

MORE THAN SQL

Продуктовый анализ данных, Лекция 3, Mirzoian Shahane



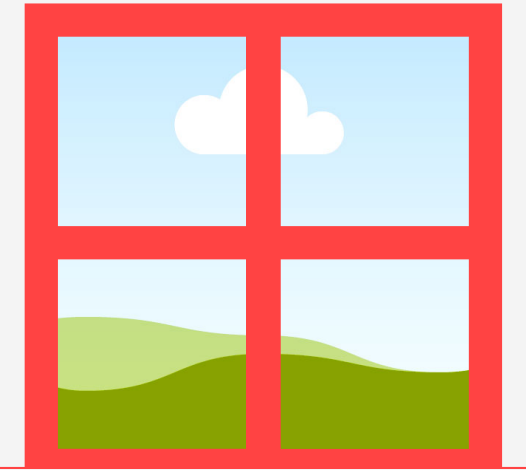
Quick Quiz

ЗАХОДИМ НА
MYQUIZ.RU/

ВВОДИМ КОД **049021**



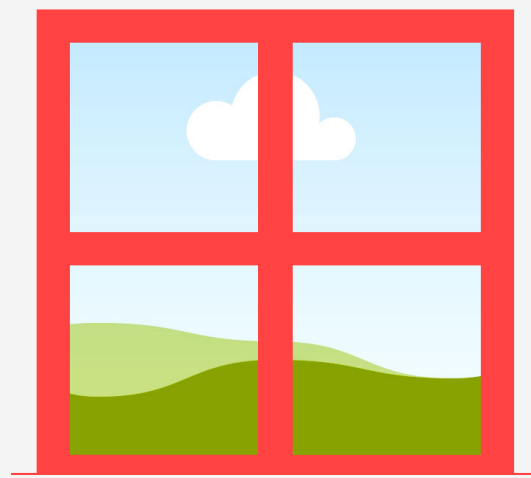
Оконные функции



Рассмотрим таблицу `user_orders`:

User_ID	Order_ID	Order_Date	...	
101	321	2020-01-01		
102	322	2020-01-01		
101	323	2020-01-02		
101	324	2020-01-03		
102	325	2020-01-03		

Оконные функции



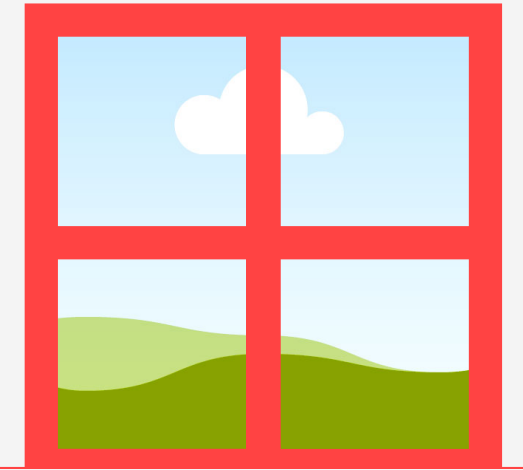
Рассмотрим таблицу `user_orders`:

User_ID	Order_ID	Order_Date	...	
101	321	2020-01-01		
102	322	2020-01-01		
101	323	2020-01-02		
101	324	2020-01-03		
102	325	2020-01-03		



Как достать
второй заказ
пользователя?
А пятый?
А предпоследний?

Оконные функции

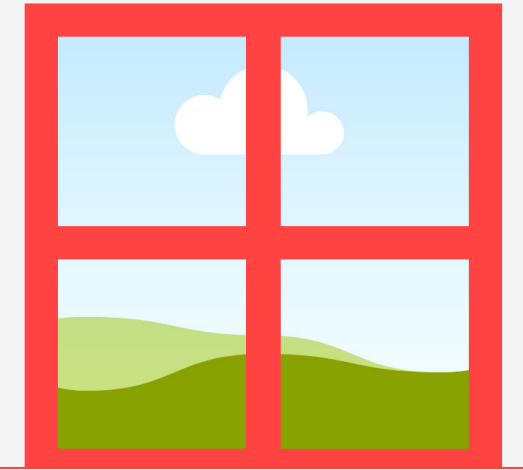


Рассмотрим таблицу `user_orders`:

User_ID	Order_ID	Order_Date	...	
101	321	2020-01-01		
102	322	2020-01-01		
101	323	2020-01-02		
101	324	2020-01-03		
102	325	2020-01-03		

Для каждого
пользователя
пронумеруем
строки
по дате заказа

Оконные функции



Рассмотрим таблицу `user_orders`:

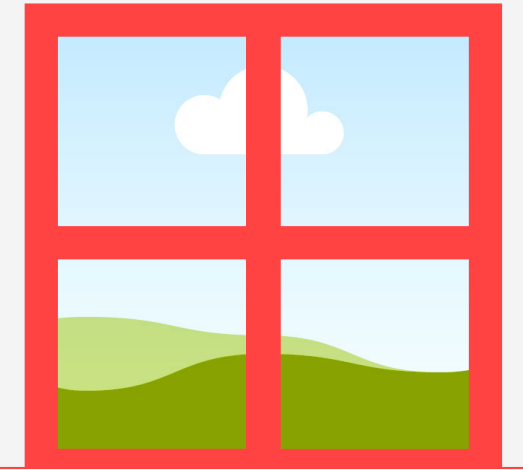
User_ID	Order_ID	Order_Date	...	
101	321	2020-01-01		
102	322	2020-01-01		
101	323	2020-01-02		
101	324	2020-01-03		
102	325	2020-01-03		

Возьмем **ОКНО** по
`user_id`



Для каждого
пользователя
пронумеруем
строки
по дате заказа

Оконные функции



Рассмотрим таблицу `user_orders`:

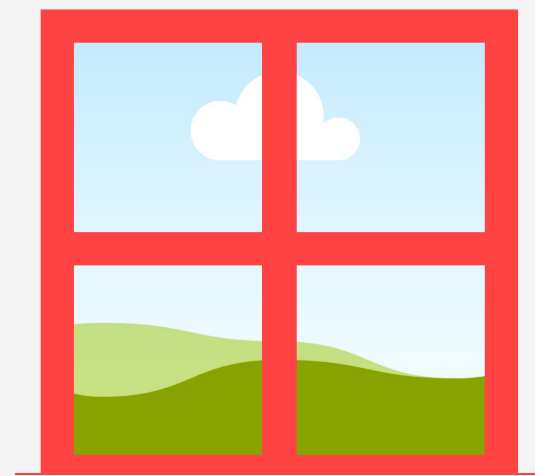
User_ID	Order_ID	Order_Date	...	
101	321	2020-01-01		
102	322	2020-01-01		
101	323	2020-01-02		
101	324	2020-01-03		
102	325	2020-01-03		

Возьмем **окно** по `user_id`

Используем **функцию** `ROW_NUMBER()`

Для каждого пользователя пронумеруем строки по дате заказа

Оконные функции



Рассмотрим таблицу `user_orders`:

User_ID	Order_ID	Order_Date	...	
101	321	2020-01-01		
102	322	2020-01-01		
101	323	2020-01-02		
101	324	2020-01-03		
102	325	2020-01-03		

Возьмем **окно** по
`user_id`

Используем
функцию
`ROW_NUMBER()`

Зададим
порядок
обработки
по `Order_Date`

Для каждого
пользователя
пронумеруем
строки
по дате заказа


```
ROW_NUMBER() OVER(PARTITION BY USER_ID ORDER BY ORDER_DATE) AS USER_ORDER_NUM
```

Используем
функцию
ROW_NUMBER()

Зададим
порядок
обработки
по Order_Date

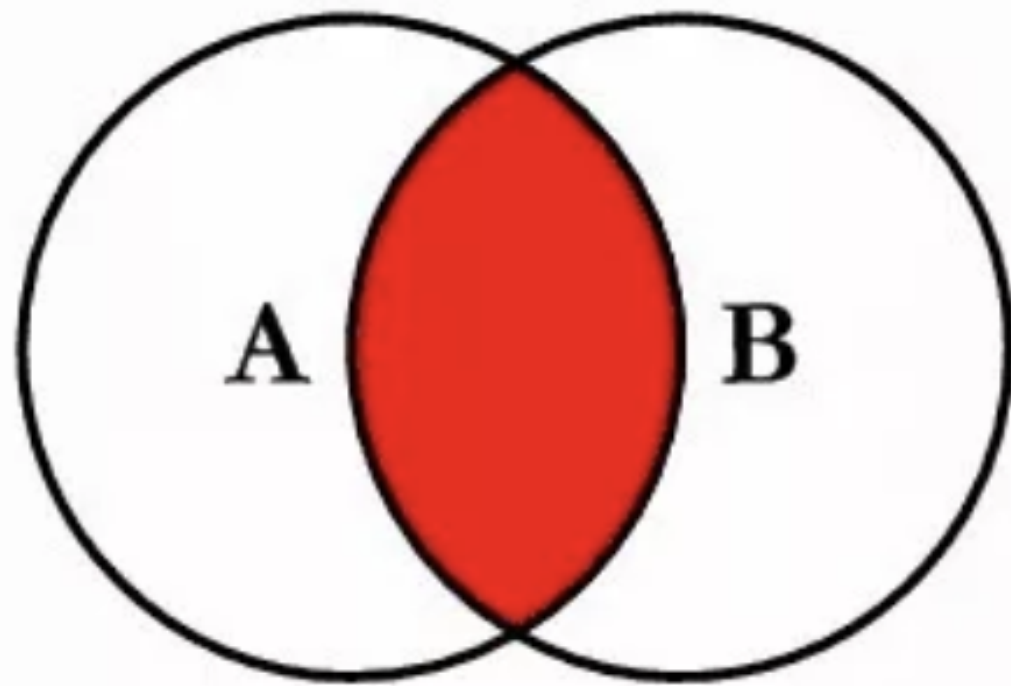


The diagram shows the SQL syntax for the ROW_NUMBER() window function. The text is: ROW_NUMBER() OVER(PARTITION BY USER_ID ORDER BY ORDER_DATE) AS USER_ORDER_NUM. Annotations include: an orange bracket above the first part of the function, a red bracket below the PARTITION BY clause, and a green bracket above the ORDER BY clause.

```
ROW_NUMBER() OVER(PARTITION BY USER_ID ORDER BY ORDER_DATE) AS USER_ORDER_NUM
```

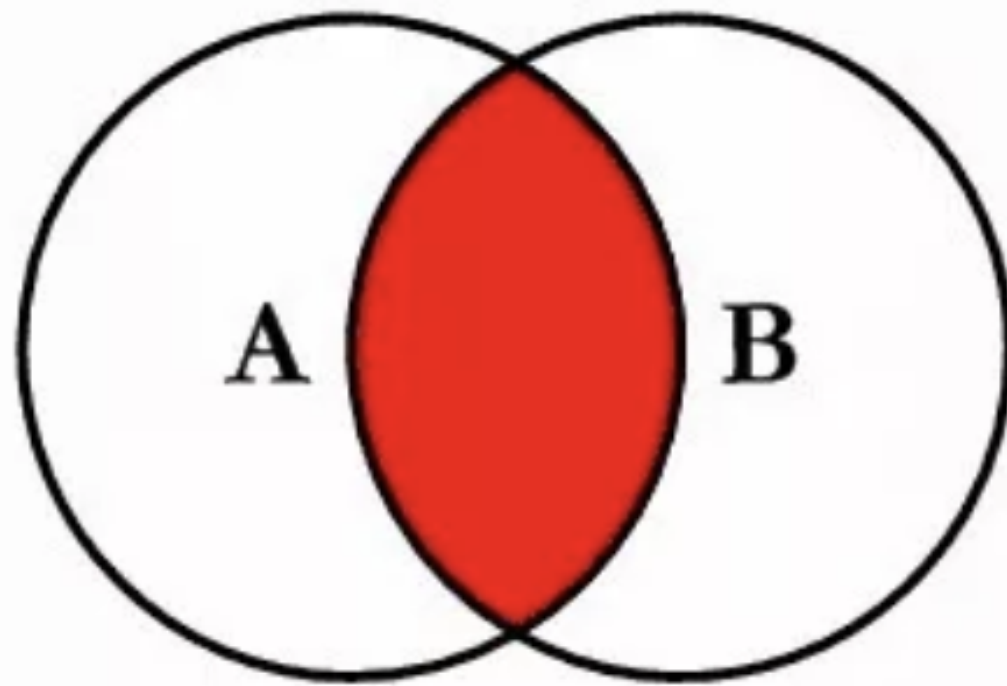
Возьмем **окно** по
user_id

ДАВАЙТЕ ЕЩЕ ОБСУДИМ



```
SELECT <select_list>  
FROM TableA A  
INNER JOIN TableB B  
ON A.Key = B.Key
```

ДАВАЙТЕ ЕЩЕ ОБСУДИМ



```
SELECT <select_list>  
FROM TableA A  
INNER JOIN TableB B  
ON A.Key = B.Key
```

Как визуализация норм,
но inner join - это
НЕ ПЕРЕСЕЧЕНИЕ МНОЖЕСТВ

Пусть у нас есть таблица заказов

User_id	Order_id	Order_Date

и таблица оплат

Payment_ID	User_id	Amount

Хотим посчитать средний чек оплат

Пусть у нас есть таблица заказов

User_id	Order_id	Order_Date

и таблица оплат

Payment_ID	User_id	Amount

Хотим посчитать средний чек оплат

```
with user_payments as
(
  select co.client_id
    , co.order_id
    , p.payment_amount
  from client_orders co
  join payments p on p.user_id = co.user_id
)

select avg(payment_amount)
from user_payments
```

Проверим устойчивость запроса

У нас появился первый покупатель с id 101

client_orders

User_id	Order_id	Order_Date
101	31	2020-02-20

payments

Payment_ID	User_id	Amount
901	101	1000

```
with user_payments as
(
  select co.client_id
    , co.order_id
    , p.payment_amount
  from client_orders co
  join payments p on p.user_id = co.user_id
)

select avg(payment_amount)
from user_payments
```

Как будет выглядеть user_payments

User_id	Order_id	Amount
101	31	1000

Вроде норм :)

И второй с id 102

client_orders

User_id	Order_id	Order_Date
101	31	2020-02-20
102	32	2020-02-21

payments

Payment_ID	User_id	Amount
901	101	1000
902	102	900



```
with user_payments as
(
  select co.client_id
    , co.order_id
    , p.payment_amount
  from client_orders co
  join payments p on p.user_id = co.user_id
)

select avg(payment_amount)
from user_payments
```



Как будет выглядеть user_payments

User_id	Order_id	Amount
101	31	1000
102	32	900

снова норм :)

Наш 101-ый купил еще одну услугу!

client_orders

User_id	Order_id	Order_Date
101	31	2020-02-20
102	32	2020-02-21
101	33	2020-02-22

payments

Payment_ID	User_id	Amount
901	101	1000
902	102	900
903	101	900



```
with user_payments as
(
  select co.client_id
    , co.order_id
    , p.payment_amount
  from client_orders co
  join payments p on p.user_id = co.user_id
)

select avg(payment_amount)
from user_payments
```

Наш 101-ый купил еще одну услугу!

client_orders

User_id	Order_id	Order_Date
101	31	2020-02-20
102	32	2020-02-21
101	33	2020-02-22

payments

Payment_ID	User_id	Amount
901	101	1000
902	102	900
903	101	900

```
with user_payments as
(
  select co.client_id
    , co.order_id
    , p.payment_amount
  from client_orders co
  join payments p on p.user_id = co.user_id
)

select avg(payment_amount)
from user_payments
```

Как будет выглядеть user_payments

User_id	Order_id	Amount
101	31	1000
102	32	900
101	33	1000
101	31	900
101	33	900

Сравним средний чек

933,3
фактический

VS

940
посчитанный

Почему так?

Вспомним, что такое декартово произведение

Пусть есть 2 таблицы t1 и t2:

```
select * from t1;
```

id
1
2
3

```
select * from t2;
```

id
4
5



```
select *  
from t1  
cross join t2;
```

id	id
1	4
1	5
2	4
2	5
3	4
3	5

Их декартово
произведение будет
содержать 6 пар

Почему так?

Конструкция

```
t1 INNER JOIN t2 ON condition
```

— это, можно сказать, всего лишь синтаксический сахар к

```
t1 CROSS JOIN t2 WHERE condition
```

**То есть наш запрос
сработал как-то так:**

Шаг 1. "Перемножить"
таблицы

колонки из users_orders		колонки из payments	
User_id	Order_id	User_id	Amount
101	31	101	1000
101	31	102	900
101	31	101	900
102	32	101	1000
102	32	102	900
102	32	101	900
101	33	101	1000
101	33	102	900
101	33	101	900

То есть наш запрос
сработал как-то так:

Шаг 1. "Перемножить"
таблицы

колонки из users_orders		колонки из payments	
User_id	Order_id	User_id	Amount
101	31	101	1000
101	31	102	900
101	31	101	900
102	32	101	1000
102	32	102	900
102	32	101	900
101	33	101	1000
101	33	102	900
101	33	101	900

То есть наш запрос
сработал как-то так:

Шаг 1. "Перемножить"
таблицы

колонки из users_orders		колонки из payments	
User_id	Order_id	User_id	Amount
101	31	101	1000
101	31	102	900
101	31	101	900
102	32	101	1000
102	32	102	900
102	32	101	900
101	33	101	1000
101	33	102	900
101	33	101	900

**То есть наш запрос
сработал как-то так:**

Шаг 1. "Перемножить"
таблицы

Шаг 2. Наложить условие
равенства user_id

колонки из users_orders		колонки из payments	
User_id	Order_id	User_id	Amount
101	31	101	1000
101	31	102	900
101	31	101	900
102	32	101	1000
102	32	102	900
102	32	101	900
101	33	101	1000
101	33	102	900
101	33	101	900

**То есть наш запрос
сработал как-то так:**

Шаг 1. "Перемножить"
таблицы

Шаг 2. Наложить условие
равенства user_id

колонки из users_orders		колонки из payments	
User_id	Order_id	User_id	Amount
101	31	101	1000
101	31	102	900
101	31	101	900
102	32	101	1000
102	32	102	900
102	32	101	900
101	33	101	1000
101	33	102	900
101	33	101	900

**То есть наш запрос
сработал как-то так:**

Шаг 1. "Перемножить"
таблицы

Шаг 2. Наложить условие
равенства user_id

User_id	Order_id	Amount
101	31	1000
102	32	900
101	33	1000
101	31	900
101	33	900



Небольшой **disclaimer**: хотя inner join логически эквивалентен cross join с фильтром, это не значит, что база будет делать именно так, в тупую: генерить все комбинации и фильтровать.
На самом деле там умные алгоритмы

Пусть у нас есть таблица заказов

User_id	Order_id	Order_Date

и таблица оплат


Payment_ID	User_id	Amount

Хотим посчитать средний чек оплат

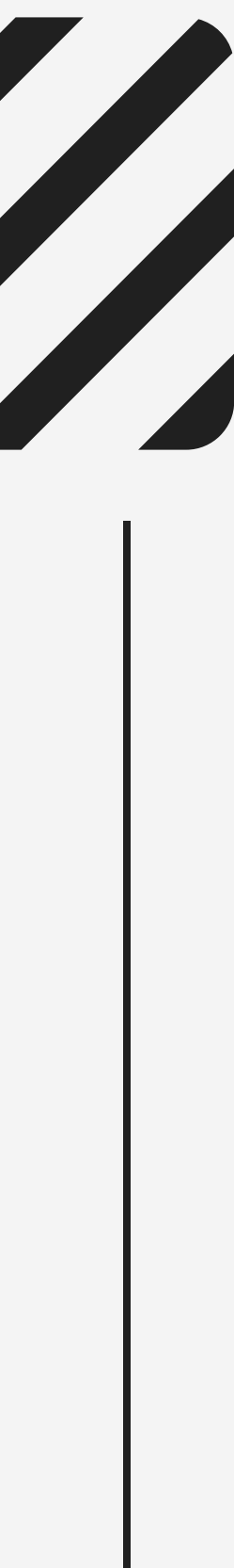
```
with user_payments as
(
  select co.client_id
    , co.order_id
    , p.payment_amount
  from client_orders co
  join payments p on p.user_id = co.user_id
)

select avg(payment_amount)
from user_payments
```


ПРАКТИКА!

- 
- В компании Y произошли изменения: раньше она продавала только подписки на видео. Теперь она продает новый тип услуги - подписку на музыку. Всем клиентам, заинтересовавшимся подпиской на видео, показывается баннер с предложением приобрести подписку на музыку.

ПРАКТИКА!

- 
- В компании Y произошли изменения: раньше она продавала только подписки на видео. Теперь она продает новый тип услуги - подписку на музыку. Всем клиентам, заинтересовавшимся подпиской на видео, показывается баннер с предложением приобрести подписку на музыку.
 - **Проблема 1.** Внутренняя инфраструктура была устроена так, что у одного клиента могла быть только одна услуга, то есть `clients` <> `orders` имели отношение 1-к-1. Во время тестирования нового типа услуги одному человеку создавали второго "фейкового" пользователя и за фейком закрепляли вторую услугу. То есть теперь **разные `client_id` не всегда обозначают разных людей.** Позже появилась возможность одному `client_id` привязать несколько `order_id`, но **дедубликацию фейковых клиентов не провели**

ПРАКТИКА!

- 
- В компании Y произошли изменения: раньше она продавала только подписки на видео. Теперь она продает новый тип услуги - подписку на музыку. Всем клиентам, заинтересовавшимся подпиской на видео, показывается баннер с предложением приобрести подписку на музыку.
 - **Проблема 1.** Внутренняя инфраструктура была устроена так, что у одного клиента могла быть только одна услуга, то есть `clients` <> `orders` имели отношение 1-к-1. Во время тестирования нового типа услуги одному человеку создавали второго "фейкового" пользователя и за фейком закрепляли вторую услугу. То есть теперь **разные `client_id` не всегда обозначают разных людей**. Позже появилась возможность одному `client_id` привязать несколько `order_id`, но **дедубликацию фейковых клиентов не провели**
 - **Проблема 2.** Из-за того, что изначально не предполагалось, что у одного клиента может быть более одной услуги, **система биллинга атрибутирует оплаты не по `order_id`, а по `client_id`** и маппинг по `order_id` технически не реализовать ближайшие полгода

ПРАКТИКА!

- В компании Y произошли изменения: раньше она продавала только подписки на видео. Теперь она продает новый тип услуги - подписку на музыку. Всем клиентам, заинтересовавшимся подпиской на видео, показывается баннер с предложением приобрести подписку на музыку.
- **Проблема 1.** Внутренняя инфраструктура была устроена так, что у одного клиента могла быть только одна услуга, то есть `clients` <> `orders` имели отношение 1-к-1. Во время тестирования нового типа услуги одному человеку создавали второго "фейкового" пользователя и за фейком закрепляли вторую услугу. То есть теперь **разные `client_id` не всегда обозначают разных людей**. Позже появилась возможность одному `client_id` привязать несколько `order_id`, но **дедубликацию фейковых клиентов не провели**
- **Проблема 2.** Из-за того, что изначально не предполагалось, что у одного клиента может быть более одной услуги, **система биллинга атрибутирует оплаты не по `order_id`, а по `client_id`** и маппинг по `order_id` технически не реализовать ближайшие полгода
- **Задача.** Собрать источник данных, с помощью которого можно анализировать кросс - продажи подписок

АНАЛИЗИРОВАТЬ КРОСС - ПРОДАЖИ ПОДПИСОК?!?

Да, тут может быть много вопросов :)

например

- насколько **эффективно** мы кросс-продаем? какой доле заинтересовавшихся в видео показали баннер с музыкой? какая конверсия из интереса подписки на видео в покупку подписки на музыку?
- насколько **целесообразно** мы кросс-продаем? нет ли такого, что клиенты "перетекают" в покупку более дешевой подписки на музыку и не покупают подписку на видео? покрываются ли такие альтернативные издержки приростом прибыли от покупок второй подписки?
- **чем отличаются** те, кто купил обе подписки, от тех, кто не купил ни одну? как мы можем из второй группы переводить пользователей в первую?
- **и много чего еще интересного**

ПРАКТИКА!

У нас есть таблицы

→ **clients**

→ **order_clients**

→ **payments**

ПРАКТИКА!

У нас есть таблицы

→ **clients**

→ **order_clients**

→ **payments**

client_id	age	geo	phone_hash		email_hash	marketing_channel
1040000	40	regions	-3219412515183298902		483045392	yandex.direct
1040001	42	spb	-422571902	8995755028835016102		google.adwors
1040002	26	spb	-6754494892527063670		419984617	google.adwors
1040003	42	regions	-489428050	6184823980854013891		google.adwors
1040004	20	msk	-769776769862558279		873195863	yandex.direct
1040005	48	othermsk	-515640226	261309823697538447		google.adwors
1040006	39	regions	1246856618720949286		715524105	google.adwors
1040007	36	regions	342454482	8797905628439951769		refferal
1040008	14	regions	-3459664961388208824		-73215476	organic
1040009	40	regions	649477292	981083197856818449		yandex.direct
1040010	44	msk	788438378430270928		-389142293	google.adwors
1040011	47	spb	549423304	6845079736020173393		other
1040012	37	regions	-2101835666144038413		611005141	organic
1040013	33	msk	-188326377	-4024106529985292705		google.adwors
1040014	44	spb	6881026658256416829		214796227	yandex.direct
1040015	19	spb	-3315171	-2747117490976367814		google.adwors
1040016	23	spb	-3970209453510529880		904772578	google.adwors
1040017	47	othermsk	268759530	7414413959530520989		google.adwors
1040018	42	msk	8426363678822508525		-147207700	google.adwors
1040019	15	othermsk	-160981425	-5817866263540781051		organic

ПРАКТИКА!

У нас есть таблицы

→ **clients**

→ **order_clients**

→ **payments**

order_id	client_id	order_type	order_date
13000001	4000001	music	2019-04-27 11:01
13000002	4000002	music	2019-05-28 12:02
13000003	4000003	music	2019-06-29 13:03
13000004	4000004	music	2019-07-01 14:04
13000005	4000005	music	2019-08-02 15:05
13000006	4000006	video	2019-09-03 16:06
13000007	4000007	music	2019-10-04 17:07
13000008	4000008	music	2019-11-05 18:08
13000009	4000009	video	2019-01-06 19:09
13000010	4000010	video	2019-02-07 20:10
13000011	4000011	music	2019-03-08 21:11

ПРАКТИКА!

У нас есть таблицы

→ **clients**

→ **order_clients**

→ **payments**

payment_id	product_type	amount	client_id
450000	video-premium	883	4002543
450002	music-light	539	4007077
450004	video-light	479	1040261
450006	video-premium	488	4004816
450008	music-light	488	4009943
450010	video-premium	428	4003438
450012	music-light	396	4006215
450014	music-light	728	4000560
450016	video-light	402	4005366

ПРАКТИКА!

- **Задача.** Собрать источник данных, с помощью которого можно анализировать кросс - продажи подписок

Создадим источник, в котором 1 строка будет отвечать за одного пользователя. Структура подойдет такая:

initial_client_id	second_client_id
initial_client_age	second_client_age
initial_client_geo	second_client_geo
initial_client_marketing_channel	second_client_marketing_channel
initial_order_id	second_order_id
initial_order_datetime	second_order_datetime
initial_order_type	second_order_type
initial_payment_id	second_payment_id
initial_payment_amount	second_payment_amount

А СЕЙЧАС

- Заходим на redash.io
- Находим в чате параметры для подключения

ДОМАШКА

→ **Вспомнить тер. вер**

понимаете, что такое

- среднее, медиана, дисперсия, квантиль
- случайная величина, вероятность, закон распределения, график плотности вероятности

→ **Поставить Anaconda**

или что угодно, где вы откроете jupyter notebook

→ **Закончить pythontutor**

можно к 2.03

→ **additional reading**

<https://habr.com/ru/post/331060/>