



# GROUP BY

GROUP BY (повторение)

Связь GROUP BY и ORDER BY

GROUP BY и NULL значения

HAVING

## GROUP BY (повторение)

```
SELECT столбец, агрегаты FROM ...  
GROUP BY столбец
```

- Разбивает данные на группы
- Агрегаты работают по группам

Агрегатные функции, такие как **COUNT**, **AVG** и **SUM** (без указания группировки) агрегируют данные в этой колонке по всей таблице. В случае, когда нужно агрегировать таблицу по частям (например, подсчитать количество записей для каждого года), нужно выражение **GROUP BY**.

GROUP BY позволяет разбить данные на группы, которые могут быть агрегированы независимо друг от друга.

### Пример:

Нужно подсчитать количество записей для каждого года.

```
SELECT yr, COUNT(*)  
FROM (  
  SELECT  
    EXTRACT (year from date) AS yr  
  FROM stocks  
) AS stocks  
GROUP BY yr;
```

## Связь GROUP BY и ORDER BY

📌 Нельзя использовать для упорядочивания поле, которого нет в группе, но можно сортировать по агрегатам. Ситуация аналогична DISTINCT.

### Пример:

Нужно найти средний объем дневных торгов для каждого месяца. Результат отсортировать в порядке убывания объема торгов.

```
SELECT mn,
       AVG(volume) AS avg_trade_volume
FROM (
  SELECT
    EXTRACT (month from date) AS mn,
    volume
  FROM stocks
) stocks
GROUP BY mn
ORDER BY AVG(volume) DESC;
```

👉 С помощью GROUP BY можно группировать по нескольким полям:

```
SELECT столбец1, столбец2, ..., агрегаты FROM ...
GROUP BY столбец1, столбец2, ...
```

Или по номерам столбцов:

```
SELECT столбец1, столбец2, ..., агрегаты FROM ...
GROUP BY 1, 2, ...
```

💬 Как и в ORDER BY, можно указывать номера столбцов вместо колонок. Но так лучше делать, если у вас слишком большое количество колонок.

### Пример:

Нужно подсчитать количество различных месяцев для каждого года.

```
SELECT
    EXTRACT(year from date),
    COUNT(DISTINCT EXTRACT(month from date))
FROM stocks
GROUP BY 1;
```

## GROUP BY и NULL значения

Аналогично DISTINCT — NULL в одной группе.

### Пример:

Нужно найти среднемесячный объем торгов за календарный год и суммарный объем торгов для остальных одним запросом для всех годов до 2011.

*Подсказка: в GROUP BY можно использовать CASE для выбора столбцов группировки*

```
SELECT
    yr,
    AVG(volume) AS volume
FROM (
    SELECT
        EXTRACT(year from date) AS yr,
        SUM(volume) AS volume
    FROM stocks
    GROUP BY 1, CASE WHEN EXTRACT(year from date) < 2011
    THEN EXTRACT(month from date) END
) t
GROUP BY 1
ORDER BY 1;
```

## HAVING

```
SELECT ... FROM ...
GROUP BY ... HAVING условие
```

HAVING позволяет осуществлять **фильтрацию по агрегатам**.

💬 Допустим, нужно получить агрегатные значения по месяцам, но при этом стоит задача учитывать только те месяцы, в которых стоимость акций не поднималась выше \$600.

Выражение WHERE не подходит, потому что не позволяет осуществлять фильтрацию по агрегатам — в таком случае используют HAVING. Сравните 2 запроса и их результаты.

```
-неправильно
SELECT "month", MAX(high)
FROM stocksWHERE high< 600
GROUP BY 1;
```

```
правильно
SELECT "month", MAX(high)
FROM stocksGROUP BY 1
HAVING MAX(high) < 600;
```

### Пример:

Нужно найти максимальную цену акций за весь период наблюдений и присвоить этому выражению алиас. Отфильтровать значение с помощью алиаса.

```
SELECT MAX(high) AS m
FROM stocks
ORDER BY m;
```

👉 В HAVING нельзя использовать алиас, используемый для именования значений агрегатной функции в предложении SELECT, но при этом можно использовать в ORDER BY — это вызвано порядком выполнения частей.

❌ Например, такой запрос выдает ошибку: “столбец "m" не существует”.

```
SELECT MAX(high) AS m
FROM stocks
HAVING m > 500;
```



# SELECT и JOIN

## SELECT: порядок выполнения частей

### JOIN

- Введение в соединения
- Анатомия соединений
- Типы соединений
- CROSS JOIN: Декартово произведение
- INNER JOIN
- Внешние соединения
- Таблицы для практики

## SELECT: порядок выполнения частей

1. FROM
2. WHERE
3. GROUP BY
4. HAVING
5. SELECT
6. ORDER BY

## JOIN

JOIN используется, когда нужно включить данные из нескольких источников данных.

```
SELECT [DISTINCT | ALL]{*  
| [выражение для столбца> [[AS] <псевдоним>]] [,...]}  
FROM <имя таблицы> [[AS] <псевдоним>] [,...]  
[WHERE <предикат>]  
[[GROUP BY <список столбцов>]
```

```
[HAVING <условие на агрегатные значения>] ]  
[ORDER BY <список столбцов>]
```

В части FROM допускается использование нескольких таблиц.

Простое перечисление через запятую используется не так часто — оно соответствует декартовому произведению.

```
FROM <таблица 1>  
{[INNER] | {{LEFT | RIGHT | FULL } [OUTER]}} JOIN <таблица 2>  
ON <предикат>
```

Чаще используется явная операция соединения с указанием ключевого слова JOIN.

## Введение в соединения

В SQL есть механизм, который позволяет объединять данные из нескольких источников в одно отношение, что позволяет производить с этими данными дальнейшие операции. Это и есть суть соединений.

### Пример

Предположим, перед вами стоит задача хранить сообщения мессенджера Авито.

Основными сущностями там будут **пользователь** и само **сообщение**.

Если хранить все данные по пользователям и сообщениям в одной таблице, это создаст проблему.

Например, когда будет нужно обновить информацию в личном кабинете (ник или настройки оповещения), потребуется изменить все сообщения в этой таблице. Поэтому лучше разделить такую таблицу на две: профиль пользователя и сообщения.

---

## Анатомия соединений

### Пример:

Нужно написать запрос для выбора средней стипендии учеников кураторов групп.

```
SELECT
    g.curator,
    AVG(s.studentship) AS avg_studentship
FROM students s
JOIN groups g
    ON s.group_code = g.group_code
GROUP BY 1;
```

- **Алиасы**

Ранее при записи простых подзапросов в части FROM требовалось задавать алиасы.

При использовании соединений алиасы задавать не обязательно — можно обращаться к столбцам по полному названию таблицы.

👉 Лучше задать короткий алиас и работать уже с ним — это удобнее.

После того, как алиасы указаны, можно обращаться к столбцам через них, как в SELECT-листе, так и в других частях запроса.

- **JOIN и ON**

После выражения FROM появляется два новых выражения: JOIN, за которым следует имя таблицы, и ON, за которым следует предикат.

📌 **ON всегда следует после JOIN**

Предикат в ON определяет, как таблицы (одна после FROM и одна после JOIN) связаны друг с другом.

В этом примере обе таблицы содержат поле group\_code, но зачастую названия полей могут быть не столь очевидны.

```
SELECT
    g.curator,
    AVG(s.studentship) AS avg_studentship
FROM students s
JOIN groups g
    ON s.group_code = g.group_code
GROUP BY 1;
```

☺️ Выражение JOIN и ON можно перевести на обычный язык так:

Соедини все строки из таблицы students со всеми строками из таблицы groups, для которых поле group\_code из таблицы students равно полю group\_code из таблицы groups.

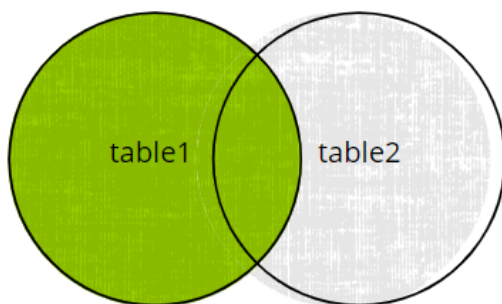
JOIN присоединяет все столбцы второй таблицы по некоторому условию. В этом можно убедиться, выполнив такой запрос.

```
SELECT
  *
FROM students s
JOIN groups g
ON s.group_code = g.group.code;
```

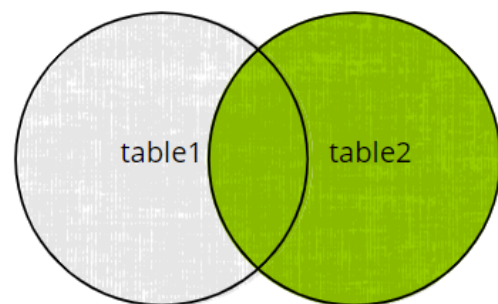
## Типы соединений

Типы соединений часто изображают с помощью диаграммы Венна:

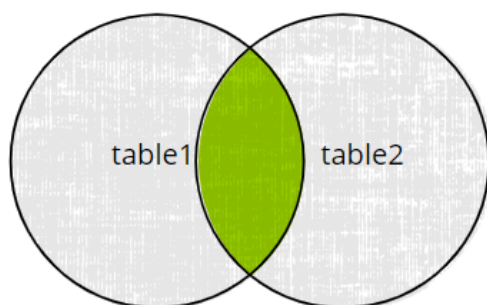
LEFT JOIN



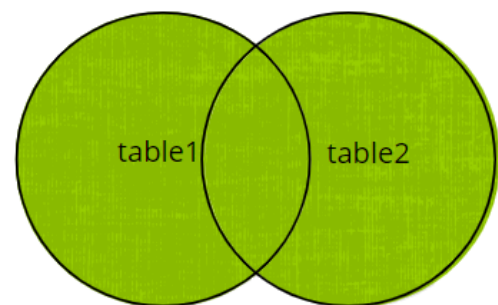
RIGHT JOIN



INNER JOIN



FULL OUTER JOIN



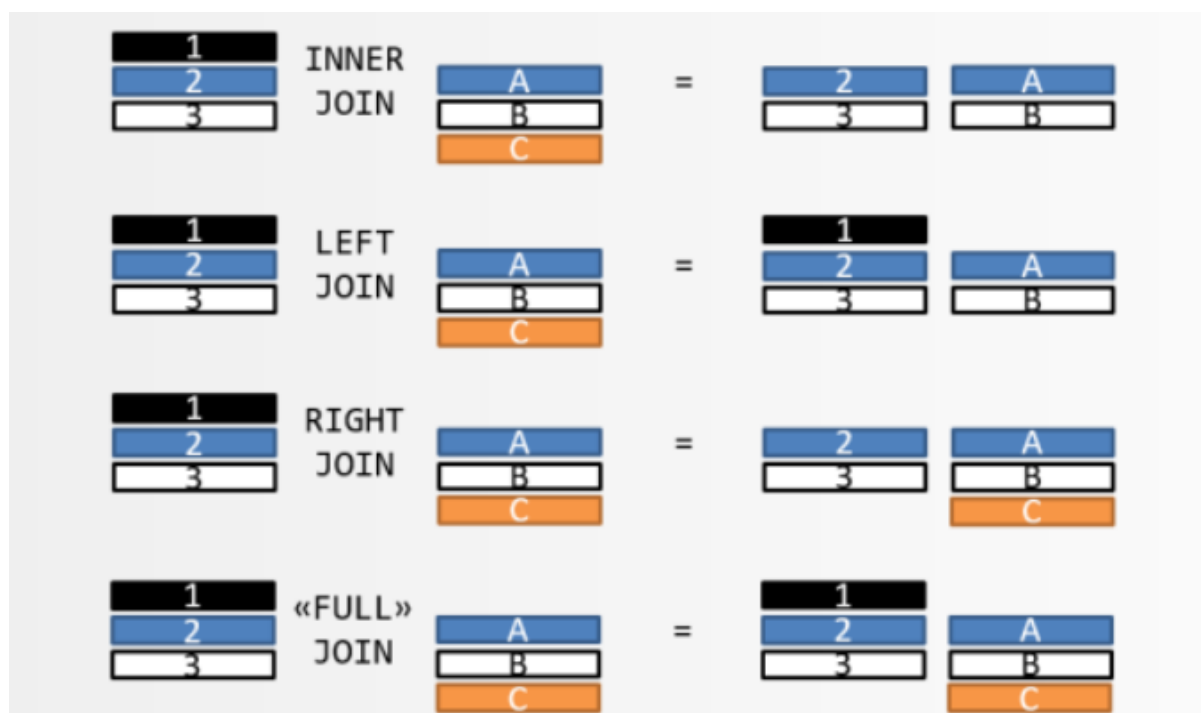
Эти диаграммы показывают, из каких таблиц значения попадут в результат (и ничего более).



Например, диаграмма Венна не отвечает на вопрос, сколько записей будет в результате выполнения запроса.

**✗** Объяснение типов соединений через диаграммы Венна ведет к **неправильному пониманию соединений**.

В таблицах не всегда ключи уникальны (хотя в реляционной модели должны быть), и в условии соединения могут участвовать не только ключи, а любой предикат.



Здесь одинаковый признак — это цвет ячеек.

## CROSS JOIN: Декартово произведение

```
A CROSS JOIN B
A, B
```

**📌** Эта операция объединяет каждую строку первой таблицы с каждой строкой второй таблицы,

то есть выдает все возможные комбинации соединения строк двух таблиц.

Это единственный тип соединения, который не требует выражения ON и предиката.

### Пример:

Нужно создать шаблон, чтобы записывать - что мы делали в определенный час. У нас есть календарь на 2021 год и 24 часа в сутках.

```
select *, 'I have been working on ...'
from
  (select '2021-01-01'::date + generate_series(0, 364) date) days
  cross join
  (select generate_series(0, 23) hh) hours
```

## INNER JOIN

```
A [INNER] JOIN B ON <условие>
```

Декартово произведение с условием — это все комбинации соединений строк с неким фильтром.

INNER JOIN — по сути есть CROSS JOIN с дополнительным условием. То есть в результирующую выборку попадают только те записи, для которых выполняется условие в конструкции ON.


### Пример:

Найти сколько студентов у каждого куратора. Учитывая только группы со студентами.

```
select curator, count(*)
from groups_day4 g
  join students on students.group_code = g.group_code
group by 1
```

## Внешние соединения

При применении внутреннего соединения строки, не имеющие совпадения в другой таблице, не появляются в результирующем отношении.

 Во внешнем соединении не совпадающие строки могут быть включены в результат.

## LEFT [OUTER] JOIN

```
A LEFT [OUTER] JOIN B ON <условие>
```

- возвращает совпавшие строки
- включает не совпавшие строки из левой таблицы

LEFT JOIN — то же самое, что и INNER JOIN, но дополнительно в результат включаются остальные записи из левой таблицы, которые не нашли соответствия в правой по условию соединения.

### Пример:

Покажите группы без студентов.

```
select *  
from groups_day4 g  
left join students s on s.group_code = g.group_code  
where s.group_code is null;
```

## RIGHT [OUTER] JOIN

```
A RIGHT [OUTER] JOIN B ON <условие>
```

- аналог LEFT, но «наоборот»
- возвращает совпавшие строки
- включает не совпавшие строки из правой таблицы

RIGHT JOIN — то же самое, что и INNER JOIN, но дополнительно в результат включаются записи из правой таблицы без совпадения с левой.

## FULL [OUTER] JOIN

```
A FULL [OUTER] JOIN B ON <условие>
```

- объединение LEFT и RIGHT
- возвращает совпавшие строки

- включает не совпавшие строки из двух таблиц

#### Пример:

Показать всех студентов без куратора и кураторов без студентов.

```
select g.curator, student_id, name
from groups_day4 g
     full join students on students.group_code = g.group_code
where student_id is null
     or curator is null
```

## JOIN и NULL

column1	column1
1	1
1	NULL
2	1
NULL	6
3	5
5	NULL
NULL	NULL

Создадим таблицу и определим результаты для различных типов JOIN.

```
SELECT *
FROM (values(1),(1),(2),(NULL),(3),(5), (NULL)) a
FULL JOIN (values(1),(NULL),(1),(6),(5),(NULL)) b
ON a.column1 = b.column1;
```

## Таблицы для практики

[companies](#) — данные по компаниям (стартапы, покупатели)

[acquisitions](#) — данные по сделкам (company\_permalink и acquirer\_permalink)

#### Пример:

Нужно найти ссылку на сайт компании, название и дату приобретения.

Может быть несколько сделок по одной компании, то есть можно найти несколько одинаковых названий с разными датами.

Попробуйте решить самостоятельно, а потом проверить себя, выделив и скопировав запрос в свою IDE.

```
SELECT
  c.permalink AS company_permalink,
  c.name AS company_name,
  a.acquired_at AS acquired_date
FROM companies c
JOIN acquisitions a
ON c.permalink = a.company_permalink;
```

---

### Пример:

Нужно найти компании, которые никогда не приобретались.

---

```
SELECT
  c.permalink AS company_permalink,
  c.name AS company_name,
  a.acquired_at AS acquired_date
FROM companies c
LEFT JOIN acquisitions a
ON c.permalink = a.company_permalink
WHERE a.company_permalink IS NULL;
```