Министерство науки и образования Российской Федерации Федеральное государственное бюджетное образовательное учреждение высшего образования

«Московский государственный технический университет имени Н.Э.

Баумана

(национальный исследовательский университет)»

 (МГТУ им. Н.Э. Баумана)



**УЧЕБНО-МЕТОДИЧЕСКОЕ ПОСОБИЕ К ЛАБОРАТОРНЫМ РАБОТАМ**

**ПО КУРСУ «Базы данных (ЮР)»**

**Лабораторная работа № 7-8** **«****Введение в PostgreSQL. Соединение таблиц»**

Авторы:

Кудрявцев А.П., kudryavtsevap@bmstu.ru

Фомин М.М.

Ланцберг А.В.

Москва, 2024

## Соединение таблиц JOIN

1.1. Общие сведения

Соединение (JOIN) - одна из самых важных операций, выполняемых реляционными системами управления базами данных (РСУБД). РСУБД используют соединения для того, чтобы сопоставить строки одной таблицы строкам другой таблицы. Например, соединения можно использовать для отображения аэропортов вылета и прилета при отображении расписания полетов в БД demo. В таблице flights аэропорты вылета и прилета являются трехсимвольными кодами (внешние ключи), ссылающимися на таблицу аэропортов airports\_data и без отображения естественных названий аэропортов разобраться в расписании будет невозможно. Например, можно сделать так:

SELECT f.scheduled\_departure AS "Время вылета",

 f.scheduled\_arrival AS "Время прилета",

 ad.airport\_name::jsonb->>'ru' AS "Аэропорт вылета",
 aa.airport\_name::jsonb->>'ru' AS "Аэропорт прилета"

FROM bookings.flights AS f

 JOIN bookings.airports\_data AS ad ON ad.airport\_code = f.departure\_airport

 JOIN bookings.airports\_data AS aa ON aa.airport\_code = f.arrival\_airport

ORDER BY f.scheduled\_departure;



 В БД demo название аэропортов хранится на русском и английских языках в формате jsonb, например:

{"en": "Vladivostok International Airport", "ru": "Владивосток"}

Для получения русского названия используется запись:

<поле>::jsonb->>'ru'

Больше мы не будем касаться формата jsonb, т. к. он не входит в текущий курс. Что касается остального в приведенном запросе, то оно будет разобрано ниже.

Оператор JOIN входит в раздел FROM операторов SELECT, UPDATE или DELETE. Ниже мы рассмотрим использование соединения в операторе SELECT.

Упрощенный синтаксис таких соединений, следующий:

SELECT список\_столбцов

FROM таблица1

 INNER JOIN | LEFT [OUTER] JOIN | RIGHT [OUTER] JOIN | FULL JOIN |
 CROSS JOIN таблица2

 ON таблица1.некий\_столбец=таблица2.некий\_столбец

В SELECT перечисляются необходимые столбцы для вывода в произвольном порядке. Выше приведены следующие типы соединения JOIN:

INNER JOIN – внутреннее соединение,

LEFT [OUTER] JOIN – левое внешнее соединение,

RIGHT [OUTER] JOIN – правое внешнее соединение,

FULL JOIN – полное соединение,

CROSS JOIN – перекрестное соединение.

1.2. Псевдонимы

Псевдонимы PostgreSQL — это инструменты, которые позволяют вам присваивать временные имена таблицам или столбцам в ваших запросах. Эти псевдонимы существуют только во время выполнения запроса, что делает ваш SQL-код более читаемым и эффективным.

Псевдонимы столбцов позволяют вам переименовывать столбцы в результирующем наборе вашего запроса, делая вывод более осмысленным:

SELECT f.scheduled\_departure AS "Время вылета",

 f.scheduled\_arrival AS "Время прилета",

…

или

SELECT f.scheduled\_departure "Время вылета",

 f.scheduled\_arrival "Время прилета",

…

Добавим в приведенный выше пример расчет длительности полета и выполним по ней сортировку:

SELECT …

 f.scheduled\_arrival-f.scheduled\_departure "Время полета"

FROM bookings.flights AS f

 JOIN bookings.airports\_data AS ad ON ad.airport\_code = f.departure\_airport

 JOIN bookings.airports\_data AS aa ON aa.airport\_code = f.arrival\_airport

ORDER BY "Время полета";

Если в качестве псевдонима использовать правила наименования идентификаторов, т.е., например, если использовать буквы и символ подчеркивания (см. документацию), то можно обойтись без кавычек:

SELECT …

 f.scheduled\_arrival-f.scheduled\_departure Время\_полета

FROM bookings.flights AS f

 JOIN bookings.airports\_data AS ad ON ad.airport\_code = f.departure\_airport

 JOIN bookings.airports\_data AS aa ON aa.airport\_code = f.arrival\_airport

ORDER BY Время\_полета;

Псевдонимы таблиц предоставляют способ сократить имена таблиц в ваших запросах, что особенно полезно для таблиц с длинными именами или при соединении нескольких таблиц:

SELECT **f.**scheduled\_departure AS "Время вылета",

 f.scheduled\_arrival AS "Время прилета",

 **ad**.airport\_name::jsonb->>'ru' AS "Аэропорт вылета",

 aa.airport\_name::jsonb->>'ru' AS "Аэропорт прилета",

 f.scheduled\_arrival-f.scheduled\_departure AS Время\_полета

FROM bookings.flights AS f

 JOIN **bookings.airports\_data AS ad** ON **ad.**airport\_code = f.departure\_airport

 JOIN bookings.airports\_data AS aa ON aa.airport\_code = f.arrival\_airport

ORDER BY Время\_полета;

Существует несколько вариантов использования псевдонима таблицы. Некоторые из них перечислены ниже:

* его можно использовать, чтобы сэкономить время нажатия клавиш и сделать ваш запрос более читаемым для таблиц с длинными именами;
* его также можно использовать при запросе данных из нескольких таблиц с одинаковыми именами столбцов;
* его можно использовать для объединения таблицы с самой собой (т.е. самосоединение).

1.3. Внутреннее соединение

С помощью ключевого слова INNER JOIN (или JOIN) задается вторая таблица и условие выбора строк из нее. Если условие выполняется, то формируется результирующий набор путем объединения всех строк из обеих таблиц.

Следующий оператор JOIN (по умолчанию INNER JOIN) объединяет первую таблицу flights со второй таблицей airports\_data, сопоставляя значения в столбцах departure\_airport из flights со airport\_code из airports\_data. departure\_airport является внешним ключом, ссылающимся на первичный ключ airport\_code таблицы airports\_data. Использование в соединении таблиц их первичные и вторичные ключи является распространенной практикой, хотя – необязательной. Такое применение целесообразно потому, что поиск подходящей строки во второй таблице по первичному ключу, будет выполняться быстрее, в отличии от сканирования всей таблицы.

SELECT …

FROM bookings.flights AS f

 **JOIN** bookings.airports\_data AS ad ON ad.airport\_code = f.departure\_airport

…

Внутреннее соединение проверяет каждую строку в первой таблице flights на равенство значения departure\_airport со значением в столбце airport\_code каждой строки во второй таблице airports\_data. Если эти значения равны, внутреннее соединение создает новую строку, содержащую столбцы из обеих таблиц, и добавляет эту новую строку в результирующий набор. Если строка во второй таблице не найдена, то эта строка из первой таблицы игнорируется.

Рассмотрим пример. Некоторый магазин продает товар, который может быть поставлен покупателю по указанному адресу, либо покупатель сам забирает покупку. Создадим две таблицы products и address:

products

(

 id\_product serial,

 name text,

 id\_address integer

)

address

(

 id\_address serial,

 address text

)

Таблица products имеет внешний ключ id\_address, ссылающийся на адрес поставки. Если значение поля id\_address «равно» NULL, то покупатель сам забрал товар.

Заполним эти таблицы:

INSERT INTO address (address)

 VALUES('Адрес 1'),

 ('Адрес 2'),

 ('Адрес 3'),

 ('Адрес 4'),

 ('Адрес 5'),

 ('Адрес 6'),

 ('Адрес 7'),

 ('Адрес 8'),

 ('Адрес 9'),

 ('Адрес 10');

INSERT INTO products (name, id\_address)

 VALUES('товар 1', 3),

 ('товар 2', NULL),

 ('товар 3', 5),

 ('товар 4', NULL),

 ('товар 5', NULL),

 ('товар 6', 9),

 ('товар 7', NULL),

 ('товар 8', NULL),

 ('товар 9', NULL),

 ('товар 10', 1),

 ('товар 11', NULL),

 ('товар 12', NULL),

 ('товар 13', NULL),

 ('товар 14', 2),

 ('товар 15', NULL),

 ('товар 16', NULL),

 ('товар 17', 3),

 ('товар 18', NULL),

 ('товар 19', NULL),

 ('товар 20', 9);

Распечатаем эти товары и адреса поставки:

SELECT p.\*, a.\*

FROM products AS p

 JOIN address AS a ON a.id\_address = p.id\_address;



 Как видим, выведены только те товары, у которых имеются адреса поставки.

 Следующая диаграмма иллюстрирует внутреннее соединение:



1.4. Внешнее левое соединение

Для задания левого внешнего соединения используется ключевое слово предложение LEFT OUTER JOIN (или LEFT JOIN) для соединения первой таблицы products c таблицей address. В контексте LEFT JOIN первая таблица называется левой таблицей, а вторая таблица называется правой таблицей. Выполним предыдущий запрос, поменяв JOIN на LEFT JOIN:

SELECT p.\*, a.\*

FROM products AS p

 LEFT JOIN address AS a ON a.id\_address = p.id\_address



Левое соединение начинает выборку данных из левой таблицы. Оно сравнивает значения в столбце id\_address со значениями в столбце id\_address в таблице address. Если эти значения равны, левое соединение создает новую строку, содержащую столбцы обеих таблиц, и добавляет эту новую строку в набор результатов. В случае, если значения не равны, левое соединение также создает новую строку, содержащую столбцы из обеих таблиц, и добавляет ее в набор результатов. Однако оно заполняет столбцы правой таблицы address значениями null.

Следующая диаграмма иллюстрирует левое соединение:



Чтобы выбрать строки из левой таблицы, которые не имеют соответствующих строк в правой таблице, используйте левое соединение с ключевым словом WHERE. Например:

SELECT p.\*, a.\*

FROM products AS p

 LEFT JOIN address AS a ON a.id\_address = p.id\_address

WHERE a.id\_address IS NULL;



Следующая диаграмма иллюстрирует левое соединение, возвращающее строки из левой таблицы, которым не соответствуют строки из правой таблицы:



1.5. Внешнее правое соединение

Правое соединение — это обратная версия левого соединения. Правое соединение начинает выборку данных из правой таблицы. Оно сравнивает значение в столбце id\_address каждой строки в правой таблице address со значением в столбце id\_address каждой строки в таблице products. Если эти значения равны, правое соединение создает новую строку, содержащую столбцы из обеих таблиц. В случае, если эти значения не равны, правое соединение также создает новую строку, содержащую столбцы из обеих таблиц. Однако оно заполняет столбцы в левой таблице значениями NULL.

Следующий оператор использует правое соединение для соединения таблицы products с таблицей address:

SELECT p.\*, a.\*

FROM products AS p

 RIGHT JOIN address AS a ON a.id\_address = p.id\_address;



Следующая диаграмма иллюстрирует правое соединение:



Аналогично можно получить строки из правой таблицы, которым не соответствуют строки из левой таблицы, добавив WHERE:

SELECT p.\*, a.\*

FROM products AS p

 RIGHT JOIN address AS a ON a.id\_address = p.id\_address

WHERE p.id\_address IS NULL;



Следующая диаграмма иллюстрирует правое соединение, которое возвращает строки из правой таблицы, которым не соответствуют строки в левой таблице:



1.6. Полное внешнее соединение

Полное внешнее соединение FULL OUTER JOIN (или FULL JOIN) возвращает результирующий набор, содержащий все строки из левой и правой таблиц, с соответствующими строками с обеих сторон, если они имеются. В случае отсутствия совпадений столбцы таблицы будут заполнены NULL:

SELECT p.\*, a.\*

FROM products AS p

 FULL JOIN address AS a ON a.id\_address = p.id\_address;



Следующая диаграмма иллюстрирует полное внешнее соединение:



Чтобы вернуть строки в таблице, которым не соответствуют строки в другой таблице, используйте полное соединение с WHERE:

SELECT p.\*, a.\*

FROM products AS p

 FULL JOIN address AS a ON a.id\_address = p.id\_address

WHERE p.id\_address IS NULL OR a.id\_address IS NULL;



Следующая диаграмма иллюстрирует полное внешнее соединение, возвращающее строки из таблицы, которым не соответствуют строки в другой таблице:



1.7. Перекрестное соединение

В PostgreSQL перекрестное соединение позволяет объединить две таблицы, объединив каждую строку из первой таблицы с каждой строкой из второй таблицы, что приводит к полному объединению всех строк. Перекрестное соединение производит декартово произведение строк двух таблиц.

CROSS JOIN не выполняет соединения строк по условию, как в LEFT JOIN или INNER JOIN. Перекрестное соединение объединяет две таблицы, объединив каждую строку из первой таблицы с каждой строкой из второй таблицы, что приводит к полному объединению всех строк.

Предположим, вам нужно выполнить CROSS JOIN из table1 и table2.

Если table1 имеет n строк и table2 имеет m строк, то CROSS JOIN вернет результирующий набор, содержащий (n x m) строк.

Поскольку CROSS JOIN может генерировать большой набор результатов, следует использовать его осторожно, чтобы избежать проблем с производительностью.

 Применим CROSS JOIN к нашим таблицам:

SELECT p.\*, a.\*

FROM products AS p

 CROSS JOIN address AS a;



Перекрестное соединение эквивалентно приведенному ниже утверждению:

SELECT p.\*, a.\*

FROM products AS p,address AS a;

1.8. Естественное соединение

Предложение ON является наиболее общим видом условия соединения: оно принимает логическое значение выражения того же типа, что используется в WHERE предложении. Пара строк из таблица1 и таблица2совпадают, если выражение в ON является истинным.

В Postgres можно использовать сокращение USING, которое позволяет воспользоваться конкретной ситуацией, когда объединение таблиц выполняется на основе равенства столбцов, имеющих одно и тоже имя. Например, приведенные ниже команды эквивалентны:

SELECT p.\*, a.\*

FROM products AS p

 JOIN address AS a USING(id\_address);

SELECT p.\*, a.\*

FROM products AS p

 JOIN address AS a ON a.id\_address = p.id\_address;

В ключевом слове USING можно указать несколько столбцов, например:

SELECT список\_столбцов

FROM таблица1

 [INNER, LEFT, RIGHT] JOIN таблица2 USING(столбец1, столбец2);

Соединение будет выполняться при условии:

SELECT список\_столбцов

FROM таблица1

 [INNER, LEFT, RIGHT] JOIN таблица2

 ON таблица1.столбец1 = таблица2.столбец1

 AND таблица1.столбец2 = таблица2.столбец2;

## 2. Оператор объединения UNION

Оператор UNION позволяет объединить наборы результатов двух или более операторов SELECT в один набор результатов.

Вот основной синтаксис оператора UNION:

SELECT список\_столбцов

FROM таблица1

UNION [ALL]

SELECT список\_столбцов

FROM таблица2;

Оператор UNION удаляет все дубликаты строк из объединенного набора данных. Чтобы сохранить дубликаты строк, используйте UNION ALL.

В этом синтаксисе запросы должны соответствовать следующим правилам:

* список\_столбцов в двух операторах SELECT должны иметь одинаковое количество столбцов,
* типы данных в соответствующих столбцах должны быть совместимыми.

Если столбцы имеют разные имена, то результирующему столбцу присваивается имя столбца из первого запроса.

Операторы UNION и UNION ALL не упорядочивают строки в конечном наборе. Чтобы отсортировать строки в конечном результирующем наборе, укажите ORDER BY после второго запроса:

SELECT \*

FROM products

WHERE id\_address < 7

UNION

SELECT \*

FROM products

WHERE 4 < id\_address

ORDER BY id\_address



Выполним те же запросы с использованием оператора UNION ALL:

SELECT \*

FROM products

WHERE id\_address < 7

UNION ALL

SELECT \*

FROM products

WHERE 4 < id\_address

ORDER BY id\_address;



 Как можем видеть появились две повторяющиеся строки – 7 и 9.

## 3. Оператор пересечения INTERSECT

Подобно операторам UNION, оператор INTERSECT объединяет результаты двух операторов SELECT в один результирующий набор, содержащий строки, доступные в обоих наборах (пересечение).

Вот основной синтаксис оператора INTERSECT:

SELECT список\_столбцов

FROM таблица1

INTERSECT

SELECT список\_столбцов

FROM таблица2;

Объединяемые запросы должны быть совместимы между собой (см. оператор UNION).

Выполним предыдущий запрос с применением оператора INTERSECT:

SELECT \*

FROM products

WHERE id\_address < 7

INTERSECT

SELECT \*

FROM products

WHERE 4 < id\_address

ORDER BY id\_address;



## 4. Оператор вычитания EXCEPT

Оператор EXCEPT возвращает строки из первого запроса, которых нет во втором запросе (вычитание).

Вот основной синтаксис оператора EXCEPT:

SELECT список\_столбцов

FROM таблица1

EXCEPT

SELECT список\_столбцов

FROM таблица2;

Запросы должны быть совместимы между собой (см. оператор UNION).

Выполним предыдущий запрос с применением оператора EXCEPT:

SELECT \*

FROM products

WHERE id\_address < 7

EXCEPT

SELECT \*

FROM products

WHERE 4 < id\_address

ORDER BY id\_address;



## 5. Контрольные вопросы

1. Что такое JOIN, какие существуют типы JOIN?
2. Что такое псевдонимы и когда их целесообразно использовать?
3. Что делают операторы UNION, INTERSECT и EXCEPT?

## 6. Практическое задание

Написать три запроса с использованием JOIN, LEFT JOIN и USING предметной области.

Возможные запросы на примере библиотеки:

* вывести все книги читателей, которые их брали и даты выдачи и возврата;
* вывести читателей, у которых на руках книги;
* вывести 10 самых активных читателей.

## 7. Список использованных источников

1. Postgres Pro Standard: Документация: <https://postgrespro.ru/docs/postgrespro>