**Тема №4 - Стандартные соединения и объединения таблиц.**

PostgreSQL оператор **UNION** используется для объединения результирующих наборов из 2 или более операторов SELECT. Он удаляет повторяющиеся строки между различными операторами SELECT.

Каждый оператор SELECT в операторе UNION должен иметь одинаковое количество полей в наборах результатов с одинаковыми типами данных.

Общий синтаксис для оператора UNION в PostgreSQL выглядит следующим образом:

SELECT expression1, expression2,… expression\_n

FROM tables

[WHERE conditions]

UNION

SELECT expression1, expression2,… expression\_n

FROM tables

[WHERE conditions];

Где:

**expression1, expression2,… expression\_n –** это столбцы или вычисления, которые вы хотите получить.

**tables –** это таблицы, из которых вы хотите получить записи. В операторе FROM должна быть указана хотя бы одна таблица.

**WHERE conditions –** это условия, которые должны быть выполнены для записей, которые будут выбраны (они являются необязательными).

**Например,** для того чтобы получить в одной таблице ширину и высоту предметов на складе, цена которых не является нулевой, можно использовать следующий запрос:

SELECT "WIDTH","LENGTH"

FROM cloth

WHERE "PRICE" IS NOT NULL

UNION

SELECT "WIDTH","LENGTH"

FROM product

WHERE "PRICE" IS NOT NULL

При работе с оператором UNION имеются примечания:

1. В обоих операторах SELECT должно быть одинаковое количество выражений.
2. Поскольку оператор UNION по умолчанию удаляет все повторяющиеся строки из набора результатов, предоставление модификатора UNION DISTINCT не влияет на результаты.
3. Имена столбцов из первого оператора SELECT в операторе UNION используются в качестве имен столбцов для набора результатов.

Через оператор UNION можно вернуть данные из полей с одним и тем же названием, но из разных таблиц (если оба поля имеют одинаковый тип данных).

**Например**, выведем все цены из двух разных таблиц в одном запросе:

SELECT "PRICE"

FROM cloth

UNION

SELECT "PRICE"

FROM product

В рамках данных таблиц, мы получили в ответ 222 строки с данными, выглядит неплохо. В этом примере оператора PostgreSQL UNION, если “PRICE” появилось как в таблице products, так и в categories, оно будет отображаться в вашем результирующем наборе один раз. Оператор PostgreSQL UNION удаляет дубликаты. Если вы не хотите удалить дубликаты, попробуйте использовать PostgreSQL оператор **UNION ALL**. Вот так:

SELECT "PRICE"

FROM cloth

UNION **ALL**

SELECT "PRICE"

FROM product

Теперь мы получаем на выходе все строки, в рамках данных таблиц их количество равно 1439. UNION ALL работает быстрее, чем UNION, т.к. отсутствует принудительная сортировка для устранения дубликатов.

Оператор PostgreSQL UNION может использовать оператор **ORDER BY** для упорядочивания результатов запроса. Например, если нам нужно отсортировать данные второго столбца (в нашем случае это столбец “LENGTH”), то в ORDER BY запишем цифру 2:

SELECT "WIDTH","LENGTH"

FROM cloth

WHERE "PRICE" IS NOT NULL

UNION

SELECT "WIDTH","LENGTH"

FROM product

WHERE "PRICE" IS NOT NULL

**ORDER BY** 2

Если нужно отсортировать по первому столбцу, запишем:

SELECT "WIDTH","LENGTH"

FROM cloth

WHERE "PRICE" IS NOT NULL

UNION

SELECT "WIDTH","LENGTH"

FROM product

WHERE "PRICE" IS NOT NULL

**ORDER BY** 1

Поскольку в SELECT мы взяли 2 столбца, если запишем после ORDER BY 0 или 3 и более, то в ответ мы получим ошибку.

Через UNION можно сочетать между собой бесконечное количество запросов. Операции над множествами тоже можно вкладывать и соединять, например:

*запрос1* UNION *запрос2* UNION *запрос3*

Такие сложные запросы выполняются следующим образом:

(*запрос1* UNION *запрос2*) UNION *запрос3*

**JOIN** — оператор языка SQL, позволяющий объединять записи из двух или более таблиц базы данных. Входит в оператор FROM и отдельно от него не используется.

**INNER JOIN –** внутреннее соединение таблиц.

Его формальный синтаксис:

|  |  |
| --- | --- |
|  | SELECT столбцы  FROM таблица1      [INNER] JOIN таблица2      ON условие1      [[INNER] JOIN таблица3      ON условие2] |

Ключевое слово INNER в запросе может быть опущено, так как эта опция в операторе JOIN действует по умолчанию. Далее после ключевого слова ON мы указываем условие соединения. Как правило, для соединения применяется первичный ключ главной таблицы и внешний ключ зависимой таблицы.

Рассмотрим пример. Допустим мы хотим, используя JOIN добавить к информации о заказах имя и фамилию пользователя сделавшего заказ.

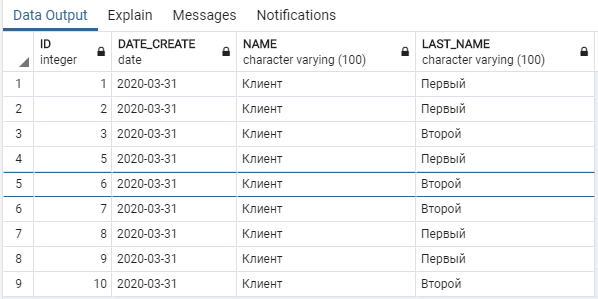
SELECT "order"."ID", "order"."DATE\_CREATE",

"users"."NAME", "users"."LAST\_NAME"

FROM "order"

JOIN "users" ON "users"."ID" = "order"."CLIENT";

В результате получаем следующую таблицу



Также условия после ключевого слова ON могут быть и более сложными. Например мы хотим вывести только те заказы с именами заказчиков, сумма которых более 1000$.

SELECT "order"."ID", "order"."DATE\_CREATE",

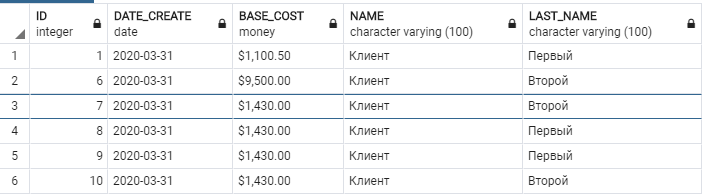
"order"."BASE\_COST", "users"."NAME", "users"."LAST\_NAME"

FROM "order"

JOIN "users" ON "users"."ID" = "order"."CLIENT"

AND "order"."BASE\_COST" > '1000$';

В результате получаем следующую таблицу



**OUTER JOIN –** внешнее соединение таблиц. Позволяет возвратить все строки одной или двух таблиц, которые участвуют в соединении.

Outer Join имеет следующий формальный синтаксис:

|  |  |
| --- | --- |
|  | SELECT столбцы  FROM таблица1      {LEFT|RIGHT|FULL} [OUTER] JOIN таблица2 ON условие1      [{LEFT|RIGHT|FULL} [OUTER] JOIN таблица3 ON условие2]... |

Перед оператором **JOIN** указывается одно из ключевых слов **LEFT**, **RIGHT** или **FULL**, которые определяют тип соединения:

**LEFT**: выборка будет содержать все строки из первой или левой таблицы

**RIGHT**: выборка будет содержать все строки из второй или правой таблицы

**FULL**: выборка будет содержать все строки из обеих таблиц

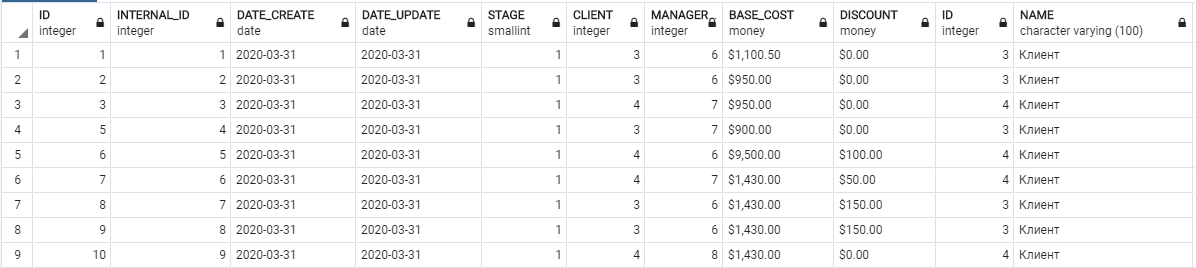
Перед оператором **JOIN** может указываться ключевое слово **OUTER**, но его применение необязательно. После **JOIN** указывается присоединяемая таблица, а затем идет условие соединения после оператора **ON**.

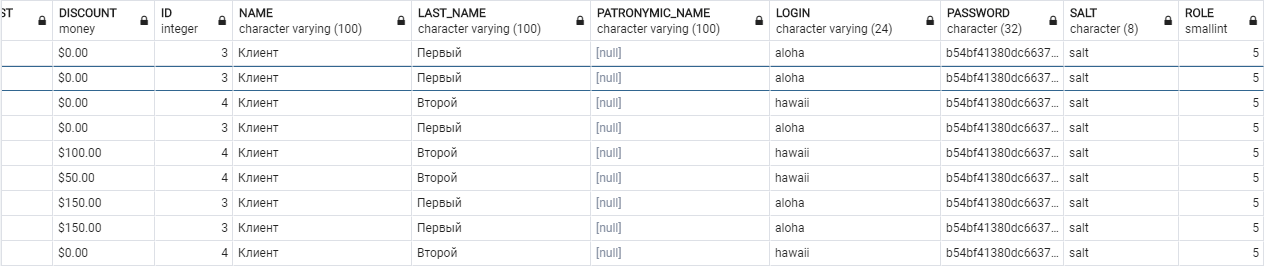
SELECT \*

FROM "order" LEFT JOIN "users"

ON "order"."CLIENT" = "users"."ID";

В результате мы получим таблицу в которой к каждому заказцу добиться полная информация о пользователе который сделал данный заказ. При этом так как таблица “order” была левой, т.е. первой таблицей, то сначала выведутся все строки из “order”, а уже затем все строки из правой, т.е. второй таблицы. Таким образом мы получаем следующий результат.





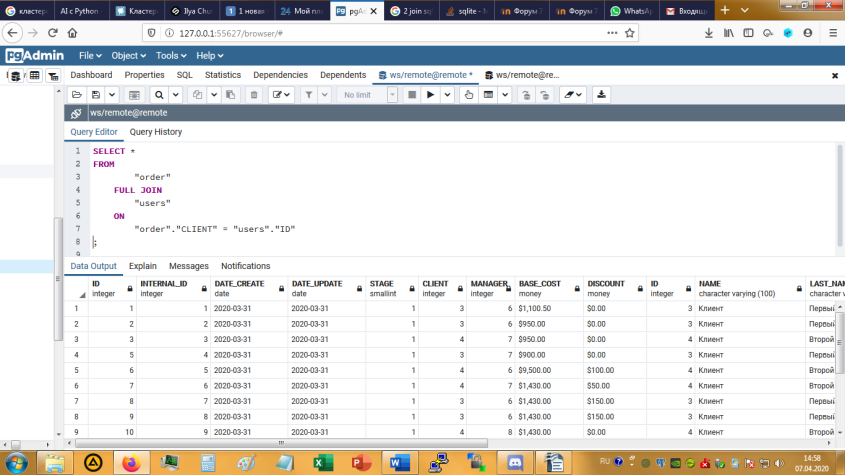
По вышеприведенному результату может показаться, что левостороннее соединение аналогично INNER Join, но это не так. Inner Join объединяет строки из дух таблиц при соответствии условию. Если одна из таблиц содержит строки, которые не соответствуют этому условию, то данные строки не включаются в выходную выборку. Left Join выбирает все строки первой таблицы и затем присоединяет к ним строки правой таблицы.

Теперь выполним запрос с Full Join

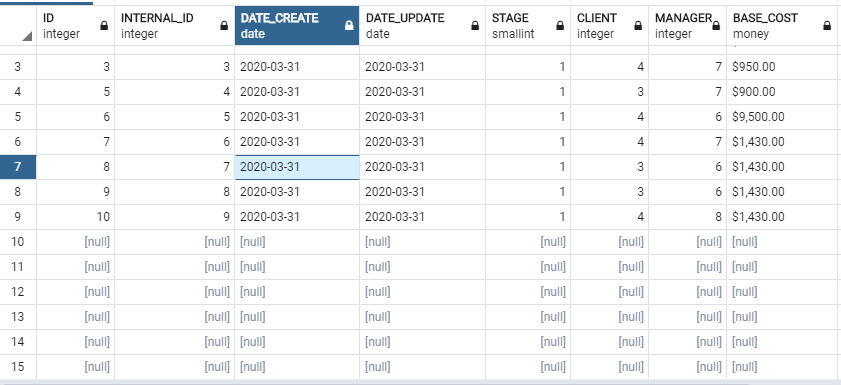
SELECT \*

FROM "order" FULL JOIN "users"

ON "order"."CLIENT" = "users"."ID";



В данном случае мы получим ту же таблицу что и в первом случае, но сюда уже будут включены пользователи без заказов, для которых соответствующие поля будут иметь значение NULL.



**Cross Join** или перекрестное соединение создает набор строк, где каждая строка из одной таблицы соединяется с каждой строкой из второй таблицы. Например, соединим таблицу заказов **order** и таблицу пользователей **users**:

SELECT \* FROM "order" CROSS JOIN "users"

Если в таблице **order** 3 строки, а в таблице **users** то же три строки, то в результате перекрестного соединения создается 3 \* 3 = 9 строк вне зависимости, связаны ли данные строки или нет.

При неявном перекрестном соединении можно опустить оператор CROSS JOIN и просто перечислить все получаемые таблицы.

SELECT \* FROM "order", "users"

## Декартово произведение

Давайте сначала вспомним определение декартова произведения. Им называют множество, элементами которого являются все возможные упорядоченные пары элементов исходных множеств. Для чего это может быть нужно в базах данных? Да для того же, что и в других областях использования понятий множества – для получения всех возможных комбинаций элементов некоторых множеств.

Для примера выполним декартово произведение таблиц цветов и рисунков, получив их возможные комбинации. Запрос и результат показаны на рисунке ааа.

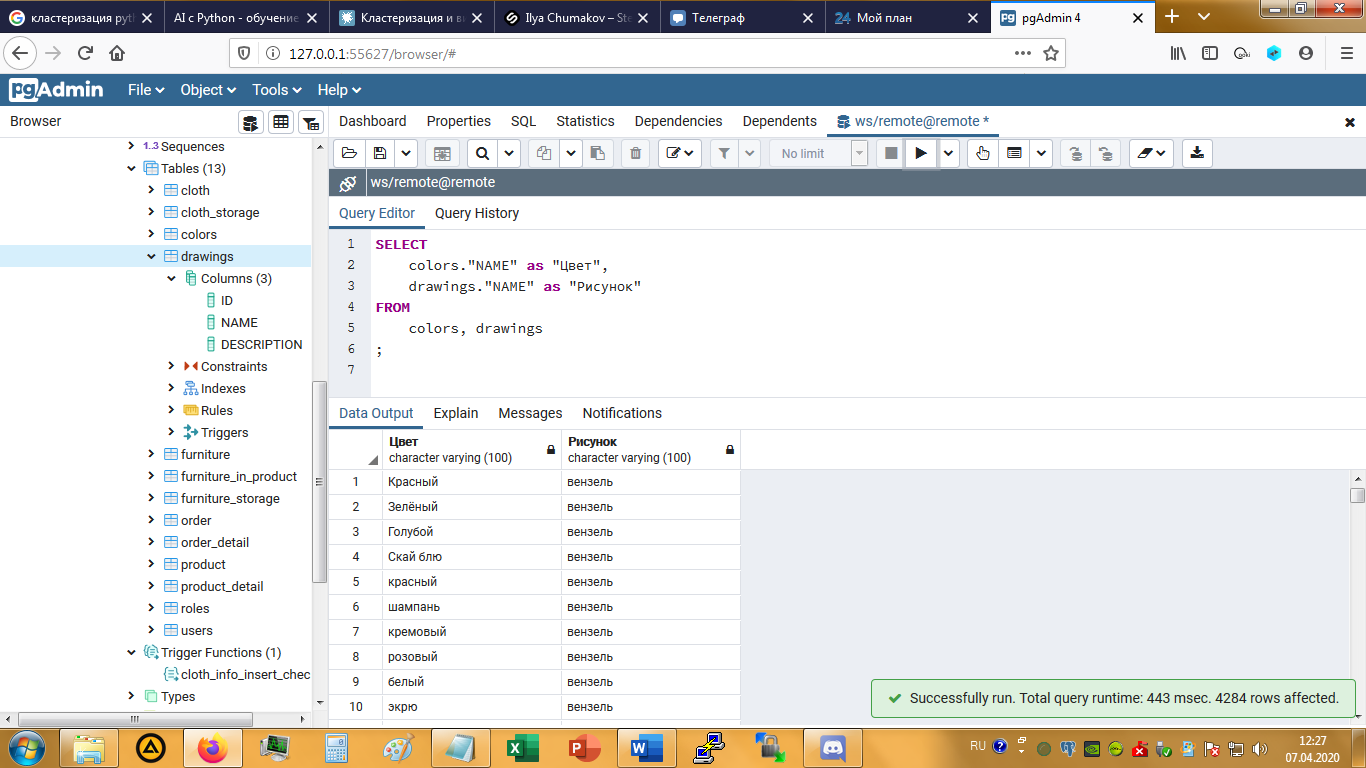


Рисунок ааа – Декартово произведение цветов и рисунков

Как известно, множества не обязательно должны быть различны. Поэтому мы можем выполнить произведение таблицы самой на себя, что и показано на рисунке ввв.

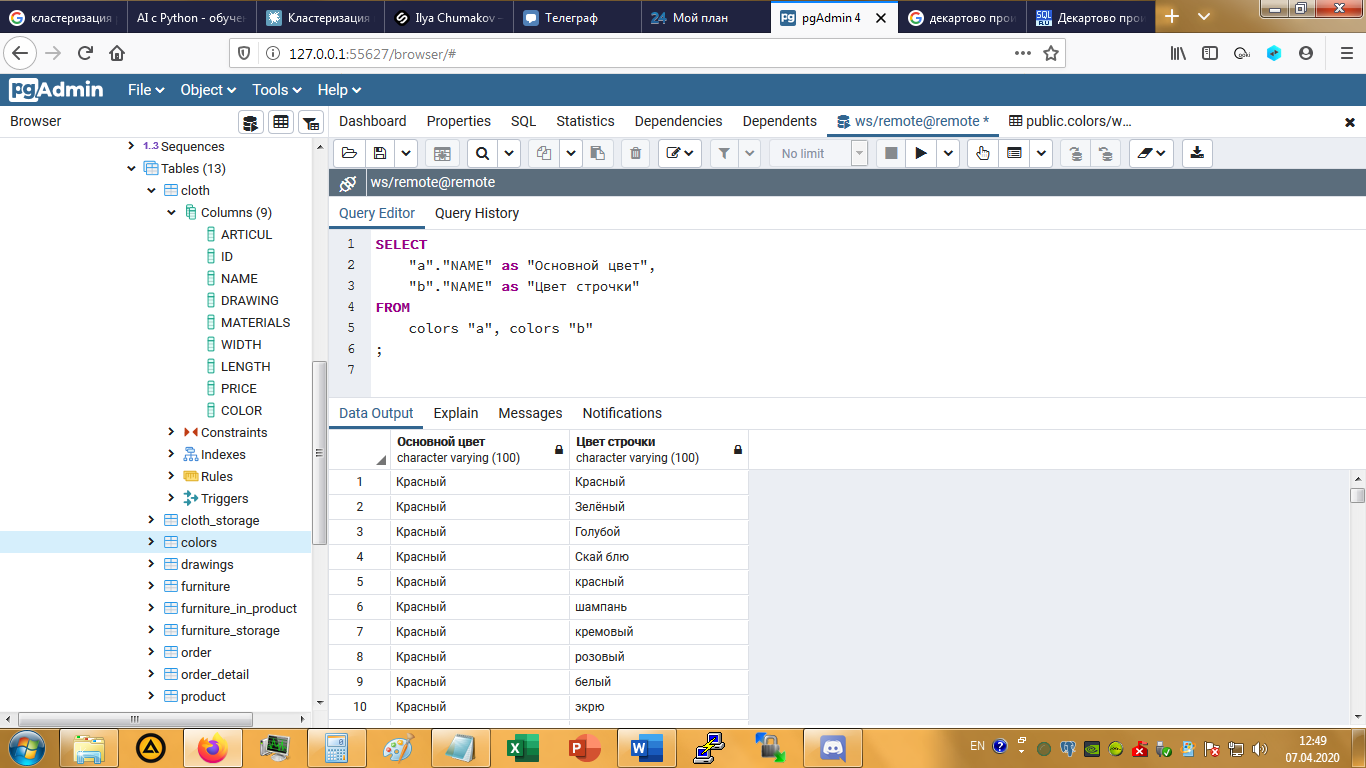


Рисунок ввв – Декартово произведение таблицы самой на себя

Подводя итог, декартово произведение таблиц – это набор всевозможных комбинаций строк из этих таблиц. Обратите внимание, что ***количество строк в результате будет равно произведению записей во всех таблицах***. В примере выше информация о цветах изделий представлена в количестве 119 записей - 119\*119=14161 строка присутствовала в выводе. Поэтому такая операция, как правило, не используется в чистом виде. Она подходит для фильтрации в подзапросах. Кроме того, рекомендуется при возможности убирать нерелевантные значения перед выборкой.

Пример объединения CROSS JOIN для базы данных рассмотренной в теме №1.

Use [univer]

Select [наименование университета], [фио студента]

From университет

Cross join студент;

## Эквисоединение

Эквисоединением называют соединения по предикату (эквивалентные). Для получения эквисоединения таблиц необходимо для декартова произведения таблиц установить имеющее смысл соответствие на основе равенства между столбцами соединяемых таблиц.

**Например,** запрос на неявное эквисоединение таблиц заказов и пользователей будет выглядеть как на рисунке ууу.

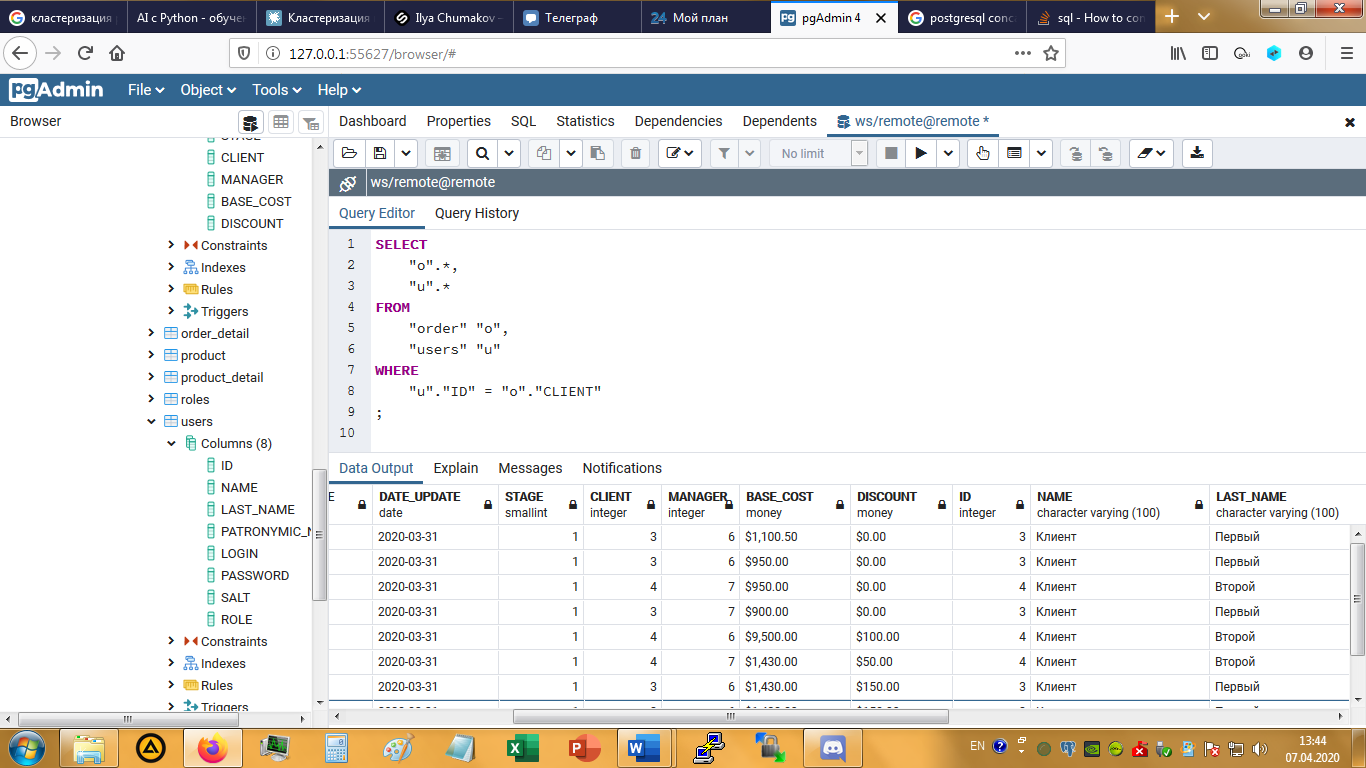


Рисунок ууу **–** Эквисоединение таблиц заказов и пользователей

Такой же результат может быть получен, если использовать запрос на явное соединение, приведенный на рисунке ннн.

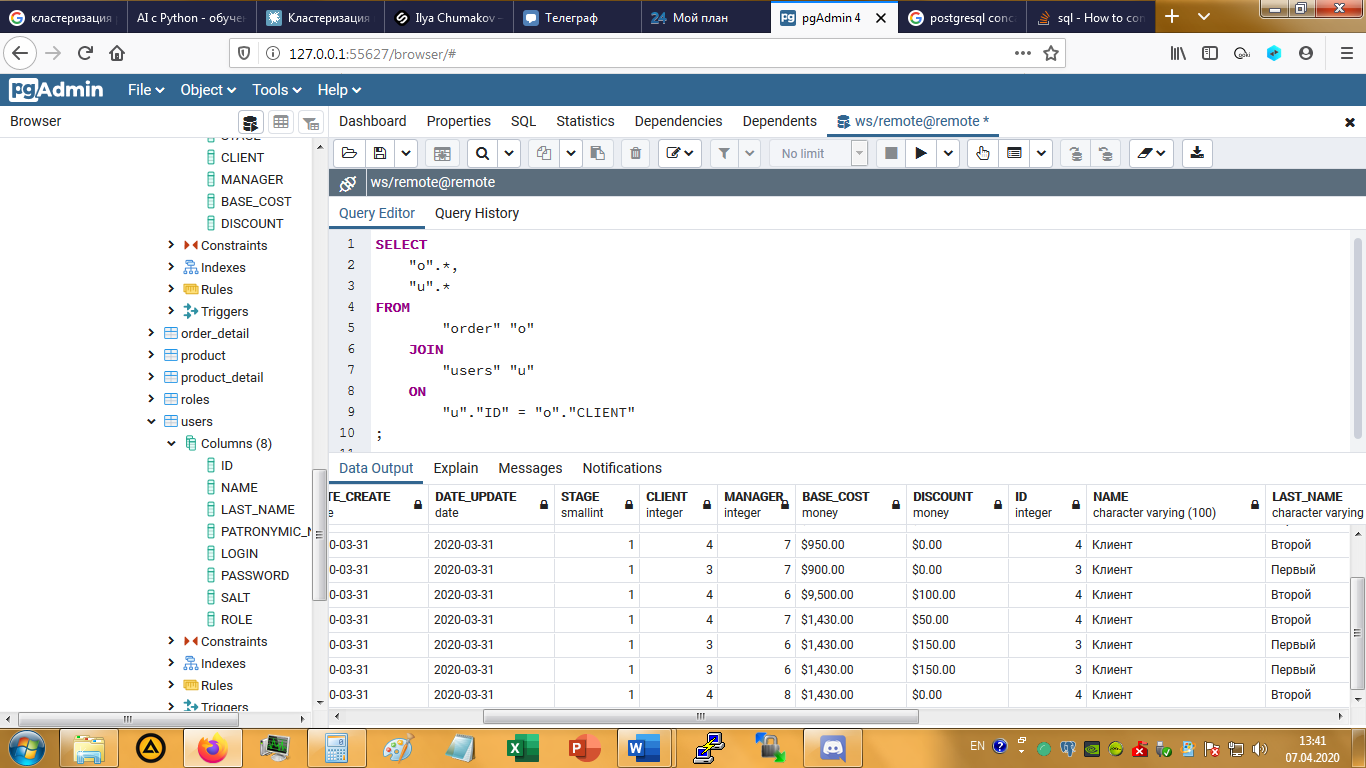
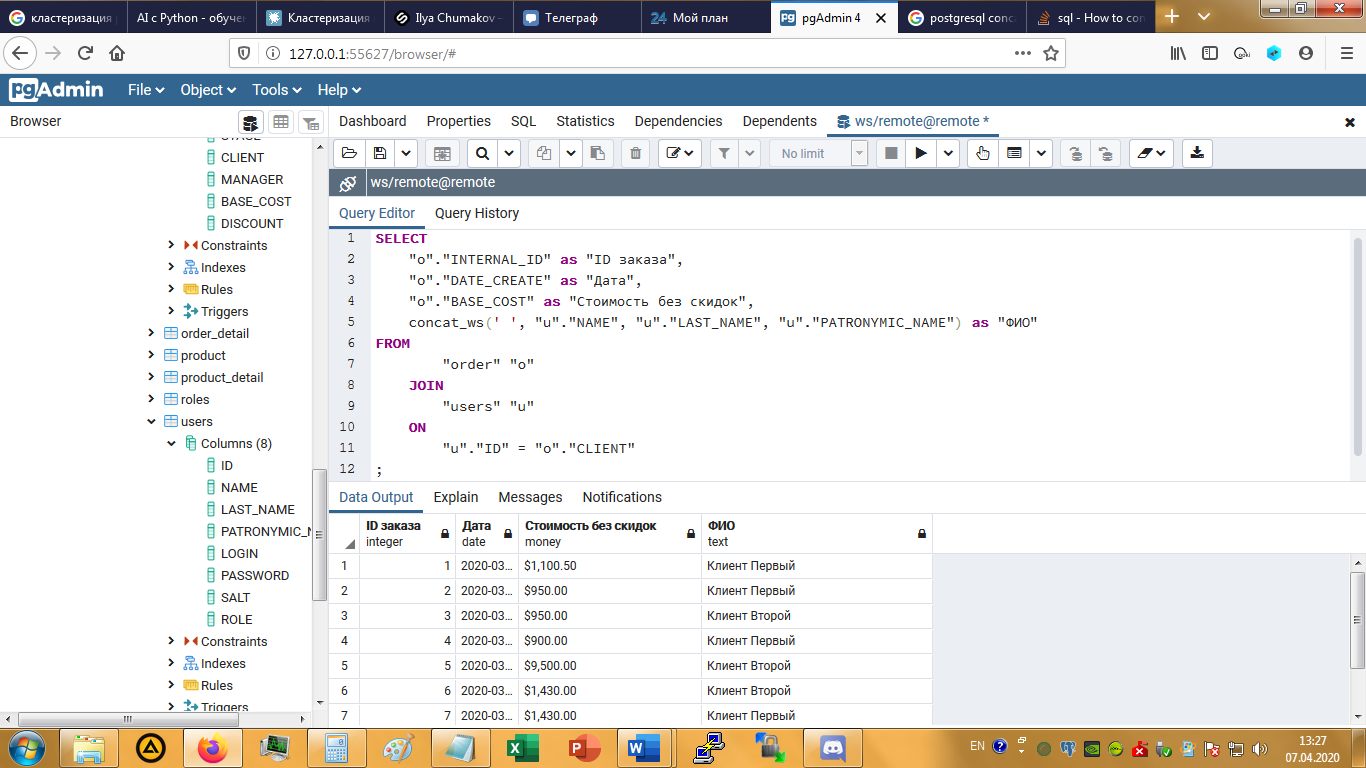


Рисунок ннн – Результат выполнения явного соединения

**Естественное соединение таблиц**

Для получения естественного соединения таблиц необходимо в эквисоединении таблиц исключить дубликаты повторяющихся столбцов (столбцов, входящих в условие соединения). Для предыдущего примера естественное соединение таблиц заказов и пользователей по столбцу USER\_ID (CLIENT) выглядит следующим образом (рисунок ццц):



Рисунокццц **-** Естественное соединение таблиц заказов и пользователей

Предложение NATURAL JOIN соединяет таблицы по колонкам с одинаковыми именами. В данном случае, мы не можем им воспользоваться, так как в этих таблицах колонка ID относится к разным сущностям.

**Композиция:**

Для создания композиции таблиц нужно исключить из вывода все столбцы, по которым проводилось соединение таблиц, например, следующим образом (рисунок ААА ПОМОГИТЕ):

SELECT product."WIDTH", product."LENGTH"

FROM cloth, product

WHERE cloth."PRICE" = product."PRICE"

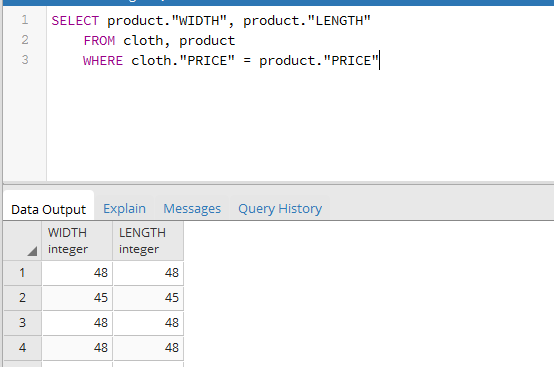


Рисунок ААА ПОМОГИТЕ – Результат выполнения композиции

**Тета-соединение:**

Тета-соединение предназначено для тех случаев, когда необходимо соединить две таблицы на основе некоторых условий, отличных от равенства. Например, получить тета – соединение таблиц cloth и product можно следующим образом (рисунок JOJO):

SELECT product.\*, product.\*

FROM cloth, product

WHERE cloth."PRICE" > product."PRICE"

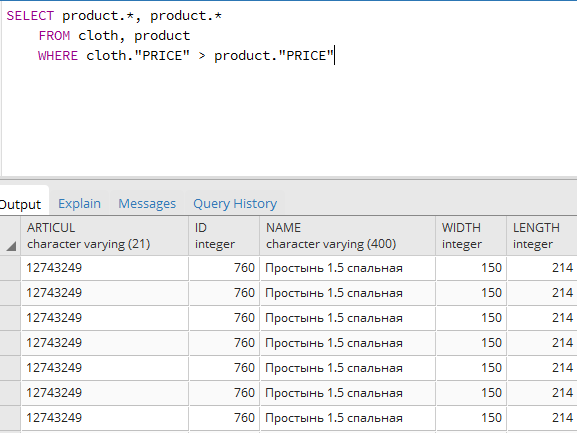


Рисунок JOJO – Результат выполнения Тета-соединениея

**Соединение таблицы со своей копией:**

В ряде приложений возникает необходимость одновременной обработки данных какой-либо таблицы и одной или нескольких ее копий, создаваемых на время выполнения запроса.

Например, при создании списков студентов (таблица Студенты) возможен повторный ввод данных о каком-либо студенте с присвоением ему второго номера зачетной книжки. Для выявления таких ошибок можно соединить таблицу Студенты с ее временной копией, установив в WHERE фразе равенство значений всех одноименных столбцов этих таблиц кроме столбцов с номером зачетной книжки (для последних надо установить условие неравенства значений).

Временную копию таблицы можно сформировать, указав имя псевдонима за именем таблицы во фразе FROM. Так, с помощью фразы FROM Товара X, Товара Y, Товара Z будут сформированы три копии таблицы Товары с именами X, Y и Z., например, чтобы найти все пары абонентов, проживающих на одной и той же улице, можно использовать следующее неявное соединение таблицы product со своей копией:

SELECT F."NAME", S."NAME"

FROM colors F, colors S

WHERE F."NAME" < S."NAME"

В этом примере для таблицы colors определены два псевдонима: F (First) и S (Second). Эти псевдонимы будут существовать, пока выполняется запрос. Дополнительное условие поиска F."NAME" < S."NAME" предназначено для удаления из ТРЗ повторяющихся строк, появляющихся в результате того, что запрос выбирает все комбинации строк с одинаковым именем.

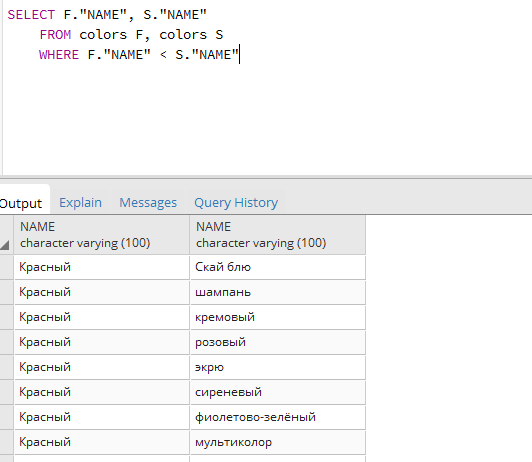


Рисунок АЙ - Результат выполнения неявного соединения

таблицы colors со своей копией

Такой же результат может быть получен, если использовать следующий запрос на явное соединение:

SELECT F."NAME", S."NAME"

FROM colors F **JOIN** colors S **ON** F."NAME" < S."NAME"

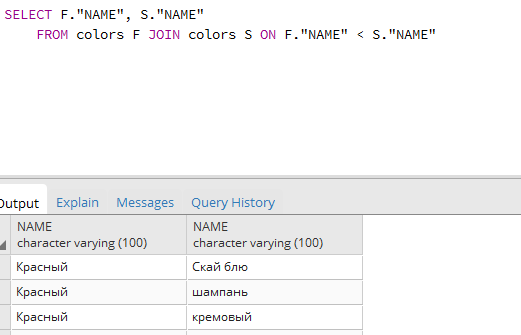


Рисунок ВЫРУБАЙ - Результат выполнения явного соединения