Postgre-DBA

otus.ru

Проверить, идет ли запись

Меня хорошо видно && слышно?

Тема вебинара

Различные виды join'ов. Применение и оптимизация

Игорь Тоескин

Ведущий разработчик СУБД

Специалист в области разработки и проектировании витрин данных в PostgreSQL, а также в области разработки хранимых процедур в таких СУБД как PostgreSQL и Oracle

Правила вебинара

Активно
участвуем

Задаем вопрос
в чат или голосом

Вопросы вижу в чате,
могу ответить не сразу

Условные обозначения

Индивидуально

Время, необходимое на активность

Пишем в чат

Говорим голосом

Документ

Ответьте себе или задайте вопрос

Маршрут вебинара

- Соединения в общем виде
- Различные варианты соединений
- Практика, сравнение запросов
- Рефлексия

Цели вебинара

К концу занятия вы сможете

1. Применять различные типы соединений в своих запросах
2. Определять, какой тип соединения вам нужен в зависимости от ситуации
3. Применять операции над множествами

Смысл

Зачем вам это уметь

1. Для написания запросов с применением различных соединений
2. Для правильного выбора соединения в зависимости от цели
3. Для написания запросов, где необходимо оперировать множествами

Введение

Вопросы

- Данные команды эквиваленты? **Inner join и join**
- Чем отличается **left join** и **right join**?
- Если у нас в обеих таблицах, состоящих из одной колонки, по 10 строк с одинаковыми значениями в обеих таблицах, то каково будет количество строк при их прямом соединении?

О соединении. Nested Loop

```
=> EXPLAIN (COSTS OFF,ANALYZE) SELECT *
  FROM tickets t JOIN ticket_flights tf ON tf.ticket_no = t.ticket_no
  WHERE t.ticket_no IN ('0005432312163','0005432312164');

                                                    QUERY PLAN
-------------------------------------------------------------------------------------------------------------------
 Nested Loop (actual time=2.533..5.056 rows=8 loops=1)
   ->  Index Scan using tickets_pkey on tickets t (actual time=1.313..1.334 rows=2 loops=1)
         Index Cond: (ticket_no = ANY ('{0005432312163,0005432312164}'::bpchar[]))
   ->  Index Scan using ticket_flights_pkey on ticket_flights tf (actual time=0.786..1.848 rows=4 loops=2)
         Index Cond: (ticket_no = t.ticket_no)
 Planning time: 0.383 ms
 Execution time: 5.090 ms
(7 rows)
```

Узел Nested Loop обращается к первому (внешнему) набору за первой строкой. Здесь это -узел Index Scan по билетам.

Затем Nested Loop обращается ко второму (внутреннему) набору и просит выдать все строки, соответствующие строке первого набора. Здесь это -узел Index Scan по перелетам.

Процесс повторяется до тех пор, пока внешний набор не исчерпает все строки.

О соединении. Hash join

```
=> EXPLAIN SELECT *
  FROM tickets t JOIN ticket_flights tf ON tf.ticket_no = t.ticket_no;

                                  QUERY PLAN
------------------------------------------------------------------------------
 Hash Join  (cost=161241.33..494089.17 rows=8391906 width=136)
   Hash Cond: (tf.ticket_no = t.ticket_no)
   ->  Seq Scan on ticket_flights tf  (cost=0.00..149997.06 rows=8391906 width=32)
   ->  Hash  (cost=78283.70..78283.70 rows=2949570 width=104)
         ->  Seq Scan on tickets t  (cost=0.00..78283.70 rows=2949570 width=104)
(5 rows)
```

Для большой выборки оптимизатор предпочитает соединение хешированием.

Узел Hash Join начинает работу с того, что обращается к дочернему узлу Hash. Тот получает от своего дочернего узла (здесь -Seq Scan) весь набор строк и строит хеш-таблицу.

Затем Hash Join обращается ко второму дочернему узлу и соединяет строки, постепенно возвращая полученные результаты.

О соединении. Merge join

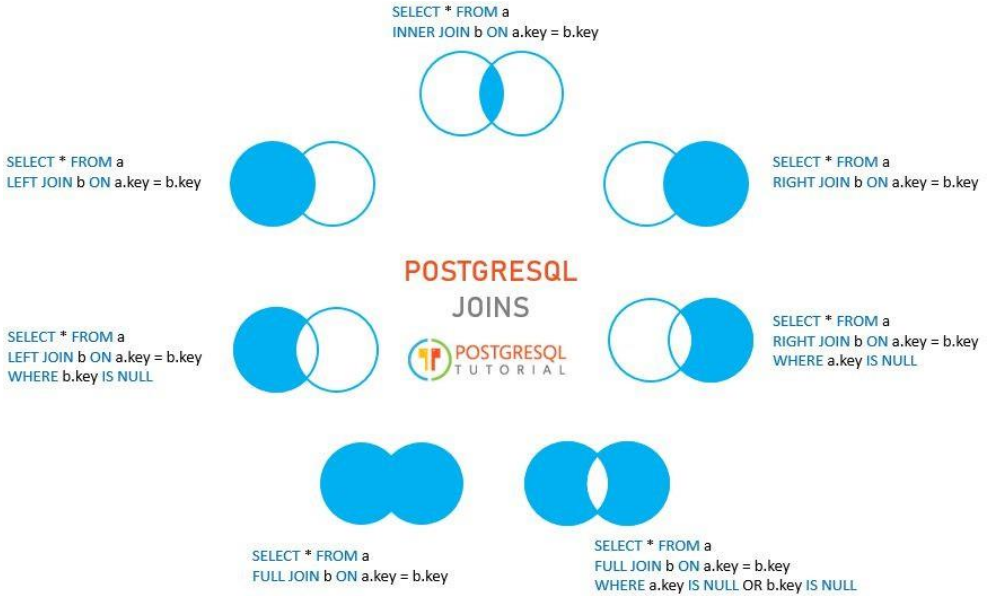
```
=> EXPLAIN SELECT *
  FROM tickets t JOIN ticket_flights tf ON tf.ticket_no = t.ticket_no
  ORDER BY t.ticket_no;

                                                  QUERY PLAN
-------------------------------------------------------------------------------------------------------------------
 Merge Join  (cost=38.10..787871.36 rows=8391906 width=150)
   Merge Cond: (t.ticket_no = tf.ticket_no)
   ->  Index Scan using tickets_pkey on tickets t  (cost=0.43..138478.98 rows=2949570 width=104)
   ->  Index Scan using ticket_flights_pkey on ticket_flights tf  (cost=0.56..537184.08 rows=8391906 width=32)
(4 rows)
```

Если результат необходим в отсортированном виде, оптимизатор может предпочесть соединение слиянием. Особенно, если данные от дочерних узлов можно получить уже отсортированными с помощью индексного сканирования.

В отличие от соединения хешированием, слияние без сортировки хорошо подходит для случая, когда надо быстро получить первые строки.

Различные виды джоинов

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



Can we stop with the SQL JOINs venn diagrams insanity?

Вопросы?

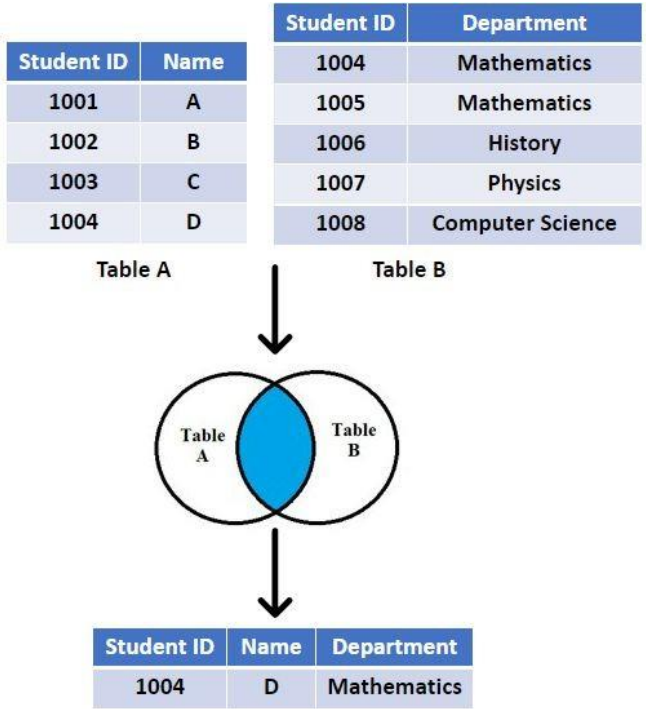
Ставим “+”,
если вопросы есть

Ставим “–”,
если вопросов нет

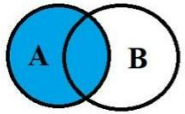
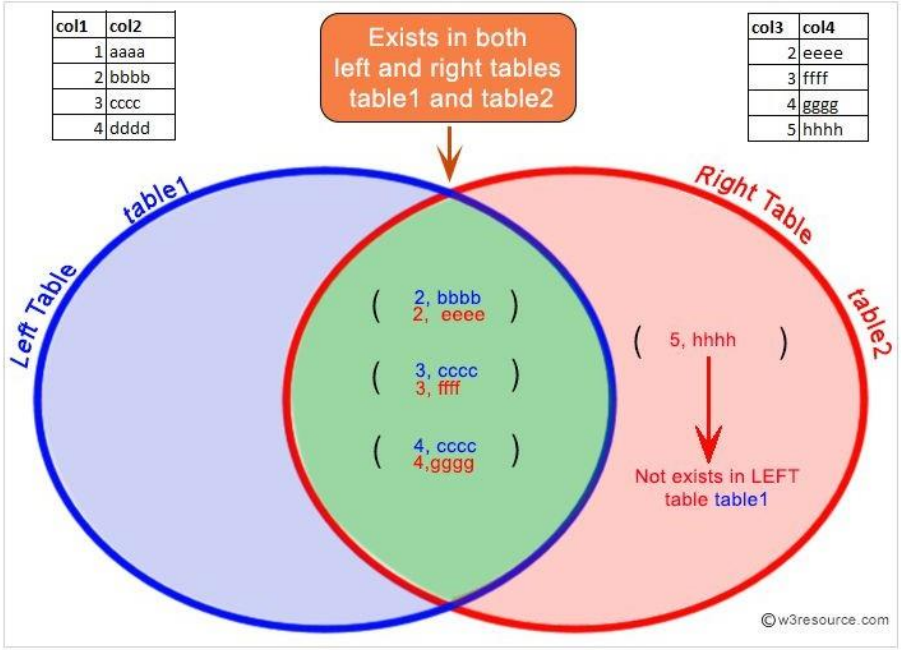
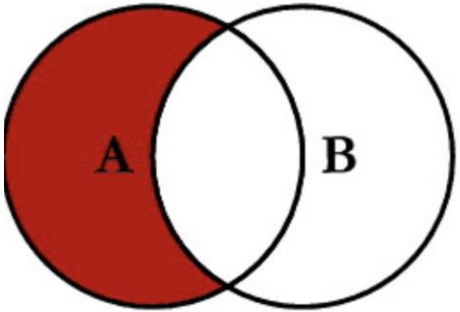
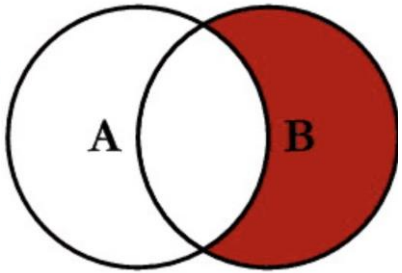
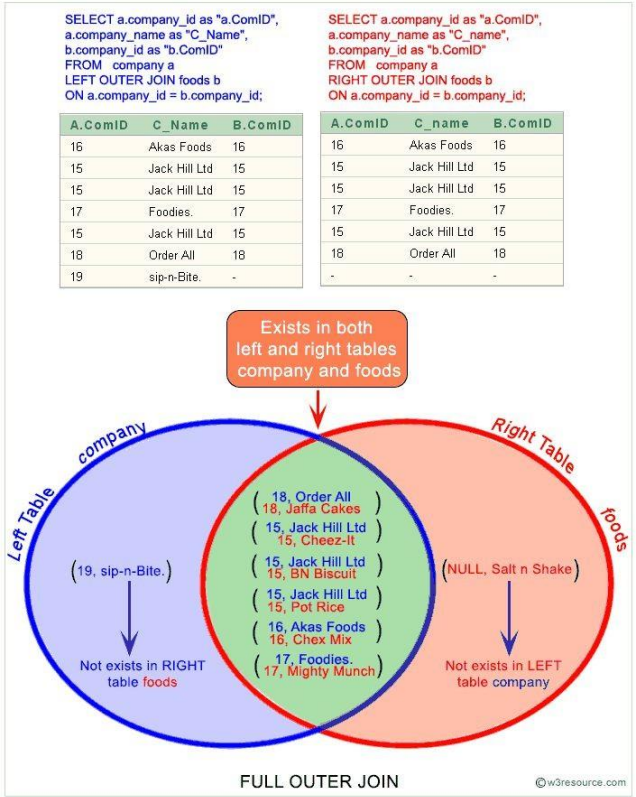
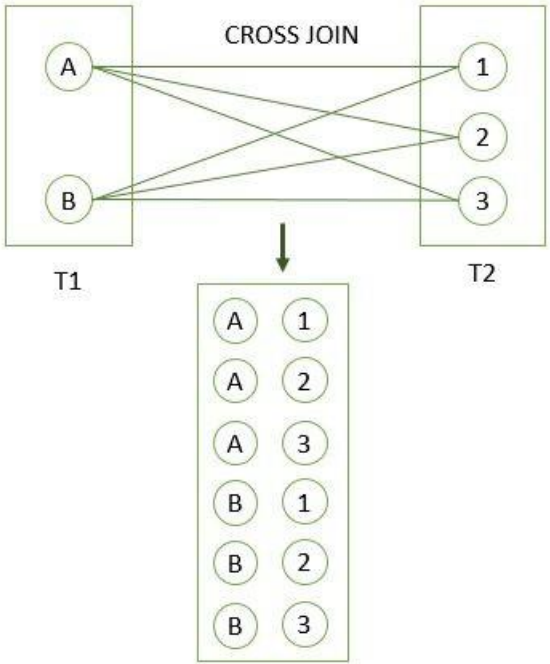
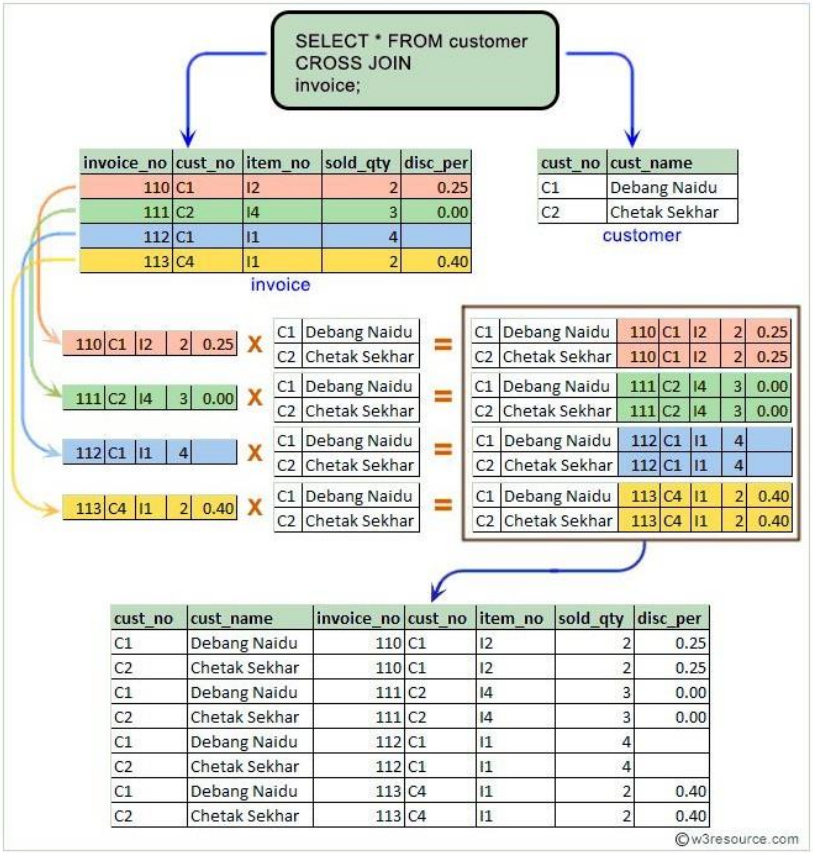
JOINs

Inner Join

- Если ключи в двух таблицах совпадают, то будет возвращена строка, содержащая колонки из обеих таблиц
- Слово INNER может быть опущено
- Можно писать соединения иначе:
  - ```
    SELECT *
    FROM TableA A, TableB B
    WHERE A.Key = B.Key
    ```



Тренировки:

- example1
- example2
    ```

Left Join

- Если ключи в двух таблицах совпадают, то будет возвращена строка, содержащая колонки из обеих таблиц
- Слово **OUTER** может быть опущено
- В случае **left join**: Если для строки из левой таблицы не будет найдено строк с тем же ключом в правой таблице, то вернётся строка из левой таблицы, но в том числе с колонками правой таблицы, в которых будет стоять null

Student ID	Name
1001	A
1002	B
1003	C
1004	D

+

Student ID	Department
1004	Mathematics
1005	Mathematics
1006	History
1007	Physics
1008	Computer Science



Student ID	Name	Department
1001	A	NULL
1002	B	NULL
1003	C	NULL
1004	D	Mathematics

Right Join

- В случае **right join**: Если для строки из правой таблицы не будет найдено строк с тем же ключом в левой таблице, то вернётся строка из правой таблицы, но в том числе с колонками левой таблицы, в которых будет стоять null



Тренировки:

- example3
- example4
- example5

Left/Right Join

- Если хотим вернуть только те строки, для которых соответствия во второй таблице не нашлось, то необходимо добавить условие:
  - В случае **left join**: WHERE b.Key is null (ключ соединения в правой таблице пуст)
  - В случае **right join**: WHERE a.Key is null (ключ соединения в левой таблице пуст)




```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
WHERE B.Key IS NULL
```




```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
```

Тренировки:

- example3
- example4

Full Join

- Если ключи в двух таблицах совпадают, то будет возвращена строка, содержащая колонки из обеих таблиц (пример)
- Если соответствия не было найдено, то колонки из противоположной таблицы будут пусты

- Можно исключить те строки, для которых было найдено соответствие (пример)
- Таким образом будут найдены только те строки, для которых нет соответствия в противоположной таблице



Cross Join





- Cross Join или перекрестное соединение создает набор строк, где каждая строка из одной таблицы соединяется с каждой строкой из второй таблицы (пример).

Lateral Join

- ```
  SELECT <target list>
  FROM <table>
  [INNER]
  {{LEFT | RIGHT | FULL} [OUTER]} JOIN LATERAL
  (<subquery using table.column>) as foo on true;
  ```
- Можно использовать для возврата первых N строк в рамках группы
- Для джойна с функциями, которые возвращают несколько строк (unnest)
- Пример
  ```

Вопросы?

Ставим “+”,
если вопросы есть

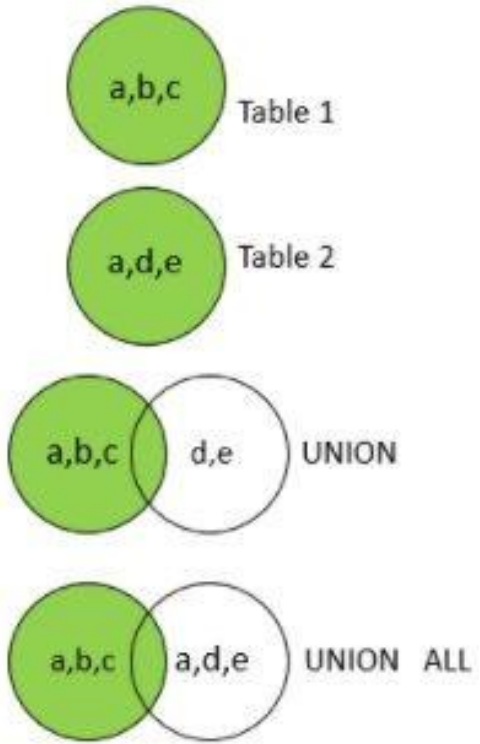
Ставим “–”,
если вопросов нет

Операции над множествами

Объединение множеств. UNION

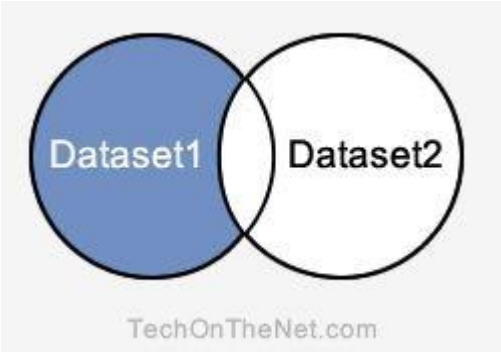
- Можно объединять результаты двух запросов (таблиц) и более
- На выходе будут возвращены строки из всех множеств
- Можно исключать дубликаты (убрав ключевое слово ALL)

Пример



Разность множеств. EXCEPT

- Можно исключить одно множество из другого
- На выходе будут возвращены строки из верхнего множества, для которых не нашлось соответствия в другом множестве
- Можно исключать дубликаты (убрав ключевое слово ALL)

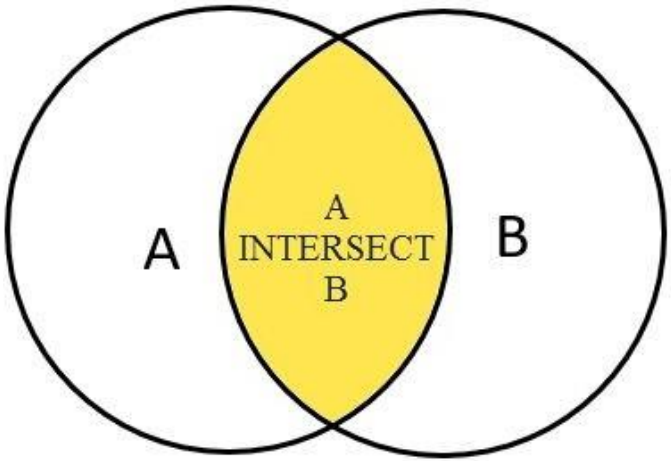


[Пример]

Пересечение множеств. INTERSECT

- Можно пересечь результаты двух запросов (таблиц) и более
- На выходе будут только те строки, которые полностью совпали
- Можно исключать дубликаты (убрав ключевое слово ALL)

Пример



Вопросы?

Ставим “+”,
если вопросы есть

Ставим “–”,
если вопросов нет

Практика

Вопросы?

Ставим “+”,
если вопросы есть

Ставим “–”,
если вопросов нет

Рефлексия

Список материалов для изучения

1. Объясняя необъяснимое. Часть 3
2. 20.7.1. Конфигурация методов планировщика
3. 14.3. Управление планировщиком с помощью явных предложений JOIN
4. Соединение таблиц. Неявное соединение таблиц
5. Пример использования LATERAL JOIN в PostgreSQL
6. UNDERSTANDING LATERAL JOINS IN POSTGRESQL

Цели вебинара

К концу занятия вы сможете

1. Применять различные типы соединений в своих запросах
2. Определять, какой тип соединения вам нужен в зависимости от ситуации
3. Применять операции над множествами

Рефлексия

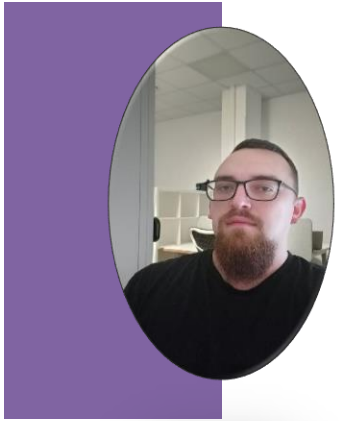
С какими впечатлениями уходите с вебинара?

Как будете применять на практике то,
что узнали на вебинаре?

Заполните, пожалуйста, опрос о занятии по ссылке в чате

Спасибо за внимание!

Приходите на следующие вебинары



Тоескин Игорь

Ведущий разработчик СУБД

Специалист в области разработки и проектировании витрин данных в PostgreSQL, а также в области разработки хранимых процедур в таких СУБД как PostgreSQL и Oracle