

Перезапуск Data Engineer

Решаем тест по предыдущей лекции
до 18.42

Типы данных

Типы данных	Размер	Описание
char(размер)	Максимальный размер 2000 байт.	Где размер — количество символов фиксированной длины. Если сохраняемое значение короче, то дополняется пробелами; если длиннее, то выдается ошибка.
nchar(размер)	Максимальный размер 2000 байт.	Где размер — количество символов фиксированной длины в кодировке Unicode. Если сохраняемое значение короче, то дополняется пробелами; если длиннее, то выдается ошибка.
nvarchar2(размер)	Максимальный размер 4000 байт.	Где размер – количество сохраняемых символов в кодировке Unicode переменной длины.
varchar2(размер)	Максимальный размер 4000 байт. Максимальный размер в PLSQL 32KB.	Где размер – количество сохраняемых символов переменной длины.
long	Максимальный размер 2GB.	Символьные данные переменной длины.
raw	Максимальный размер 2000 байт.	Содержит двоичные данные переменной длины
long raw	Максимальный размер 2GB.	Содержит двоичные данные переменной длины

Типы данных

Типы данных	Размер	Описание
date	date может принимать значения от 1 января 4712 года до н.э. до 31 декабря 9999 года нашей эры.	date
bfile	Максимальный размер файла 4 ГБ.	Файл locators, указывает на двоичный файл в файловой системе сервера (вне базы данных).
blob	Хранит до 4 ГБ двоичных данных.	Хранит неструктурированные двоичные большие объекты.
clob	Хранит до 4 ГБ символьных данных.	Хранит однобайтовые и многобайтовые символьные данные.
nclob	Хранит до 4 ГБ символьных текстовых данных.	Сохраняет данные в кодировке unicode.

Типы данных

Типы данных	Размер	Описание
number(точность,масштаб)	Точность может быть в диапазоне от 1 до 38. Масштаб может быть в диапазоне от -84 до 127.	Например,number (14,5) представляет собой число, которое имеет 9 знаков до запятой и 5 знаков после запятой.
BINARY_FLOAT	32-битный тип данных с плавающей запятой одинарной точности	Каждое значение BINARY_FLOAT требует 5 байтов, включая байт длины
BINARY_DOUBLE	64-битный тип данных с плавающей запятой двойной точности	Каждое значение BINARY_DOUBLE требует 9 байтов, включая байт длины.

Типы данных

Типы данных	Размер	Описание
rowid	Формат строки: BBBBBBB.RRRR.FFFFF, Где BBBBBBB — это блок в файле базы данных; RRRR — строка в блоке; FFFFF — это файл базы данных.	Двоичные данные фиксированной длины. Каждая запись в базе данных имеет физический адрес или идентификатор строки (rowid).
BOOLEAN	TRUE или FALSE. Может принимать значение NULL	Хранит логические значения, которые вы можете использовать в логических операциях.

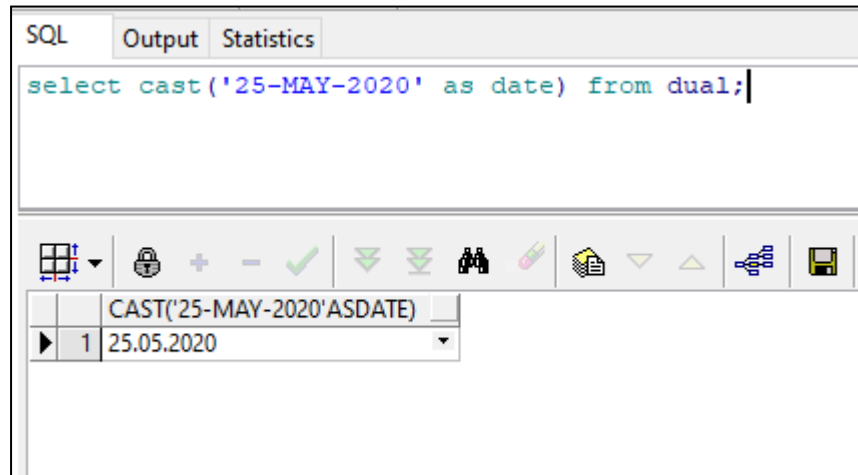
ROWID

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

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

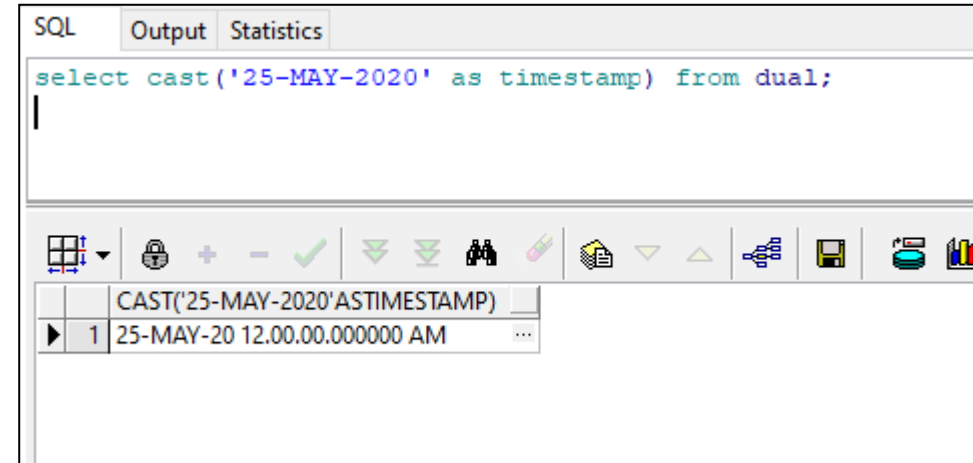
Преобразование типов CAST

CAST(что-то AS что-то)



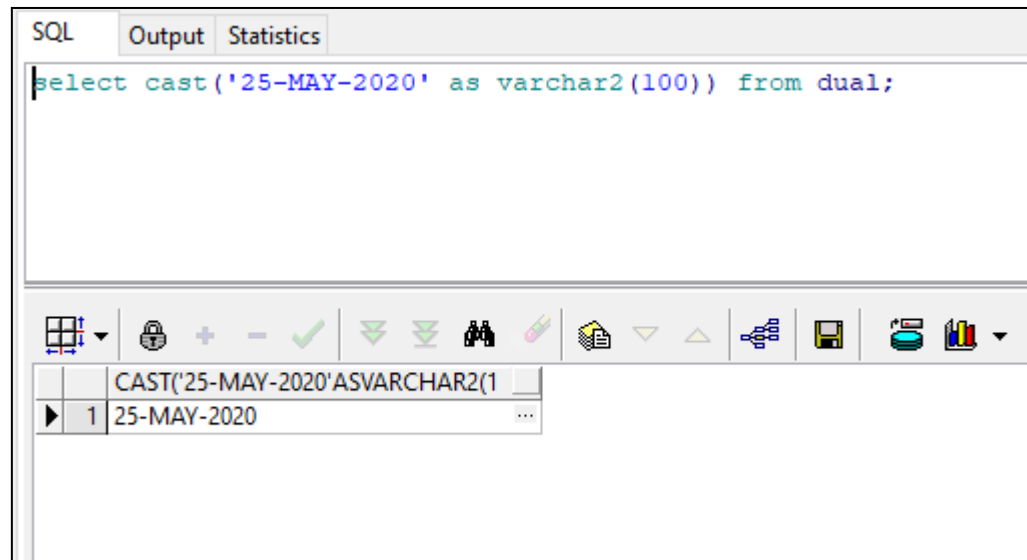
The screenshot shows the SQL Developer interface with the SQL tab selected. The SQL editor contains the query: `select cast('25-MAY-2020' as date) from dual;`. Below the editor is a toolbar with various icons. The output pane at the bottom displays the result of the query in a table format.

	CAST('25-MAY-2020'ASDATE)
1	25.05.2020



The screenshot shows the SQL Developer interface with the SQL tab selected. The SQL editor contains the query: `select cast('25-MAY-2020' as timestamp) from dual;`. Below the editor is a toolbar with various icons. The output pane at the bottom displays the result of the query in a table format.

	CAST('25-MAY-2020'ASTIMESTAMP)
1	25-MAY-20 12.00.00.000000 AM



The screenshot shows the SQL Developer interface with the SQL tab selected. The SQL editor contains the query: `select cast('25-MAY-2020' as varchar2(100)) from dual;`. Below the editor is a toolbar with various icons. The output pane at the bottom displays the result of the query in a table format.

	CAST('25-MAY-2020'ASVARCHAR2(100))
1	25-MAY-2020

Календарный тип данных

YEAR хранит год

DATE хранит дату с точностью до дня

TIMESTAMP также хранит дату и время

Тип	Описание
YEAR	0000
DATE	'0000-00-00'
TIMESTAMP	'0000-00-00 00:00:00'

Преобразование типов

❖ TO_DATE

❖ TO_NUMBER

❖ TO_CHAR

DECODE

Oracle/PLSQL функция DECODE имеет функциональные возможности оператора IF-THEN-ELSE.

Синтаксис:
















DECODE(expression , search , result [, search , result]... [, default])

SQL

Output

Statistics

```
select first_name, last_name, salary, department_id
      , decode(department_id, 90, 'Executive', 100, 'Finance', 50, 'Shipping', 'OTHER')
from employees
```

	FIRST_NAME	LAST_NAME	SALARY	DEPARTMENT_ID	DECODE(DEPARTMENT_ID,90,'EXECU
1	Steven	King	24000,00	90	Executive
2	Neena	Kochhar	17000,00	90	Executive
3	Lex	De Haan	17000,00	90	Executive
4	Alexander	Hunold	9000,00	60	OTHER
5	Bruce	Ernst	6000,00	60	OTHER
6	David	Austin	4800,00	60	OTHER
7	Valli	Pataballa	4800,00	60	OTHER
8	Diana	Lorentz	4200,00	60	OTHER
9	Nancy	Greenberg	12000,00	100	Finance
10	Daniel	Faviet	9000,00	100	Finance
11	John	Chen	8200,00	100	Finance
12	Ismael	Sciarra	7700,00	100	Finance
13	Jose Manuel	Urman	7800,00	100	Finance
14	Luis	Popp	6900,00	100	Finance
15	Den	Raphaely	11000,00	30	OTHER
16	Alexander	Khoo	3100,00	30	OTHER
17	Shelli	Baida	2900,00	30	OTHER
18	Sigal	Tobias	2800,00	30	OTHER
19	Guy	Himuro	2600,00	30	OTHER
20	Karen	Colmenares	2500,00	30	OTHER
21	Matthew	Weiss	8000,00	50	Shipping
22	Adam	Fripp	8200,00	50	Shipping
23	Payam	Kaufling	7900,00	50	Shipping
24	Shanta	Vollman	6500,00	50	Shipping
25	Kevin	Mourgos	5800,00	50	Shipping
26	Julia	Nayer	3200,00	50	Shipping
27	Irene	Mikkilineni	2700,00	50	Shipping
28	James	Landry	2400,00	50	Shipping
29	Steven	Markle	2200,00	50	Shipping
30	Laura	Bissot	3300,00	50	Shipping
31	Mozhe	Atkinson	2800,00	50	Shipping
32	James	Marlow	2500,00	50	Shipping
33	TJ	Olson	2100,00	50	Shipping
34	Jason	Mallin	3300,00	50	Shipping
35	Michael	Rogers	2900,00	50	Shipping

CASE

Оператор **CASE** имеет функциональность IF-THEN-ELSE

Синтаксис:

















```
CASE [ expression ]  
WHEN condition_1 THEN result_1  
WHEN condition_2 THEN result_2  
...  
WHEN condition_n THEN result_n  
ELSE result  
END
```

SQL

Output

Statistics












```
select first_name, last_name, salary, hire_date
      , case when hire_date > to_date('01-01-2000','DD-MM-YYYY')
            then '21 century'
            else '20 century' end as century
from employees
```

	FIRST_NAME	LAST_NAME	SALARY	HIRE_DATE	CENTURY
11	John	Chen	8200,00	28.09.1997	20 century
12	Ismael	Sciarra	7700,00	30.09.1997	20 century
13	Jose Manuel	Urman	7800,00	07.03.1998	20 century
8	Diana	Lorentz	4200,00	07.02.1999	20 century
9	Nancy	Greenberg	12000,00	17.08.1994	20 century
10	Daniel	Faviet	9000,00	16.08.1994	20 century
28	James	Landry	2400,00	14.01.1999	20 century
40	John	Seo	2700,00	12.02.1998	20 century
41	Joshua	Patel	2500,00	06.04.1998	20 century
42	Trenna	Rajs	3500,00	17.10.1995	20 century
36	Ki	Gee	2400,00	12.12.1999	20 century
38	Renske	Ladwig	3600,00	14.07.1995	20 century
39	Stephen	Stiles	3200,00	26.10.1997	20 century
43	Curtis	Davies	3100,00	29.01.1997	20 century
47	Karen	Partners	13500,00	05.01.1997	20 century
48	Alberto	Errazuriz	12000,00	10.03.1997	20 century
49	Gerald	Cambrault	11000,00	15.10.1999	20 century
44	Randall	Matos	2600,00	15.03.1998	20 century
45	Peter	Vargas	2500,00	09.07.1998	20 century
46	John	Russell	14000,00	01.10.1996	20 century
33	TJ	Olson	2100,00	10.04.1999	20 century
34	Jason	Mallin	3300,00	14.06.1996	20 century
35	Michael	Rogers	2900,00	26.08.1998	20 century
32	James	Marlow	2500,00	16.02.1997	20 century
31	Mozhe	Atkinson	2800,00	30.10.1997	20 century
30	Laura	Bissot	3300,00	20.08.1997	20 century
29	Steven	Markle	2200,00	08.03.2000	21 century
50	Eleni	Zlotkey	10500,00	29.01.2000	21 century
74	Sundita	Kumar	6100,00	21.04.2000	21 century
80	Charles	Johnson	6200,00	04.01.2000	21 century
37	Hazel	Philtanker	2200,00	06.02.2000	21 century
66	David	Lee	6800,00	23.02.2000	21 century
65	Mattea	Marvins	7200,00	24.01.2000	21 century
67	Sundar	Ande	6400,00	24.03.2000	21 century
100	Douglas	Grant	2600,00	13.01.2000	21 century
84	Girard	Geoni	2800,00	03.02.2000	21 century
68	Amit	Banda	6200,00	21.04.2000	21 century

ROUND

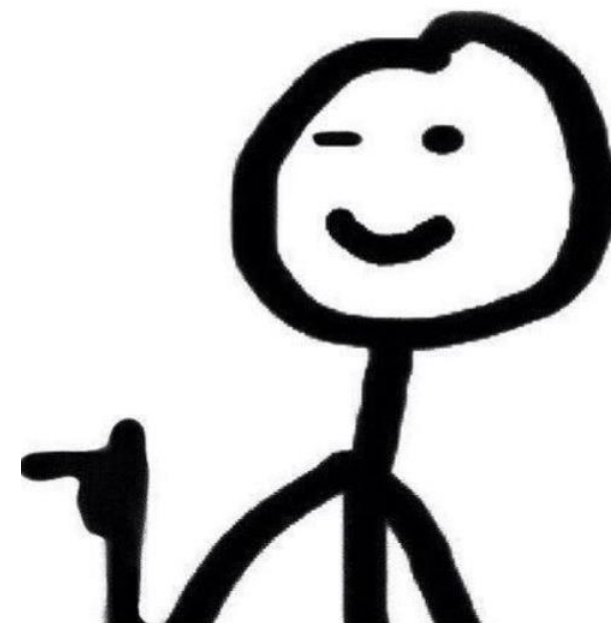
ROUND(number, [decimal_places])

SQL		Output	Statistics
<pre>select round(22.765,2) from dual</pre>			
    			
		    	
		ROUND(22.765,2)	
	1	22,77	

LOWER, UPPER

LOWER – приведение к нижнему регистру
UPPER - приведение к верхнему регистру

ДА ТЫ КЭП



Подзапросы

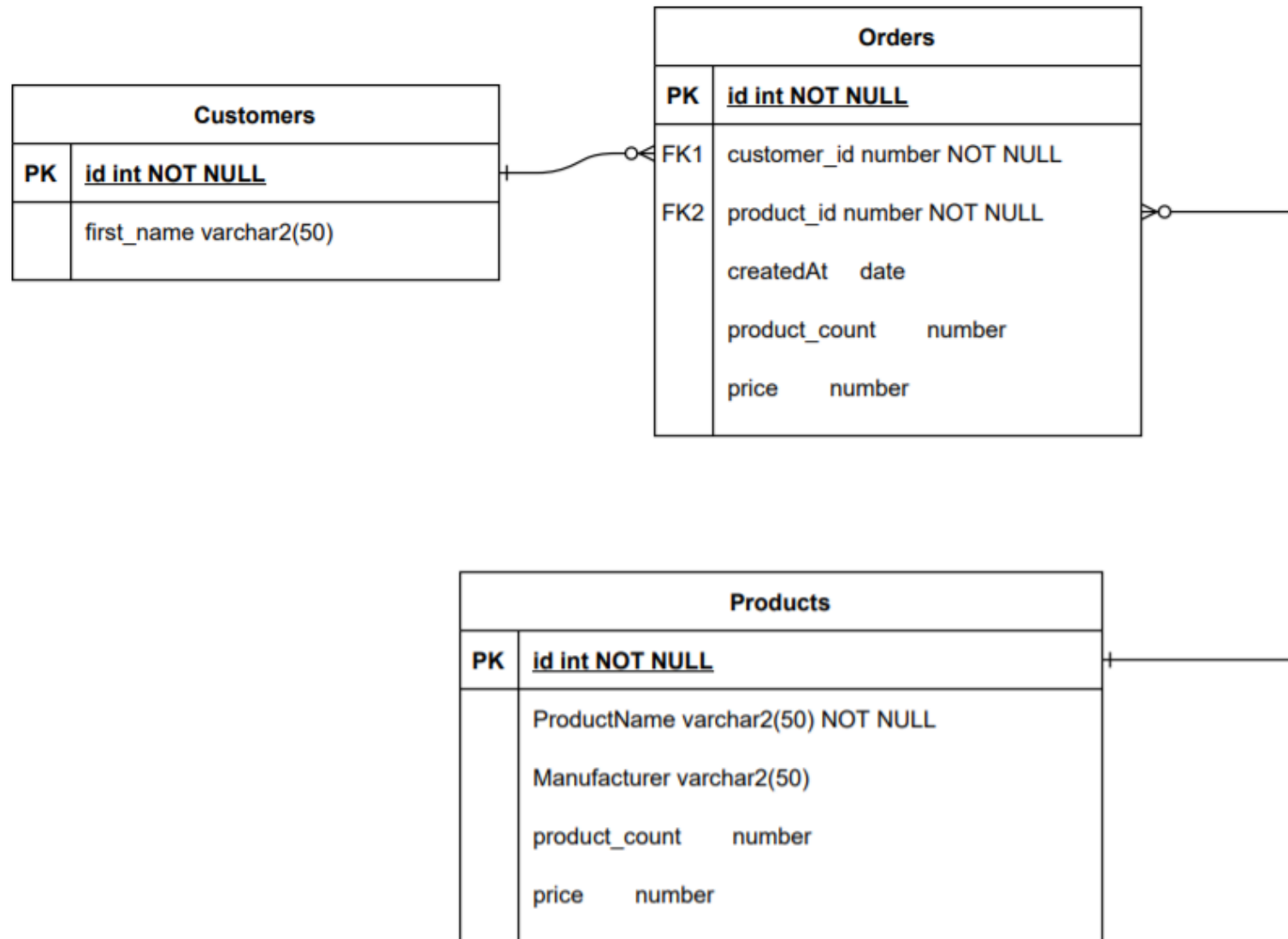
В выражении SELECT мы можем вводить подзапросы четырьмя способами:

- В условии в выражении WHERE
- В условии в выражении HAVING
- В качестве таблицы для выборки в выражении FROM
- В качестве спецификации столбца в выражении SELECT

```
select *  
from employees  
where salary > (select avg(salary) from employees)
```

Подзапросы

CUSTOMERS
ORDERS
PRODUCTS



Подзапросы

WHERE

```
SELECT * FROM Products  
WHERE Id IN (SELECT ProductId FROM Orders)
```

SELECT

```
SELECT o.*,  
(SELECT ProductName FROM Products WHERE Id = o.ProductId) AS Product  
FROM Orders o
```

FROM

```
select *  
from Orders o,  
(select * from Customers) cu  
where cu.id = o.customerid
```












Подзапросы

SQL

Output

Statistics

```
SELECT *  
FROM Products  
WHERE Id IN (SELECT ProductId FROM Orders)
```












	ID	PRODUCTNAME	MANUFACTURER	PRODUCTCOUNT	PRICE		
▶ 1	4	Galaxy S8	...	Samsung	...	2	46000
▶ 2	2	iPhone 6S	...	Apple	...	2	41000

SQL

Output

Statistics

```
SELECT o.*,  
(SELECT ProductName FROM Products WHERE Id = o.ProductId) AS Product  
FROM Orders o
```












	ID	PRODUCTID	CUSTOMERID	CREATEDAT	PRODUCTCOUNT	PRICE	PRODUCT
▶ 1	1	4	1	11.07.2017	2	46000	Galaxy S8 ...
▶ 2	2	2	1	13.07.2017	1	41000	iPhone 6S ...
▶ 3	3	2	2	11.07.2017	1	41000	iPhone 6S ...

SQL

Output

Statistics

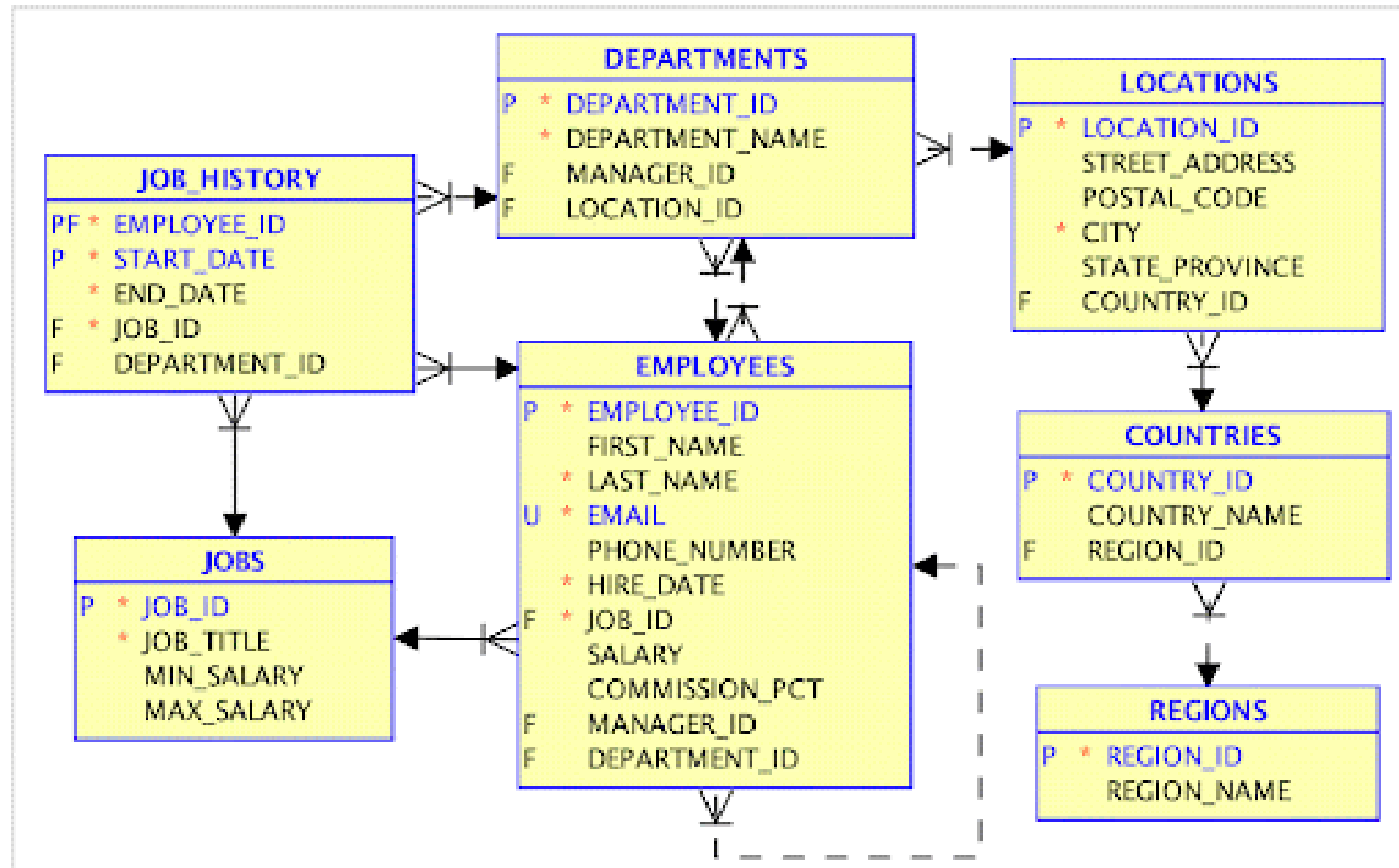
```
select *  
from Orders o,  
(select * from Customers) cu  
where cu.id = o.customerid  
|
```

	ID	PRODUCTID	CUSTOMERID	CREATEDAT	PRODUCTCOUNT	PRICE	ID	FIRSTNAME
▶ 1	2	2	1	13.07.2017	1	41000	1	Tom
▶ 2	1	4	1	11.07.2017	2	46000	1	Tom
▶ 3	3	2	2	11.07.2017	1	41000	2	Bob

SQL



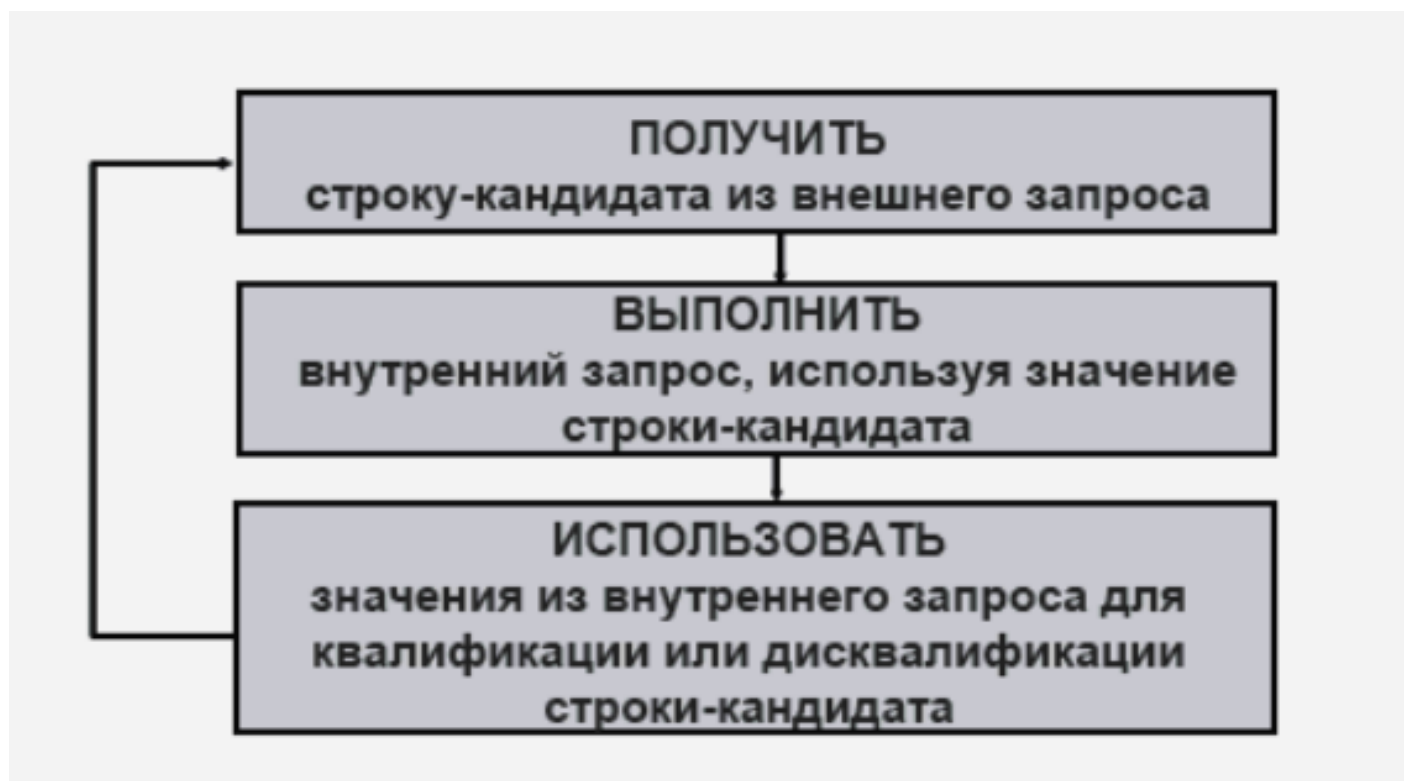


Коррелирующие подзапросы

Коррелирующие подзапросы позволяют иногда очень кратко написать запросы, которые могут выглядеть весьма громоздко при использовании других языковых средств. Напомним, что коррелирующий подзапрос — это подзапрос, который содержит ссылку на столбцы из включающего его запроса (назовем его основным). Таким образом, коррелирующий подзапрос будет выполняться для каждой строки основного запроса, так как значения столбцов основного запроса будут меняться.

```
select *  
from employees  
where department_id in (10,50)
```

Коррелирующие подзапросы



Коррелирующие подзапросы

```
SELECT o.createdat  
      , o.price  
      , (SELECT ProductName FROM Products p WHERE p.Id = o.ProductId) AS Product  
FROM orders o;
```

```
SELECT ProductName  
      , Manufacturer  
      , Price  
      , (SELECT AVG(Price)  
          FROM Products SubProds  
          WHERE SubProds.Manufacturer=Prods.Manufacturer) AS AvgPrice  
FROM Products Prods  
WHERE Price >  
      (SELECT AVG(Price)  
        FROM Products SubProds  
        WHERE SubProds.Manufacturer=Prods.Manufacturer)
```

Коррелирующие подзапросы

SQL Output Statistics

```
SELECT o.createdat, o.price  
      , (SELECT ProductName FROM Products p WHERE p.Id = o.ProductId) AS Product  
FROM orders o;
```

Grid View | Lock | + | - | ✓ | Filter | Sort | Group By | Save | Print | Chart

	CREATEDAT	PRICE	PRODUCT
▶ 1	11.07.2017	46000	Galaxy S8 ...
2	13.07.2017	41000	iPhone 6S ...
3	11.07.2017	41000	iPhone 6S ...

SQL Output Statistics

```
SELECT ProductName  
      , Manufacturer  
      , Price  
      , (SELECT AVG(Price)  
          FROM Products SubProds  
          WHERE SubProds.Manufacturer=Prods.Manufacturer) AS AvgPrice  
FROM Products Prods  
WHERE Price >  
      (SELECT AVG(Price)  
        FROM Products SubProds  
        WHERE SubProds.Manufacturer=Prods.Manufacturer)
```

Grid View | Lock | + | - | ✓ | Filter | Sort | Group By | Save | Print | Chart

	PRODUCTNAME	MANUFACTURER	PRICE	AVGPRICE
▶ 1	Galaxy S8 Plus ...	Samsung ...	56000	51000
2	iPhone 7 ...	Apple ...	52000	43000

Вертикальное соединение таблиц

✓ UNION











✓ INTERSECT

✓ MINUS

UNION

SQL Output Statistics

```
select 'ololol' from dual
union
select '1' from dual
union
select 'zazazaza' from dual
```














		'OLOLOL'
▶	1	1
	2	ololol
	3	zazazaza

SQL Output Statistics

```
select *
from employees
where job_id = 'FI_ACCOUNT'

UNION

select *
from employees
where job_id = 'IT_PROG'
```

	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_PCT	MANAGER_ID	DEPARTMENT_ID
▶	1	103	Alexander	Hunold	AHUNOLD	590.423.4567	03.01.1990	IT_PROG	9000,00	102	60
	2	104	Bruce	Ernst	BERNST	590.423.4568	21.05.1991	IT_PROG	6000,00	103	60
	3	105	David	Austin	DAUSTIN	590.423.4569	25.06.1997	IT_PROG	4800,00	103	60
	4	106	Valli	Pataballa	VPATABAL	590.423.4560	05.02.1998	IT_PROG	4800,00	103	60
	5	107	Diana	Lorentz	DLORENTZ	590.423.5567	07.02.1999	IT_PROG	4200,00	103	60
	6	109	Daniel	Faviet	DFAVIET	515.124.4169	16.08.1994	FI_ACCOUNT	9000,00	108	100
	7	110	John	Chen	JCHEN	515.124.4269	28.09.1997	FI_ACCOUNT	8200,00	108	100
	8	111	Ismael	Sciarra	ISCIARRA	515.124.4369	30.09.1997	FI_ACCOUNT	7700,00	108	100
	9	112	Jose Manuel	Urman	JMURMAN	515.124.4469	07.03.1998	FI_ACCOUNT	7800,00	108	100
	10	113	Luis	Popp	LPOPP	515.124.4567	07.12.1999	FI_ACCOUNT	6900,00	108	100

INTERSECT

```
with t1 as
(
select '1' as id, 'ololol' as name from dual
UNION
select '2' as id, 'ololol2' as name from dual
UNION
select '3' as id, 'ololol3' as name from dual
), t2 as
(
select '1' as id, 'ololol' as name from dual
UNION
select '2' as id, 'ololol2' as name from dual
UNION
select '6' as id, 'ololol6' as name from dual
)
```

```
select * from t1
INTERSECT
select * from t2
```

SQL Output Statistics

```
with t1 as
(
select '1' as id, 'ololol' as name from dual
UNION
select '2' as id, 'ololol2' as name from dual
UNION
select '3' as id, 'ololol3' as name from dual
), t2 as
(
select '1' as id, 'ololol' as name from dual
UNION
select '2' as id, 'ololol2' as name from dual
UNION
select '6' as id, 'ololol6' as name from dual
)

select * from t1
INTERSECT
select * from t2
```

	ID	NAME
1	1	ololol
2	2	ololol2

MINUS

SQL Output Statistics

```
with t1 as
(
select '1' as id, 'ololol1' as name from dual
UNION
select '2' as id, 'ololol2' as name from dual
UNION
select '3' as id, 'ololol3' as name from dual
), t2 as
(
select '1' as id, 'ololol1' as name from dual
UNION
select '2' as id, 'ololol2' as name from dual
UNION
select '6' as id, 'ololol6' as name from dual
)

select * from t1
MINUS
select * from t2
```

Grid View | Lock | + | - | ✓ | ↘ | ↙ | 🔍 | 🖌️ | 📄 | ⏏ | ⏮ | ⏭ | 📊 | 📁 | 🖨️ | 🔄

	ID	NAME
▶ 1	3	ololol3

SQL Output Statistics

```
with t1 as
(
select '1' as id, 'ololol1' as name from dual
UNION
select '2' as id, 'ololol2' as name from dual
UNION
select '3' as id, 'ololol3' as name from dual
), t2 as
(
select '1' as id, 'ololol1' as name from dual
UNION
select '2' as id, 'ololol2' as name from dual
UNION
select '6' as id, 'ololol6' as name from dual
)

|
select * from t2
MINUS
select * from t1
```

Grid View | Lock | + | - | ✓ | ↘ | ↙ | 🔍 | 🖌️ | 📄 | ⏏ | ⏮ | ⏭ | 📊 | 📁 | 🖨️ | 🔄

	ID	NAME
▶ 1	6	ololol6

Расширенные возможности группировки (GROUPING SETS, ROLLUP, CUBE)

ROLLUP — оператор, который формирует промежуточные итоги для каждого указанного элемента и общий итог.

CUBE — оператор, который формирует результаты для всех возможных перекрестных вычислений.

GROUPING SETS — оператор, который формирует результаты нескольких группировок в один набор данных, другими словами, он эквивалентен UNION ALL указанным группам.

Расширенные возможности группировки (GROUPING SETS, ROLLUP, CUBE)

```
select --d.department_name, grouping (d.department_name)
       decode (grouping (d.department_name),1,'ALL
departments',d.department_name)
--, e.job_id, grouping(e.job_id)
, decode (grouping (e.job_id),1,'ALL jobs',e.job_id)
, count(*) as Total
, round(avg(e.salary),2) as AVVG
from employees e
join departments d on d.department_id = e.department_id
group by rollup(d.department_name, e.job_id);
```