ТЕХНОТРЕК

Занятие № 3

Работа с данными. Выборка.

Дина Сафина, Лысак Павел

Работа с данными. Выборка.

1. SELECT
2. JOIN
3. UNION
4. GROUP BY(Aggregation)
5. SUBQUERY
6. VIEW

SELECT

```
SELECT
    [ALL | DISTINCT | DISTINCTROW ]
      [HIGH_PRIORITY]
      [STRAIGHT_JOIN]
      [SQL_SMALL_RESULT] [SQL_BIG_RESULT] [SQL_BUFFER_RESULT]
      SQL_NO_CACHE [SQL_CALC_FOUND_ROWS]
    select_expr [, select_expr ...]
    [FROM table_references
      [PARTITION partition_list]
    [WHERE where_condition]
    [GROUP BY {col_name | expr | position}, ... [WITH ROLLUP]]
    [HAVING where_condition]
    [WINDOW window_name AS (window_spec)
        [, window_name AS (window_spec)] ...]
    [ORDER BY {col_name | expr | position}
      [ASC | DESC], ... [WITH ROLLUP]]
    [LIMIT {[offset,] row_count | row_count OFFSET offset}]
```

SELECT

```
1 • SELECT *
2   FROM db_track.users
3   WHERE user_id <= 5;
```

#	user_id	login	reg_dttm
1	1	login_1	2018-08-07 23:29:55
2	2	login_2	2018-09-29 22:31:33
3	3	login_3	2018-08-10 12:56:02
4	4	login_4	2018-09-14 01:26:52
5	5	login_5	2018-08-08 05:53:20

Comparison Operators

Name	Description
BETWEEN ... AND ...	Check whether a value is within a range of values
COALESCE()	Return the first non-NULL argument
=	Equal operator
<=>	NULL-safe equal to operator
>	Greater than operator
>=	Greater than or equal operator
GREATEST()	Return the largest argument
IN()	Check whether a value is within a set of values
INTERVAL()	Return the index of the argument that is less than the first argument
IS	Test a value against a boolean
IS NOT	Test a value against a boolean
IS NOT NULL	NOT NULL value test
IS NULL	NULL value test
ISNULL()	Test whether the argument is NULL
LEAST()	Return the smallest argument
<	Less than operator
<=	Less than or equal operator
LIKE	Simple pattern matching
NOT BETWEEN ... AND ...	Check whether a value is not within a range of values
!=, <>	Not equal operator
NOT IN()	Check whether a value is not within a set of values
NOT LIKE	Negation of simple pattern matching

Comparison Operators

```
mysql> SELECT 1 <=> 1, NULL <=> NULL, 1 <=> NULL;
        -> 1, 1, 0
mysql> SELECT 1 = 1, NULL = NULL, 1 = NULL;
        -> 1, NULL, NULL
```

```
mysql> SELECT 2 BETWEEN 1 AND 3, 2 BETWEEN 3 and 1;
        -> 1, 0
mysql> SELECT 1 BETWEEN 2 AND 3;
        -> 0
mysql> SELECT 'b' BETWEEN 'a' AND 'c';
        -> 1
mysql> SELECT 2 BETWEEN 2 AND '3';
        -> 1
```

```
mysql> SELECT COALESCE(NULL,1);
        -> 1
mysql> SELECT COALESCE(NULL,NULL,NULL);
        -> NULL
```

Numeric Functions and Operators

Name	Description
DIV	Integer division
/	Division operator
-	Minus operator
%, MOD	Modulo operator
+	Addition operator
*	Multiplication operator
-	Change the sign of the argument

Name	Description
ABS()	Return the absolute value
ACOS()	Return the arc cosine
ASIN()	Return the arc sine
ATAN()	Return the arc tangent
ATAN2(), ATAN()	Return the arc tangent of the two arguments
CEIL()	Return the smallest integer value not less than the argument
CEILING()	Return the smallest integer value not less than the argument
CONV()	Convert numbers between different number bases
COS()	Return the cosine
COT()	Return the cotangent
CRC32()	Compute a cyclic redundancy check value
DEGREES()	Convert radians to degrees
EXP()	Raise to the power of
FLOOR()	Return the largest integer value not greater than the argument
LN()	Return the natural logarithm of the argument
LOG()	Return the natural logarithm of the first argument

Numeric Functions and Operators

```
mysql> SELECT SQRT(4);
        -> 2
mysql> SELECT SQRT(20);
        -> 4.4721359549996
mysql> SELECT SQRT(-16);
        -> NULL
```

```
mysql> SELECT PI();
        -> 3.141593
mysql> SELECT PI()+0.0000000000000000;
        -> 3.1415926535897931
```

```
mysql> SELECT POW(2,2);
        -> 4
mysql> SELECT POW(2,-2);
        -> 0.25
```

String Functions

Name	Description
ASCII()	Return numeric value of left-most character
BIN()	Return a string containing binary representation of a number
BIT_LENGTH()	Return length of argument in bits
CHAR()	Return the character for each integer passed
CHAR_LENGTH()	Return number of characters in argument
CHARACTER_LENGTH()	Synonym for CHAR_LENGTH()
CONCAT()	Return concatenated string
CONCAT_WS()	Return concatenate with separator

String Functions

```
mysql> SELECT 'David!' LIKE 'David_';
        -> 1
mysql> SELECT 'David!' LIKE '%D%v%';
        -> 1
```

```
mysql> SELECT 'Michael!' REGEXP '.*';
+-------------------------+
| 'Michael!' REGEXP '.*' |
+-------------------------+
|                       1 |
+-------------------------+
```

Date and Time Functions

Name	Description
ADDDATE()	Add time values (intervals) to a date value
ADDTIME()	Add time
CONVERT_TZ()	Convert from one time zone to another
CURDATE()	Return the current date
CURRENT_DATE(), CURRENT_DATE	Synonyms for CURDATE()
CURRENT_TIME(), CURRENT_TIME	Synonyms for CURTIME()
CURRENT_TIMESTAMP(), CURRENT_TIMESTAMP	Synonyms for NOW()
CURTIME()	Return the current time
DATE()	Extract the date part of a date or datetime expression
DATE_ADD()	Add time values (intervals) to a date value
DATE_FORMAT()	Format date as specified
DATE_SUB()	Subtract a time value (interval) from a date
DATEDIFF()	Subtract two dates
DAY()	Synonym for DAYOFMONTH()
DAYNAME()	Return the name of the weekday

Date and Time Functions

```
mysql> SELECT DATE_ADD('2013-01-01', INTERVAL 1 DAY);
        -> '2013-01-02'
mysql> SELECT DATE_ADD('2013-01-01', INTERVAL 1 HOUR);
        -> '2013-01-01 01:00:00'
```

```
mysql> SELECT DATE('2003-12-31 01:02:03');
         -> '2003-12-31'
```

Logical Operators

Name	Description
AND, &&	Logical AND
NOT, !	Negates value
||, OR	Logical OR
XOR	Logical XOR

```
mysql> SELECT 10 IS TRUE;
-> 1
mysql> SELECT -10 IS TRUE;
-> 1
mysql> SELECT 'string' IS NOT NULL;
-> 1
```

Control Flow Functions

Name	Description
CASE	Case operator
IF()	If/else construct
IFNULL()	Null if/else construct
NULLIF()	Return NULL if expr1 = expr2

CASE

```
CASE
    WHEN search_condition THEN statement_list
    [WHEN search_condition THEN statement_list] ...
    [ELSE statement_list]
END CASE
```

```
SELECT db_track.payments.*,
    CASE
    WHEN db_track.payments.payment_dttm > CURDATE()
    THEN 1 ELSE 0
    END AS future
FROM db_track.payments
ORDER BY future DESC;
```

#	payment_id	user_id	payment_sum	payment_dttm	future
1	2	1	75	2019-08-23 14:52:00	1
2	5	1	779	2019-09-16 05:47:55	1
3	8	1	200	2019-09-21 17:07:53	1
4	272	99	500	2019-09-29 03:34:42	1
5	1	1	407	2018-09-29 02:08:32	0
6	3	1	694	2018-08-17 23:05:26	0

alias for columns

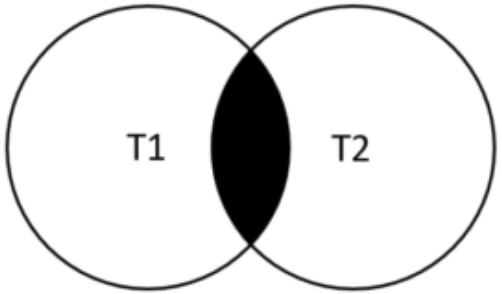
```sql
SELECT
 orderNumber `Order no.`,
 SUM(priceEach * quantityOrdered) total
FROM
 orderdetails
GROUP BY
 `Order no.`
HAVING
 total > 60000;
```

INITIAL TABLES

#	user_id	login	reg_dttm
1	1	login_1	2018-08-07 23:29:55
2	2	login_2	2018-09-29 22:31:33
3	3	login_3	2018-08-10 12:56:02
4	4	login_4	2018-09-14 01:26:52
5	5	login_5	2018-08-08 05:53:20

#	payment_id	user_id	payment_sum	payment_dttm
1	1	1	407	2018-09-29 02:08:32
2	2	1	75	2018-08-23 14:52:00
3	3	1	694	2018-08-17 23:05:26
4	4	1	438	2018-08-30 05:30:07
5	5	1	779	2018-09-16 05:47:55
6	6	1	669	2018-09-20 05:23:20
7	7	1	807	2018-08-27 10:46:21
8	8	1	200	2018-09-21 17:07:53
9	9	3	615	2018-08-13 09:30:52
10	10	3	161	2018-08-15 16:56:18
11	11	3	520	2018-08-14 13:53:11
12	12	3	330	2018-08-12 07:47:48
13	13	3	508	2018-08-20 16:12:08

INNER JOIN



#	id	data
1	1	1
2	2	2
3	3	3
4	3	4

#	id	data
1	1	10
2	2	20
3	5	50
4	6	60

```
1 •  use db_track;
2 •  SELECT *
3    FROM t1
4    JOIN t2 ON t1.id = t2.id;
```

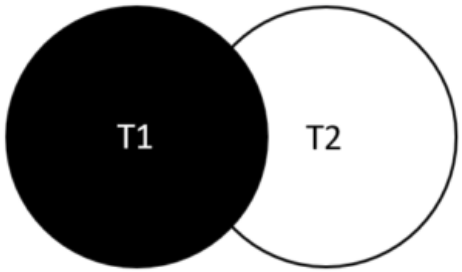
#	id	data	id	data
1	1	1	1	10
2	2	2	2	20

JOIN, INNER JOIN

```
1 • SELECT *
2   FROM track_2018.users
3   JOIN track_2018.payments ON (users.user_id = payments.user_id)
```

#	user_id	login	reg_dttm	payment_id	user_id	payment_sum	payment_dttm
1	1	login_1	2018-08-07 23:29:55	1	1	407	2018-09-29 02:08:32
2	1	login_1	2018-08-07 23:29:55	2	1	75	2018-08-23 14:52:00
3	1	login_1	2018-08-07 23:29:55	3	1	694	2018-08-17 23:05:26
4	1	login_1	2018-08-07 23:29:55	4	1	438	2018-08-30 05:30:07
5	1	login_1	2018-08-07 23:29:55	5	1	779	2018-09-16 05:47:55
6	1	login_1	2018-08-07 23:29:55	6	1	669	2018-09-20 05:23:20
7	1	login_1	2018-08-07 23:29:55	7	1	807	2018-08-27 10:46:21
8	1	login_1	2018-08-07 23:29:55	8	1	200	2018-09-21 17:07:53
9	3	login_3	2018-08-10 12:56:02	9	3	615	2018-08-13 09:30:52
10	3	login_3	2018-08-10 12:56:02	10	3	161	2018-08-15 16:56:18
11	3	login_3	2018-08-10 12:56:02	11	3	520	2018-08-14 13:53:11
12	3	login_3	2018-08-10 12:56:02	12	3	330	2018-08-12 07:47:48
13	3	login_3	2018-08-10 12:56:02	13	3	508	2018-08-20 16:12:08

LEFT JOIN



#	id	data
1	1	1
2	2	2
3	3	3
4	3	4

#	id	data
1	1	10
2	2	20
3	5	50
4	6	60

```
1 •  use db_track;
2 •  SELECT *
3    FROM t1 LEFT JOIN t2 ON t1.id = t2.id;
```

#	id	data	id	data
1	1	1	1	10
2	2	2	2	20
3	3	3	NULL	NULL
4	3	4	NULL	NULL

RIGHT JOIN

```
1 •  SELECT *
2    FROM t2 LEFT JOIN t1 ON t1.id = t2.id;
3
4 •  SELECT *
5    FROM t1 RIGHT JOIN t2 ON t1.id = t2.id;
```

#	id	data	id	data
1	1	10	1	1
2	2	20	2	2
3	5	50	NULL	NULL
4	6	60	NULL	NULL

LEFT JOIN

```
1 • SELECT *
2   FROM track_2018.users
3   LEFT JOIN track_2018.payments ON (users.user_id = payments.user_id)
```

#	user_id	login	reg_dttm	payment_id	user_id	payment_sum	payment_dttm
1	1	login_1	2018-08-07 23:29:55	1	1	407	2018-09-29 02:08:32
2	1	login_1	2018-08-07 23:29:55	2	1	75	2018-08-23 14:52:00
3	1	login_1	2018-08-07 23:29:55	3	1	694	2018-08-17 23:05:26
4	1	login_1	2018-08-07 23:29:55	4	1	438	2018-08-30 05:30:07
5	1	login_1	2018-08-07 23:29:55	5	1	779	2018-09-16 05:47:55
6	1	login_1	2018-08-07 23:29:55	6	1	669	2018-09-20 05:23:20
7	1	login_1	2018-08-07 23:29:55	7	1	807	2018-08-27 10:46:21
8	1	login_1	2018-08-07 23:29:55	8	1	200	2018-09-21 17:07:53
9	2	login_2	2018-09-29 22:31:33	NULL	NULL	NULL	NULL
10	3	login_3	2018-08-10 12:56:02	9	3	615	2018-08-13 09:30:52
11	3	login_3	2018-08-10 12:56:02	10	3	161	2018-08-15 16:56:18
12	3	login_3	2018-08-10 12:56:02	11	3	520	2018-08-14 13:53:11
13	3	login_3	2018-08-10 12:56:02	12	3	330	2018-08-12 07:47:48
14	3	login_3	2018-08-10 12:56:02	13	3	508	2018-08-20 16:12:08
15	4	login_4	2018-09-14 01:26:52	NULL	NULL	NULL	NULL
16	5	login_5	2018-08-08 05:53:20	NULL	NULL	NULL	NULL

LEFT JOIN (RIGHT JOIN)

```
1 • SELECT *
2   FROM track_2018.payments
3   LEFT JOIN track_2018.users ON (users.user_id = payments.user_id);
```

#	payment_id	user_id	payment_sum	payment_dttm	user_id	login	reg_dttm
1	1	1	407	2018-09-29 02:08:32	1	login_1	2018-08-07 23:29:55
2	2	1	75	2018-08-23 14:52:00	1	login_1	2018-08-07 23:29:55
3	3	1	694	2018-08-17 23:05:26	1	login_1	2018-08-07 23:29:55
4	4	1	438	2018-08-30 05:30:07	1	login_1	2018-08-07 23:29:55
5	5	1	779	2018-09-16 05:47:55	1	login_1	2018-08-07 23:29:55
6	6	1	669	2018-09-20 05:23:20	1	login_1	2018-08-07 23:29:55
7	7	1	807	2018-08-27 10:46:21	1	login_1	2018-08-07 23:29:55
8	8	1	200	2018-09-21 17:07:53	1	login_1	2018-08-07 23:29:55
9	9	3	615	2018-08-13 09:30:52	3	login_3	2018-08-10 12:56:02
10	10	3	161	2018-08-15 16:56:18	3	login_3	2018-08-10 12:56:02
11	11	3	520	2018-08-14 13:53:11	3	login_3	2018-08-10 12:56:02
12	12	3	330	2018-08-12 07:47:48	3	login_3	2018-08-10 12:56:02
13	13	3	508	2018-08-20 16:12:08	3	login_3	2018-08-10 12:56:02

CROSS JOIN

```sql
SELECT *
FROM track_2018.payments
CROSS JOIN track_2018.users;
```

```sql
SELECT *
FROM track_2018.payments, track_2018.users;
```

#	payment_id	user_id	payment_sum	payment_dttm	user_id	login	reg_dttm
1	1	1	407	2018-09-29 02:08:32	1	login_1	2018-08-07 23:29:55
2	1	1	407	2018-09-29 02:08:32	2	login_2	2018-09-29 22:31:33
3	1	1	407	2018-09-29 02:08:32	3	login_3	2018-08-10 12:56:02
4	1	1	407	2018-09-29 02:08:32	4	login_4	2018-09-14 01:26:52
5	1	1	407	2018-09-29 02:08:32	5	login_5	2018-08-08 05:53:20
6	2	1	75	2018-08-23 14:52:00	1	login_1	2018-08-07 23:29:55
62	13	3	508	2018-08-20 16:12:08	2	login_2	2018-09-29 22:31:33
63	13	3	508	2018-08-20 16:12:08	3	login_3	2018-08-10 12:56:02
64	13	3	508	2018-08-20 16:12:08	4	login_4	2018-09-14 01:26:52
65	13	3	508	2018-08-20 16:12:08	5	login_5	2018-08-08 05:53:20

UNION

#	user_id	login	reg_dttm
1	1	login_1	2018-10-01 00:33:01
*	NULL	NULL	NULL

#	user_id	login	reg_dttm
1	1	login_1	2018-08-07 23:29:55
2	2	login_2	2018-09-29 22:31:33
3	3	login_3	2018-08-10 12:56:02
4	4	login_4	2018-09-14 01:26:52
5	5	login_5	2018-08-08 05:53:20

UNION

```sql
SELECT *
FROM track_2018.another_users
UNION
SELECT *
FROM track_2018.users;
```

#	user_id	login	reg_dttm
1	1	login_1	2018-10-01 00:33:01
2	1	login_1	2018-08-07 23:29:55
3	2	login_2	2018-09-29 22:31:33
4	3	login_3	2018-08-10 12:56:02
5	4	login_4	2018-09-14 01:26:52
6	5	login_5	2018-08-08 05:53:20

UNION

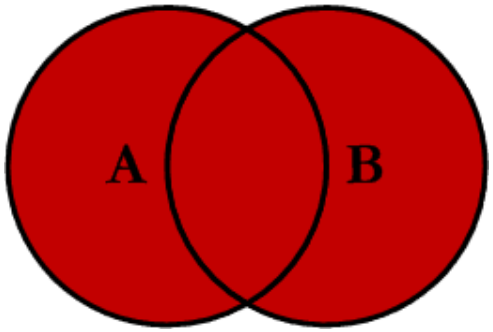
```sql
SELECT *
FROM track_2018.another_users
UNION
SELECT *
FROM track_2018.another_users;
```

#	user_id	login	reg_dttm
1	1	login_1	2018-10-01 00:33:01

```sql
SELECT *
FROM track_2018.another_users
UNION ALL
SELECT *
FROM track_2018.another_users;
```

#	user_id	login	reg_dttm
1	1	login_1	2018-10-01 00:33:01
2	1	login_1	2018-10-01 00:33:01

FULL JOIN



#	id	data	id	data
1	1	1	1	10
2	2	2	2	20
3	3	3	NULL	NULL
4	3	4	NULL	NULL

#	id	data	id	data
1	1	1	1	10
2	2	2	2	20
3	3	3	NULL	NULL
4	3	4	NULL	NULL

```
1 •  SELECT *
2    FROM t1 LEFT JOIN t2 ON t1.id = t2.id
3    UNION ALL
4    SELECT *
5    FROM t1 RIGHT JOIN t2 ON t1.id = t2.id
6    WHERE t1.id IS NULL;
```

#	id	data	id	data
1	1	1	1	10
2	2	2	2	20
3	3	3	NULL	NULL
4	3	4	NULL	NULL
5	NULL	NULL	5	50
6	NULL	NULL	6	60

GROUP BY FUNCTIONS

Name	Description
AVG()	Return the average value of the argument
BIT_AND()	Return bitwise AND
BIT_OR()	Return bitwise OR
BIT_XOR()	Return bitwise XOR
COUNT()	Return a count of the number of rows returned
COUNT(DISTINCT)	Return the count of a number of different values
GROUP_CONCAT()	Return a concatenated string
JSON_ARRAYAGG()	Return result set as a single JSON array
JSON_OBJECTAGG()	Return result set as a single JSON object
MAX()	Return the maximum value
MIN()	Return the minimum value
STD()	Return the population standard deviation
STDDEV()	Return the population standard deviation
STDDEV_POP()	Return the population standard deviation
STDDEV_SAMP()	Return the sample standard deviation
SUM()	Return the sum
VAR_POP()	Return the population standard variance
VAR_SAMP()	Return the sample variance
VARIANCE()	Return the population standard variance

GROUP BY

```sql
SELECT user_id, SUM(payment_sum)
FROM track_2018.payments
WHERE payment_sum > 100
GROUP BY user_id;
```

#	user_id	SUM(payment_sum)
1	1	3994
2	3	2134

#	payment_id	user_id	payment_sum	payment_dttm
1	1	1	407	2018-09-29 02:08:32
2	2	1	75	2018-08-23 14:52:00
3	3	1	694	2018-08-17 23:05:26
4	4	1	438	2018-08-30 05:30:07
5	5	1	779	2018-09-16 05:47:55
6	6	1	669	2018-09-20 05:23:20
7	7	1	807	2018-08-27 10:46:21
8	8	1	200	2018-09-21 17:07:53
9	9	3	615	2018-08-13 09:30:52
10	10	3	161	2018-08-15 16:56:18
11	11	3	520	2018-08-14 13:53:11
12	12	3	330	2018-08-12 07:47:48
13	13	3	508	2018-08-20 16:12:08

GROUP BY WITH ROLLUP

```sql
SELECT user_id, SUM(payment_sum)
FROM track_2018.payments
WHERE payment_sum > 100
GROUP BY user_id WITH ROLLUP;
```

#	user_id	SUM(payment_sum)
1	1	3994
2	3	2134
3	NULL	6128

GROUP BY HAVING

```sql
SELECT user_id, SUM(payment_sum) as sum
FROM track_2018.payments
WHERE payment_sum > 100
GROUP BY user_id
HAVING sum > 2500;
```

#	user_id	sum
1	1	3994

ORDER BY

```
1 •  SELECT * FROM track_2018.payments
2    ORDER BY MONTH(payment_dttm), payment_sum DESC;
```

#	payment_id	user_id	payment_sum	payment_dttm
1	7	1	807	2018-08-27 10:46:21
2	3	1	694	2018-08-17 23:05:26
3	9	3	615	2018-08-13 09:30:52
4	11	3	520	2018-08-14 13:53:11
5	13	3	508	2018-08-20 16:12:08
6	4	1	438	2018-08-30 05:30:07
7	12	3	330	2018-08-12 07:47:48
8	10	3	161	2018-08-15 16:56:18
9	2	1	75	2018-08-23 14:52:00
10	5	1	779	2018-09-16 05:47:55
11	6	1	669	2018-09-20 05:23:20
12	1	1	407	2018-09-29 02:08:32
13	8	1	200	2018-09-21 17:07:53

LIMIT

```
1 • SELECT * FROM track_2018.payments
2   ORDER BY MONTH(payment_dttm), payment_sum DESC
3   LIMIT 5;
```

#	payment_id	user_id	payment_sum	payment_dttm
1	7	1	807	2018-08-27 10:46:21
2	3	1	694	2018-08-17 23:05:26
3	9	3	615	2018-08-13 09:30:52
4	11	3	520	2018-08-14 13:53:11
5	13	3	508	2018-08-20 16:12:08

```
1 • SELECT * FROM track_2018.payments
2   ORDER BY MONTH(payment_dttm), payment_sum DESC
3   LIMIT 5, 5;
```

#	payment_id	user_id	payment_sum	payment_dttm
1	4	1	438	2018-08-30 05:30:07
2	12	3	330	2018-08-12 07:47:48
3	10	3	161	2018-08-15 16:56:18
4	2	1	75	2018-08-23 14:52:00
5	5	1	779	2018-09-16 05:47:55

SQL_CALC_FOUND_ROWS

```
mysql> SELECT SQL_CALC_FOUND_ROWS * FROM tbl_name
    -> WHERE id > 100 LIMIT 10;
mysql> SELECT FOUND_ROWS();
```

SUBQUERY (as Scalar Operand)

```
1  CREATE TABLE t1 (s1 INT, s2 CHAR(5) NOT NULL);
2  INSERT INTO t1 VALUES(100, 'abcde');
3  SELECT (SELECT s2 FROM t1);
```

Comparisons Using Subqueries

```sql
SELECT * FROM t1
  WHERE column1 = (SELECT MAX(column2) FROM t2);
```

```sql
SELECT * FROM t1 AS t
  WHERE 2 = (SELECT COUNT(*) FROM t1 WHERE t1.id = t.id);
```

Subqueries with ANY, IN, or SOME

```
operand comparison_operator ANY (subquery)
operand IN (subquery)
operand comparison_operator SOME (subquery)
```

```
SELECT s1 FROM t1 WHERE s1 > ANY (SELECT s1 FROM t2);
```

```
SELECT s1 FROM t1 WHERE s1 = ANY (SELECT s1 FROM t2);
SELECT s1 FROM t1 WHERE s1 IN    (SELECT s1 FROM t2);
```

```
SELECT s1 FROM t1 WHERE s1 <> ANY  (SELECT s1 FROM t2);
SELECT s1 FROM t1 WHERE s1 <> SOME (SELECT s1 FROM t2);
```

Subqueries with ALL

```
operand comparison_operator ALL (subquery)
```

```
SELECT s1 FROM t1 WHERE s1 > ALL (SELECT s1 FROM t2);
```

```
SELECT s1 FROM t1 WHERE s1 <> ALL (SELECT s1 FROM t2);
SELECT s1 FROM t1 WHERE s1 NOT IN (SELECT s1 FROM t2);
```

Row Subqueries

```
1  SELECT * FROM t1
2    WHERE (col1,col2) = (SELECT col3, col4 FROM t2 WHERE id = 10);
3  SELECT * FROM t1
4    WHERE ROW(col1,col2) = (SELECT col3, col4 FROM t2 WHERE id = 10);
```

```
1  SELECT * FROM t1 WHERE (column1,column2) = (1,1);
2  SELECT * FROM t1 WHERE column1 = 1 AND column2 = 1;
```

Subqueries with EXISTS or NOT EXISTS

```sql
SELECT DISTINCT store_type FROM stores
  WHERE EXISTS (SELECT * FROM cities_stores
                WHERE cities_stores.store_type = stores.store_type);
```

```sql
SELECT DISTINCT store_type FROM stores
  WHERE NOT EXISTS (SELECT * FROM cities_stores
                    WHERE cities_stores.store_type = stores.store_type);
```

Derived Tables

```sql
INSERT INTO t1 VALUES (1,'1',1.0);
INSERT INTO t1 VALUES (2,'2',2.0);
SELECT sb1,sb2,sb3
  FROM (SELECT s1 AS sb1, s2 AS sb2, s3*2 AS sb3 FROM t1) AS sb
  WHERE sb1 > 1;
```

Rewriting Subqueries as Joins

For example, this query:

```
SELECT * FROM t1 WHERE id IN (SELECT id FROM t2);
```

Can be rewritten as:

```
SELECT DISTINCT t1.* FROM t1, t2 WHERE t1.id=t2.id;
```

The queries:

```
SELECT * FROM t1 WHERE id NOT IN (SELECT id FROM t2);
SELECT * FROM t1 WHERE NOT EXISTS (SELECT id FROM t2 WHERE t1.id=t2.id);
```

Can be rewritten as:

```
SELECT table1.*
  FROM table1 LEFT JOIN table2 ON table1.id=table2.id
  WHERE table2.id IS NULL;
```

VIEW

```
CREATE [OR REPLACE]
[ALGORITHM = {UNDEFINED | MERGE | TEMPTABLE}]
VIEW view_name [(column_list)]
AS select_statement
```

```
CREATE VIEW v AS SELECT subject, num_views/num_replies AS param FROM topics WHERE
num_replies>0;
```

VIEW MERGE

```
SELECT subject, param FROM v WHERE param>1000;
```

```
SELECT subject, num_views/num_replies AS param FROM topics WHERE num_replies>0 AND
num_views/num_replies>1000;
```

VIEW TEMPTABLE

```
CREATE VIEW v AS SELECT forum_id, count(*) AS num FROM topics GROUP BY forum_id;
```

```
SELECT MAX(num) FROM v;
```

```
CREATE TEMPORARY TABLE tmp_table SELECT forum_id, count(*) AS num FROM topics GROUP BY
forum_id;
SELECT MAX(num) FROM tmp_table;
DROP TABLE tpm_table;
```

VIEW

```
mysql> CREATE TABLE t (qty INT, price INT);
mysql> INSERT INTO t VALUES(3, 50), (5, 60);
mysql> CREATE VIEW v AS SELECT qty, price, qty*price AS value FROM t;
mysql> SELECT * FROM v;
+------+-------+-------+
| qty  | price | value |
+------+-------+-------+
|    3 |    50 |   150 |
|    5 |    60 |   300 |
+------+-------+-------+
mysql> SELECT * FROM v WHERE qty = 5;
+------+-------+-------+
| qty  | price | value |
+------+-------+-------+
|    5 |    60 |   300 |
+------+-------+-------+
```

Домашнее задание №3

- Вывести логины трёх пользователей, которые заплатили больше всего денег

- Посчитать среднее количество сессий у пользователей

Срок сдачи

10.10.18

Полезные ссылки

- To SQL_CALC_FOUND_ROWS or not to SQL_CALC_FOUND_ROWS?
- Views tutorial
- Mysql Documentation
- http://www.sql-ex.ru/

ТЕХНОТРЕК

Спасибо за внимание!