ТЕХНОТРЕК

Занятие №4

# Аналитические запросы в MySQL

Дина Сафина

# Аналитические запросы

1. Разбор домашнего задания
2. Представления
3. Common Table Expressions
4. GROUP_CONCAT
5. Оконные функции
6. Рефлексия
7. Продуктовые метрики

# Представления

```
CREATE
    [OR REPLACE]
    [ALGORITHM = {UNDEFINED | MERGE | TEMPTABLE}]
    [DEFINER = { user | CURRENT_USER }]
    [SQL SECURITY { DEFINER | INVOKER }]
    VIEW view_name [(column_list)]
    AS select_statement
    [WITH [CASCADED | LOCAL] CHECK OPTION]
```

```
CREATE OR REPLACE VIEW db_course.payed_users_vw AS
SELECT u.user_id, u.login, u.reg_dttm
  FROM db_course.users u
 WHERE u.user_id IN (SELECT user_id FROM db_course.payments);
```

# Синтаксис WITH (Common Table Expression)

```
WITH [RECURSIVE]
    cte_name [(col_name [, col_name] ...)] AS (subquery)
    [, cte_name [(col_name [, col_name] ...)] AS (subquery)] ...
```

```
WITH payed_users AS
(
    SELECT u.user_id, u.login, u.reg_dttm
      FROM db_course.users u
     WHERE u.user_id IN (SELECT user_id FROM db_course.payments)
)
SELECT COUNT(1) FROM payed_users
```

# Recursive Common Table Expression

```
WITH RECURSIVE cte AS
(
    SELECT MIN(CAST(payment_dttm AS DATE)) AS dt FROM payments
        UNION ALL
    SELECT dt + INTERVAL 1 DAY
      FROM cte
     WHERE dt + INTERVAL 1 DAY <= (SELECT MAX(CAST(payment_dttm AS DATE)) FROM payments)
)
SELECT cte.dt, COALESCE(SUM(payment_sum), 0) AS money
  FROM cte LEFT JOIN payments ON cte.dt = CAST(payment_dttm AS DATE)
 GROUP BY cte.dt
 ORDER BY cte.dt;
```

| dt | money |
|---|---|
| 2018-08-07 | 902 |
| 2018-08-08 | 0 |
| 2018-08-09 | 0 |
| 2018-08-10 | 317 |
| 2018-08-11 | 270 |
| 2018-08-12 | 330 |
| 2018-08-13 | 615 |
| 2018-08-14 | 1020 |
| 2018-08-15 | 1488 |
| 2018-08-16 | 28 |
| 2018-08-17 | 1463 |
| 2018-08-18 | 839 |
| 2018-08-19 | 763 |
| 2018-08-20 | 508 |

# Recursive Common Table Expression

```
WITH RECURSIVE cte AS
(
    SELECT id, pid, item, CAST(id AS CHAR(200)) AS pth
      FROM rec_items
     WHERE pid IS NULL
        UNION ALL
    SELECT rec.id, rec.pid, rec.item, CONCAT(cte.pth, '-', CAST(rec.id AS CHAR(200))) AS pth
      FROM cte LEFT JOIN rec_items AS rec ON cte.id = rec.pid
     WHERE rec.pid IS NOT NULL
)
SELECT * FROM cte ORDER BY pth;
```

| | id | pid | item |
|---|---|---|---|
| ▶ | 1 | NULL | one |
| | 2 | 1 | two |
| | 3 | 2 | three |
| | 4 | 1 | four |
| | 5 | 1 | five |
| | 6 | 5 | six |

| | id | pid | item | pth |
|---|---|---|---|---|
| ▶ | 1 | NULL | one | 1 |
| | 2 | 1 | two | 1-2 |
| | 3 | 2 | three | 1-2-3 |
| | 4 | 1 | four | 1-4 |
| | 5 | 1 | five | 1-5 |
| | 6 | 5 | six | 1-5-6 |

# GROUP_CONCAT

```
GROUP_CONCAT([DISTINCT] expr [,expr ...]
             [ORDER BY {unsigned_integer | col_name | expr}
                 [ASC | DESC] [,col_name ...]]
             [SEPARATOR str_val])
```

```
SELECT CAST(reg_dttm AS DATE) dt,
       GROUP_CONCAT(login ORDER BY login SEPARATOR ', ') AS logins
  FROM users
 GROUP BY dt;
```

| dt | logins |
|---|---|
| 2018-08-01 | login_22 |
| 2018-08-02 | login_6 |
| 2018-08-03 | login_78 |
| 2018-08-04 | login_43, login_8 |
| 2018-08-06 | login_10, login_39, login_59, login_65 |
| 2018-08-07 | login_1, login_17, login_66 |
| 2018-08-08 | login_48, login_5, login_94 |
| 2018-08-09 | login_33, login_96 |
| 2018-08-10 | login_28, login_3 |
| 2018-08-11 | login_30 |
| 2018-08-12 | login_18, login_45 |
| 2018-08-13 | login_46, login_50 |
| 2018-08-14 | login_31, login_90 |
| 2018-08-15 | login_75, login_95 |

# Оконные функции

- Разбивает выборку на партиции
- Сортирует каждую партицию
- SELECT оперирует окном строк из той же партиции

# Пример оконной функции

```
SELECT user_id,
       payment_sum,
       ROW_NUMBER() OVER (PARTITION BY user_id ORDER BY payment_dttm) AS rn,
       SUM(payment_sum) OVER (PARTITION BY user_id ORDER BY payment_dttm) AS cum_sum,
       MAX(payment_sum) OVER (PARTITION BY user_id ORDER BY payment_dttm) AS cum_max,
       AVG(payment_sum) OVER (PARTITION BY user_id ORDER BY payment_dttm) AS cum_avg,
       payment_dttm
  FROM payments
```

| user_id | payment_sum | rn | cum_sum | cum_max | cum_avg | payment_dttm |
|---|---|---|---|---|---|---|
| 11 | 426 | 1 | 426 | 426 | 426 | 2018-09-08 15:58:02 |
| 11 | 959 | 2 | 1385 | 959 | 692.5 | 2018-09-12 23:38:54 |
| 11 | 60 | 3 | 1445 | 959 | 481.6666666666667 | 2018-09-16 03:03:20 |
| 11 | 351 | 4 | 1796 | 959 | 449 | 2018-09-27 08:17:09 |
| 14 | 968 | 1 | 968 | 968 | 968 | 2018-09-27 06:16:20 |
| 14 | 903 | 2 | 1871 | 968 | 935.5 | 2018-09-29 15:37:23 |
| 14 | 250 | 3 | 2121 | 968 | 707 | 2018-09-30 08:01:40 |
| 16 | 796 | 1 | 796 | 796 | 796 | 2018-09-27 19:16:20 |
| 19 | 232 | 1 | 232 | 232 | 232 | 2018-09-30 06:29:55 |
| 19 | 246 | 2 | 478 | 246 | 239 | 2018-09-30 08:23:25 |
| 19 | 106 | 3 | 584 | 246 | 194.66666666666666 | 2018-09-30 11:42:42 |

# Коллективные окна

```
WINDOW window_name AS (window_spec)
    [, window_name AS (window_spec)] ...

window_spec:
    [window_name] [partition_clause] [order_clause] [frame_clause]
```

```
SELECT user_id,
       payment_sum,
       ROW_NUMBER() OVER w AS rn,
       SUM(payment_sum) OVER w AS cum_sum,
       MAX(payment_sum) OVER w AS cum_max,
       AVG(payment_sum) OVER w AS cum_avg,
       payment_dttm
  FROM payments
WINDOW w AS (PARTITION BY user_id ORDER BY payment_dttm);
```

# Список оконных функций

| Name | Description |
|---|---|
| CUME_DIST() | Cumulative distribution value |
| DENSE_RANK() | Rank of current row within its partition, without gaps |
| FIRST_VALUE() | Value of argument from first row of window frame |
| LAG() | Value of argument from row lagging current row within partition |
| LAST_VALUE() | Value of argument from last row of window frame |
| LEAD() | Value of argument from row leading current row within partition |
| NTH_VALUE() | Value of argument from N-th row of window frame |
| NTILE() | Bucket number of current row within its partition. |
| PERCENT_RANK() | Percentage rank value |
| RANK() | Rank of current row within its partition, with gaps |
| ROW_NUMBER() | Number of current row within its partition |

# RANK, DENSE_RANK, PERCENT_RANK

```
SELECT user_id,
       payment_sum,
       ROW_NUMBER() OVER w AS rn,
       RANK() OVER w AS _rank,
       DENSE_RANK() OVER w AS _dense_rank,
       PERCENT_RANK() OVER w AS _percent_rank,
       CAST(payment_dttm AS DATE) AS dt
  FROM payments
WINDOW w AS (PARTITION BY user_id ORDER BY CAST(payment_dttm AS DATE));
```

| user_id | payment_sum | rn | _rank | _dense_rank | _percent_rank | dt |
|---|---|---|---|---|---|---|
| 20 | 415 | 1 | 1 | 1 | 0 | 2018-09-26 |
| 20 | 750 | 2 | 1 | 1 | 0 | 2018-09-26 |
| 20 | 291 | 3 | 3 | 2 | 0.25 | 2018-09-27 |
| 20 | 943 | 4 | 4 | 3 | 0.375 | 2018-09-28 |
| 20 | 767 | 5 | 4 | 3 | 0.375 | 2018-09-28 |
| 20 | 404 | 6 | 4 | 3 | 0.375 | 2018-09-28 |
| 20 | 277 | 7 | 7 | 4 | 0.75 | 2018-09-29 |
| 20 | 823 | 8 | 8 | 5 | 0.875 | 2018-09-30 |
| 20 | 683 | 9 | 8 | 5 | 0.875 | 2018-09-30 |
| 21 | 364 | 1 | 1 | 1 | 0 | 2018-09-24 |
| 21 | 845 | 2 | 1 | 1 | 0 | 2018-09-24 |
| 21 | 485 | 3 | 3 | 2 | 0.6666666666666666 | 2018-09-29 |
| 21 | 511 | 4 | 4 | 3 | 1 | 2018-09-30 |

# LAG, LEAD, FIRST_VALUE, LAST_VALUE, NTH_VALUE

```
SELECT user_id,
       payment_sum,
       ROW_NUMBER() OVER w AS rn,
       FIRST_VALUE(payment_dttm) OVER w AS fist_pay_dttm,
       LAST_VALUE(payment_dttm) OVER w AS last_pay_dttm,
       LAG(payment_sum) OVER w AS prev_pay,
       LEAD(payment_sum) OVER w AS next_pay,
       NTH_VALUE(payment_sum, 3) OVER w AS third_pay,
       payment_dttm
  FROM payments
WINDOW w AS (PARTITION BY user_id ORDER BY payment_dttm);
```

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

# Фреймы в оконных функциях

```
frame_clause:
    frame_units frame_extent

frame_units:
    {ROWS | RANGE}
frame_extent:
    {frame_start | frame_between}

frame_between:
    BETWEEN frame_start AND frame_end

frame_start, frame_end: {
    CURRENT ROW
  | UNBOUNDED PRECEDING
  | UNBOUNDED FOLLOWING
  | expr PRECEDING
  | expr FOLLOWING
}
```

# Пример указания фрейма

```
SELECT user_id,
       payment_sum,
       ROW_NUMBER() OVER (PARTITION BY user_id ORDER BY payment_dttm) AS rn,
       SUM(payment_sum) OVER (PARTITION BY user_id ORDER BY payment_dttm) AS cum_sum,
       SUM(payment_sum) OVER (PARTITION BY user_id ORDER BY payment_dttm
                              ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) AS unbounded_sum,
       SUM(payment_sum) OVER (PARTITION BY user_id ORDER BY payment_dttm
                              ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING) AS three_pays_sum,
       payment_dttm
  FROM payments
```

| user_id | payment_sum | rn | cum_sum | unbounded_sum | three_pays_sum | payment_dttm |
|---|---|---|---|---|---|---|
| 11 | 426 | 1 | 426 | 426 | 1385 | 2018-09-08 15:58:02 |
| 11 | 959 | 2 | 1385 | 1385 | 1445 | 2018-09-12 23:38:54 |
| 11 | 60 | 3 | 1445 | 1445 | 1370 | 2018-09-16 03:03:20 |
| 11 | 351 | 4 | 1796 | 1796 | 411 | 2018-09-27 08:17:09 |
| 14 | 968 | 1 | 968 | 968 | 1871 | 2018-09-27 06:16:20 |
| 14 | 903 | 2 | 1871 | 1871 | 2121 | 2018-09-29 15:37:23 |
| 14 | 250 | 3 | 2121 | 2121 | 1153 | 2018-09-30 08:01:40 |
| 16 | 796 | 1 | 796 | 796 | 796 | 2018-09-27 19:16:20 |
| 19 | 232 | 1 | 232 | 232 | 478 | 2018-09-30 06:29:55 |
| 19 | 246 | 2 | 478 | 478 | 584 | 2018-09-30 08:23:25 |
| 19 | 106 | 3 | 584 | 584 | 352 | 2018-09-30 11:42:42 |

# Рефлексия

```
SHOW DATABASES;

SHOW TABLES FROM INFORMATION_SCHEMA;

SELECT * FROM INFORMATION_SCHEMA.TABLES;
SELECT * FROM INFORMATION_SCHEMA.COLUMNS;
```

| TABLE_SCHEMA | TABLE_NAME | TABLE_TYPE | ENGINE | VERSION | ROW_FORMAT | TABLE_ROWS | AVG_ROW_LENGTH | DATA_LENGTH | INDEX_LENGTH |
|---|---|---|---|---|---|---|---|---|---|
| db_course | payments | BASE TABLE | InnoDB | 10 | Dynamic | 271 | 60 | 16384 | 16384 |
| db_course | sessions | BASE TABLE | InnoDB | 10 | Dynamic | 3591 | 45 | 163840 | 81920 |
| db_course | users | BASE TABLE | InnoDB | 10 | Dynamic | 100 | 163 | 16384 | 0 |

# Как вы будете оценивать свой проект?

# Продуктовые метрики

DAU MAU CCU

CPI ARPPU

ROI Revenue Retention

Churn Virality Conversion

# DAU/WAU/MAU

**DAU** — Daily Active Users — количество уникальных пользователей, которые зашли в приложение в течение суток

**WAU** — Weekly Active Users — количество уникальных пользователей, которые зашли в приложение в течение недели

**MAU** — Monthly Active Users — количество уникальных пользователей, которые зашли в приложение в течение месяца

# Revenue, Cumulative Revenue

**Revenue** – сумма платежей по проекту за день

**Cumulative Revenue** – кумулятивная сумма платежей

# CCU, PCCU

**CCU** – Concurrent Users – среднее число пользователей, одновременно находящихся в приложении

**PCCU** – Peak Concurrent Users – максимальное количество пользователей, одновременно находящихся в приложении

# PU, PPU

**PU** – Paying Users – количество платящих пользователей

**PPU** – Percentage of Paying Users – доля платящей аудитории относительно DAU

# ARPU

**ARPU** — Average Revenue Per User — средний доход с пользователя

Рассчитывается по формуле:

Выручка приложения / Количество всех пользователей, посетивших приложение за период полученной выручки

# ARPPU

**ARPPU** — Average Revenue Per Paying User — средний доход с одного платящего пользователя

Рассчитывается по формуле:

Выручка приложения / Количество пользователей, совершивших платеж

# Retention X Day

**Retention 7 Day** – процент пользователей, удержанных на седьмой день

Рассчитывается по формуле:

Количество пользователей, которые зашли в игру на седьмой день после регистрации / Количество регистраций за день.

День регистрации считается первым днём

# Rolling Retention

**Rolling Retention 7 Day** – процент пользователей, удержанных начиная с седьмого дня

Рассчитывается по формуле:

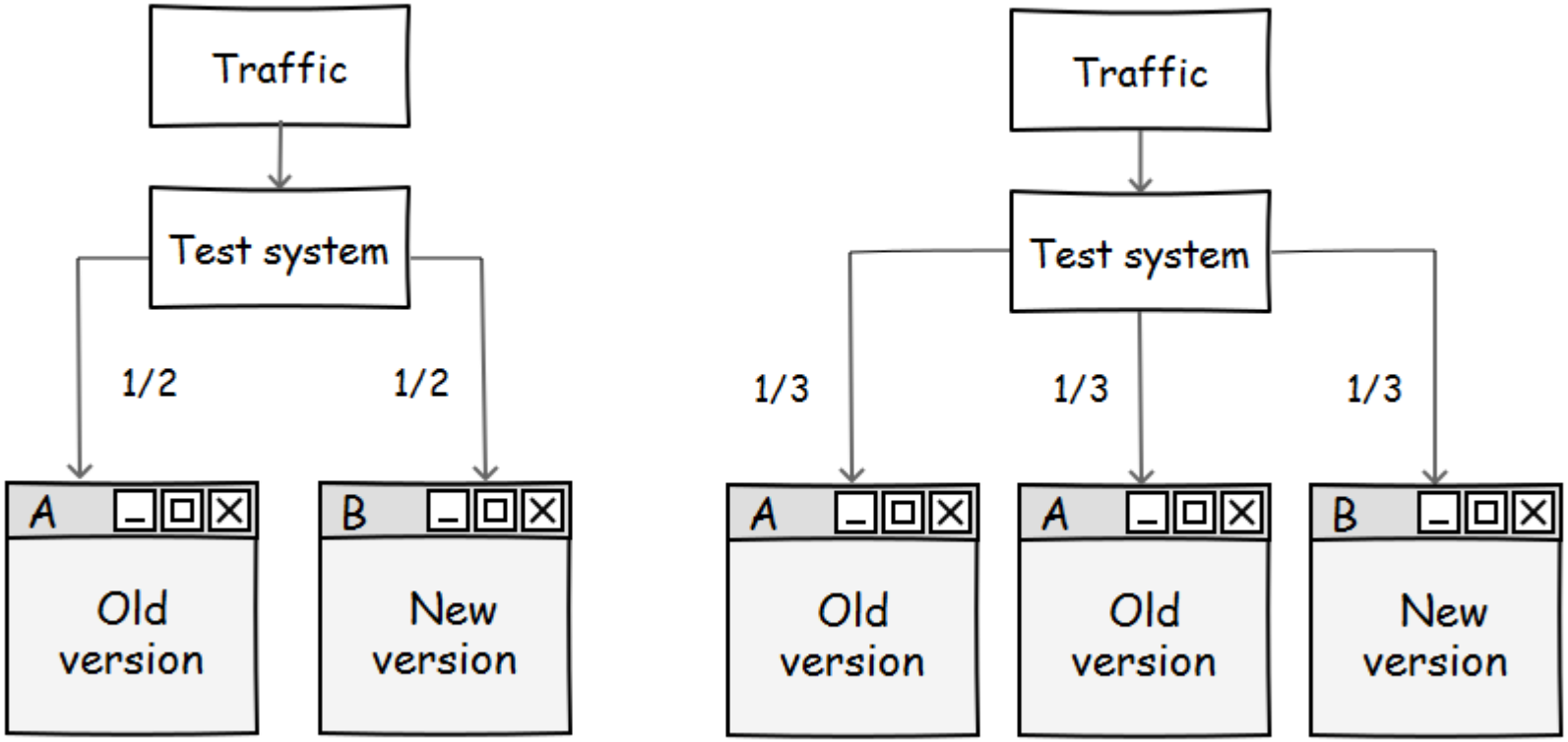
Количество пользователей, которые зашли в игру начиная с седьмого дня после регистрации / Количество регистраций за день.

День регистрации считается первым днём


INTERNATIONAL CARGO
TRANSPORTER
ANTONOV 225
UR-82060


15:01:02
FFFF
EEE
1938
2630
2112
EEE
EEE
EEE
EEE
101
255
-EEEE
219
37E+0
43E+0
0.0
229
1006
203
000
4 0.0
0.0
2.0
5 10
60.0
069

# A/B-тесты, AAB-тесты



# Домашнее задание №1

Посчитать следующие метрики:

- WAU
- PPU
- ARPPU

Выложить на GitHub скрипт и результаты запроса

**Срок сдачи**

*17 октября 2018*