

## ЗАНЯТИЕ 1.3

# ОСНОВЫ SQL



## **Алексей Кузьмин**

Директор разработки; Data Scientist

**ДомКлик.ру**



[aleksej.kyzmin@gmail.com](mailto:aleksej.kyzmin@gmail.com)

---

О ЧЁМ ПОГОВОРИМ И ЧТО  
СДЕЛАЕМ

- 
- Напишем простые запросы
  - Посмотрим на сложные запросы
  - Применим соединения и функции
  - Разберемся в хронологии связывания таблиц

—

# Соединения

---

Нередко возникает ситуация, когда нам надо получить данные из нескольких таблиц одновременно. Для этого используется специальный класс операций, называемый соединения (join).

Рассмотрим 2 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**

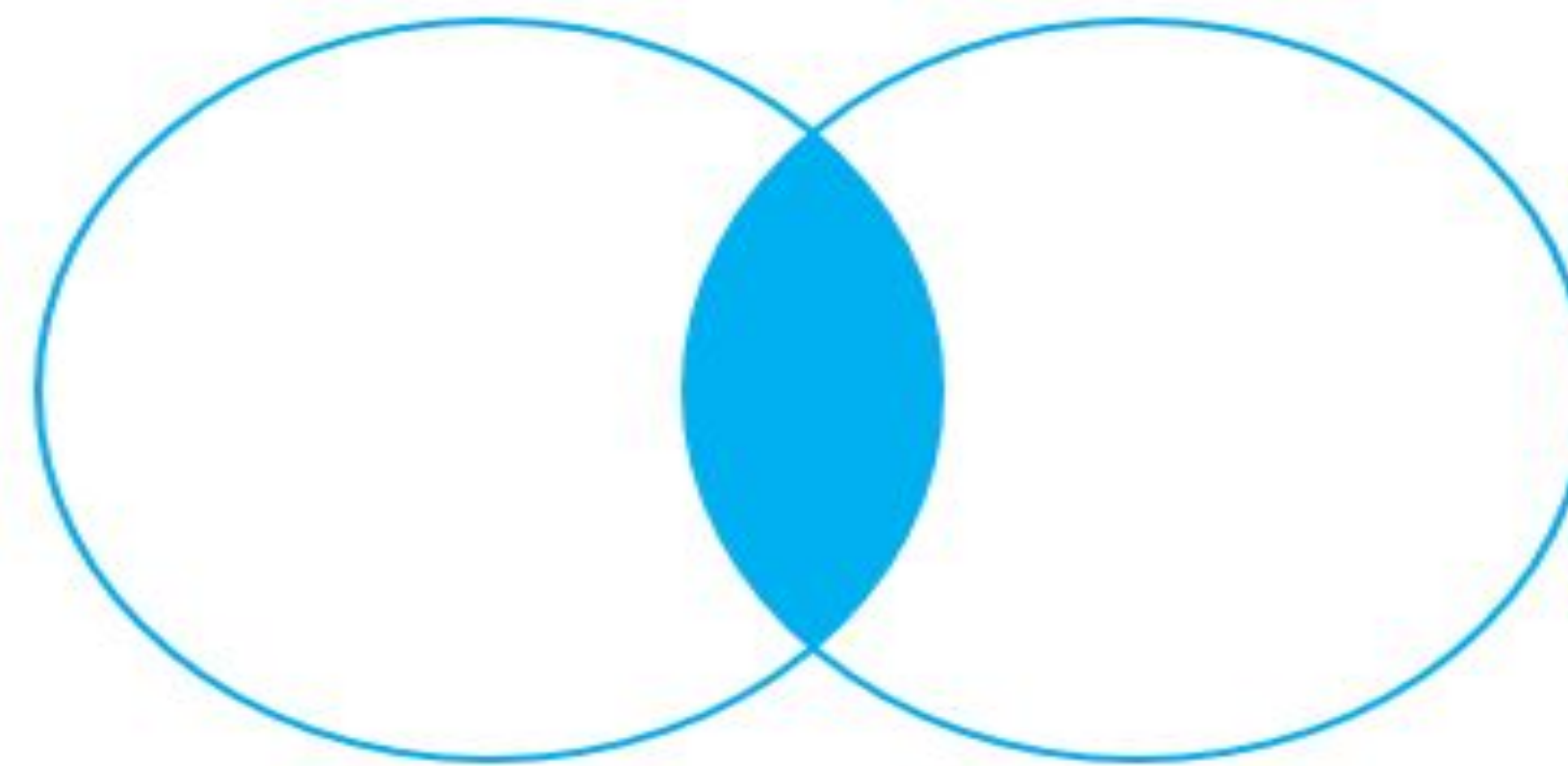
basket\_a a

**INNER JOIN** basket\_b b **ON** a.fruit = b.fruit;

---

id_a	fruit_d	id_a	fruit_d
1	'Apple'	2	'Apple'
2	'Orange'	1	'Orange'

Легко видеть, что `inner join` возвращает данные по строкам, содержащим одинаковые значения. Если смотреть на таблицы как на множества строк, то результат их выполнения можно представить на следующей диаграмме Вена:



INNER JOIN



## PostgreSQL left join

**SELECT**

a.id id\_a,  
a.fruit fruit\_a,  
b.id id\_b,  
b.fruit fruit\_b

**FROM**

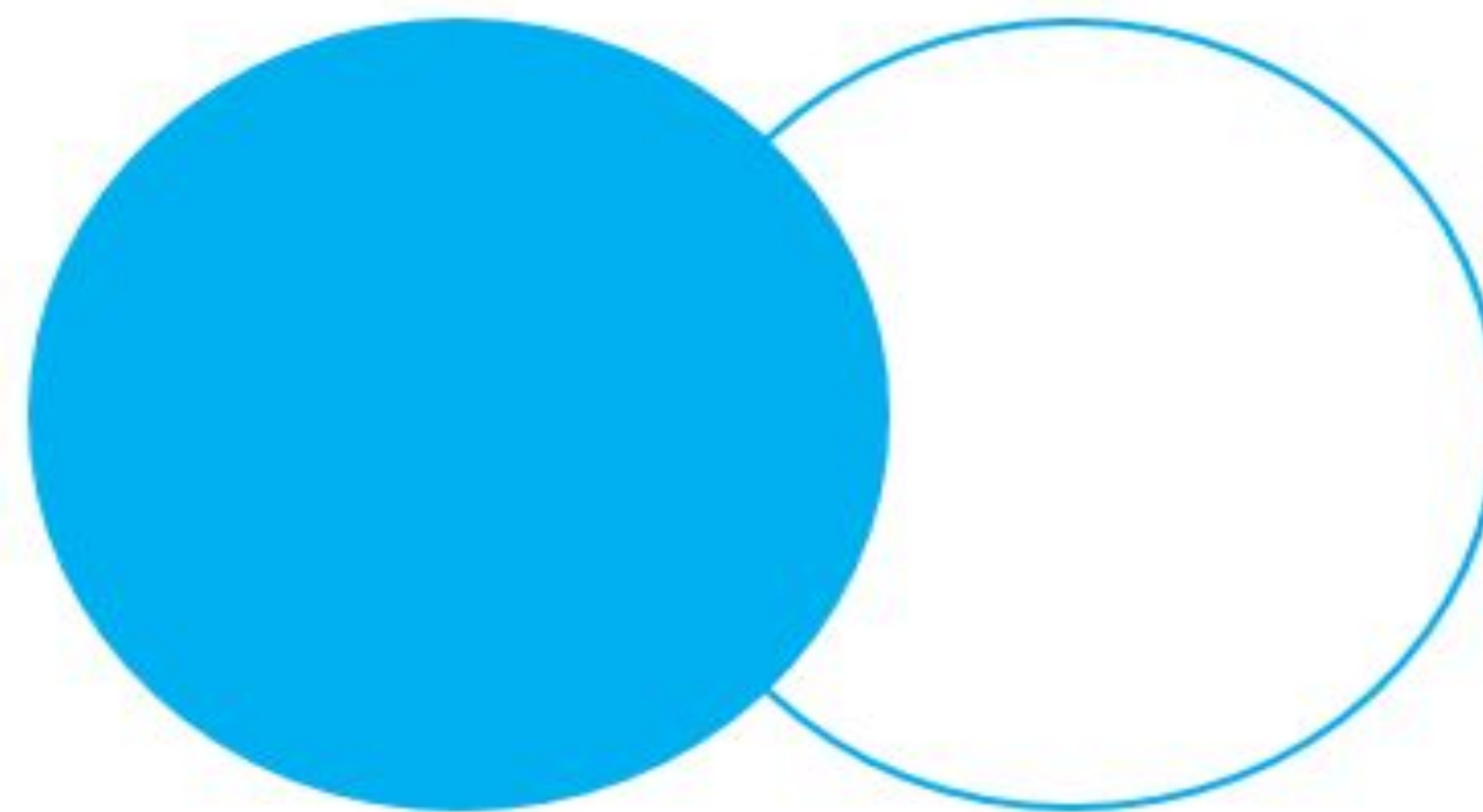
basket\_a a

**LEFT JOIN** basket\_b b **ON** a.fruit = b.fruit;

---

id_a	fruit_d	id_a	fruit_d
1	'Apple'	2	'Apple'
2	'Orange'	1	'Orange'
3	'Banana'	(null)	(null)
4	'Cucumber'	(null)	(null)

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



LEFT OUTER JOIN

## PostgreSQL right join

Это обратная версия left join'a. Рассмотрим пример:

**SELECT**

a.id id\_a,  
a.fruit fruit\_a,  
b.id id\_b,  
b.fruit fruit\_b

**FROM**

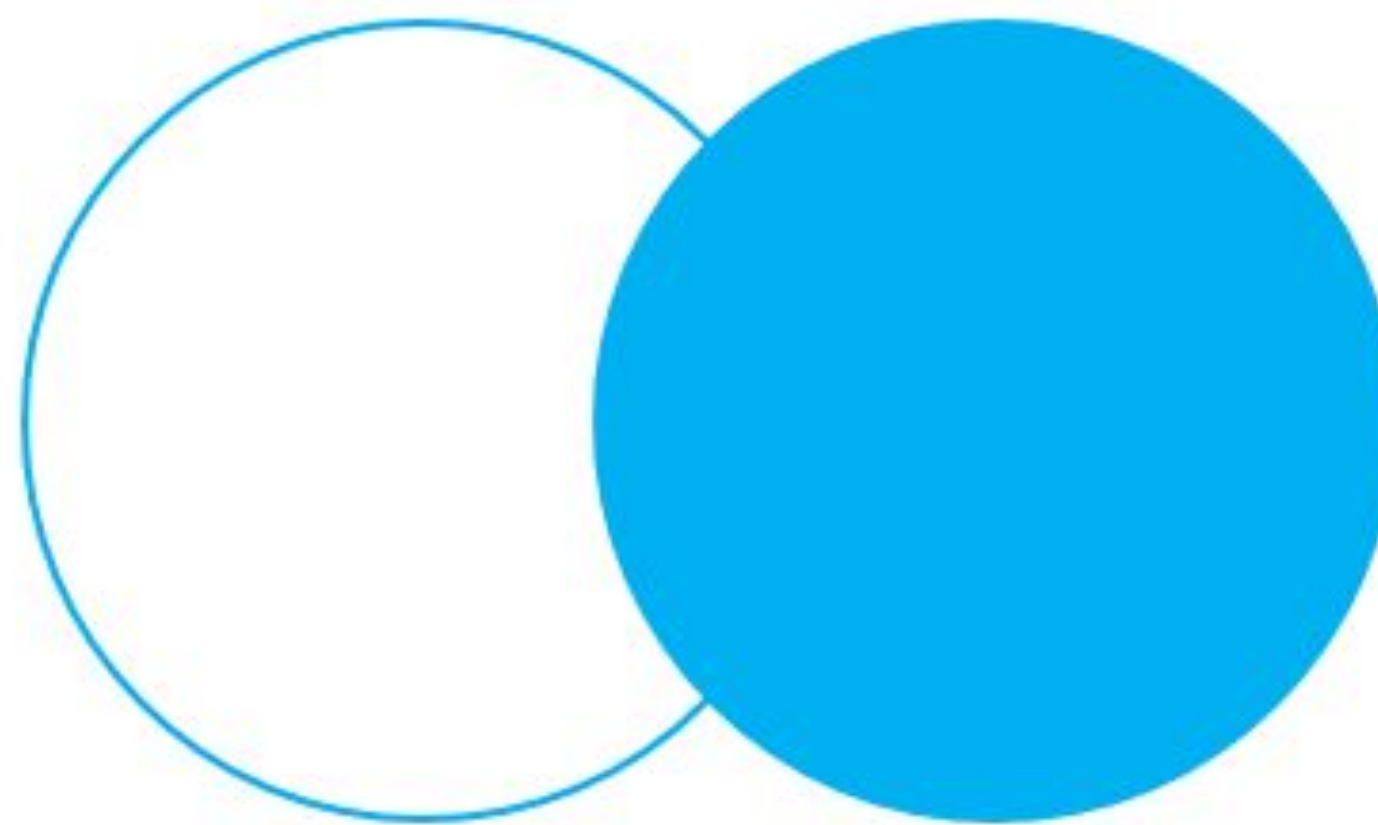
basket\_a a

**RIGHT JOIN** basket\_b b **ON** a.fruit = b.fruit;

---

id_a	fruit_d	id_a	fruit_d
1	'Apple'	2	'Apple'
2	'Orange'	1	'Orange'
(null)	(null)	3	'Watermelon'
(null)	(null)	4	'Pear'

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



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

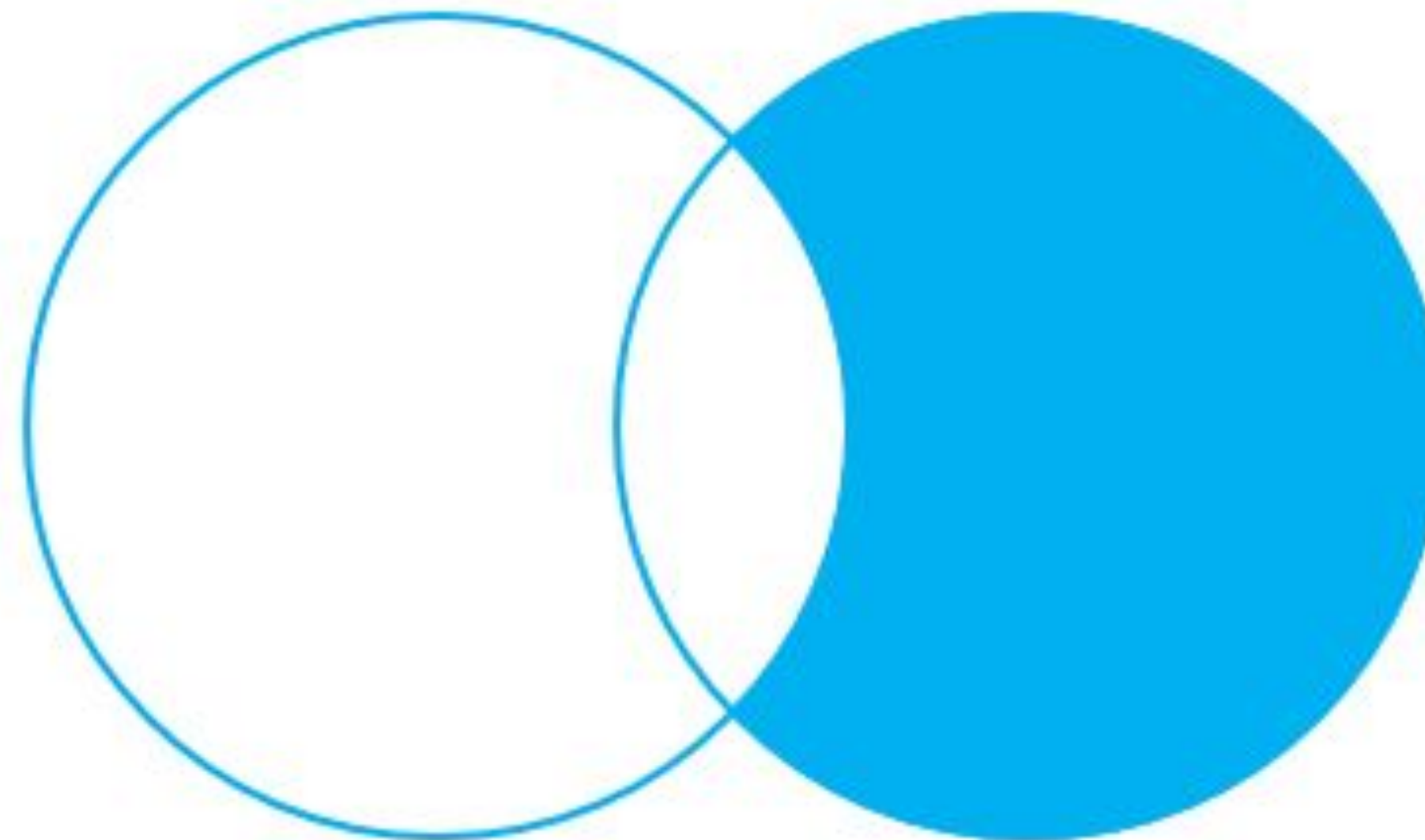
**RIGHT JOIN** basket\_b b **ON** a.fruit = b.fruit

**WHERE** a.id **IS NULL**;

---

id_a	fruit_d	id_a	fruit_d
(null)	(null)	3	'Watermelon'
(null)	(null)	4	'Pear'

Диаграмма Вена для соответствующего запроса:



RIGHT OUTER JOIN – only  
rows from the right table

## PostgreSQL full outer join

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

**SELECT**

a.id id\_a,  
a.fruit fruit\_a,  
b.id id\_b,  
b.fruit fruit\_b

**FROM**

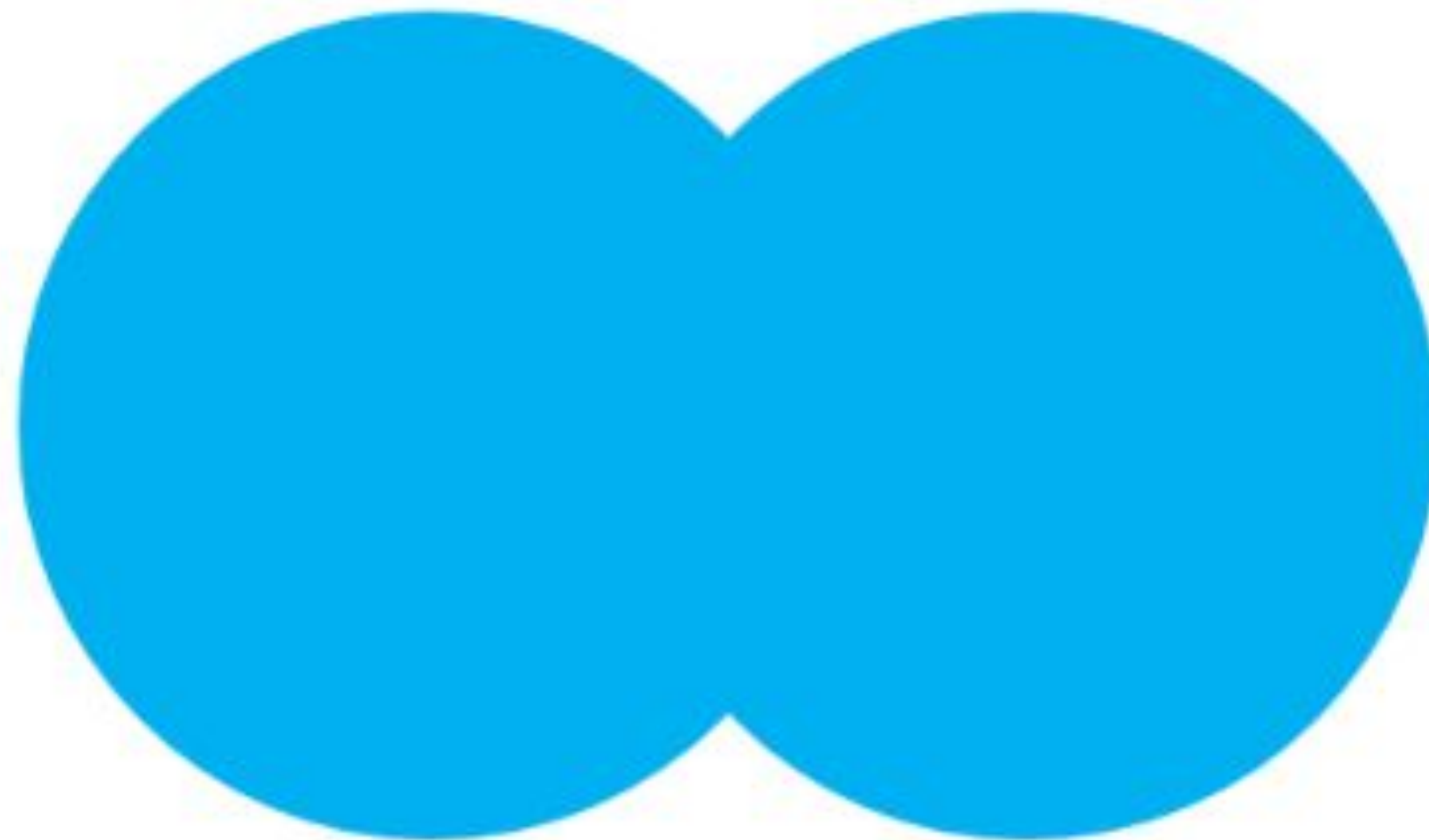
basket\_a a

**FULL OUTER JOIN** basket\_b b **ON** a.fruit = b.fruit;



\_\_\_\_\_

id_a	fruit_d	id_a	fruit_d
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



---

Чтобы получить список уникальных строк из обеих таблиц, можно так же воспользоваться оператором WHERE:

**SELECT**

a.id id\_a,  
a.fruit fruit\_a,  
b.id id\_b,  
b.fruit fruit\_b

**FROM**

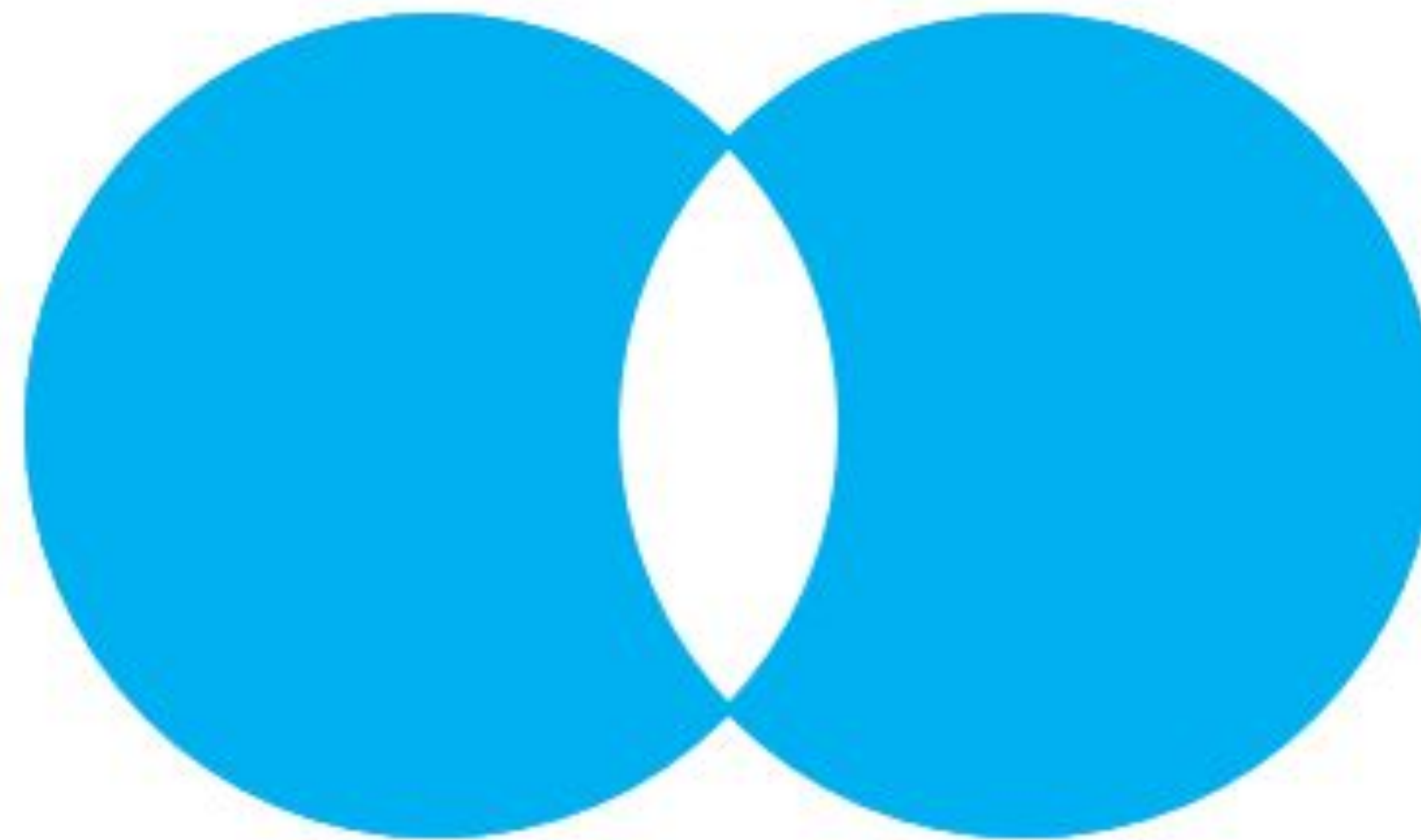
basket\_a a

**FULL JOIN** basket\_b b **ON** a.fruit = b.fruit

**WHERE** a.id **IS NULL OR** b.id **IS NULL**;

---

id_a	fruit_d	id_a	fruit_d
3	'Banana'	(null)	(null)
4	'Cucumber'	(null)	(null)
(null)	(null)	3	'Watermelon'
(null)	(null)	4	'Pear'



FULL OUTER JOIN – only  
rows unique to both tables

—

ВРЕМЯ ПРАКТИКИ

## Практика 1 (dvdrental)

1. Выведите список названий всех фильмов и их языков (таблица language)
2. Выведите список всех актеров, снимавшихся в фильме Lambs Cincinatti (film\_id = 508). Надо использовать 2 join'a и один where

---

# Агрегатные функции

---

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 ключа 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 странные

```
SELECT SUM(rating) FROM public.ratings;
```

Результат:

```
sum
```

```
-----
```

```
91816043
```

```
(1 row)
```

## COUNT

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

```
SELECT
```

```
    COUNT(userId) as count,
```

```
    COUNT(DISTINCT userId) as count_distinct,
```

```
    COUNT(DISTINCT userId)/CAST(COUNT(userId) as float)
```

```
unique_fraction
```

```
FROM public.ratings;
```



Результат:

count | count\_distinct | unique\_fraction

-----+-----+-----

777776 | 7956 | 0.0102291662380943

(1 row)

Несколько особенностей запроса

- несколько агрегатов в одной строке
- использовали alias - дали имя колонке
- применили арифметическую операцию к результатам запроса (деление) - посчитали отношение уникальных userId к общему числу записей.



## AVG

AVG (AVERAGE) - вычисление среднего значения

```
SELECT AVG(rating) from public.ratings;
```

Результат:

avg

-----

3.52809035436088

(1 row)

—

ВРЕМЯ ПРАКТИКИ

---

## Практика 2.

1. Подсчитайте количество актеров в фильме Grosse Wonderful (id - 384)

---

Базовые статистики по  
группам: GROUP BY

---

Кроме 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 поставил больше всего оценок

```
SELECT
  userId,
  COUNT(rating) as activity
FROM public.ratings
GROUP BY userId
ORDER BY activity DESC
LIMIT 5;
```



Результат:

userid | activity

-----+-----

45811 | 18276

8659 | 9279

270123 | 7638

179792 | 7515

228291 | 7410

(5 rows)



Группировать можно по нескольким полям

```
SELECT
  userId,
  to_char(to_timestamp(timestamp), 'YYYY/MM/DD') as dt,
  COUNT(rating) as activity
FROM public.ratings
GROUP BY 1,2
ORDER BY activity
DESC LIMIT 5;
```



---

Результат:

userid	dt	activity
-----+	-----+	-----
270123	2015/07/05	6663
45811	2015/12/15	5085
24025	2016/03/27	4946
101276	2016/05/09	4834
258253	2017/02/10	4227

(5 rows)

---

Фильтрация: HAVING

---

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

```
SELECT  
  userId,  
  AVG(rating) as avg_rating  
FROM public.ratings  
GROUP BY userId  
HAVING AVG(rating) < 3.5  
LIMIT 5;
```



Результат:

userid	avg_rating
5761	3.41922005571031
5468	1.66666666666667
7662	3.26373626373626
4326	3.33783783783784
2466	3.4375

(5 rows)

—

ВРЕМЯ ПРАКТИКИ

## Практика 3

Выведите список фильмов, в которых снималось больше 10 актеров

—

Подзапросы

---

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 (вложенные запросы)**

Не что иное, как запрос внутри запроса.

Обычно, подзапрос используется в конструкции WHERE.

В большинстве случаев, подзапрос используется, когда вы можете получить значение с помощью запроса, но не знаете конкретного результата.

# Подзапросы - каков результат работы?

Таблица    Одномерный массив    Отдельные значения

Query Editor    Query History

1    SELECT id, sum(weight)  
2    FROM public.products  
3    group by id

Data Output    Explain    Messages

	id character varying	sum bigint
1	P2	17
2	P1	12
3	P6	19
4	P4	14
5	P5	12
6	P3	17

в соединениях

postgres on postgres@pg

Query Editor    Query History

1    SELECT id  
2    FROM public.products  
3    group by id  
4    having sum(weight) > 15;

Data Output    Explain    Messages

	id character varying
1	P2
2	P6
3	P3

в условиях in/exist/any/all

Query Editor    Query History

1    SELECT name  
2    FROM public.products  
3    where id = 'P1'

Data Output    Explain    Messages

	name character varying
1	Гайка

в условиях с операторами =/>/<

# Подзапросы в соединениях

Query Editor   Query History

1   SELECT id, sum(weight)

2   FROM public.products

3   group by id

Data Output   Explain   Messages

	id character varying	sum bigint
1	P2	17
2	P1	12
3	P6	19
4	P4	14
5	P5	12
6	P3	17

Query Editor   Query History

1   SELECT \*

2   FROM public.supply inner join

3   (

4   SELECT id, sum(weight)

5   FROM public.products

6   group by id) as pr

7   on pr.id = pid

8   where sid = 'S1'


9

Data Output   Explain   Messages   Notifications

	sid character varying	pid character varying	jid character varying	num integer	id character varying	sum bigint
1	S1	P1	J4	700	P1	12
2	S1	P1	J1	200	P1	12



# Подзапросы, возвращающие массивы

 postgres on postgres@pg

Query Editor

Query History

1

2

3

4

```
SELECT id
FROM public.products
group by id
having sum(weight) > 15;
```

Data Output

Explain

Messages

	id	
	character varying	
1	P2	
2	P6	
3	P3	

1

2

3

4

5

6

7

8

9

```
SELECT *
FROM public.supply
where pid in (
SELECT id
FROM public.products
group by id
having sum(weight) > 15
) and sid in( 'S2', 'S3', 'S4')
```

Data Output

Explain

Messages

Notifications

	sid	pid	jid	num
	character varying	character varying	character varying	integer
1	S4	P6	J7	300
2	S4	P6	J3	300
3	S3	P3	J1	200
4	S2	P3	J7	800
5	S2	P3	J6	400
6	S2	P3	J5	600
7	S2	P3	J4	500
8	S2	P3	J3	200
9	S2	P3	J2	200
10	S2	P3	J1	400

# Подзапросы, единственное значение

Query Editor

Query History

1

SELECT name

2

FROM public.products

3

where id = 'P1'

Data Output

Explain

Messages

	name	
	character varying	
1	Гайка	

Query Editor

Query History

1

select \*

2

from products

3

where name = (

4

SELECT name

5

FROM public.products

6

where id = 'P3'

7

)

8

Data Output

Explain

Messages

Notifications

	color	id	name	weight	city
	character varying	character varying	character varying	integer	character varying
1	Голубой	P3	Винт	17	Рим
2	Красный	P4	Винт	14	Лондон

---

Оконные (аналитические)  
функции



**Оконные функции** - полезный инструмент для построения сложных аналитических запросов.

Для их использования нужно задать параметры окна и функцию, которую хотим посчитать на каждом объекте внутри окна.

Простой пример - функция ROW\_NUMBER(). Эта функция нумерует строки внутри окна. Пронумеруем контент для каждого пользователя в порядке убывания рейтингов.

Номер поставщика	Номер детали	Номер изделия	Количество
S1	P1	J1	200
S1	P1	J4	700
S5	P1	J4	100
S5	P2	J2	200
S5	P2	J4	100
S2	P3	J1	400
S3	P3	J1	200
S2	P3	J2	200
S2	P3	J3	200
S5	P3	J4	200
S2	P3	J4	500
S2	P3	J5	600
S2	P3	J6	400
S2	P3	J7	800
S3	P4	J2	500
S5	P4	J4	800
S2	P5	J2	100
S5	P5	J4	400
S5	P5	J5	500
S5	P5	J7	100
S5	P6	J2	200
S4	P6	J3	300
S5	P6	J4	500
S4	P6	J7	300

---

```
SELECT
  userId, movieId, rating,
  ROW_NUMBER() OVER (PARTITION BY userId ORDER BY rating DESC) as
  movie_rank
FROM (
  SELECT DISTINCT
    userId, movieId, rating
  FROM ratings
  WHERE userId <> 1 LIMIT 1000
) as sample
ORDER BY
  userId,
  rating DESC,
  movie_rank
LIMIT 20;
```





Результат:

userid | movieid | rating | movie\_rank

-----+-----+-----+-----

2	1356	5	1
2	339	5	2
2	648	4	3
2	605	4	4
2	1233	4	5
2	1210	4	6
2	377	4	7
2	260	4	8
2	79	4	9
2	628	4	10
2	64	4	11
2	58	3	12
2	25	3	13
2	762	3	14

---

Параметры запроса:

- ROW\_NUMBER - функция, которую применяем к окну
- OVER - описание окна

Описание окна содержит:

- PARTITION BY - поле (или список полей), которые описывают группу строк для применения оконной функции
- ORDER BY - поле, которое задаёт порядок записей внутри окна. Для полей внутри ORDER BY можно применять стандартные модификаторы DESC, ASC

Оконная функция никак не меняет количество строк в выдаче, но к каждой строке добавляется полезная информация - например, про порядковый номер строки внутри окна

Названия функций обычно отражают их смысл. Ниже будут приведены примеры использования и результаты запросов

## SUM()

Суммирует значения внутри окна. Посчитаем странную метрику - разделим каждое значение рейтинга на сумму всех рейтингов этого пользователя.

```
SELECT userId, movieId, rating,  
       rating / SUM(rating) OVER (PARTITION BY userId) as  
       strange_rating_metric  
FROM (SELECT DISTINCT userId, movieId, rating FROM ratings WHERE  
      userId <> 1 LIMIT 1000) as sample  
ORDER BY userId, rating DESC  
LIMIT 20;
```

Результат:

userid	movieid	rating	strange_rating_metric
2	339	5	0.0684931506849315
2	1356	5	0.0684931506849315
2	648	4	0.0547945205479452
2	64	4	0.0547945205479452
2	79	4	0.0547945205479452
2	260	4	0.0547945205479452
2	1233	4	0.0547945205479452
2	1210	4	0.0547945205479452
2	377	4	0.0547945205479452
2	605	4	0.0547945205479452
2	628	4	0.0547945205479452
2	762	3	0.0410958904109589
2	141	3	0.0410958904109589
2	780	3	0.0410958904109589
2	5	3	0.0410958904109589
2	58	3	0.0410958904109589
2	25	3	0.0410958904109589
2	1475	3	0.0410958904109589
2	32	2	0.0273972602739726
2	1552	2	0.0273972602739726

(20 rows)

## **COUNT(), AVG()**

Счётчик элементов внутри окна, а так же функция Average(). Для наглядности воспользуемся ими одновременно - результаты не должны отличаться. Вычислим полезную метрику - отклонение рейтинга пользователя от среднего рейтинга, который он склонен выставлять

```
SELECT userId, movieId, rating,  
       rating - AVG(rating) OVER (PARTITION BY userId) rating_deviance_simplex,  
       rating - SUM(rating) OVER (PARTITION BY userId) /COUNT(rating) OVER  
(PARTITION BY userId) as rating_deviance_complex  
FROM (SELECT DISTINCT userId, movieId, rating FROM ratings WHERE userId <>1  
LIMIT 1000) as sample  
ORDER BY userId, rating DESC  
LIMIT 20;
```

\_\_\_\_\_

Результат:

userid | movieid | rating | rating\_deviance\_simplex | rating\_deviance\_complex

-----+-----+-----+-----+-----				
2	339	5	1.68181818181818	1.68181818181818
2	1356	5	1.68181818181818	1.68181818181818
2	648	4	0.681818181818182	0.681818181818182
2	64	4	0.681818181818182	0.681818181818182
2	79	4	0.681818181818182	0.681818181818182
2	260	4	0.681818181818182	0.681818181818182
2	1233	4	0.681818181818182	0.681818181818182
2	1210	4	0.681818181818182	0.681818181818182
2	377	4	0.681818181818182	0.681818181818182
2	605	4	0.681818181818182	0.681818181818182
2	628	4	0.681818181818182	0.681818181818182

—

ВРЕМЯ ПРАКТИКИ

## Практика 4

Выведите таблицу с 3-мя полями: название фильма, имя актера и количества фильмов, в которых он снимался



—

# ИТОГИ ЗАНЯТИЯ

---

# Мы сегодня сделали:

Разобрали и увидели работу соединений (внешнее, внутреннее, левое, правое, декартово)

- Увидели работу группировок (Group By, Having)
- Узнали отличия и применения агрегирующих и оконных функций, и подзапросов

---

# Полезные материалы

## 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](http://www.skillz.ru/dev/php/article-Obyasnenie_SQL_obedinenii_JOIN_INNER_OUTER.html)



НЕТОЛОГИЯ  
групп

Спасибо за  
внимание!

Алексей Кузьмин



[aleksej.kyzmin@gmail.com](mailto:aleksej.kyzmin@gmail.com)