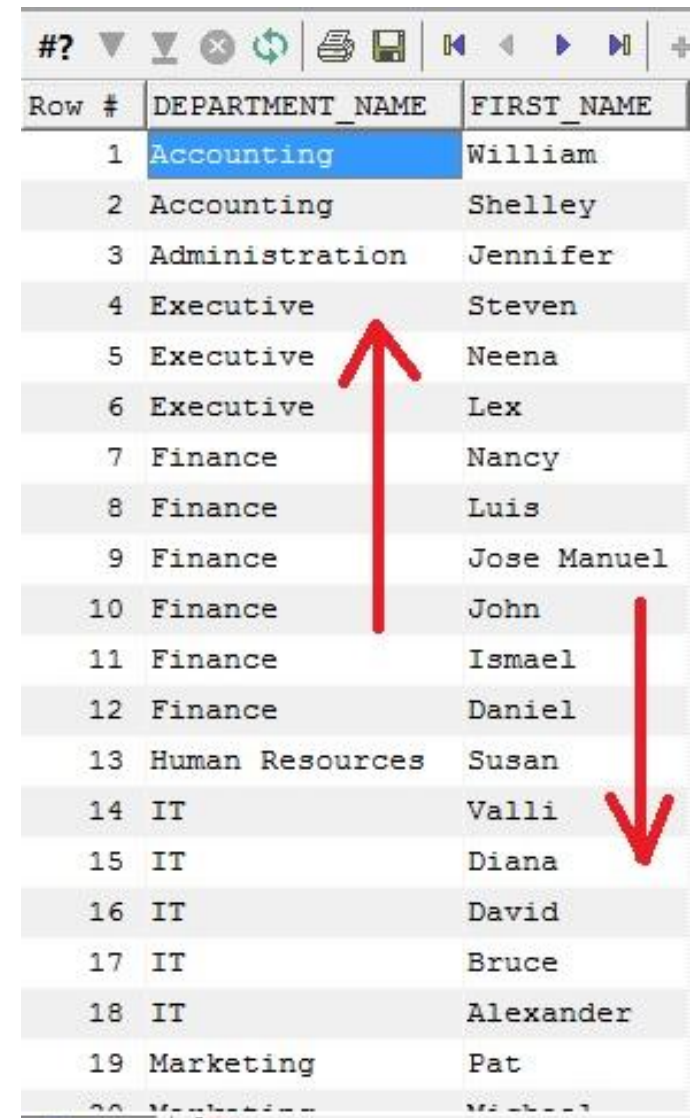
*FROM* table1[,table2,...,tableN]

*ORDER BY* column1, column2, column **DESC**;

Спецификация *ASC/ DESC* относится лишь к столбцу, после которого определена.

# Сортировка по вложению

```
SELECT d.department_name,  
e.first_name  
FROM hr.departments d,  
hr.employees e  
WHERE  
d.department_id=e.department_id  
ORDER BY d.department_name,  
e.first_name desc
```



Row #	DEPARTMENT_NAME	FIRST_NAME
1	Accounting	William
2	Accounting	Shelley
3	Administration	Jennifer
4	Executive	Steven
5	Executive	Neena
6	Executive	Lex
7	Finance	Nancy
8	Finance	Luis
9	Finance	Jose Manuel
10	Finance	John
11	Finance	Ismael
12	Finance	Daniel
13	Human Resources	Susan
14	IT	Valli
15	IT	Diana
16	IT	David
17	IT	Bruce
18	IT	Alexander
19	Marketing	Pat
20	Marketing	Neena

# Работа со значениями *NULL*

**ORDER BY** *column* [**ASC** | **DESC**]

[**NULLS FIRST** | **NULLS LAST**]

Row #	FIRST_NAME	DEPARTMENT_ID
89	Elizabeth	80
90	Sundita	80
91	Ellen	80
92	Alyssa	80
93	Jonathon	80
94	Jack	80
95	Charles	80
96	Steven	90
97	Neena	90
98	Lex	90
99	Nancy	100
100	Daniel	100
101	John	100
102	Ismael	100
103	Jose Manuel	100
104	Luis	100
105	Shelley	110
106	William	110
107	Kimberely	

Row #	FIRST_NAME	DEPARTMENT_ID
1	Kimberely	
2	Jennifer	10
3	Pat	20
4	Michael	20
5	Sigal	30
6	Karen	30
7	Shelli	30
8	Alexander	30
9	Den	30
10	Guy	30
11	Susan	40
12	Timothy	50
13	Randall	50
14	Sarah	50
15	Britney	50
16	Samuel	50
17	Vance	50
18	Alana	50
19	Kevin	50

```
SELECT e.first_name,  
e.department_id  
FROM hr.employees e  
ORDER BY  
department_id nulls  
first
```



# Обработка *NULL* значений

Функция *NVL* (*имя\_столбца, значение*) – функция принимает два параметра. Первый – имя столбца, которые может содержать «незаполненные» значения. Второй параметр – значение, **НА** которое следует заменить «незаполненные» поля в итоговой выборке.

Row #	FIRST_NAME	SALARY	COMMISSION_PCT
56	Stephen	3200	
57	Alexander	9000	
58	Bruce	6000	
59	David	4800	
60	Valli	4800	
61	Diana	4200	
62	Hermann	10000	
63	Sundita	6100	0,1
64	Ellen	11000	0,3
65	Alyssa	8800	0,25
66	Jonathon	8600	0,2
67	Jack	8400	0,2
68	Charles	6200	0,1
69	John	14000	0,4
70	Karen	13500	0,3
71	Alberto	12000	0,3

```
SELECT emp.first_name, emp.salary, NVL(emp.commission_pct,0) as "Премия"
FROM hr.employees emp
```

Row #	FIRST_NAME	SALARY	Премия
56	Stephen	3200	0
57	Alexander	9000	0
58	Bruce	6000	0
59	David	4800	0
60	Valli	4800	0
61	Diana	4200	0
62	Hermann	10000	0
63	Sundita	6100	0,1
64	Ellen	11000	0,3
65	Alyssa	8800	0,25
66	Jonathon	8600	0,2
67	Jack	8400	0,2
68	Charles	6200	0,1



# Конструкция WHERE

Конструкция *WHERE table1. column1 > value* применяется для фильтрации строк, выбираемых из таблицы *table1*.

```
SELECT column1, columnN  
FROM table1[,table2,...,tableN]  
WHERE table1. column1 > value1  
[AND | OR] table1. column1 = value2  
ORDER BY column1, column2, column DESC;
```

## Операторы сравнения

>, <, >=, <=, =, !=, <>



# Диапазоны, ограничения, включение / исключение

<b>[NOT]</b>	BETWEEN... AND ...	Границы диапазона
	LIKE	Поиск по шаблону
	IN	Вхождение в список
	IS <b>[NOT]</b> NULL	Проверка на <i>NULL</i>

# Диапазон значений

***BETWEEN*** (начальное\_знач.\_диапазона) ***AND***  
(конечное\_знач.\_диапазона)

***SELECT*** column1, column2

***FROM*** table1

***WHERE*** column1 ***BETWEEN*** value1 ***AND*** value2;

# Поиск по шаблону

```
SELECT column1, column2  
FROM table1  
WHERE column1 LIKE '%value1_  
OR  
column1 LIKE '_value2%',
```

где маска '*%value1\_*' указывает, что **до** искомой подстроки может быть любое количество символов, а **после** искомой подстроки – один символ.

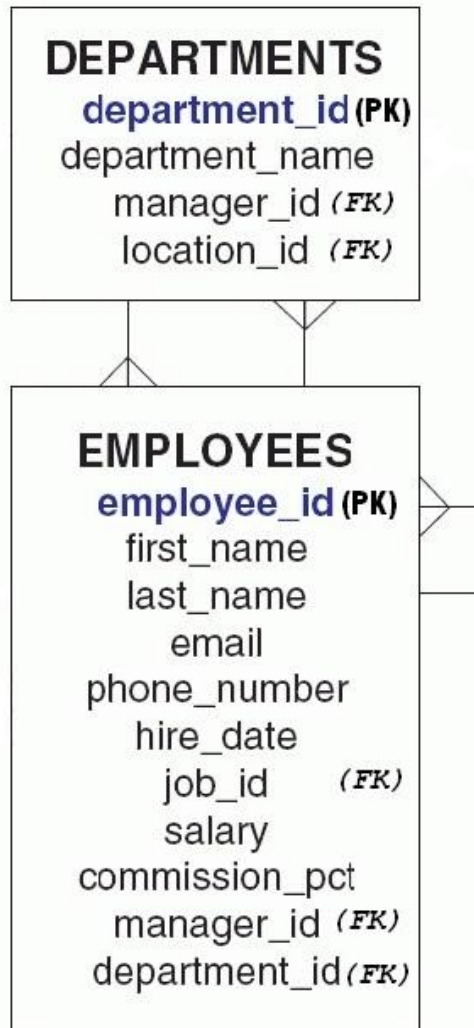
## Вхождение в список

```
SELECT column1, column2  
FROM table1  
WHERE column1 IN (value1, value2, ..., valueN)
```

## Проверка на *NULL*

```
SELECT column1, column2  
FROM table1  
WHERE column2 IS NULL
```

# Соединение таблиц



Чаще всего для соединения таблиц используют **внешние** ключи. В этом случае таблицы, необходимые для выполнения запросы перечисляются в выражении *FROM*, а условие соединения указывается в предложении *WHERE*.

# Соединение таблиц по ключам

```
SELECT emp.first_name, emp.last_name, dep.department_id  
FROM employees emp, departments dep  
WHERE emp.department_id=dep.department_id
```

где *emp* и *dep* – псевдонимы соответствующих таблиц.

**ИНАЧЕ: РЕЗУЛЬТАТ – ДЕКАРТОВО ПРОИЗВЕДЕНИЕ!!!**

Если число таблиц, перечисленных в предложении *FROM* равно **N**. Тогда **минимальное** количество условий соединения является разность (**N-1**).

# Ключевое слово *JOIN*

*JOIN* в **SQL** также используется для соединения нескольких таблиц.

Существует три типа инструкций *JOIN*:

***INNER JOIN;***

***OUTER JOIN;***

***CROSS JOIN.***

В свою очередь *OUTER JOIN* может быть:

***LEFT outer JOIN;***

***RIGHT outer JOIN;***

***FULL outer JOIN.***

При использовании в SQL-запросах слово *outer* обычно опускается.



# Общий синтаксис конструкции *JOIN*

*SELECT* \*

*FROM* *table\_1* *join\_type* *JOIN* *table\_2* *ON* *condition...*

где *join\_type* – тип *JOIN* –выражения [*left, right, full, inner, cross*];

*table\_1* – имя таблицы к которой присоединяются столбцы другой таблицы;

*table\_2* – имя таблицы которая присоединяется к таблице *table\_1*;

*condition* – условие соединения таблиц.



# Использование *JOIN* на примере двух отношений

Одна таблица содержит информацию о пользователях, другая – о web-ресурсах, которые администрируют эти пользователи.

Таблица 1 – *Users*

<i>id_user</i>	<i>nick</i>
1	user1
3	user3
4	user4

Таблица 2 – *Resources*

<i>id_res</i>	<i>res_name</i>	<i>userid</i>
1	res1	3
2	res2	1
3	res3	2
5	res5	3

*id\_user* – первичный ключ таблицы *Users*;

*id\_res* – первичный ключ таблицы *Resources*;

# Конструкция *INNER JOIN*

Конструкция *INNER JOIN* – необходима для получения только тех записей, для которых существует однозначное соответствие записей в другой таблице. Для *INNER JOIN* верно утверждение, что условие *CONDITION* выполняется для каждой записи.

*SELECT Resources.res\_name, Users.nick*

*FROM Resources INNER JOIN Users ON*

*Users.id\_user= Resources.userid*

<i>id_user</i>	<i>nick</i>
1	user1
3	user3
4	user4

<i>id_res</i>	<i>res_name</i>	<i>userid</i>
1	res1	3
2	res2	1
3	res3	2
5	res5	3

Результат запроса:

<i>res_name</i>	<i>nick</i>
res2	user1
res1	user3
res5	user3

# **INNER JOIN *vs* WHERE**

Конструкция **INNER JOIN** идентична использованию условия соединения таблиц по равенству (по внешнему ключу) в предложении **WHERE**:

```
SELECT Resources.res_name, Users.nick  
FROM Resources, Users  
WHERE Users.id_user= Resources.Userid
```

Результат запроса:

<i><b>res_name</b></i>	<i><b>nick</b></i>
res2	user1
res1	user3
res5	user3

# Конструкция *LEFT JOIN*

Предложение *LEFT JOIN* подразумевает, что из первой таблицы будут выбраны все записи, даже если в присоединяемой таблице не найдено соответствий. Для строк, которые не удовлетворяют условию ***ON condition***, возвращаются значения ***NULL***.

*SELECT Resources.res\_name, Users.nick*

*FROM Resources LEFT JOIN Users ON*

*Users.id\_user= Resources.Userid*

<i>id_user</i>	<i>nick</i>
1	user1
3	user3
4	user4

<i>id_res</i>	<i>res_name</i>	<i>userid</i>
1	res1	3
2	res2	1
3	res3	2
5	res5	3

Результат запроса:

<i>res_name</i>	<i>nick</i>
res1	user3
res2	user1
res3	
res5	user3

# **LEFT JOIN *vs* WHERE**

Операции внешнего соединения в Oracle можно указывать в предложении **WHERE** следующим образом:

```
SELECT Resources.res_name, Users.nick  
FROM Resources, Users  
WHERE Users.id_user(+) = Resources.userid;
```

Оператор внешнего соединения (+) ставится возле той таблицы, где записей в возвращаемой выборке потенциально меньше, и возле наименования столбца, по которому выполняется соединение.

результат запроса:

<i>res_name</i>	<i>nick</i>
<i>res1</i>	<i>user3</i>
<i>res2</i>	<i>user1</i>
<i>res3</i>	
<i>res5</i>	<i>user3</i>

# Конструкция *RIGHT JOIN*

Будут возвращены все записи, которые удовлетворяют правой части условия **ON condition**, даже если не найдено соответствующих записей в таблице, к которой присоединяют другую таблицу.

*SELECT Resources.res\_name, Users.nick*

*FROM Resources RIGHT JOIN Users ON*

*Users.id\_user= Resources.userid*

<i>id_user</i>	<i>nick</i>
1	user1
3	user3
4	user4

<i>id_res</i>	<i>res_name</i>	<i>userid</i>
1	res1	3
2	res2	1
3	res3	2
5	res5	3

Результат запроса:

<i>res_name</i>	<i>nick</i>
res2	user1
res1	user3
res5	user3
	user4

# **RIGHT JOIN *vs* WHERE**

```
SELECT Resources.res_name, Users.nick  
FROM Resources, Users  
WHERE Users.id_user= Resources.userid(+);
```

Результат запроса:

<i><b>res_name</b></i>	<i><b>nick</b></i>
res2	user1
res1	user3
res5	user3
	user4

# Конструкция *FULL JOIN*

Объединение результатов *LEFT JOIN* и *RIGHT JOIN* – *FULL JOIN*.

*SELECT Resources.res\_name, Users.nick*

*FROM Resources FULL JOIN Users ON*

*Users.id\_user= Resources.Userid*

Возвращает все строки из двух таблиц. Для записей, которые не удовлетворяют условию **ON condition**, возвращаются значения **NULL**.

<i>id_user</i>	<i>nick</i>
1	user1
3	user3
4	user4

<i>id_res</i>	<i>res_name</i>	<i>userid</i>
1	res1	3
2	res2	1
3	res3	2
5	res5	3

Результат запроса:

<i>res_name</i>	<i>nick</i>
res1	user3
res2	user1
res3	
res5	user3
	user4



# Классы функций языка SQL

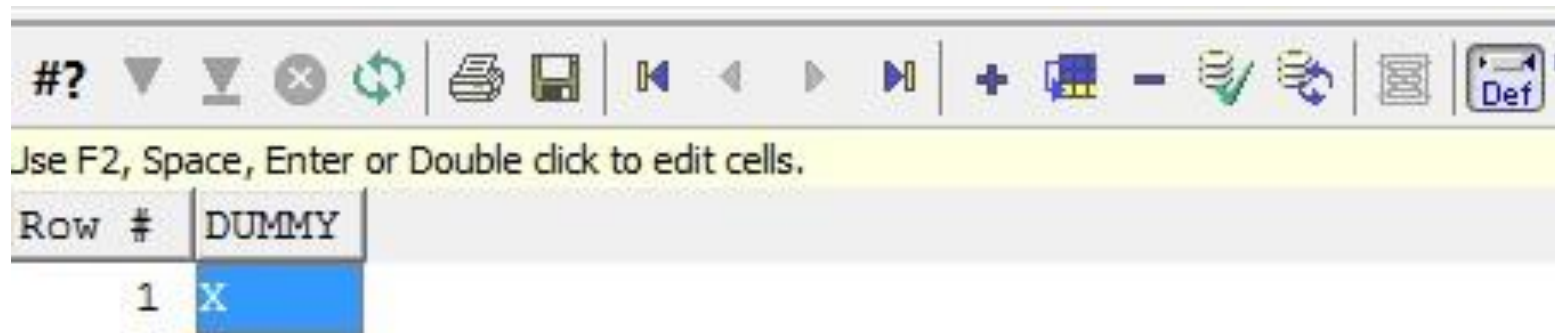
1. Строковые функции;
2. Числовые функции;
3. Функции для работы с датами;
4. Функции конвертирования типов данных;
5. Универсальные функции для работы с любыми типами данных;
6. Агрегатные (групповые) функции;
  - 6.1 Аналитические функции.



# Служебная таблица DUAL

**DUAL** – системная таблица, которая содержит 1 строку и 1 столбец. Используется для выбора различных констант, которые задает пользователь.

**SELECT \***  
**FROM dual;**



Use F2, Space, Enter or Double click to edit cells.

Row #	DUMMY
1	X

# Строковые функции

**CONCAT(s1,s2)** Соединение строк s1 и s2 (||);

**LOWER(column/val)** Перевод в нижний регистр;

**UPPER(column/val)** Перевод в верхний регистр;

**INITCAP(column/val)** Перевод в верхний регистр только первого символа;

```
select concat(last_name,concat(' ',  
first_name)) as "ФИО" from hr.employees;
```

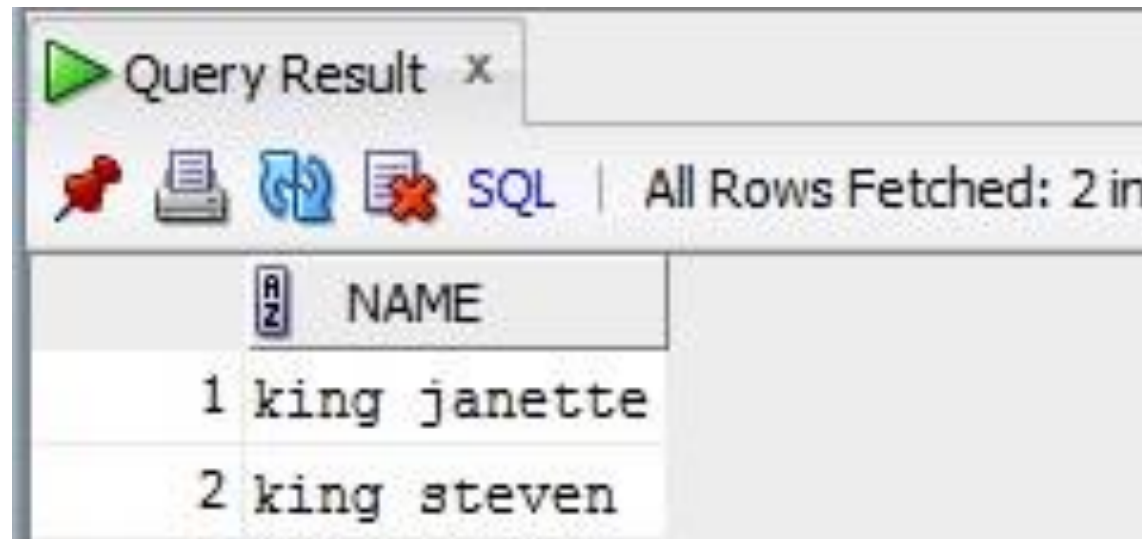
```
результат: /* Abel Ellen  
              Ande Sundar  
              Atkinson Mozhe */
```

```
select last_name, first_name, hire_date
       from hr.employees emp
       where last_name like 'king'
```

--Запрос не возвращает строк--

/\*-----\*/

```
select LOWER(last_name || ' ' || first_name)
       from hr.employees emp
       where UPPER(last_name) like 'KING'
```



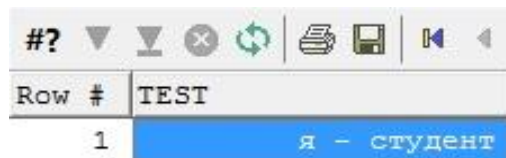
The screenshot shows a 'Query Result' window in Oracle SQL Developer. The window title is 'Query Result x'. Below the title bar, there are icons for a pin, a printer, a refresh, and a document with a red 'X'. To the right of these icons, it says 'SQL | All Rows Fetched: 2 in'. The main area of the window displays a table with two columns: 'NAME' and an unlabeled column. The table has two rows of data: '1 king janette' and '2 king steven'.

	NAME
1	king janette
2	king steven

**LPAD(s1,n [,s2])** Заполняет строку **s1** пробелами слева, чтобы довести общую длину строки до заданного числа символов **n**. Необязательный параметр **s2** может быть указан для определения строки, которой будут заполнены все свободные символы слева;

**RPAD(s1,n [,s2])** Аналогично **LPAD**, но строка дополняется справа;

```
select LPAD('я - студент',20) test from dual
```



A screenshot of a SQL query result window. The window has a toolbar with icons for undo, redo, print, save, and navigation. Below the toolbar is a table with two columns: 'Row #' and 'TEST'. The first row shows '1' in the 'Row #' column and 'я - студент' in the 'TEST' column.

Row #	TEST
1	я - студент

```
select RPAD('я - студент',20,'*****')  
test from dual
```



A screenshot of a SQL query result window. The window has a toolbar with icons for undo, redo, print, save, and navigation. Below the toolbar is a table with two columns: 'Row #' and 'TEST'. The first row shows '1' in the 'Row #' column and 'я - студент\*\*\*\*\*' in the 'TEST' column.

Row #	TEST
1	я - студент*****

**LTRIM(s1 [,s2])** Удаление символов из левой части строки **s1**. Дополнительный параметр **s2** определяет символы, которые должны быть удалены; если **s2** не задан, удаляются начальные пробелы.

**RTRIM(s1 [,s2])** Аналогично **LTRIM**, но **s1** обрезается справа;

**TRIM([s2 FROM ] s1)** Удаление в строке **s1** заполнителей **s2** справа и слева; по умолчанию удалятся пробелы.

```
select LTRIM('      я - студент') t from dual
select TRIM('*' FROM '**** я - студент****' )
test from dual
```

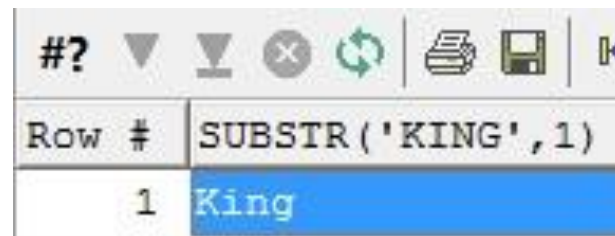


#?	▼	⌵	⊗	↺	🖨	📱
Row #	TEST					
1	я - студент					

**SUBSTR(s1,n1 [,n2])** Выделение из строки s1 подстроки, начиная с позиции n1 длиной n2 символов;

```
select substr('King',0) from dual;
```

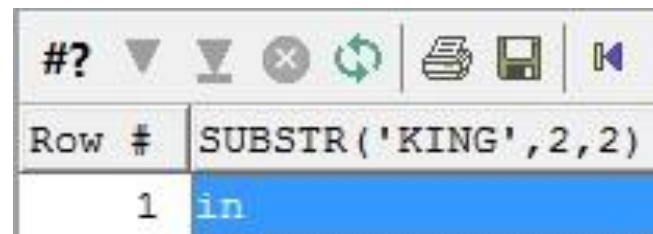
```
select substr('King',1) from dual;
```



A screenshot of a SQL query result window. The title bar shows a toolbar with icons for query execution, undo, redo, and save. The window displays a table with two columns: 'Row #' and 'SUBSTR('KING', 1)'. The first row shows the value '1' in the 'Row #' column and 'King' in the 'SUBSTR('KING', 1)' column.

Row #	SUBSTR('KING', 1)
1	King

```
select substr('King',2,2) from dual;
```

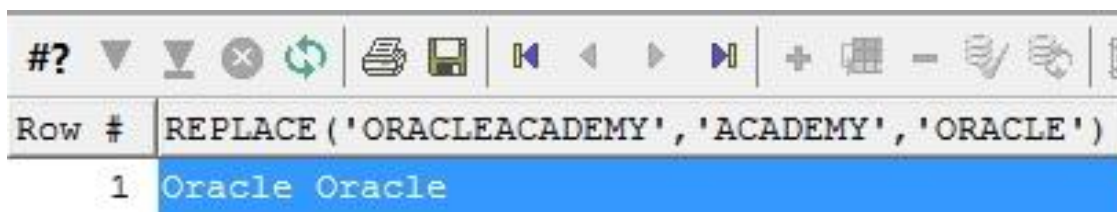


A screenshot of a SQL query result window. The title bar shows a toolbar with icons for query execution, undo, redo, and save. The window displays a table with two columns: 'Row #' and 'SUBSTR('KING', 2, 2)'. The first row shows the value '1' in the 'Row #' column and 'in' in the 'SUBSTR('KING', 2, 2)' column.

Row #	SUBSTR('KING', 2, 2)
1	in

**REPLACE(s1,s2,s3)** Замена в строке s1 подстроки s2 на подстроку s3;

```
select REPLACE('Oracle Academy',  
              'Academy', 'Oracle') from dual
```



The screenshot shows a database query result in a table. The table has two columns: 'Row #' and the result of the REPLACE function. The first row shows the result 'Oracle Oracle'.

Row #	REPLACE('ORACLEACADEMY','ACADEMY','ORACLE')
1	Oracle Oracle

**TRANSLATE(s1,s2,s3)** Перекодировка строки s1 из алфавита s2 в алфавит s3.

```
select TRANSLATE(314253, 12345, 'abcdef') from dual  
cadbec
```

```
select TRANSLATE(314253, 12346, 'abcdef') from dual  
cadb5c
```



**CHR(x)** Возвращает символ, код ASCII которого **x**;

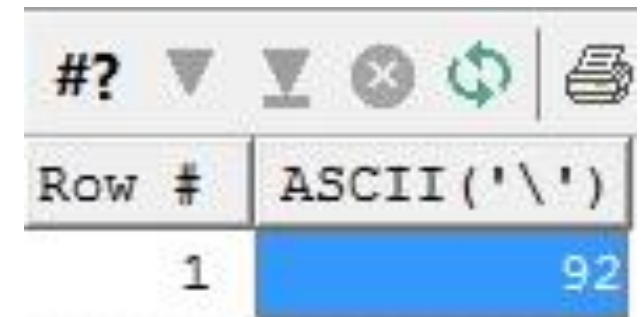
```
select chr(92) from dual
```



Row #	CHR (92)
1	\

**ASCII(x)** Возвращает код символа;

```
select ascii('\') from dual
```

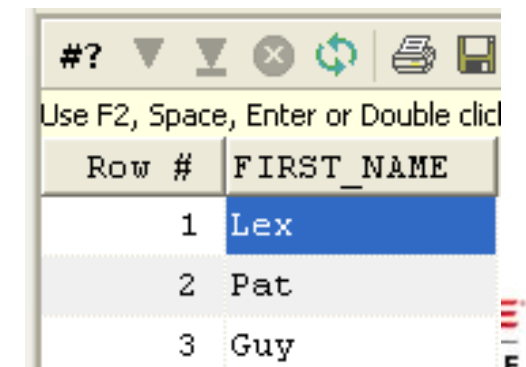


Row #	ASCII ('\')
1	92

**LENGTH(s1)** Возвращает длину строки **s1**.

```
select LENGTH('first') from dual /*5*/
```

```
SELECT first_name FROM hr.employees  
WHERE LENGTH(first_name)=3
```



Row #	FIRST_NAME
1	Lex
2	Pat
3	Guy


**INSTR(s1,s2[,n1,n2])** Поиск в строке s1, начиная с позиции n1, n2-ого вхождения строки s2.

```
select INSTR('я - студент','т', 1) from dual  
select INSTR('я - студент','т') from dual
```



Row #	INSTR('Я-СТУДЕНТ','Т',1)
1	6

```
select INSTR('я - студент','т', 1,2) from dual
```



Row #	INSTR('Я-СТУДЕНТ','Т',1,2)
1	11

```
select INSTR('я - студент','т', -1,1) from dual
```



Row #	INSTR('Я-СТУДЕНТ','Т',1,2)
1	11

# Числовые функции

**ABS(x)** Абсолютное значение выражения (числа) x;

**POWER(x, y)** Возведение x в степень y;

**SQRT(x)** Извлечение квадратного корня из x;

**SIGN(x)** Возвращает знак выражения x: 
$$\left\{ \begin{array}{l} -1 - x \text{ отрицательное;} \\ 0 - x = 0; \\ 1 - x \text{ положительное.} \end{array} \right\}$$

**COS(x)** Косинус x;

**SIN(x)** Синус x;

**TAN(x)** Тангенс x;

**EXP(x)** Число e в степени x;

**LOG(x, y)** Логарифм x по основанию y;

**LN(x)** Натуральный логарифм x;



**CEIL(x)** Возвращает наименьшее целое число, которое больше/равно x;

```
select ceil(-11.1) from dual          /*-11*/
```

**FLOOR(x)** Возвращает наибольшее целое число, которое меньше/равно x;

```
select floor(-11.1) from dual        /*-12*/
```

**ROUND(x [,y])** Округление x до y знаков:

$$y = \begin{cases} 1 - \text{округление до десятых;} \\ 0 - \text{округление до целых;} \\ -1 - \text{округление до десятков;} \\ -2 - \text{округление до сотен;} \\ \text{и т.д.} \end{cases}$$

**TRUNC(x, y)** Усечение x до y знаков (по умолчанию – отбрасывает дробную часть);

**MOD(x, y)** Остаток от деления x на y.

# Функции преобразования типов данных

*(если формат преобразования возможен)*

Тип данных Функция преобразования	<i>number</i>	<i>char</i>	<i>varchar2</i>	<i>date</i>
<i>to_date</i> ('date', [format_of_date])	+	+	+	+
<i>to_char</i> ('строка', [format])	+	+	+	+
<i>to_number</i> (число, [format])	+	+	+	+

Функция **CAST()** также используется для преобразования одного встроенного типа данных к другому и имеет следующий синтаксис:

*CAST('строка\_для\_конвертирования' AS  
тип\_данных(размер))*

**SELECT** *CAST ('test\_string' AS varchar2(20))*  
**FROM** *dual;*



# Работа с датой

Для работы с датой и временем ORACLE использует тип данных **DATE**.

**1: Век; 2: Год; 3: Месяц; 4: День;**  
**5: Час; 6: Минута; 7: Секунда.**

**SYSDATE** – Псевдостолбец, который возвращает системную дату и время.

**CURRENT\_DATE** – Псевдостолбец, возвращающий системную дату для часового пояса, в котором выполняется сеанс.



# Формат представления даты

```
SELECT 'string' test, sysdate  
FROM dual;
```

TEST	SYSDATE
<i>string</i>	<i>24-фев-2014 13:39:59</i>

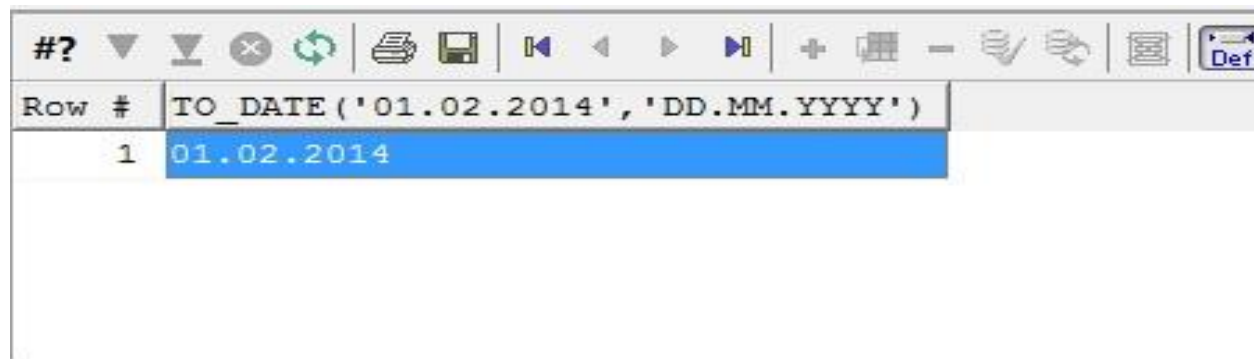




# Преобразование даты

```
SELECT to_date('01-фев-2014', 'dd-mon-yyyy')  
      FROM dual;
```

```
SELECT to_date('01-02-2014', 'dd-mm-yyyy')  
      FROM dual;
```












Row #	TO_DATE('01.02.2014', 'DD.MM.YYYY')
1	01.02.2014

# Вывод даты на экран

**SELECT** sysdate **FROM** dual;



#?																			
Row	#	SYSDATE																	
1		25.02.2014 13:00:49																	

**SELECT** to\_char(sysdate, 'dd-mon-yyyy')  
**FROM** dual;

#?																			
Row	#	TO_CHAR(SYSDATE, 'DD-MON-YYYY')																	
1		25-фев-2014																	

# Функции для работы с датами

**ДАТА - ЧИСЛО** Добавляет/вычитает определенное количество дней к дате, возвращает дату;

**ДАТА – ДАТА** Возвращает количество дней между указанными датами; `select to_date('01.02.2014', 'dd.mm.yyyy') - to_date('01.01.2014', 'dd.mm.yyyy') from dual;`

**ADD\_MONTHS(d, n)** Возвращает дату d с учетом прибавленных n месяцев (n – только целое число).

`select ADD_MONTHS(sysdate, 2) from dual`

Row #	ADD_MONTHS(SYSDATE, 2)
1	02-май-2015 18:51:33

**MONTHS\_BETWEEN(d1, d2)** Возвращает количество месяцев между двумя датами.

`select MONTHS_BETWEEN(sysdate, to_date('01.02.2011', 'dd.mm.yyyy')) test from dual;`

результат: /\*49,05772438769414575866188769414575866189\*/

**ROUND (date, ['format'])** Возвращает дату, округленную до формата, заданного вторым (необязательным) параметром:


*round(date) – округление до полуночи;*

*round(date, 'month') – округление до 1-ого числа месяца;*

*round (date, 'year') – округление до 1 января.*

---

```
select round(sysdate, 'MONTH') from dual
```



1	01-мар-2015
---	-------------

```
select round(sysdate, 'year') from dual
```

---

**EXTRACT(part\_of\_date FROM date)** Выделяет часть даты из полного значения DATE.

```
select extract(month from sysdate) from dual
```

результат: /\*3\*/ (т.к. sysdate=02.03.2015)

# Универсальные функции

**NVL (имя\_столбца, значение)** – функция возвращает **значение**, если **имя\_столбца** имеет значение **null**.

**NVL2 (имя\_столбца, значение 1, значение 2)** – функция возвращает **значение1**, если **имя\_столбца** не имеет значение **null**, в противном случае – возвращает **значение 2**.

```
select manager_id, NVL2(to_char(manager_id), 'yes', 'no')
from hr.employees
```

Row #	MANAGER_ID	NVL2 (TO_CHAR (
1		no
2	100	yes
3	100	yes
4	102	yes

**DECODE (имя\_столбца / значение, если\_знач.\_1, то\_итог\_1, если\_знач.\_2, то\_итог\_2,..., если\_знач.\_N, то\_итог\_N, Иначе\_значение\_N);**

```
SELECT department_id, decode(department_id, 10, 'ok', 50, 'test',
'other') dep from hr.departments
```

Row #	DEPARTMENT_ID	DEP
1	10	ok
2	20	other
3	30	other
4	40	other
5	50	test

# Варианты работы функции CASE

*CASE* имя\_ст. / знач

*WHEN* условие1 *THEN* результат\_1

[*WHEN* условие2 *THEN* результат\_2

*WHEN* условие3 *THEN* результат\_3 ]

*ELSE* результат\_4 *END*;



```
SELECT department_id, CASE department_id WHEN 10 THEN  
'ok' ELSE 'other' END dep from hr.departments
```

Row #	DEPARTMENT_ID	DEP
1	10	ok
2	20	other
3	30	other
4	40	other

# CASE с поиском



**CASE**

**WHEN** условие1 **THEN** результат\_1  
[**WHEN** условие2 **THEN** результат\_2  
**WHEN** условие3 **THEN** результат\_3 ]  
**ELSE** результат\_4 **END**;

---

**SELECT** last\_name, department\_id, salary,

**CASE**

**WHEN** salary<5000 **AND** department\_id=50 **THEN** 'O\_o'  
**WHEN** salary>5000 **AND** department\_id=50 **THEN** 'Good'  
**ELSE** 'other' **END** dep **from** hr.employees

Row #	LAST_NAME	DEPARTME... ▲	SALARY	DEP
20	Colmenares	30	2500	other
104	Mavris	40	6500	other
21	Weiss	50	8000	Good
22	Fripp	50	8200	Good
23	Kaufling	50	7900	Good
24	Vollman	50	6500	Good
25	Mourgos	50	5800	Good
26	Nayer	50	3200	O_o
27	Mikkilineni	50	2700	O_o
28	Landry	50	2400	O_o

# Групповые функции

Функции возвращают результат, обработанной группы значений.

**AVG(столбец/выражение)** Возвращает среднее значение по группе;

**COUNT(столбец/выражение)** Счетчик группы;

**MIN(столбец/выражение)** Возвращает минимальное значение по группе;

**MAX(столбец/выражение)** Возвращает максимальное значение по группе;

**SUM(столбец/выражение)** Подсчет суммы в группе;



```
select sum(salary) from hr.employees
```

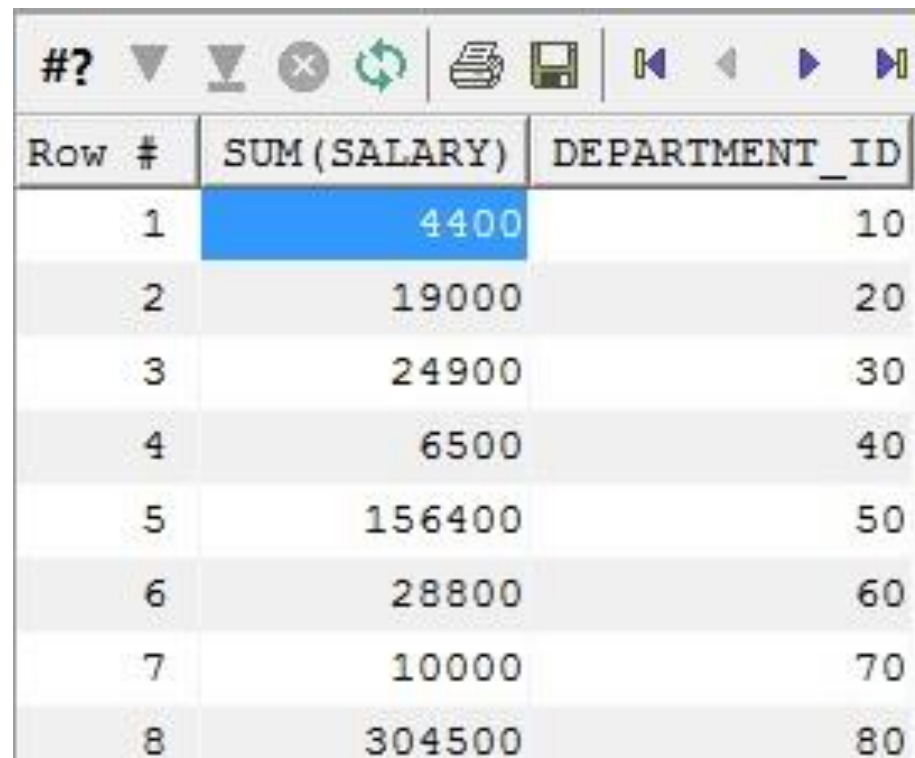
*результат: 1 строка - /\* 691416\*/*



# Конструкция *GROUP BY*

Для фрагментарной работы групповых функции используется конструкция **GROUP BY**. При работе функции для каждой отдельной группы, значение групповой функции «сбрасывается».

```
select sum(salary), department_id
      from hr.employees
     group by department_id
     order by department_id;
```

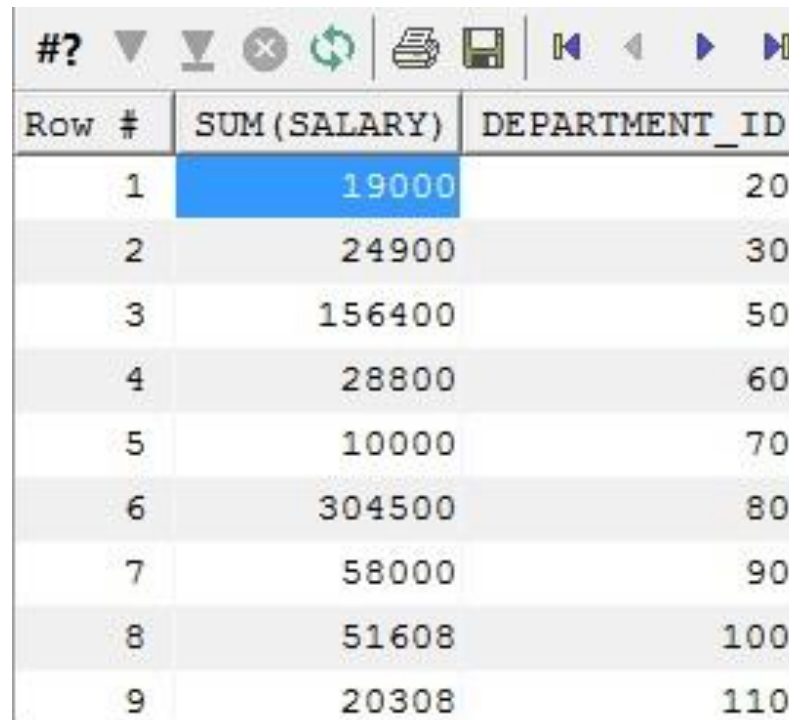


Row #	SUM(SALARY)	DEPARTMENT_ID
1	4400	10
2	19000	20
3	24900	30
4	6500	40
5	156400	50
6	28800	60
7	10000	70
8	304500	80

# Конструкция *HAVING*

Используется для наложения условий на группу значений.

```
select sum(salary), department_id
      from hr.employees
      group by department_id
      having sum(salary)>6500
      order by department_id;
```



Row #	SUM(SALARY)	DEPARTMENT_ID
1	19000	20
2	24900	30
3	156400	50
4	28800	60
5	10000	70
6	304500	80
7	58000	90
8	51608	100
9	20308	110

