



Тинькофф

ОСНОВЫ SQL

Tinkoff.ru

Февраль, 2020

- ✓ Мавлютова Анна Павловна
- ✓ 3 года в компании
- ✓ Начала свой путь джуном в Тинькофф
- ✓ Руководжу бизнес-направлением SME



Предыстория





-  Введение в SQL
-  Основные блоки SQL запроса
-  Фильтрация
-  Операторы и функции
-  Группировка и агрегаты
-  Механика SQL запроса



SQL (Structured Query Language) — декларативный язык программирования, применяемый для создания, модификации и управления данными в реляционной базе данных, управляемой соответствующей системой управления базами данных (СУБД).





- ✓ Говорим, **что** делать, а не **как** делать (декларативный язык)

Find all the data about the customer whose name is Ivan.

```
Select * from customer where first_nm = 'ИВАН';
```

- ✓ Работает с **реляционной** моделью – представление данных посредством таблиц
- ✓ Независимость от конкретной СУБД
- ✓ Наличие стандартов SQL



- ✓ Запрос - одна SQL команда
- ✓ В SQL запросы разделяются точкой с запятой

```
CREATE TABLE my_super_table (  
    id INT  
    ,name VARCHAR(30)  
);  
  
INSERT INTO my_super_table  
VALUES (1, 'Cosa Nostra');  
  
SELECT *  
from my_super_table;
```



Data Definition Language

- ✓ CREATE - Создает объекты базы данных
- ✓ ALTER - Изменяет структуру и объекты базы данных
- ✓ DROP - Удаляет объекты базы данных
- ✓ TRUNCATE - Удаляет все записи из таблицы



Data Manipulation Language

- ✓ SELECT - Возвращает данные из базы данных
- ✓ INSERT - Вставляет данные в таблицу
- ✓ UPDATE - Обновляет существующие данные в таблице
- ✓ DELETE - Удаляет записи в таблице

Основа для
аналитики



Привилегии пользователей

✓ db_owner

✓ db_datawriter

✓ db_ddladmin

✓ db_datareader

```
CREATE ROLE Mafiosi ;  
GRANT ALL ON <Schema.Table> TO Mafiosi ;  
  
CREATE ROLE Svetochka ;  
GRANT SELECT ON <Schema.Table> TO Svetochka ;
```



Всегда надо разграничивать роли, кому можно выполнять определенные действия.



✓ Числовые типы

Name	Size	Description	Example
bigint	8 bytes	large range integer	9223372036854775807
integer	4 bytes	usual choice for integer	2147483647
smallint	2 bytes	small range integer	32767
numeric (decimal) [(p, s)]	variable	user-specified precision, exact	12,3450 (p = 6, s = 4)

✓ Дата и время

Name	Size	Description	Example
date	4 bytes	calendar date (year, month, day)	2020-02-25
time [(p)]	8 bytes	time of day only	00:05:14[.120070] (p = 6)
timestamp [(p)]	8 bytes	both date and time	2020-02-25 19:30:23 (p = 14)

* P – точность, S – масштаб



✓ Строковые

Name	Size	Description	Example
character [(n)]	1 byte + n	fixed-length, blank padded	string1
character varying [(n)]	1 byte + string size	variable-length with limit	string2
text	1 byte + string size	variable unlimited length	string3

✓ Логический тип - boolean



Преобразование типов

- ✓ Явное преобразование ТД

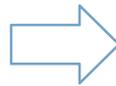
```
CAST('2020-02-01' as date)
```

```
'2020-02-01'::date
```

- ✓ Неявное преобразование ТД

- ✓ Строка автоматически преобразуется в число в выражениях, требующих числа

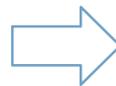
```
'25' - 5
```



```
20
```

- ✓ Число автоматически преобразуется к строке в выражениях, требующих строки

```
'Баланс = ' || 50
```



```
'Баланс = 50'
```

Преобразование типов



From \ To	binary	varbinary	char	varchar	nchar	nvarchar	datetime	smalldatetime	date	time	datetimeoffset	datetime2	decimal	numeric	float	real	bigint	int(INT4)	smallint(INT2)	tinyint(INT1)	money	smallmoney	bit	timestamp	uniqueidentifier	image	ntext	text	sql_variant	xml	CLR UDT	hierarchyid
binary		●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	
varbinary	●		●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	
char	●	●		●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	
varchar	●	●	●		●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	
nchar	●	●	●	●		●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	
nvarchar	●	●	●	●	●		●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	
datetime	●	●	●	●	●	●		●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	
smalldatetime	●	●	●	●	●	●	●		●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	
date	●	●	●	●	●	●	●	●		●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	
time	●	●	●	●	●	●	●	●	●		●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	
datetimeoffset	●	●	●	●	●	●	●	●	●	●		●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	
datetime2	●	●	●	●	●	●	●	●	●	●	●		●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	
decimal	●	●	●	●	●	●	●	●	●	●	●	●	●		●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	
numeric	●	●	●	●	●	●	●	●	●	●	●	●	●	●		●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	
float	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●		●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	
real	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●		●	●	●	●	●	●	●	●	●	●	●	●	●	●	
bigint	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●		●	●	●	●	●	●	●	●	●	●	●	●	●	
int(INT4)	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●		●	●	●	●	●	●	●	●	●	●	●	●	
smallint(INT2)	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●		●	●	●	●	●	●	●	●	●	●	●	
tinyint(INT1)	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●		●	●	●	●	●	●	●	●	●	●	
money	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●		●	●	●	●	●	●	●	●	●	
smallmoney	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●		●	●	●	●	●	●	●	●	
bit	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●		●	●	●	●	●	●	●	
timestamp	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●		●	●	●	●	●	●	
uniqueidentifier	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●		●	●	●	●	●	
image	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●		●	●	●	●	●	
ntext	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●		●	●	●	●	
text	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●		●	●	●	
sql_variant	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	
xml	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	
CLR UDT	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	
hierarchyid	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	

- Explicit conversion
- Implicit conversion
- ✗ Conversion not allowed
- ◆ Requires explicit CAST to prevent the loss of precision or scale that might occur in an implicit conversion.
- Implicit conversions between xml data types are supported only if the source or target is untyped xml. Otherwise, the conversion must be explicit.



NULL – «пустое поле»

- ✓ Состояние в любом поле, означающее, что значение **неизвестно**
- ✓ Может встретиться в любом поле с любым типом данных

id	money_produce	currency
55	1256000,00000000	RUB
56	155300,00000000	USD
56	245300,00000000	USD
57	1005680,00000000	RUB
57	714568,00000000	EUR
654	NULL	NULL

- ✓ Любая арифметическая операция с NULL возвращает NULL ($money_produce * 2.5$)

id	?column?	currency
55	3140000,00000000	RUB
56	388250,00000000	USD
56	613250,00000000	USD
57	2514200,00000000	RUB
57	1786420,00000000	EUR
654	NULL	NULL



MONEY_MAKER

```
CREATE TABLE money_maker
```

```
(
```

```
  id INTEGER,
```

```
  country CHARACTER VARYING(30),
```

```
  city CHARACTER VARYING(30),
```

```
  issue_date DATE,
```

```
  money_produce NUMERIC(24, 7),
```

```
  currency CHARACTER VARYING(3),
```

```
  actuality_start timestamp without time zone,
```

```
  actuality_end timestamp without time zone
```

```
);
```



id	country	city	issue_date	money_produce	currency	actuality_start	actuality_end
55	Россия	Ростов-на-Дону	2007-09-15 00:00:00	1256000,0000000	RUB	2019-10-20 22:37:48	5999-01-01 00:00:00
56	США	Нью-Йорк	2015-05-14 00:00:00	245300,0000000	USD	2018-11-07 14:58:22	2019-06-06 08:09:09
56	Канада	Торронтто	2015-05-14 00:00:00	155300,0000000	USD	2019-06-06 08:09:10	5999-01-01 00:00:00
57	Германия	Берлин	2017-02-10 00:00:00	714568,0000000	EUR	2019-01-10 12:13:55	2019-09-07 15:07:44
57	Германия	Мюнхен	2017-02-10 00:00:00	1005680,0000000	RUB	2019-09-07 15:07:45	5999-01-01 00:00:00



Блоки запроса

```
SELECT . . .  
FROM . . .  
WHERE . . .  
GROUP BY . . .  
HAVING . . .  
ORDER BY . . .
```

- ✓ Важна последовательность блоков
- ✓ Не все блоки обязательные

SELECT



SELECT - оператор запроса в языке SQL, возвращающий набор данных (таблицу).

```
SELECT . . .  
FROM . . .  
WHERE . . .  
GROUP BY . . .  
HAVING . . .  
ORDER BY . . .
```

SELECT



В блоке SELECT указываем набор полей выходной таблицы и способ их получения:

```
SELECT 'Hello world!';
```

```
SELECT id, money_produce  
FROM money_maker;
```

```
SELECT *  
FROM money_maker;
```



SELECT

В качестве аргументов в SELECT можно задавать поля, выражения и функции с этими полями, константы и операторы

```
SELECT
    country || ' ' || city || '.',
    'Производительность',
    money_produce * 12
FROM money_maker;
```

?column?	▼ ▢	?column?	▼ ▢	?column?	↓ Σ ▼ ▢
Германия Берлин.		Производительность		8574816,0000000	
Германия Мюнхен.		Производительность		12068160,0000000	
Канада Торранто.		Производительность		1863600,0000000	
Россия Ростов-на-...		Производительность		15072000,0000000	
США Нью-Йорк.		Производительность		2943600,0000000	
Франция Париж.		Производительность		NULL	



Можно добавлять алиасы для выбираемых полей

```
SELECT
  country||' '||city ||'.' as address,
  'Производительность' produce,
  mm.money_produce * 12 as "desirable produce"
FROM money_maker as mm;
```

address	produce	desirable produce
Германия Берлин.	Производительность	8574816,0000000
Германия Мюнхен.	Производительность	12068160,0000000
Франция Париж.	Производительность	NULL
Россия Ростов-на-Дону.	Производительность	15072000,0000000
США Нью-Йорк.	Производительность	2943600,0000000
Канада Торранто.	Производительность	1863600,0000000



- ✓ В некоторых случаях алиасы можно использовать дальше в запросе
- ✓ Алиасы можно давать не только выбираемым полям, но и таблицам

```
SELECT
```

```
    mm.country as lc, count(*) as cnt
```

```
FROM money_maker as mm
```

```
GROUP BY lc
```

```
ORDER BY cnt;
```

lc	∇ ⇨	cnt	↓ ∑ ∇ ⇨
Россия		1	
США		1	
Франция		1	
Канада		1	
Германия		2	

DISTINCT



Отбрасывает дубликаты, возвращает
ТОЛЬКО **уникальные** значения

```
SELECT
  id,
  country,
  city
FROM money_maker;
```



id	country	city
55	Россия	Ростов-на-Дону
55	Россия	Ростов-на-Дону
55	Россия	Ростов-на-Дону
56	Канада	Торронтто
56	Канада	Торронтто
56	Канада	Торронтто
57	Германия	Мюнхен
57	Германия	Мюнхен
57	Германия	Мюнхен
654	Франция	Париж
654	Франция	Париж
654	Франция	Париж

```
SELECT distinct
  id,
  country,
  city
FROM money_maker;
```



id	country	city
55	Россия	Ростов-на-Дону
56	Канада	Торронтто
57	Германия	Мюнхен
654	Франция	Париж

Блоки запроса



SELECT . . .

FROM . . .

WHERE . . .

GROUP BY . . .

HAVING . . .

ORDER BY . . .

Конвейер FROM



FROM требует таблицу

SELECT возвращает таблицу



```
SELECT * FROM (SELECT ...);
```

Может быть много уровней вложенности





Конвейер FROM

Сколько уникальных стран
(country)?

```
select count (*)  
FROM (  
    select distinct country  
    from money_maker) as c;
```

id	country	city
55	Россия	Ростов-на-Дону
55	Россия	Ростов-на-Дону
55	Россия	Ростов-на-Дону
56	Канада	Торронтто
56	Канада	Торронтто
56	Канада	Торронтто
57	Германия	Мюнхен
57	Германия	Мюнхен
57	Германия	Мюнхен
654	Франция	Париж
654	Франция	Париж
654	Франция	Париж



В **GreenPlum** при использовании подзапросов необходимо давать им **алиас**

Блоки запроса



SELECT . . .

FROM . . .

WHERE . . .

GROUP BY . . .

HAVING . . .

ORDER BY . . .

WHERE



- ✓ Фильтрует записи, пришедшие из FROM
- ✓ Должно содержать условие
 - Условие - это **логическое** выражение любой сложности

```
SELECT * FROM money_maker  
WHERE (логическое_выражение) ;
```

Примеры логических выражений



- ✓ Равенство `country = 'США'`
- ✓ Неравенство `country <> 'США'`
- ✓ Сравнение `money_produce > 10000`
- ✓ Принадлежность интервалу `id between 50 and 60`
- ✓ Сравнение строки с маской `city LIKE ('%МОС%')`
- ✓ Сравнение со значениями из списка `id in (18,19,20)`

```
SELECT * FROM money_maker  
WHERE country = 'США'
```



- ✓ Условия можно комбинировать при помощи операторов `and`, `or`, `not`
- ✓ Порядок применения логических операций в выражении
 1. NOT
 2. AND
 3. OR

Если логических операторов несколько, лучше

использовать скобки
NOT (`money_produce >1000` **AND**
`money_produce <5000`)



BETWEEN VS Двойное условие

```
SELECT *  
FROM money_maker  
WHERE issue_date BETWEEN  
date('1990-01-01') and  
date('1999-12-31');
```

VS

```
SELECT *  
FROM money_maker  
WHERE issue_date >= date('1990-01-01')  
and issue_date <= date('1999-12-31');
```



Логические выражения могут вернуть:

✓ TRUE

✓ FALSE

✓ UNKNOWN (NULL)

ⓘ WHERE отберёт только те строки, где вернулось TRUE



В Greenplum сравнение с NULL возвращает
NULL!

```
SELECT * FROM money_maker  
WHERE money_produce = NULL;
```

Вернет 0 строк!



IS NULL - единственная возможность проверить, что
поле равно NULL

```
SELECT *  
FROM money_maker  
WHERE money_produce IS NULL;
```



```
SELECT *  
FROM money_maker  
WHERE id = 3 or id = 5;
```

Как записать
компактнее?

```
SELECT *  
FROM money_maker  
WHERE id in (3, 5);
```

И ЧОВА NULL



```
SELECT *  
FROM money_maker  
WHERE id in (3, NULL);
```

VS



```
SELECT *  
FROM money_maker  
WHERE id not in (3, NULL);
```



```
SELECT *  
FROM money_maker  
WHERE id = 3  
      or id = NULL;
```

TRUE(FALSE)
or NULL

VS

```
SELECT *  
FROM money_maker  
WHERE id <> 3  
      and id <> NULL;
```

TRUE(FALSE)
and NULL

NOT IN



Эти два запроса
эквивалентны.

```
SELECT *  
FROM money_maker  
WHERE id not in (3, 4);
```

```
SELECT *  
FROM money_maker  
WHERE not id in (3, 4);
```

Задач а



Выбрать уникальные станки (id), у которых **когда-нибудь была** производительность более 1 000 000



id	money_produce	actuality_start	actuality_end
55	1256000,0000000	2019-10-20 22:37:48	5999-01-01 00:00:00
56	245300,0000000	2018-11-07 14:58:22	2019-06-06 08:09:09
56	155300,0000000	2019-06-06 08:09:10	5999-01-01 00:00:00
57	714568,0000000	2019-01-10 12:13:55	2019-09-07 15:07:44
57	1005680,0000000	2019-09-07 15:07:45	5999-01-01 00:00:00
654	NULL	2015-09-07 15:07:45	5999-01-01 00:00:00

```
SELECT distinct id  
FROM money_maker  
WHERE money_produce > 1000000;
```



Задача

Вывести всю информацию о тех станках, которые **в данный момент** печатают рубли.



id	country	city	issue_date	money_produce	currency	actuality_start	actuality_end
55	Россия	Ростов-на-Дону	2007-09-15 00:00:00	1256000,0000000	RUB	2019-10-20 22:37:48	5999-01-01 00:00:00
56	США	Нью-Йорк	2015-05-14 00:00:00	245300,0000000	USD	2018-11-07 14:58:22	2019-06-06 08:09:09
56	Канада	Торронтто	2015-05-14 00:00:00	155300,0000000	USD	2019-06-06 08:09:10	5999-01-01 00:00:00
57	Германия	Берлин	2017-02-10 00:00:00	714568,0000000	EUR	2019-01-10 12:13:55	2019-09-07 15:07:44
57	Германия	Мюнхен	2017-02-10 00:00:00	1005680,0000000	RUB	2019-09-07 15:07:45	5999-01-01 00:00:00

```
SELECT *  
FROM money_maker  
WHERE currency = 'RUB'  
and actuality_end = '5999-01-01';
```

id	country	city	issue_date	money_produce	currency	actuality_start	actuality_end
55	Россия	Ростов-на-Дону	2007-09-15 00:00:00	1256000,0000000	RUB	2019-10-20 22:37:48	5999-01-01 00:00:...
57	Германия	Мюнхен	2017-02-10 00:00:00	1005680,0000000	RUB	2019-09-07 15:07:45	5999-01-01 00:00:...



LIKE возвращает TRUE, если строка похожа на шаблон

Подстановочное выражение	Описание
%	Ноль и более символов
_ (подчеркивание)	Ровно один символ

```
SELECT * FROM money_maker  
WHERE city like 'САН-%'
```



Возвращает тот или иной результат в зависимости от условия

```
SELECT id,  
(  
CASE  
    WHEN money_produce is NULL THEN 'Неизвестно'  
    WHEN money_produce < 100000 THEN 'Мало'  
    ELSE 'Много'  
END  
) as income_str  
FROM money_maker
```



Операторы и функции

- ✓ Арифметические операции
- ✓ Строковые операции
- ✓ Операции с данными даты и времени
- ✓ Разнообразные системные операции



Арифметические операции

- ✓ Унарные операторы тождества и отрицания
- ✓ Бинарные операторы умножения и деления (+, -)
- ✓ Бинарные операторы сложения и вычитания (*, /)

! В Greenplum операция «/» является целочисленным делением, если работаем с целыми числами ($5/2 = 2$). Чтобы получить дробное число, можно сделать, например, так: *5/2::numeric*.



Строковые операции

- ✓ || `select 'S' || 'Q' || 'L'; --SQL`
- ✓ substr() `select substr('noSQL', 3, 3); --SQL`
- ✓ Trim(), Ltrim(), Rtrim()
`select trim('noSQL', 'no'); --SQL`
- ✓ upper(), lower() `select upper('sql'); --SQL`
- ✓ length() `select length('SQL'); --3`
- ✓ position() `select position('QL' in 'SQL'); --2`



ФУНКЦИИ И NULL

```
1 CASE
    WHEN выр_1 IS NOT NULL THEN выр_1
    WHEN выр_2 IS NOT NULL THEN выр_2
    ELSE выр_3
END
```

Чем заменить?

```
COALESCE (выр_1, выр_2, выр_3);
```

```
2 CASE
    WHEN выр_1 = значение_1 THEN рез_1
    WHEN выр_1 = значение_2 THEN рез_2
    ELSE рез_3
END
```

А это выражение?

```
DECODE (выр_1, значение_1, рез_1, значение_2, рез_2, рез_3);
```



- Арифметические операции с данными даты, времени и интервалами

```
select '2019-03-02 14:15:24'::timestamp -  
'2019-02-01 12:20:01'::timestamp +  
interval '1 hour'; --29.02:55:23
```

- Выделение части даты или интервала

```
select extract(week from '2019-02-27'::date); --9
```

- Округление значения даты или интервала

```
select date_trunc('week', '2019-02-27'::date); --2019-02-25
```

- Извлечение текущих даты и времени

```
current_date, current_time, current_timestamp
```



Функции, обрабатывающие сразу все строки входной таблицы

- ✓ SUM - сумма
- ✓ AVG - среднее значение
- ✓ MAX - максимальное значение
- ✓ MIN - минимальное значение
- ✓ COUNT - количество строк

```
SELECT AVG(money_produce)  
from money_maker
```



Агрегация и NULL

→ `select sum(money_produce)`
`from money_maker ;`

17500

→ `select avg(money_produce)`
`from money_maker ;`

3500

→ `select max(money_produce)`
`from money_maker ;`

4500

→ `select min(money_produce)`
`from money_maker ;`

2500

→ `select count(money_produce)`
`from money_maker ;`

5

→ `select count(*)`
`from money_maker ;`

6

id	money_produce
1	2500
2	3000
3	3500
4	4000
5	4500
6	NULL



Функции агрегации
не учитывают NULL,
за исключением
`count(*)`



COUNT and DISTINCT

Сколько уникальных стран
(location_country)?

```
select count(*)  
FROM (  
    select distinct country  
    from dev_wrk.money_maker) a;
```

id	country	city
55	Россия	Ростов-на-Дону
55	Россия	Ростов-на-Дону
55	Россия	Ростов-на-Дону
56	Канада	Торронтто
56	Канада	Торронтто
56	Канада	Торронтто
57	Германия	Мюнхен
57	Германия	Мюнхен
57	Германия	Мюнхен
654	Франция	Париж
654	Франция	Париж
654	Франция	Париж

```
select count(distinct country)  
FROM money_maker ;
```



Агрегаты

Суммарная производительность станков за всё время?

```
SELECT sum(money_produce) as income_sum  
FROM money_maker  
;
```

issue_date	money_produce
1998-02-04	50 000
1941-05-06	60 000
1941-10-10	10 000
1967-09-17	70 000
1921-02-08	50 000
1967-09-24	60 000
1941-04-15	10 000
1998-10-20	80 000



income_sum
390 000



Агрегаты

Суммарная производительность станков , выпущенных в 1941 году?

```
SELECT sum(money_produce) as income_sum  
FROM money_maker  
WHERE extract('year' from issue_date) = 1941;
```

issue_date	money_produce
1998-02-04	50 000
1941-05-06	60 000
1941-10-10	10 000
1967-09-17	70 000
1921-02-08	50 000
1967-09-24	60 000
1941-04-15	10 000
1998-10-20	80 000



income_sum
80 000

Блоки запроса



```
SELECT . . .  
FROM . . .  
WHERE . . .  
GROUP BY . . .  
HAVING . . .  
ORDER BY . . .
```



GROUP BY

Где и сколько станков?

```
SELECT country, count (*) as cnt  
FROM money_maker  
GROUP BY country;
```

country
Россия
Россия
Сша
Германия
Германия
Сша
Сша
Франция



country	cnt
Россия	2
Сша	3
Германия	2
Франция	1

Каждая группа преобразуется ровно в одну строку.



GROUP BY

- ✓ Поля, выводимые группировкой, могут быть любым выражением, но только над колонками из GROUP BY
- ✓ Нельзя добавлять в SELECT поля, не участвующие в группировке

SELECT

`extract('year' from issue_date) as year,`

 `location_country,`

`avg(money_produce) as income_sum`

FROM money_maker

GROUP BY **year;**



Но можно добавлять **константы**

```
SELECT
    'Год' as year,
    extract('year' from issue_date) as
    issue_year,
    avg(money_produce) as income_sum
FROM dev_wrk.money_maker
GROUP BY [year], issue_year;
```



- ✓ Полей с агрегацией может быть сколько угодно.
- ✓ Можно использовать выражения.

```
select
  extract('year' from issue_date) as issue_year,
  avg(money_produce) as income_sum,
  count(distinct id) as id_cnt,
  max(money_produce) - min(money_produce) as
  income_delta
FROM money_maker
GROUP BY issue_year;
```



Агрегирующих функций может и не быть при группировке

```
SELECT  
extract('year' from issue_date) as issue_year  
FROM money_maker  
GROUP BY issue_year;
```

Блоки запроса



```
SELECT . . .  
FROM . . .  
WHERE . . .  
GROUP BY . . .  
HAVING . . .  
ORDER BY . . .
```



HAVING

Вывести только те страны, где за всё время было более 2 станков

```
select country, cnt from (  
  select country,  
  count(*) as cnt  
  FROM money_maker GROUP BY country) a
```

WHERE

country
Россия
Россия
Сша
Германия
Германия
Сша
Сша
Франция



country	cnt
Россия	2
Сша	3
Германия	2
Франция	1



country	cnt
Сша	3



Аналогичный результат с использованием **HAVING**

```
SELECT
    country, count (*) as cnt
FROM money_maker
GROUP BY country
HAVING count (*) > 2;
```



HAVING

- ! В условии HAVING может быть любая функция агрегации. Даже функция, которая не встречается в SELECT
- ! Также можно включать логические выражения и поля группировки

```
SELECT
    extract('year' from issue_date) as year,
    sum(money_produce) as income_sum
FROM money_maker
GROUP BY year
HAVING count(*) > 1 and
extract('year' from issue_date) in (1980,
1981);
```

Блоки запроса



```
SELECT . . .  
FROM . . .  
WHERE . . .  
GROUP BY . . .  
HAVING . . .  
ORDER BY . . .
```



- ✔ Позволяет упорядочить строки результата по значению

```
SELECT * FROM money_maker ORDER BY id;
```
- ✔ Можно упорядочивать сразу по нескольким столбцам, порядок столбцов важен



ORDER BY

1 `SELECT * FROM money_maker
ORDER BY id, actuality_start`

id	↓ ∑ ▾ ▹ ▸	actuality_start	▾ ▹ ▸	actuality_end	▾ ▹ ▸
55		2019-10-20 22:37:48		5999-01-01 00:00:00	
56		2018-11-07 14:58:22		2019-06-06 08:09:09	
56		2019-06-06 08:09:10		5999-01-01 00:00:00	
57		2019-01-10 12:13:55		2019-09-07 15:07:44	
57		2019-09-07 15:07:45		5999-01-01 00:00:00	
654		2015-09-07 15:07:45		5999-01-01 00:00:00	

!=

2 `SELECT * FROM money_maker
ORDER BY actuality_start, id`

id	↓ ∑ ▾ ▹ ▸	actuality_start	▾ ▹ ▸	actuality_end	▾ ▹ ▸
654		2015-09-07 15:07:45		5999-01-01 00:00:00	
56		2018-11-07 14:58:22		2019-06-06 08:09:09	
57		2019-01-10 12:13:55		2019-09-07 15:07:44	
56		2019-06-06 08:09:10		5999-01-01 00:00:00	
57		2019-09-07 15:07:45		5999-01-01 00:00:00	
55		2019-10-20 22:37:48		5999-01-01 00:00:00	

ORDER BY



- ✓ ASC - от меньшего к большему (по умолчанию)
- ✓ DESC - от большего к меньшему

```
SELECT * FROM money_maker  
ORDER BY id ASC, actuality_start DESC
```

```
SELECT * FROM money_maker  
ORDER BY id DESC, actuality_start
```



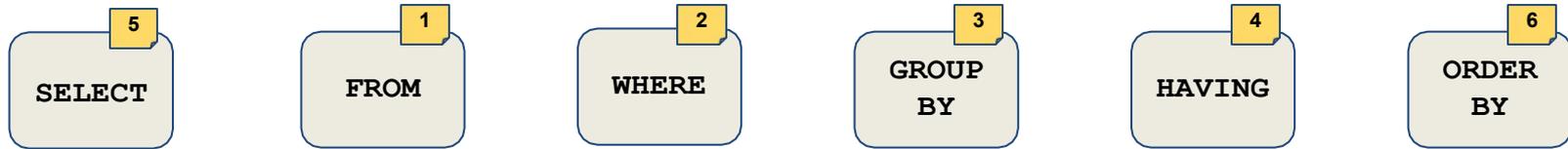
- ✓ ORDER BY влияет только на порядок сортировки при выводе команды SELECT, например на экран
- ✓ Если результат команды SELECT с ORDER BY занести в таблицу, то упорядоченность строк пропадет, т. к. таблица
- это неупорядоченное множество

Можно, но бессмысленно:

```
CREATE table new_table AS  
SELECT * from money_maker  
ORDER BY id;
```



Последовательность выполнения запроса



1. Взять все строки входной таблицы
2. Оставить только строки, где логическое выражение в `WHERE` равно `TRUE`
3. Сгруппировать
 - a. Разбить дошедшие строки на группы
 - b. Каждую группу схлопнуть (сагрегировать) до одной строки
4. Оставить только те строки, где логическое выражение `HAVING` равно `TRUE`
5. Определить столбцы результирующего набора
6. Отсортировать результат



- ✓ Необходимо подключиться к Greenplum и самостоятельно написать 3 запроса. Проверить их работоспособность запуском в БД. Результат работы на портале – прикрепленные запросы.
- ✓ Дедлайн выполнения ДЗ – 0:00 с 2 на 3 марта



По таблице `public.money_maker` вывести количество станков(`id`) в разбивке по году выпуска, которые когда-либо печатали деньги в размере 50 000 и когда-нибудь печатали деньги в размере 60 000.

- ✓ Ответ – SQL запрос
- ✓ SQL-запрос должен корректно обрабатываться на стендовом Greenplum



По таблице `public.money_maker` вывести все станки (`id`) и их максимальную производительность (`money_produce`), но только тех, у которых максимальная производительность за всю историю как минимум в два раза больше, чем минимальная за всю историю и которые за всю историю печатали банкноты в нескольких валютах

- ✓ (currency)
Ответ – SQL запрос
- ✓ SQL-запрос должен корректно обрабатывать на стендовом Greenplum



По таблице `public.money_maker` необходимо вывести максимальную производительность в разбивке по году производства станка (`issue_date`), произведённых в этом веке, предварительно рассчитав среднюю производительность по каждому станку. Производительность округлить до сотых.

- ✓ Ответ – SQL запрос
- ✓ SQL-запрос должен корректно обрабатываться на стендовом Greenplum



Тинькофф

Дальше действовать будем
мы!

Tinkoff.ru