



# WHERE или ON

## WHERE или ON

Обычно процесс фильтрации выполняется в части WHERE, когда обе таблицы уже соединены.

При этом существует возможность фильтровать записи еще до выполнения соединения — можно указывать в этом предикате дополнительные условия.

### Пример:

Нужно написать запрос, который покажет название компании, статус и количество уникальных инвесторов в компанию. Выбрать только компании из Нью-Йорка. Отсортировать по убыванию уникальных инвесторов.

```
SELECT c.name AS company_name,  
       c.status,  
       COUNT(DISTINCT i.investor_permalink) AS unique_investors  
FROM companies c  
LEFT JOIN investments i  
  ON c permalink = i.company_permalink  
WHERE c.state_code = 'NY'  
GROUP BY 1,2  
ORDER BY 3 DESC
```

👉 Фильтр в WHERE убирает NULL из результатов. Если указывать фильтр в WHERE на поле из правой таблички, то из LEFT получается INNER JOIN, исключение составляет явная проверка на null ("`<column> is null`").

### Пример (LEFT JOIN + WHERE):

Нужно написать запрос, который покажет все компании и выведет цену покупки, если она была приобретена за евро (EUR)

```
— неправильно (получим INNER JOIN)  
select c.permalink, a.price_amount, a.price_currency_code  
FROM companies c  
LEFT JOIN acquisitions a  
  ON c.permalink = a.company_permalink
```

```
where a.price_currency_code = 'EUR'
order by a.price_amount
```

```
— правильно
select c.permlink, a.price_amount, a.price_currency_code
FROM companies c
LEFT JOIN acquisitions a
  ON c.permlink = a.company_permlink
  and a.price_currency_code = 'EUR'
order by a.price_amount
```

```
— тоже правильно
select c.permlink, a.price_amount, a.price_currency_code
FROM companies c
LEFT JOIN acquisitions a
  ON c.permlink = a.company_permlink
where a.price_currency_code = 'EUR' or a.price_currency_code is null
order by a.price_amount
```

### Пример (FULL JOIN):

Нужно написать запрос, который покажет количество компаний:

- которые есть в таблице companies
- которые есть в двух таблицах
- которые есть в таблице acquisitions

```
SELECT
  COUNT(CASE WHEN c.permlink IS NOT NULL
    AND a.company_permlink IS NULL
    THEN 1 ELSE NULL END) AS companies_only,
  COUNT(CASE WHEN c.permlink IS NOT NULL
    AND a.company_permlink IS NOT NULL
    THEN 1 ELSE NULL END) AS both_tables,
  COUNT(CASE WHEN c.permlink IS NULL
    AND a.company_permlink IS NOT NULL
    THEN 1 ELSE NULL END) AS acquisitions_only
FROM companies c
FULL JOIN acquisitions a
  ON c.permlink = a.company_permlink
```

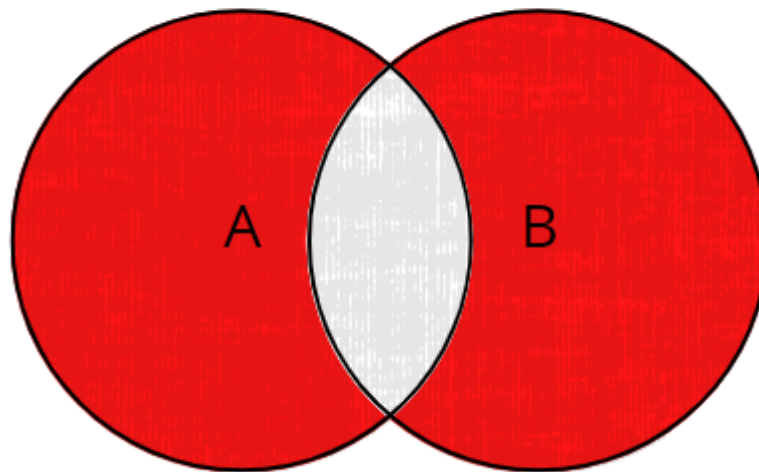


# UNION JOIN

## UNION JOIN

Команда UNION объединяет данные из нескольких таблиц в одну при выборке.

## UNION JOIN



```
SELECT <select list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
OR B.Key IS NULL
```

**✗ UNION JOIN** был в стандарте SQL-92, но убран в SQL-2003 и обычно не используется.



# EQUI JOIN

NATURAL JOIN: естественное соединение

USING

NON EQUI JOIN

SELF JOIN

Накопительный итог

MULTIPLE JOIN

MULTI JOINS

Эквисоединения — это соединения с условием по равенству столбцов.

Для эквисоединения по столбцам с одинаковыми именами имеются отдельные синтаксические формы:

- естественное соединение (NATURAL JOIN)
- соединение, использующее имена столбцов (USING)

## NATURAL JOIN: естественное соединение

```
A NATURAL <тип> JOIN B
```

Автоматически определяет столбцы, имеющие одинаковые имена в обеих соединяемых таблицах.

NATURAL добавляется перед типом соединения. Предикат здесь не нужен, поскольку он подразумевается (а именно попарное равенство всех столбцов с одинаковыми именами в обеих таблицах).

Сравните два вида записи:

```
A NATURAL INNER JOIN B  
A INNER JOIN B ON A.a = B.a and A.b = B.b
```

Если у обеих соединяемых таблиц есть столбцы a и b, то естественное соединение будет эквивалентно такому соединению по предикату.

```
SELECT *  
FROM natural_join_a  
NATURAL JOIN natural_join_b;
```

При естественном соединении одноименные столбцы будут присутствовать в выборке в одном экземпляре.

Более того, в некоторых базах данных к таким столбцам нельзя обращаться с указанием таблицы. В PostgreSQL можно.

💬 NATURAL JOIN поддерживают не все СУБД.

## USING

```
A <тип> JOIN B USING(<список столбцов>)
```

USING применяется, если требуется выполнить эквисоединение не по всем столбцам с совпадающими именами, а только по их части.

👉 Список столбцов содержит те столбцы, по которым выполняется эквисоединение.

Поэтому в этом списке могут присутствовать только те из столбцов, имена которых совпадают в обеих соединяемых таблицах.

Перепишем предыдущий запрос так, чтобы эквисоединение выполнялось только по столбцу a.

```
SELECT *  
FROM natural_join_a  
JOIN natural_join_b  
USING (a);
```

## NON EQUI JOIN

В ON можно задавать любой предикат, в том числе и неравенство.

📌 **NON EQUI JOIN** вместо равенства использует операторы сравнения, такие как `>`, `<`, `>=`, `<=`, `BETWEEN` в условиях.

#### 💻 **Пример:**

Нужно вывести компании и количество их инвестиций, которые были осуществлены после пяти лет существования компании.

```
SELECT
    c.name,
    COUNT(i.company_permalink) AS cnt
FROM companies c
LEFT JOIN investments i
ON c.permlink = i.company_permalink
and (EXTRACT(year from i.funded_at)
    > EXTRACT(year from c.founded_at) + 5)
GROUP BY 1
ORDER BY 2 DESC
```

Если использовать `WHERE` вместо второго условия в `ON`, тогда не будут выбраны компании, у которых не было инвестиций.

## SELF JOIN

```
A JOIN A ON <условие>
```

В SQL можно комбинировать данные из нескольких источников. При этом из одного и того же источника можно комбинировать данные несколько раз: строки из некоторой таблицы сопоставляются с другими строками той же самой таблицы.

**SELF JOIN** — соединение таблицы самой на себя.

💬 Специального оператора для этого типа соединения в SQL нет, но SQL позволяет использовать любой обычный тип соединений при использовании одной таблицы несколько раз. Разница между обычным JOIN и SELF JOIN лишь в том, что и слева и справа от выражения JOIN указывается одна таблица.

#### 💻 **Пример**

В таблице `employee` есть `id` работника, имя, зарплата и `id` руководителя работника.

Если посмотреть на данные в таблице, то видно, что все за исключением босса имеют руководителей.

Вот запрос, который вернет непосредственного начальника каждого сотрудника.

```
SELECT
  e.id, e.name, e.salary,
  m.name AS boss_name
FROM employee e
JOIN employee m
ON e.manager_id = m.id
```

Используется одна таблица и в левой, и в правой части соединения. Чтобы как-то их отличать, им присваиваются разные алиасы.

Нужно использовать алиасы при self-join, потому что все столбцы в двух источниках одинаковые и их нужно отличать. Несмотря на то, что фактически используется одна и та же таблица, SQL считает их двумя различными источниками данных.


Самый главный босс не показывается в результатах — у него нет менеджера, и внутреннее соединение отбрасывает его. С таким запросом босс будет выводиться в результатах:

```
SELECT
  e.id, e.name, e.salary,
  m.name AS boss_name
FROM employee e
LEFT JOIN employee m
ON e.manager_id = m.id
```

## Накопительный итог

SELF JOIN можно использовать для вычисления накопительных итогов — задачи, часто возникающей на практике.

В предположении некоторой упорядоченности строк накопительный итог для каждой строки представляет собой сумму значений некоторого числового столбца для этой строки и всех строк, расположенных до данной.

 Накопительный итог для первой строки в упорядоченном наборе будет равен значению в этой строке.

Для любой другой строки накопительный итог будет равен сумме значения в этой строке и накопительного итога в предыдущей строке.

### Пример:

Дана таблица total. Необходимо найти накопительный итог по полю income в разрезе id.

```
SELECT
  id, date, SUM(income)
FROM
(
  SELECT t1.id, t1.date, t2.income
  FROM total t1
  JOIN total t2
  ON t1.id = t2.id
    and t1.date >= t2.date
) t
GROUP BY 1, 2
ORDER BY 1, 2
```

## MULTIPLE JOIN

Множественные соединения: разбор примера.

```
SELECT *
FROM DMA.click_stream_raw csr
INNER JOIN dds.L_Track_SearchEvent se
  ON csr.track_id = se.track_id and csr.event_no = se.event_no
INNER JOIN dds.S_Track_ObjectsCount oc
  ON csr.track_id = oc.track_id and csr.event_no = oc.event_no
LEFT JOIN dds.S_Track_Position wp
  ON csr.track_id = wp.track_id and csr.event_no = wp.event_no
LEFT JOIN dds.H_SearchEvent hse
  ON se.SearchEvent_id = hse.SearchEvent_id
```



```

SELECT *
FROM DMA.click_stream_raw csr
INNER JOIN dds.L_Track_SearchEvent se
    ON csr.track_id = se.track_id and csr.event_no = se.event_no
INNER JOIN dds.S_Track_ObjectsCount oc
    ON csr.track_id = oc.track_id and csr.event_no = oc.event_no
LEFT JOIN dds.S_Track_Position wp
    ON csr.track_id = wp.track_id and csr.event_no = wp.event_no
LEFT JOIN dds.H_SearchEvent hse
    ON se.SearchEvent_id = hse.SearchEvent_id

```

Сначала может показаться, что каждая таблица в текущем запросе соединяется с предыдущей таблицей; выражение ON показывает это ссылкой на колонки в предыдущих таблицах.

Выглядит так, что таблицы L\_Track\_SearchEvent, S\_Track\_ObjectsCount и S\_Track\_Position соединяются с click\_stream\_raw, а H\_SearchEvent с L\_Track\_SearchEvent.

На самом деле в множественном соединении происходит другое.

```

SELECT *
FROM DMA.click_stream_raw csr
INNER JOIN dds.L_Track_SearchEvent se
    ON csr.track_id = se.track_id and csr.event_no = se
INNER JOIN dds.S_Track_ObjectsCount oc
    ON csr.track_id = oc.track_id and csr.event_no = oc.event_no
LEFT JOIN dds.S_Track_Position wp
    ON csr.track_id = wp.track_id and csr.event_no = wp.event_no
LEFT JOIN dds.H_SearchEvent hse
    ON se.SearchEvent_id = hse.SearchEvent_id

```

Мульти-соединение выполняет серию одиночных соединений между только двумя таблицами одновременно. Каждое одиночное соединение порождает производную (или промежуточную) таблицу (derived/intermediate table), которая уже соединяется со следующей таблицей и так далее — как это показано здесь:

```

SELECT *
FROM DMA.click_stream_raw csr
INNER JOIN dds.L_Track_SearchEvent se
    ON csr.track_id = se.track_id and csr.event_no = se.event_no
INNER JOIN dds.S_Track_ObjectsCount oc
    ON csr.track_id = oc.track_id and csr.event_no = oc.event_no
LEFT JOIN dds.S_Track_Position wp
    ON csr.track_id = wp.track_id and csr.event_no = wp.event_no
LEFT JOIN dds.H_SearchEvent hse
    ON se.SearchEvent_id = hse.SearchEvent_id

```

Правильный способ чтения выражений ON не в том, что новая таблица соединяется с предыдущей. Единственное соединение, которое работает подобным образом — это первое. Все последующие присоединяются к производным таблицам, которые являются результатами предыдущих соединений.

## MULTI JOINS

- Каждый JOIN соединяет 2 таблицы
- JOIN производит промежуточную таблицу, которая используется в следующих соединениях
- Можно использовать любые типы соединений
- Порядок важен

### Пример:

Выбраны три таблицы: workers, vehicles, countries.

Если посмотреть на содержимое таблицы workers, то можно определить, что один сотрудник никуда не поехал, а для еще одного неизвестен тип транспорта.

Вот так соединяются все таблицы с помощью внутреннего соединения :

```

SELECT w.name, c.name, v.name
FROM workers w
INNER JOIN vehicles v
    ON w.vehicle_id = v.id
INNER JOIN countries c
    ON w.dest_country_id = c.id

```

В этом случае всего 2 строки соответствует критерию, которые задаются для внутренних соединений.

В случае, если нужно оставить в результате транспортные средства, на которых никто не передвигался, можно использовать RIGHT JOIN вместо INNER.

```
SELECT w.name, c.name, v.name
FROM workers w
RIGHT JOIN vehicles v
ON w.vehicle_id = v.id
INNER JOIN countries c
ON w.dest_country_id = c.id
```

Такой запрос неверен из-за порядка выполнения. Значит, нужно изменить порядок.

```
SELECT w.name, c.name, v.name
FROM workers w
INNER JOIN countries c
ON w.dest_country_id = c.id
RIGHT JOIN vehicles v
ON w.vehicle_id = v.id
```

Для того, чтобы получить вообще все строки из всех таблиц (то есть не декартово произведение друг на друга, а какой-то другой тип соединения, которое позволит увидеть нам записи, ссылки на которые отсутствуют в других таблицах), можно применить FULL

```
SELECT w.name, c.name, v.name
FROM workers w
FULL JOIN countries c
ON w.dest_country_id = c.id
FULL JOIN vehicles v
ON w.vehicle_id = v.id
```

### Пример:

Дана таблица sequence. В ней содержится последовательность чисел с пропусками. Нужно написать запрос, который выведет пропущенные интервалы. Ожидаемый формат ответа:

start	stop
3	4
6	9
13	14
16	16

```
SELECT 1.n + 1 AS start
FROM sequence s1
LEFT JOIN sequence s2 on s1.n = s2.n - 1
WHERE s2.n is NULL
```

Таким способом можно найти левую границу пропущенных интервалов.

```
SELECT 1.n + 1 AS start, min(s3.n) - 1 AS stop
FROM sequence s1
LEFT JOIN sequence s2 ON s1.n = s2.n - 1
LEFT JOIN sequence s3 ON s1.n < s3.n
WHERE s2.n is null and s3.n is not null
GROUP BY s1.n
ORDER BY s1.n
```