Основы SQL

Занятие 1.3

Алексей Кузьмин
Директор разработки,
Data Scientist ДомКлик.ру

Алексей Кузьмин

Директор разработки,
Data Scientist ДомКлик.ру

О чём поговорим и что сделаем

1. Присоединение таблиц
2. Агрегация данных
3. Группировка данных
4. Вложенные запросы
5. Условия с использованием оператора CASE

Соединения JOIN

1

Алексей Кузьмин
Директор разработки,
Data Scientist ДомКлик.ру

Соединения JOIN

Нередко возникает ситуация, когда нам надо получить данные из нескольких таблиц одновременно. Для этого используется специальный класс операций — соединения **JOIN**.

Рассмотрим две корзины с товарами:

basket_a:

id	fruit
1	‘Apple’
2	‘Orange’
3	‘Banana’
4	‘Cucumber’

basket_b:

id	fruit
1	‘Orange’
2	‘Apple’
3	‘Watermelon’
4	‘Pear’

PostgreSQL INNER JOIN

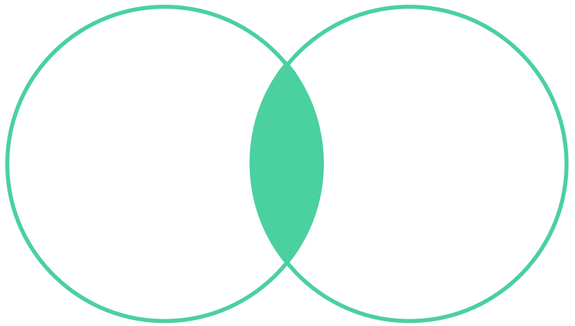
Рассмотрим запрос, соединяющий данные из первой и второй таблицы, используя значения колонки fruit:

```
SELECT
    a.id id_a,
    a.fruit fruit_a,
    b.id id_b,
    b.fruit fruit_b
FROM
    basket_a a
INNER JOIN basket_b b ON a.fruit = b.fruit;
```

Легко увидеть, что **inner join** возвращает данные по строкам, содержащим одинаковые значения.

id_a	fruit_a	id_b	fruit_b
1	‘Apple’	2	‘Apple’
2	‘Orange’	1	‘Orange’

Если смотреть на таблицы как на множества строк, то результат их выполнения можно представить на следующей диаграмме Венна:



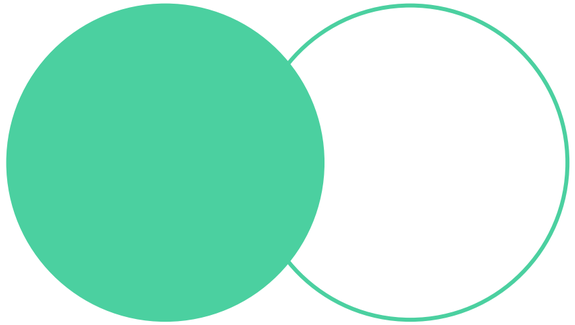
INNER JOIN

PostgreSQL LEFT JOIN

```
SELECT
    a.id id_a,
    a.fruit fruit_a,
    b.id id_b,
    b.fruit fruit_b
FROM
    basket_a a
LEFT JOIN basket_b b ON a.fruit = b.fruit;
```

left join возвращает все данные из первой таблицы.
Если по ним есть совпадения в правой, они обогащаются соответствующими данными, иначе туда записывается специальное значение **NULL**.

id_a	fruit_a	id_b	fruit_b
1	‘Apple’	2	‘Apple’
2	‘Orange’	1	‘Orange’
3	‘Banana’	(null)	(null)
4	‘Cucumber’	(null)	(null)



LEFT OUTER JOIN

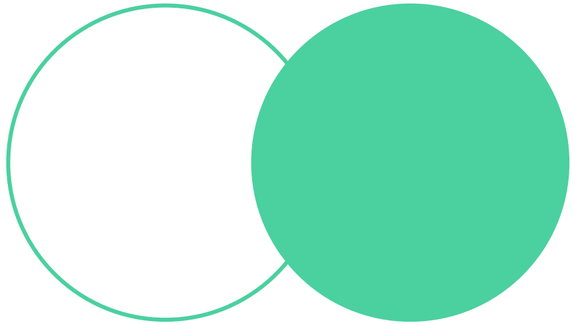
PostgreSQL RIGHT JOIN

Это обратная версия **LEFT JOIN**. Рассмотрим пример:

```
SELECT
    a.id id_a,
    a.fruit fruit_a,
    b.id id_b,
    b.fruit fruit_b
FROM
    basket_a a
RIGHT JOIN basket_b b ON a.fruit = b.fruit;
```

Правое соединение возвращает все данные из правой таблицы.
Если есть совпадения в левой, они обогащают данные.
Иначе — вместо них **NULL**.

id_a	fruit_a	id_b	fruit_b
1	‘Apple’	2	‘Apple’
2	‘Orange’	1	‘Orange’
(null)	(null)	3	‘Watermelon’
(null)	(null)	4	‘Pear’



RIGHT OUTER JOIN

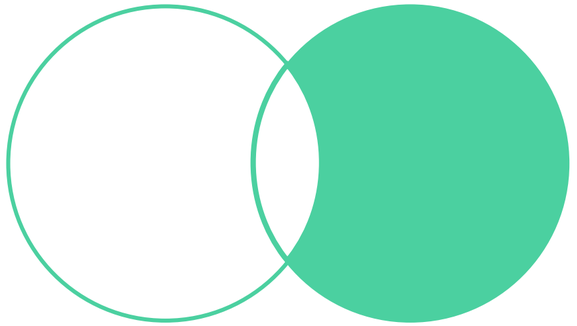
Кстати

Чтобы получить только те строки, которые не содержат данных в левой таблице, можно использовать оператор **WHERE**:

```
SELECT
    a.id id_a,
    a.fruit fruit_a,
    b.id id_b,
    b.fruit fruit_b
FROM
    basket_a a
RIGHT JOIN basket_b b ON a.fruit = b.fruit
WHERE a.id IS NULL;
```

id_a	fruit_a	id_b	fruit_b
(null)	(null)	3	‘Watermelon’
(null)	(null)	4	‘Pear’

Диаграмма Венна для соответствующего запроса:



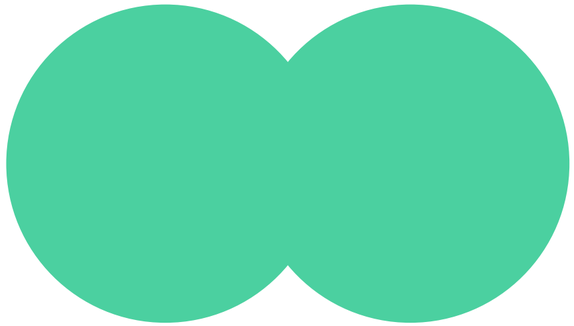
RIGHT OUTER JOIN – только строки из правой таблицы

PostgreSQL FULL OUTER JOIN

Аналогично можно получить сопоставление по всем строкам
в обеих таблицах. Для этого используется оператор **FULL OUTER JOIN**:

```
SELECT
    a.id id_a,
    a.fruit fruit_a,
    b.id id_b,
    b.fruit fruit_b
FROM
    basket_a a
FULL OUTER JOIN basket_b b ON a.fruit = b.fruit;
```

id_a	fruit_a	id_b	fruit_b
1	‘Apple’	2	‘Apple’
2	‘Orange’	1	‘Orange’
3	‘Banana’	(null)	(null)
4	‘Cucumber’	(null)	(null)
(null)	(null)	3	‘Watermelon’
(null)	(null)	4	‘Pear’



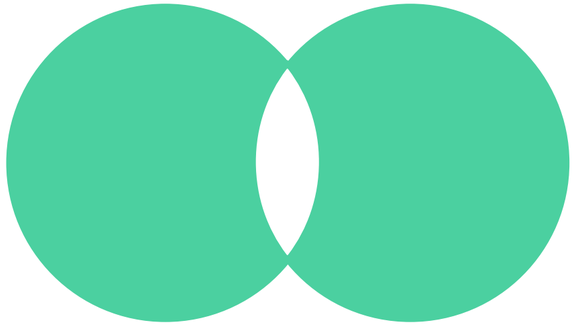
FULL OUTER JOIN

Кстати

Чтобы получить список уникальных строк из обеих таблиц, можно также воспользоваться оператором **WHERE**:

```
SELECT
    a.id id_a,
    a.fruit fruit_a,
    b.id id_b,
    b.fruit fruit_b
FROM
    basket_a a
FULL JOIN basket_b b ON a.fruit = b.fruit
WHERE a.id IS NULL OR b.id IS NULL;
```

id_a	fruit_a	id_b	fruit_b
3	‘Banana’	(null)	(null)
4	‘Cucumber’	(null)	(null)
(null)	(null)	3	‘Watermelon’
(null)	(null)	4	‘Pear’



FULL OUTER JOIN — только уникальные строки из обеих таблиц

USING

Если в двух отношениях, которые нужно объединить, атрибуты, по которым будет происходить объединение, имеют одинаковое название, можно использовать ключевое слово **USING**:

```
SELECT
    a.id id_a,
    a.fruit fruit_a,
    b.id id_b,
    b.fruit fruit_b
FROM
    basket_a a
JOIN basket_b b USING (fruit);
```

Соединения UNION

Алексей Кузьмин
Директор разработки,
Data Scientist ДомКлик.ру

Соединения UNION

Если при работе с **JOIN** соединение данных происходит «слева» или «справа», то при работе с операторами **UNION** или **EXCEPT** работа происходит «сверху» и «снизу». Давайте возьмём две таблицы из примеров с **JOIN**.

Рассмотрим две корзины с товарами:

basket_a:

id	fruit
1	‘Apple’
2	‘Orange’
3	‘Banana’
4	‘Cucumber’

basket_b:

id	fruit
1	‘Orange’
2	‘Apple’
3	‘Watermelon’
4	‘Pear’

UNION

При объединении данных через оператор **UNION** в результате будет список уникальных значений для двух таблиц.

```
SELECT
    a.fruit fruit_a
FROM
    basket_a a
UNION
SELECT
    b.fruit fruit_b
FROM
    basket_b b;
```

fruit_a
‘Apple’
‘Orange’
‘Banana’
‘Cucumber’
‘Watermelon’
‘Pear’

UNION ALL

При объединении данных через оператор **UNION ALL**
в результате будет список всех значений для двух таблиц:

```
SELECT
    a.fruit fruit_a
FROM
    basket_a a
UNION ALL
SELECT
    b.fruit fruit_b
FROM
    basket_b b;
```

fruit_a
‘Apple’
‘Orange’
‘Banana’
‘Cucumber’
‘Apple’
‘Orange’
‘Watermelon’
‘Pear’

EXCEPT

При вычитании данных через оператор **EXCEPT** из значений, полученных в верхней части запроса, будут вычтены значения, которые совпадут со значениями, полученными в нижней части запроса:

```
SELECT
   a.fruit fruit_a
FROM
   basket_a a
EXCEPT
SELECT
   b.fruit fruit_b
FROM
   basket_b b;
```

fruit_a
‘Banana’
‘Cucumber’

Время практики

Алексей Кузьмин
Директор разработки,
Data Scientist ДомКлик.ру

Практика 1 (dvdrental)

- Выведите список названий всех фильмов и их языков (таблица language)

- Выведите список всех актёров, снимавшихся в фильме Lambs Cincinatti (film_id = 508). Надо использовать **JOIN** два раза и один раз **WHERE**

Практика 1. Решение

```
SELECT f.title, l."name"
FROM film f
INNER JOIN "language" l ON l.language_id = f.language_id
```

```
SELECT f.title, a.last_name
FROM actor a
LEFT JOIN film_actor fa ON fa.actor_id = a.actor_id
LEFT JOIN film f ON f.film_id = fa.film_id
WHERE f.film_id = 508
```

Агрегатные функции

2

Алексей Кузьмин
Директор разработки,
Data Scientist ДомКлик.ру

До сих пор мы занимались простыми выборками из БД.

Для задач аналитики и машинного обучения требуется создавать на основе выборок агрегаты —

данные группируются по ключу, в качестве которого выступает один или несколько атрибутов, и внутри каждой группы вычисляются некоторые статистики.

SUM

Простое суммирование, в качестве аргумента принимает имя колонки.

Примечание: признак должен быть числовой, иначе результаты могут быть странными.

```
SELECT SUM(amount) FROM payment;

Результат:
    sum
  ----------
  61312.04
  (1 row)
```

COUNT

Простой счётчик записей. Если передать модификатор **DISTINCT** — получим только уникальные записи.

```
SELECT
    COUNT(customer_id) as count,
    COUNT(DISTINCT customer_id) as count_distinct,
    COUNT(DISTINCT customer_id)/CAST(COUNT(customer_id) as float)
unique_fraction
FROM payment;
```

Несколько особенностей запроса

- Несколько агрегатов в одной строке
- Использовали alias — дали имя колонке
- Применили деление к результатам запроса — посчитали отношение уникальных customer_id к общему числу записей

Результат:

```
count     | count_distinct |  unique_fraction
-----------+--------------------+-------------------
   14596|                599|             0.0410|
  (1 row)
```

AVG

AVG (average) — вычисление среднего значения

```
SELECT AVG(amount) FROM payment;
Результат:
      avg
-----------------
4.200605645382296 5
(1 row)
```

STRING_AGG

STRING_AGG (string) — агрегация значений строк в одну строку

```
SELECT STRING_AGG(title, ', ')
FROM film
WHERE film_id < 3;
Результат:
string_agg                                |
------------------------------------------|
Academy Dinosaur, Ace Goldfinger|
(1 row)
```

Время практики

Алексей Кузьмин
Директор разработки,
Data Scientist ДомКлик.ру

Практика 2

Подсчитайте количество актёров
в фильме Grosse Wonderful (id – 384)

Практика 2. Решение

```
SELECT COUNT(fa.actor_id)
FROM film f
INNER JOIN film_actor fa ON fa.film_id = f.film_id
WHERE f.film_id = 384
```

Группировка данных

3

Алексей Кузьмин
Директор разработки,
Data Scientist ДомКлик.ру

Помимо расчёта статистик по всей таблице, можно считать значения статистик внутри групп с помощью агрегирующего оператора **GROUP BY**:

Например, можем найти самых активных пользователей — тех, кто совершил больше всего платежей:

```
SELECT
    customer_id,
    COUNT(payment_id) as activity
FROM payment
GROUP BY customer_id
ORDER BY activity
DESC LIMIT 5;
```

Результат:

```
customer_id  | activity
            ------------+----------
                        148|      45|
                       526|      42|
                        144|      40|
                         75|      39|
                       236|      39|
(5 rows)
```

Группировать можно по нескольким полям.

```
SELECT
    customer_id,
    date_trunc('month', payment_date) as dt,
    COUNT(payment_id) as activity
FROM payment
GROUP BY 1,2
ORDER BY activity
DESC LIMIT 5;
```

Результат:

```
customer_id|dt                          |activity|
-----------|----------------------------|--------|
       148|2007-04-01 00:00:00|      22|
       102|2007-04-01 00:00:00|      21|
        64|2007-04-01 00:00:00|      20|
        75|2007-04-01 00:00:00|      20|
       526|2007-04-01 00:00:00|      20|
(5 rows)
```

Фильтрация: HAVING

Алексей Кузьмин
Директор разработки,
Data Scientist ДомКлик.ру

Аналогично **WHERE** оператор **HAVING** позволяет проводить фильтрацию. Разница в том, что фильтруются поля с агрегирующими функциями.

```
SELECT
    customer_id,
    AVG(amount) as avg_amount
FROM payment
GROUP BY customer_id
HAVING AVG(amount) > 5
DESC LIMIT 5;
```

Результат:

```
customer_id|avg_amount                    |
-----------|-----------------------------|
           3|5.4483333333333333   |
          19|5.4900000000000000|
         137|5.0426315789473684    |
         181|5.0809090909090909 |
         187|5.6196296296296296   |


(5 rows)
```

Логический порядок выполнения инструкции SELECT

Алексей Кузьмин
Директор разработки,
Data Scientist ДомКлик.ру

Порядок выполнения алгоритма запроса:

1. FROM
2. ON
3. JOIN
4. WHERE
5. GROUP BY
6. WITH CUBE или WITH ROLLUP
7. HAVING
8. SELECT
9. DISTINCT
10. ORDER BY

Время практики

Алексей Кузьмин
Директор разработки,
Data Scientist ДомКлик.ру

Практика 3

Выведите список фильмов, в которых снималось больше 10 актёров

Практика 3. Решение

```
SELECT f.title, COUNT(fa.actor_id), f.description
FROM film f
INNER JOIN film_actor fa ON fa.film_id = f.film_id
GROUP BY f.film_id
HAVING COUNT(fa.actor_id) > 10
```

Обратите внимание, что группировка идет по film.film_id — первичному ключу отношения film. Перечислять в группировке все атрибуты отношения film, указанные в SELECT, не нужно, так как сработает функциональная зависимость.

Подзапросы

4

Алексей Кузьмин
Директор разработки,
Data Scientist ДомКлик.ру

Способ разделения логики формирования выборки — подзапросы. Подзапрос — это **SELECT**, результаты которого используются в другом **SELECT**.

```
SELECT DISTINCT
    userId
FROM public.ratings
WHERE
    rating < (
            SELECT AVG(rating)
            FROM public.ratings
    )
LIMIT 5;
```

Подзапросы

- Запрос внутри запроса
- Обычно используется в конструкции **WHERE**
- В большинстве случаев подзапрос используется, когда вы можете получить значение с помощью запроса, но не знаете конкретный результат

Подзапросы — какой результат работы?

Таблица

```
select customer_id, sum(amount)
from payment
group by customer_id
```

ment 1

ct customer_id, sum(amount) from paymer

123 customer_id	123 sum
184	80,8
87	137,72
477	106,79
273	130,72
550	151,69
51	123,7
394	77,8
272	65,87
70	75,83
190	102,75

в соединениях

Одномерный массив

```
select customer_id
from rental
group by customer_id
having count(rental_id) > 10
```

al 1

ct customer_id from rental group by custo

123 customer_id
87
184
477
273
550
394
51

в условиях in/exist/any/all

Отдельные значения

```
select count(film_id)
from film
where rating::text ilike '%NC%'
```

ультат 1

ct count(film_id) from film where rating::text il

123 count
210

в условиях с операторами =/>/<

Подзапросы в соединениях

```sql
select customer_id, sum(amount)
from payment
group by customer_id
```

ment 1

ct customer_id, sum(amount) from paymen

123 customer_id	123 sum
184	80,8
87	137,72
477	106,79
273	130,72
550	151,69
51	123,7
394	77,8
272	65,87
70	75,83
190	102,75

```sql
select concat(c.last_name, ' ', c.first_name), t.sum
from customer c
inner join
    (select customer_id, sum(amount)
    from payment
    group by customer_id) as t on t.customer_id = c.customer_id
where t.sum > 100
```

ультат 1

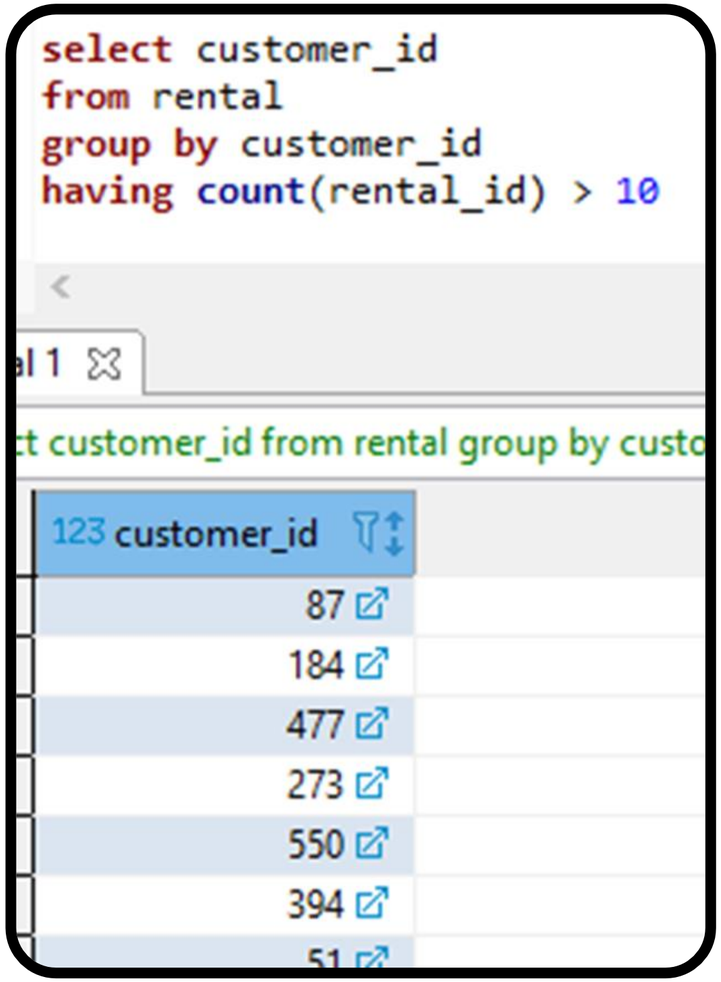
ct concat(c.last_name, ' ', c.first_name), t.sum from customer c inn

Введите SQL вы

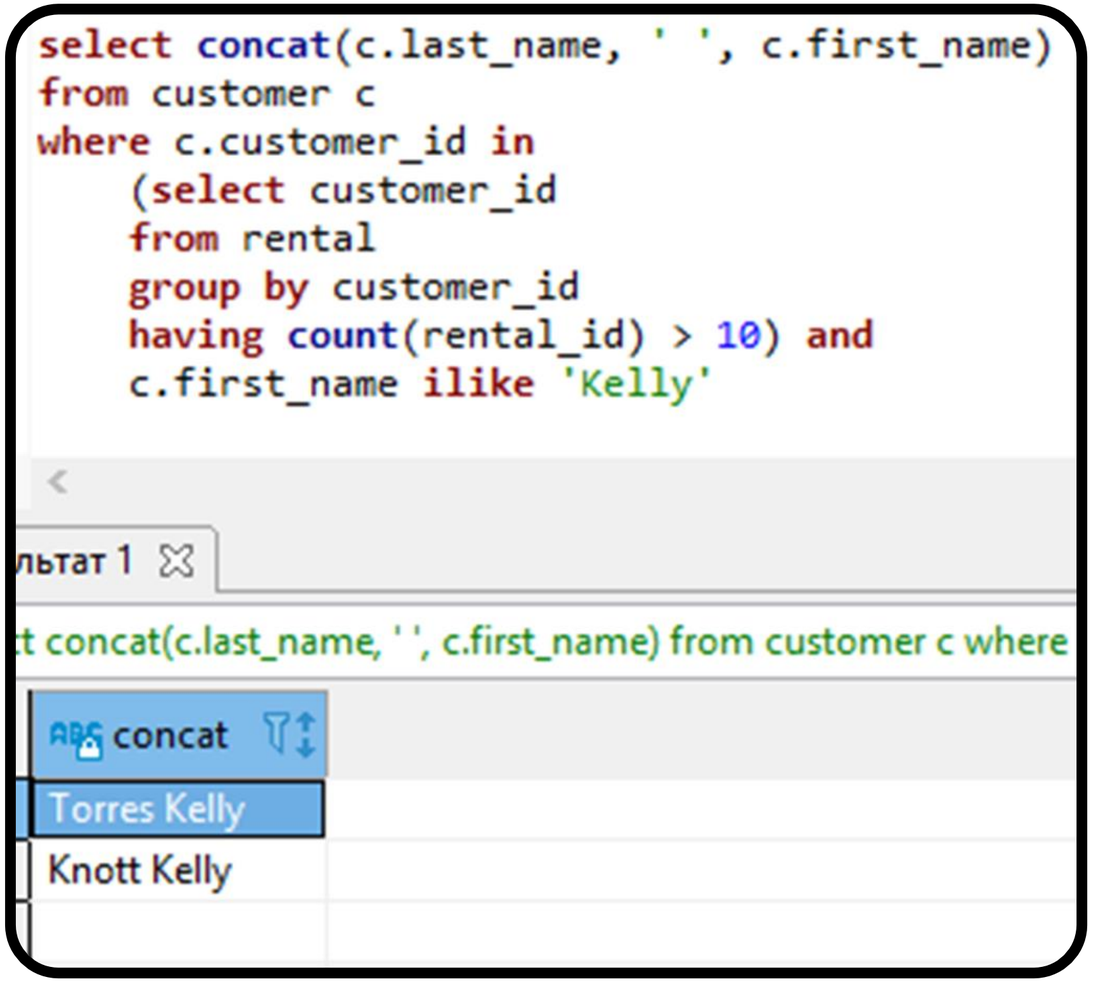
ABC concat	123 sum
Brown Elizabeth	134,65
Miller Maria	130,72
Jackson Karen	131,73
Harris Helen	134,68
Martin Sandra	109,75
Robinson Sharon	103,73
Clark Michelle	146,68
Lewis Sarah	106,73
Walker Deborah	104,73
Hall Jessica	146,68
Allen Shirley	123,7

Подзапросы, возвращающие массивы

```sql
select customer_id
from rental
group by customer_id
having count(rental_id) > 10
```

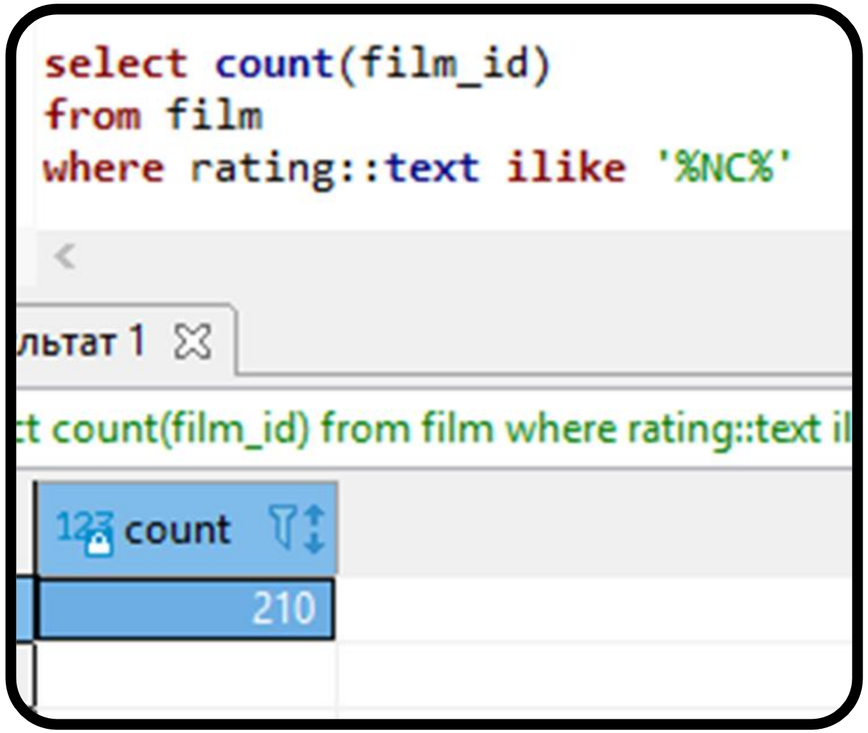



```sql
select concat(c.last_name, ' ', c.first_name)
from customer c
where c.customer_id in
    (select customer_id
    from rental
    group by customer_id
    having count(rental_id) > 10) and
    c.first_name ilike 'Kelly'
```

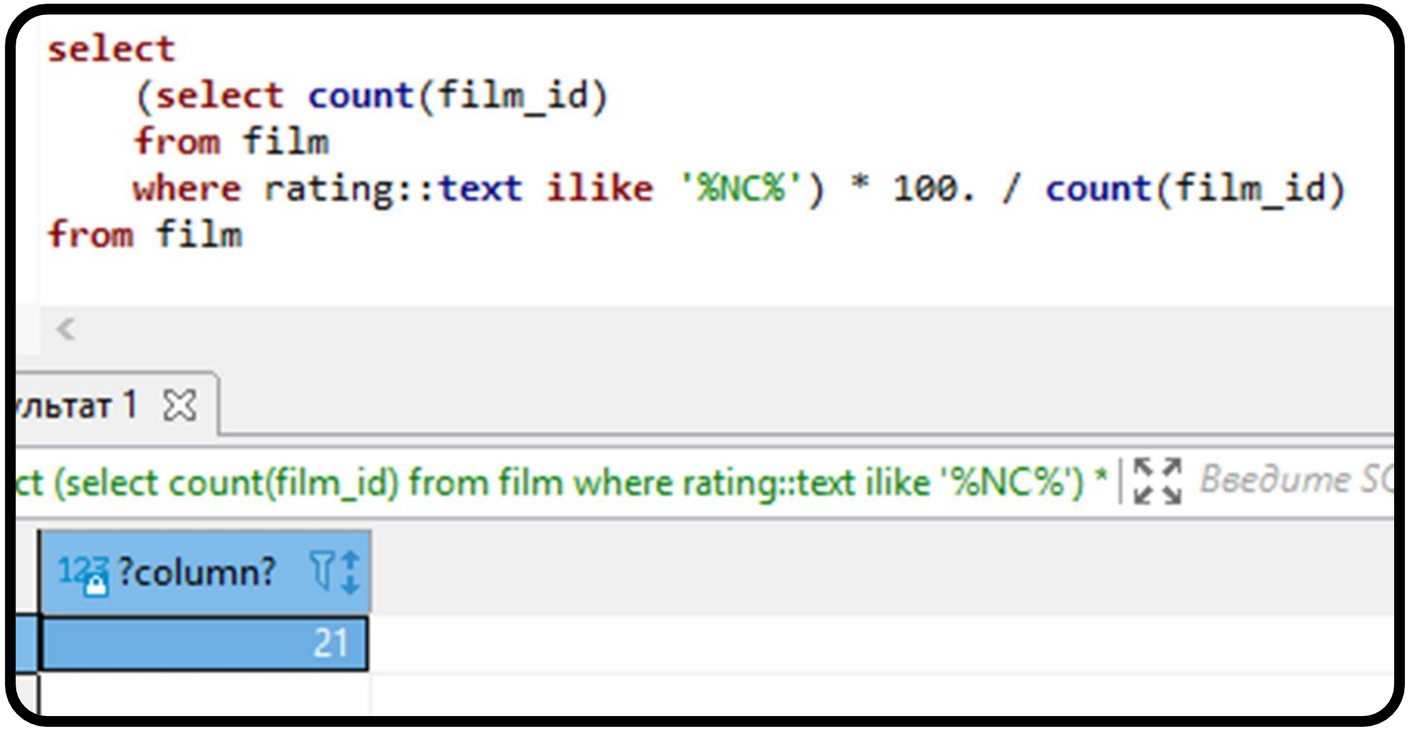


Подзапросы, единственное значение

```sql
select count(film_id)
from film
where rating::text ilike '%NC%'
```




```sql
select
    (select count(film_id)
    from film
    where rating::text ilike '%NC%') * 100. / count(film_id)
from film
```



Оператор CASE

5

Алексей Кузьмин
Директор разработки,
Data Scientist ДомКлик.ру

CASE

Выражение **CASE** в SQL представляет собой общее условное выражение, напоминающее операторы if/else в других языках программирования. Типы данных всех выражений результатов должны приводиться к одному выходному типу.

В запросе мы проверяем, что если пользователь купил более чем на 200 у. е., то он хороший клиент, если менее чем на 200, то не очень хороший, в остальных случаях — «средний».

```
SELECT      customer_id, sum(amount),
        CASE
                WHEN SUM(amount) > 200 THEN 'Good user'
                WHEN SUM(amount) < 200 THEN 'Bad user'
                ELSE 'Avarage user'
        END
FROM payment
GROUP BY customer_id
ORDER BY SUM(amount) DESC
LIMIT 5;
```

Результат:

```
customer_id|sum     |case         |
-----------------|----------|--------------|
           148|211.55  |Good user|
           526|208.58|Good user|
           178|194.61   |Bad user   |
           137|191.62   |Bad user   |
           144|189.60  |Bad user   |


(5 rows)
```

Итоги занятия

Алексей Кузьмин
Директор разработки,
Data Scientist ДомКлик.ру

Что мы сегодня сделали?

- Разобрали и увидели работу соединений — внешнее, внутреннее, левое, правое, декартово
- Увидели работу группировок (Group By, Having)
- Узнали отличия и применения агрегирующих функций и подзапросов

Полезные материалы

Алексей Кузьмин
Директор разработки,
Data Scientist ДомКлик.ру

Полезные материалы

- https://habr.com/ru/post/268983/
- http://www.postgresqltutorial.com/postgresql-aggregate-functions/
- http://www.skillz.ru/dev/php/article-Obyasnenie_SQL_obedinenii_JOIN_INNER_OUTER.html

Спасибо за внимание!

Алексей Кузьмин
Директор разработки,
Data Scientist ДомКлик.ру