




[КАК СТАТЬ АВТОРОМ](#) [Зарплаты IT-специалистов](#)[Где работать в следующем году](#)**varanio**

17 апр 2019 в 21:48

Понимание джойнов сломано. Это точно не пересечение кругов, честно

 4 мин  260K

Разработка веб-сайтов*, PostgreSQL*, Программирование*, SQL*

Так получилось, что я провожу довольно много собеседований на должность веб-программиста. Один из обязательных вопросов, который я задаю — это чем отличается INNER JOIN от LEFT JOIN.

Чаще всего ответ примерно такой: "inner join — это как бы пересечение множеств, т.е. остается только то, что есть в обеих таблицах, а left join — это когда левая таблица остается без изменений, а от правой добавляется пересечение множеств. Для всех остальных строк добавляется null". Еще, бывает, рисуют пересекающиеся круги.

Я так устал от этих ответов с пересечениями множеств и кругов, что даже перестал поправлять людей.

Дело в том, что этот ответ в общем случае неверен. Ну или, как минимум, не точен.

Давайте рассмотрим почему, и заодно затронем еще парочку тонкостей join-ов.

Во-первых, таблица — это вообще не множество. По математическому определению, во множестве все элементы уникальны, не повторяются, а в таблицах в общем случае это вообще-то не так. Вторая беда, что термин "пересечение" только путает.

(Update. В комментах идут жаркие споры о теории множеств и уникальности. Очень интересно, много нового узнал, спасибо)

INNER JOIN

Давайте сразу пример.

Итак, создадим две одинаковых таблицы с одной колонкой id, в каждой из этих таблиц пусть будет по две строки со значением 1 и еще что-нибудь.

```
INSERT INTO table1
(id)
VALUES
(1),
(1)
(3);

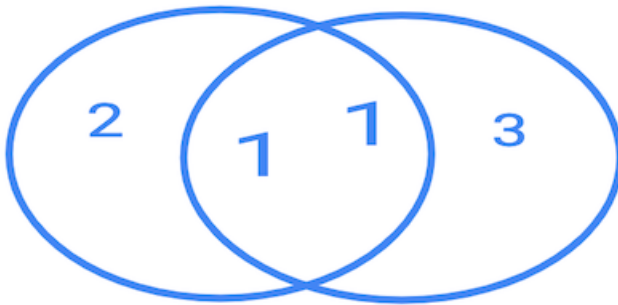
INSERT INTO table2
```

```
(id)
VALUES
(1),
(1),
(2);
```

Давайте, их, что ли, поджойним

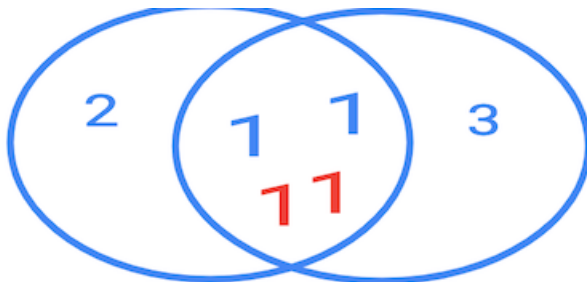
```
SELECT *
FROM table1
  INNER JOIN table2
    ON table1.id = table2.id;
```

Если бы это было "пересечение множеств", или хотя бы "пересечение таблиц", то мы бы увидели две строки с единицами.



На практике ответ будет такой:

id	id
1	1
1	1
1	1
1	1



Но как??

Для начала рассмотрим, что такое CROSS JOIN. Вдруг кто-то не в курсе.

CROSS JOIN — это просто все возможные комбинации соединения строк двух таблиц. Например, есть две таблицы, в одной из них 3 строки, в другой — 2:

```
select * from t1;
```

```
id
----
1
2
3
```

```
select * from t2;
```

```
id
----
4
5
```

Тогда CROSS JOIN будет порождать 6 строк.

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

```
id | id
----+----
1 | 4
1 | 5
2 | 4
2 | 5
3 | 4
3 | 5
```

Так вот, вернемся к нашим баранам.
Конструкция

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

— это, можно сказать, всего лишь синтаксический сахар к

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

Т.е. по сути `INNER JOIN` — это все комбинации соединений строк с неким фильтром `condition`. В общем-то, можно это представлять по разному, кому как удобнее, но точно не как пересечение каких-то там кругов.

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 более интересные алгоритмы.

LEFT JOIN

Если вы считаете, что левая таблица всегда остается неизменной, а к ней присоединяется или значение из правой таблицы или `null`, то это в общем случае не так, а именно в случае когда есть повторы данных.

Опять же, создадим две таблицы:

```
insert into t1
(id)
values
(1),
(1),
(3);

insert into t2
(id)
values
(1),
(1),
(4),
(5);
```

Теперь сделаем `LEFT JOIN`:

```
SELECT *
FROM t1
LEFT JOIN t2
ON t1.id = t2.id;
```

Результат будет содержать 5 строк, а не по количеству строк в левой таблице, как думают очень многие.

```
| id | id |
```

	---		---	
	1		1	
	1		1	
	1		1	
	1		1	
	3			

Так что, LEFT JOIN — это тоже самое что и INNER JOIN (т.е. все комбинации соединений строк, отфильтрованных по какому-то условию), и плюс еще записи из левой таблицы, для которых в правой по этому фильтру ничего не совпало.

LEFT JOIN можно переформулировать так:

```
SELECT *
FROM t1
      CROSS JOIN t2
      WHERE t1.id = t2.id

UNION ALL

SELECT t1.id, null
FROM t1
WHERE NOT EXISTS (
      SELECT
      FROM t2
      WHERE t2.id = t1.id
)
```

Сложноватое объяснение, но что поделать, зато оно правдивее, чем круги с пересечениями и т.д.

Условие ON

Удивительно, но по моим ощущениям 99% разработчиков считают, что в условии ON должен быть id из одной таблицы и id из второй. На самом деле там любое булево выражение.

Например, есть таблица со статистикой юзеров users_stats, и таблица с ip адресами городов. Тогда к статистике можно прибавить город

```
SELECT s.id, c.city
FROM users_stats AS s
      JOIN cities_ip_ranges AS c
      ON c.ip_range && s.ip
```

где && — оператор пересечения (см. расширение погреса [ip4r](#))

Если в условии ON поставить true, то это будет полный аналог CROSS JOIN

```
"table1 JOIN table2 ON true" == "table1 CROSS JOIN table2"
```

Производительность

Есть люди, которые боятся join-ов как огня. Потому что "они тормозят". Знаю таких, где есть полный запрет join-ов по проекту. Т.е. люди скачивают две-три таблицы себе в код и джойнят вручную в каком-нибудь php.

Это, прямо скажем, странно.

Если джойнов немного, и правильно сделаны индексы, то всё будет работать быстро. Проблемы будут возникать скорее всего лишь тогда, когда у вас таблиц будет с десяток в одном запросе. Дело в том, что планировщику нужно определить, в какой последовательности осуществлять джойны, как выгоднее это сделать.

Сложность этой задачи $O(n!)$, где n — количество объединяемых таблиц. Поэтому для большого количества таблиц, потратив некоторое время на поиски оптимальной последовательности, планировщик прекращает эти поиски и делает такой план, какой успел придумать. В этом случае иногда бывает выгодно вынести часть запроса в [подзапрос CTE](#); например, если вы точно знаете, что, поджойнив две таблицы, мы получим очень мало записей, и остальные джойны будут стоить копейки.

Кстати, Еще маленький совет по производительности. Если нужно просто найти элементы в таблице, которых нет в другой таблице, то лучше использовать не 'LEFT JOIN... WHERE... IS NULL', а конструкцию EXISTS. Это и читабельнее, и быстрее.

Выводы

Как мне кажется, не стоит использовать диаграммы Венна для объяснения джойнов. Также, похоже, нужно избегать термина "пересечение".

Как объяснить **на картинке** джойны корректно, я, честно говоря, не представляю. Если вы знаете — расскажите, плиз, и киньте в коменты.

Update В этом видео я наглядно объясняю, как правильно визуализировать джойны (English):

Update2 Продолжение статьи здесь: <https://habr.com/ru/post/450528/>

Больше полезного можно найти на telegram-канале о разработке "Cross Join", где мы обсуждаем базы данных, языки программирования и всё на свете!

Теги: [джойны](#)

Хабы: [Разработка веб-сайтов](#), [PostgreSQL](#), [Программирование](#), [SQL](#)

♦ +97

🔖 692



💬 225

Редакторский дайджест



Присылаем лучшие статьи раз в месяц



239

Карма

70

Рейтинг

Антон Околелов [@varanio](#)

Go-тимлид, веду канал <https://t.me/crossjoin>

Задонатить

[Facebook](#) [Twitter](#) [Telegram](#)