**Тема №1 - Язык SQL. Проектирование структуры базы данных**

## Основные теоретические сведения

Язык SQL (Structured Query Language - структурированный язык запросов) представляет собой стандартный высокоуровневый язык описания данных и манипулирования ими в системах управления базами данных (СУБД), построенных на основе реляционной модели данных. SQL относится к классу непроцедурных языков программирования. Программа на языке SQL представляет собой простую линейную последовательность операторов языка SQL. Язык SQL в своем чистом виде операторов управления порядком выполнения запросов к БД (типа циклов, ветвлений, переходов) не имеет.

Операторы языка SQL строятся с применением:

* зарезервированных ключевых слов;
* идентификаторов таблиц и столбцов таблиц;
* логических, арифметических и строковых выражений, используемых для формирования критериев поиска информации в БД и для вычисления значений ячеек результирующих таблиц;
* идентификаторов (имен) операций и функций, используемых в выражениях.

В языке SQL не делается различия между прописными и строчными буквами, т.е., например, строки SELECT, Select, select представляют собой одно и то же ключевое слово. Для конструирования имен таблиц и их столбцов допустимо использовать буквы, цифры и знак подчеркивание, но первым символом имени обязательно должна быть буква. Запрещено использование ключевых слов и имен функций в качестве идентификаторов таблиц и имен столбцов. Оператор начинается с ключевого слова, например CREATE - создать, UPDATE - обновить, SELECT - выбрать, и заканчивается знаком точка с запятой. Оператор записывается в свободном формате и может занимать несколько строк. Допустимыми разделителями лексических единиц в операторе являются:

* один или несколько пробелов,
* один или несколько символов табуляции,
* один или несколько символов типа новая строка.

В приводимых в пособии примерах построения SQL-запросов используется база данных в основном одна и та же база данных «Университет». В ряде отдельных случаев запросы в целях наглядности формируются для таблиц, не входящих данную структуру. В этом случае перед выполнением запроса приводится таблица с исходной информацией.

## Создание базы данных

Итак, спроектируем структуру базы данных «Университет». Для этого войдём в панель управления СУБД pgAdmin. Вводим пароль от ПУ (не от сервера БД) и раскрываем вкладку «Серверы», её расположение изображено на рисунке 1.

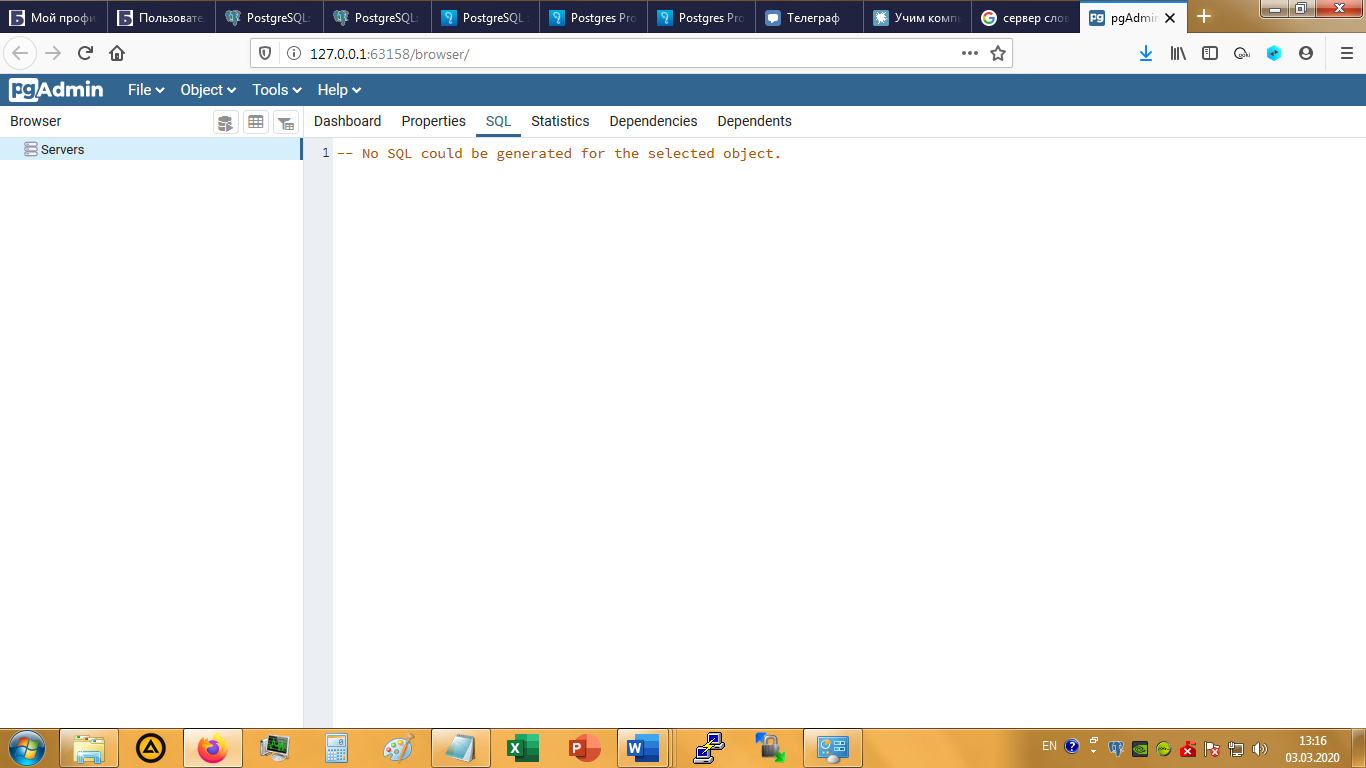


Рисунок 1 – Элемент «Серверы БД»

Кликаем правой кнопкой мыши и выбираем пункт «Создать −> Сервер» (рисунок 2).

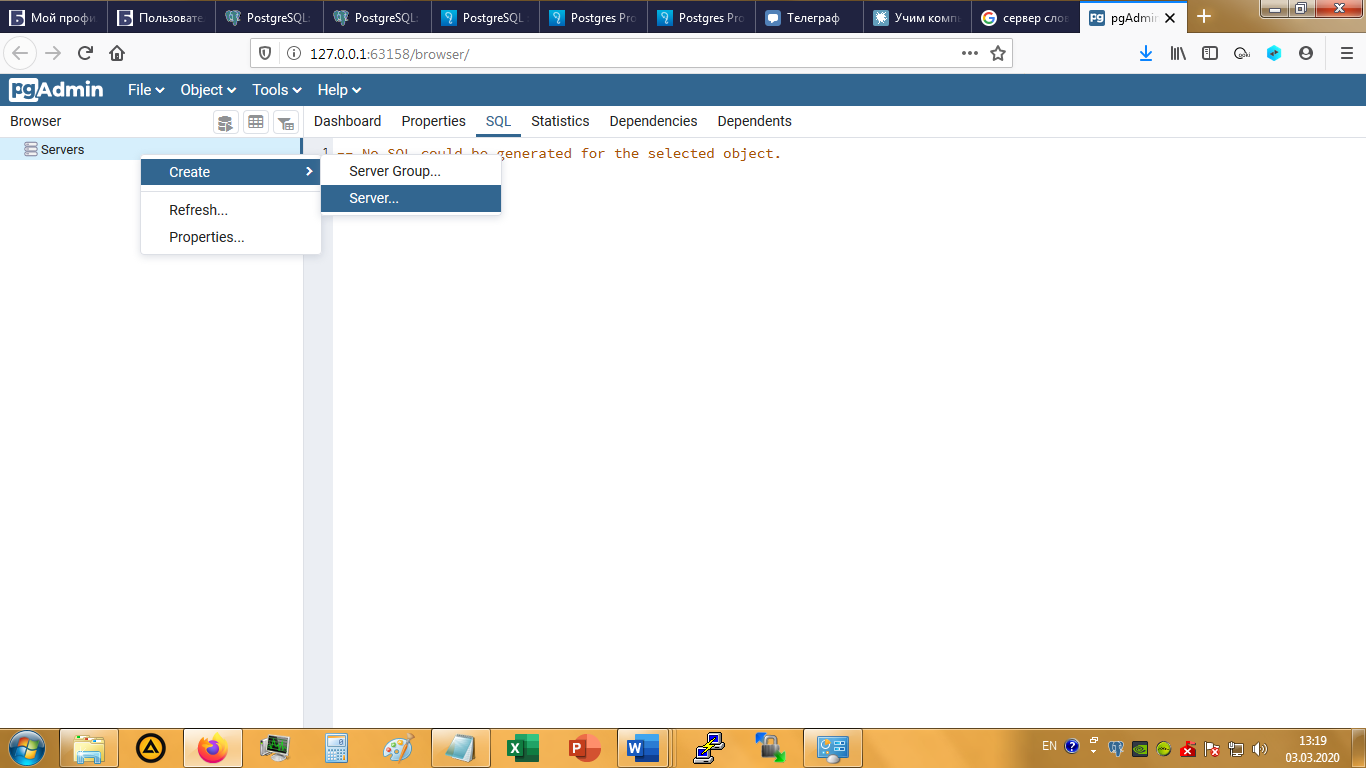


Рисунок 2 – Добавляем сервер БД

В открывшемся окне задаём желаемое имя, затем на вкладке «Подключение» задаём необходимые параметры:

• Хост: localhost;

• Порт, если не задан по умолчанию, 5432;

• Данные пользователя в соответствие с теми, что были заданы при установке

Этого будет достаточно, чтобы подключить наш сервер БД к pgAdmin. В случае успеха на вкладке серверов появится элемент, содержащий одну базу postgres, как показано на рисунке 3.

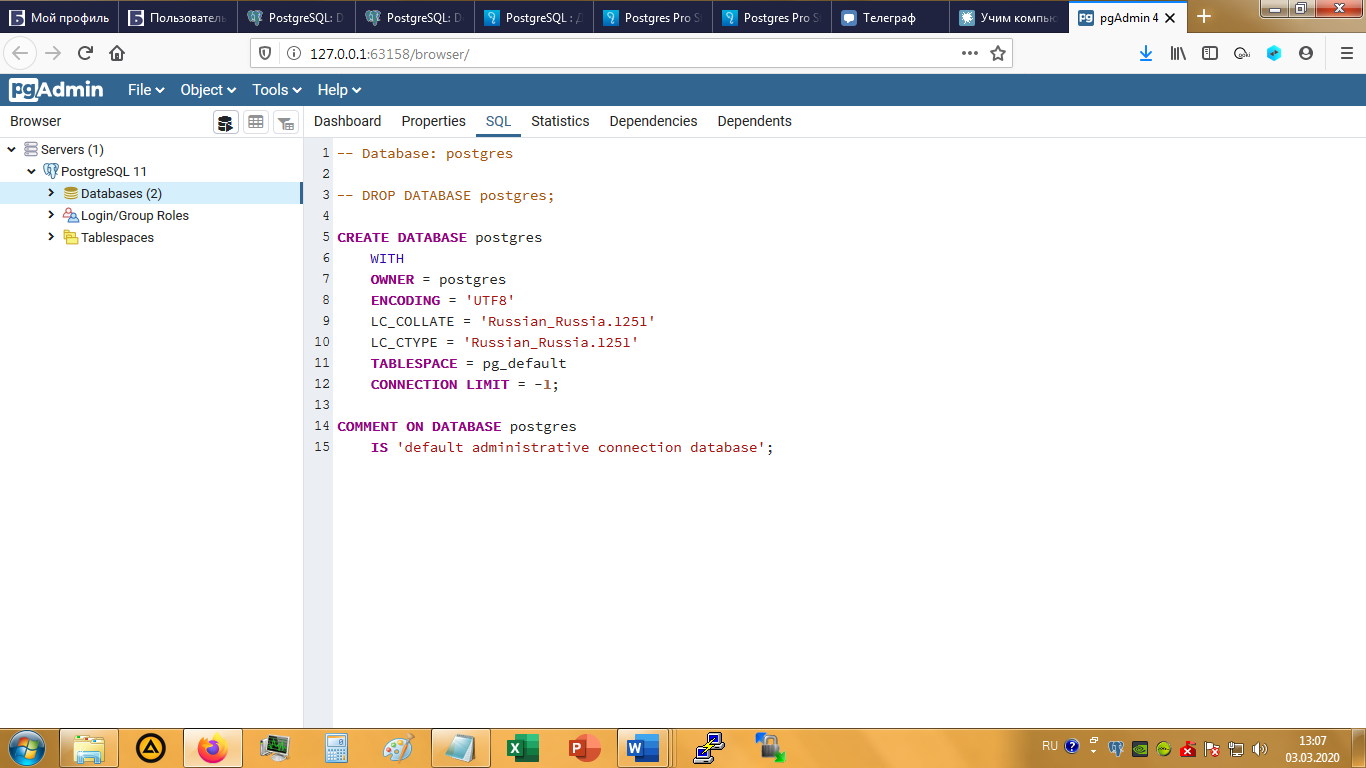


Рисунок 3 – Успешное подключение БД

Когда сервер БД подключен, мы можем создать новую базу данных. Для этого нужно кликнуть на элемент «Базы данных» и выбрать «Создать −> Базу данных». В открывшемся окне задаём имя в нижнем регистре, (аналогично для таблиц, иначе возможны ошибки с именами при работе в psql) владельца, комментарий, кодировку и прочие параметры по мере необходимости. Когда всё готово, нажмите «Сохранить». Все шаги проиллюстрированы на рисунке 4.

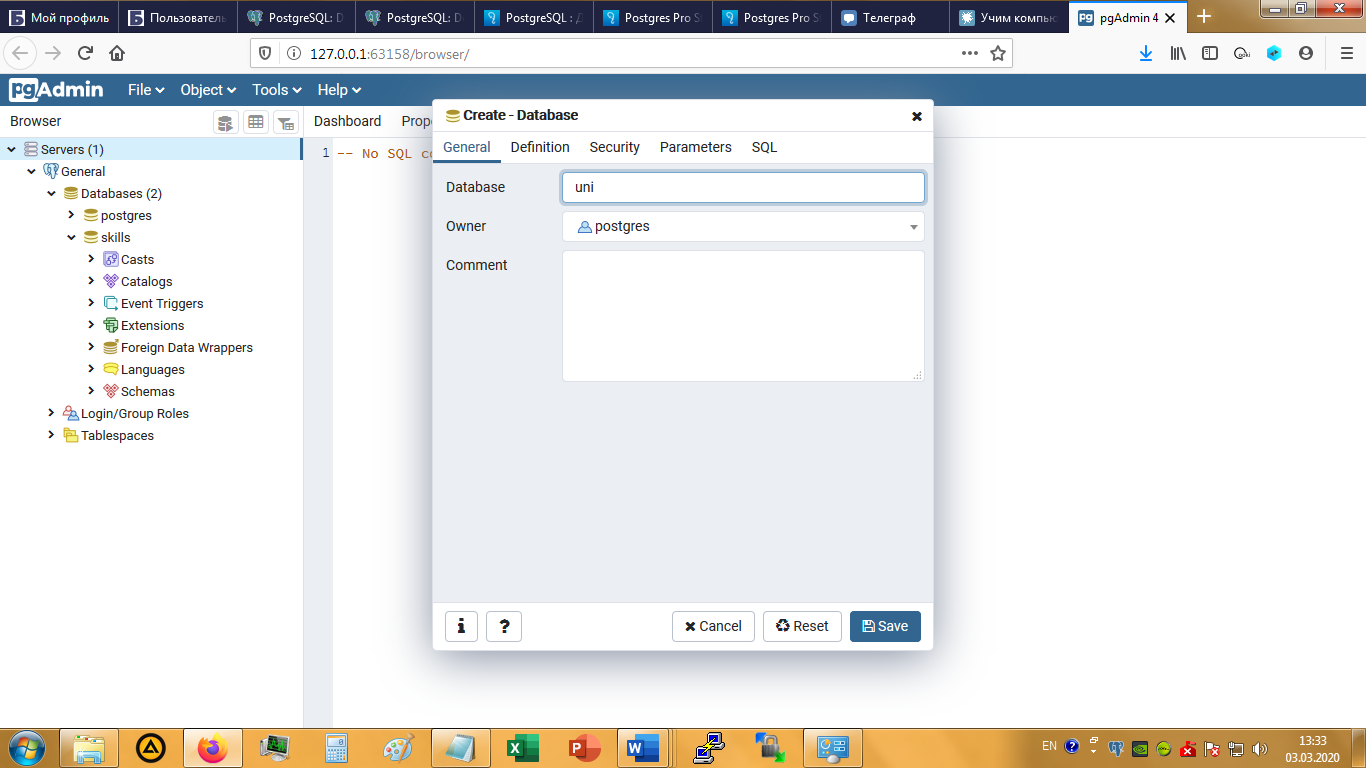


Рисунок 4 – Создание новой БД.

## Создание таблиц

Добавим несколько таблиц в нашу БД. Развернем вкладку созданной БД, в ней находим вкладку «Schemas −> Public −> Tables». Кликаем ПКМ и добавляем таблицу. Задаём ей имя и необходимые поля на вкладке «Колонки» (см. рисунок 5).

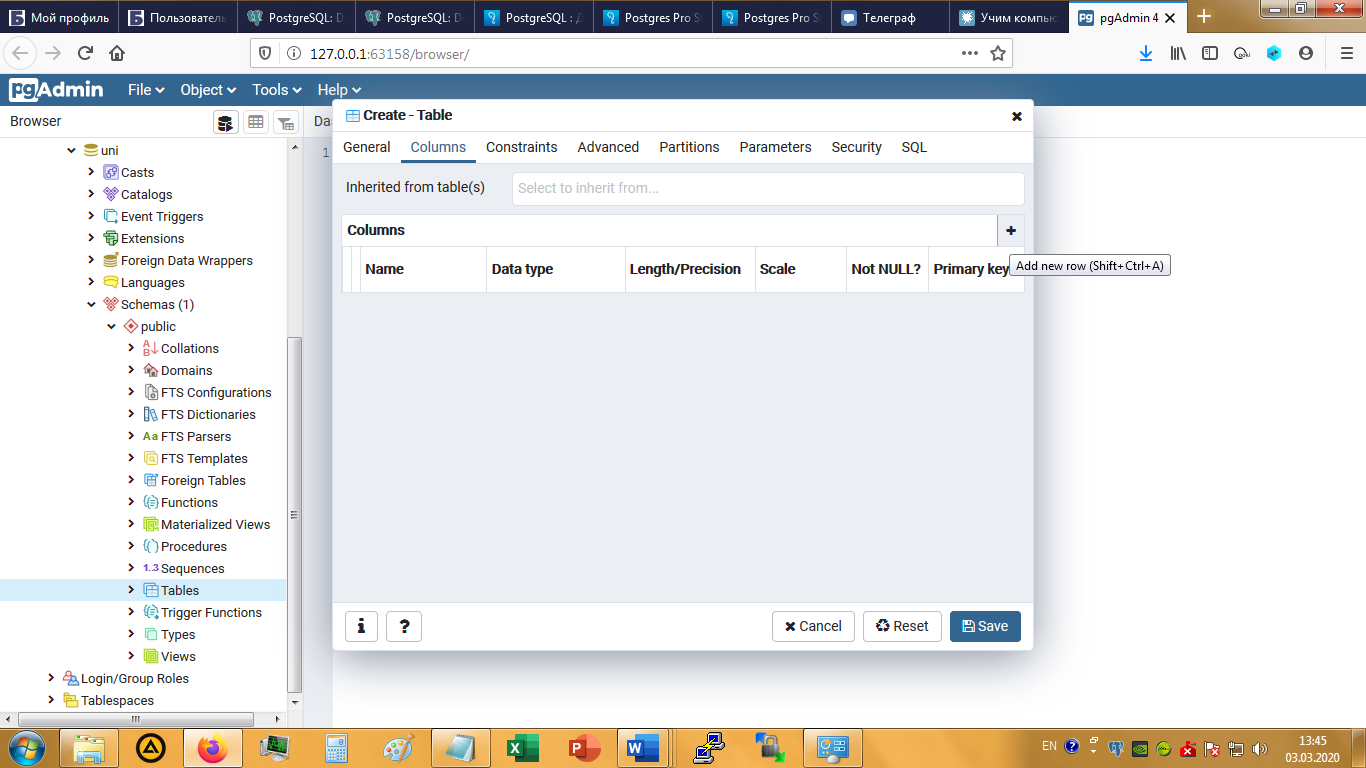


Рисунок 5 – Добавление полей

Чтобы добавить поле в таблицу, нажмите на иконку «Плюс». Задайте имя, тип и параметры поля. Например, создадим поле ID для таблицы «Product». Для этого поля подойдёт тип «Serial», который представляет собой четырёхбайтное целое с автоувеличением (каждый новый элемент будет иметь идентификатор на единицу больше, при удалении элементов освободившиеся диапазоны повторно не заполняются – это позволяет однозначно определять записи, в дальнейшем он будет отображаться по умолчанию как integer). В столбце «NOT NULL?» устанавливается флажок, определяющий возможность оставлять ячейку не заполненной (пустой). Для поля ID это недопустимо, оставим выключенным.

Ключевое поле создается при помощи флажка «Первичный ключ». Под первичным ключом понимают поле или набор полей, однозначно (уникально) идентифицирующих запись. Первичный ключ должен быть минимально достаточным: в нем не должно быть полей, удаление которых из первичного ключа не отразится на его уникальности. Правила хорошего тона при разработке структур баз данных, и чисто практические соображения должны побудить разработчика всегда определять первичный ключ для таблицы базы данных. В подавляющем большинстве случаев поле ID создаётся в каждой таблице и назначается первичным ключом. Поэтому мы зададим ей соответствующий атрибут. Итоговый набор заполненных параметров для поля ID изображен на рисунке 6.

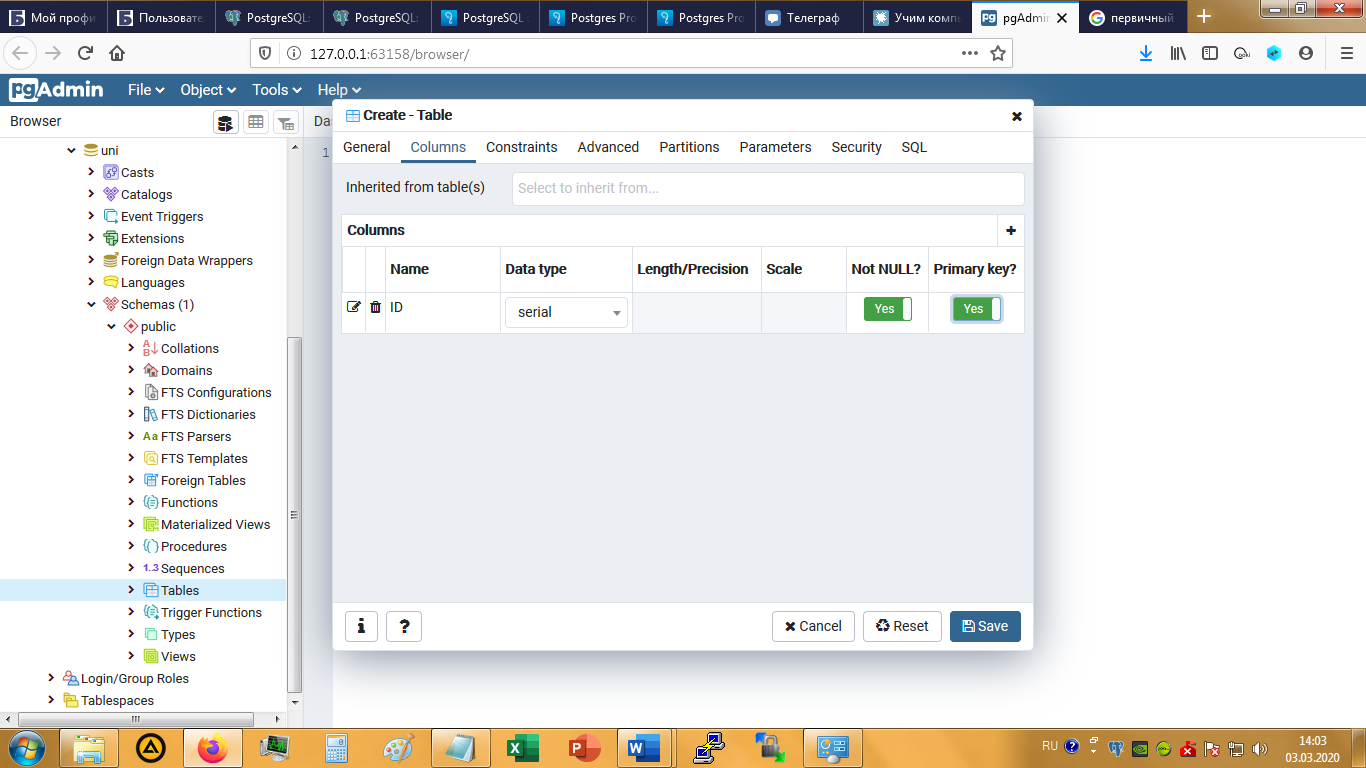


Рисунок 6 – Создание поля ID

Также создадим поля и для них выберем следующие типы данных:

* «ARTICUL» – character varying (длинны 21)
* «ID» - serial (он же integer)
* «NAME» - character varying (длинный 400)
* «WIDTH» - integer
* «LENGTH» - integer
* «IMAGE» - text
* «COMMENT» - text

Итого список полей будет выглядеть примерно таким образом, как на рисунке 7.

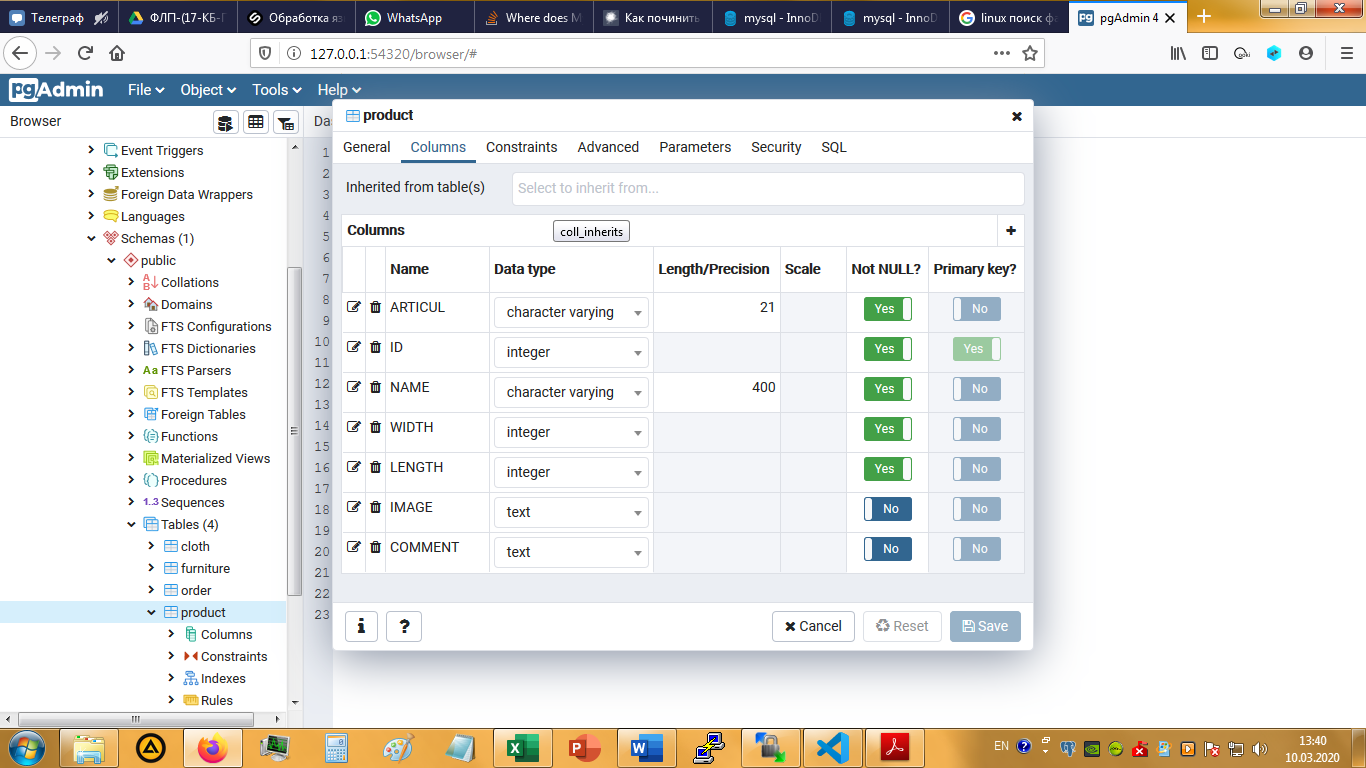


Рисунок 7 – Список полей для таблицы «product»

Когда все поля созданы, нажмём кнопку «Сохранить», если всё указано, таблица будет добавлена в нашу базу данных.

Аналогичным образом создаем остальные таблицы.

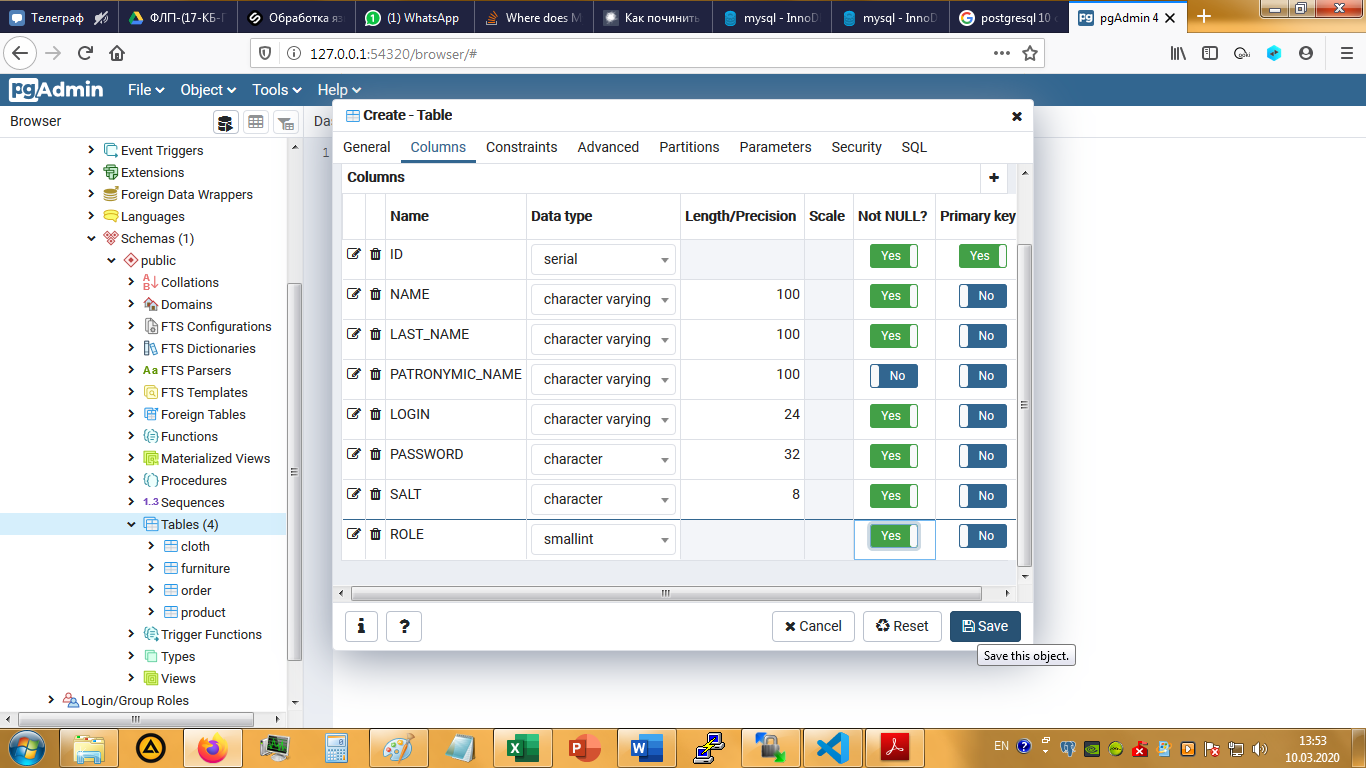


Рисунок 8 – Список полей для таблицы «users»

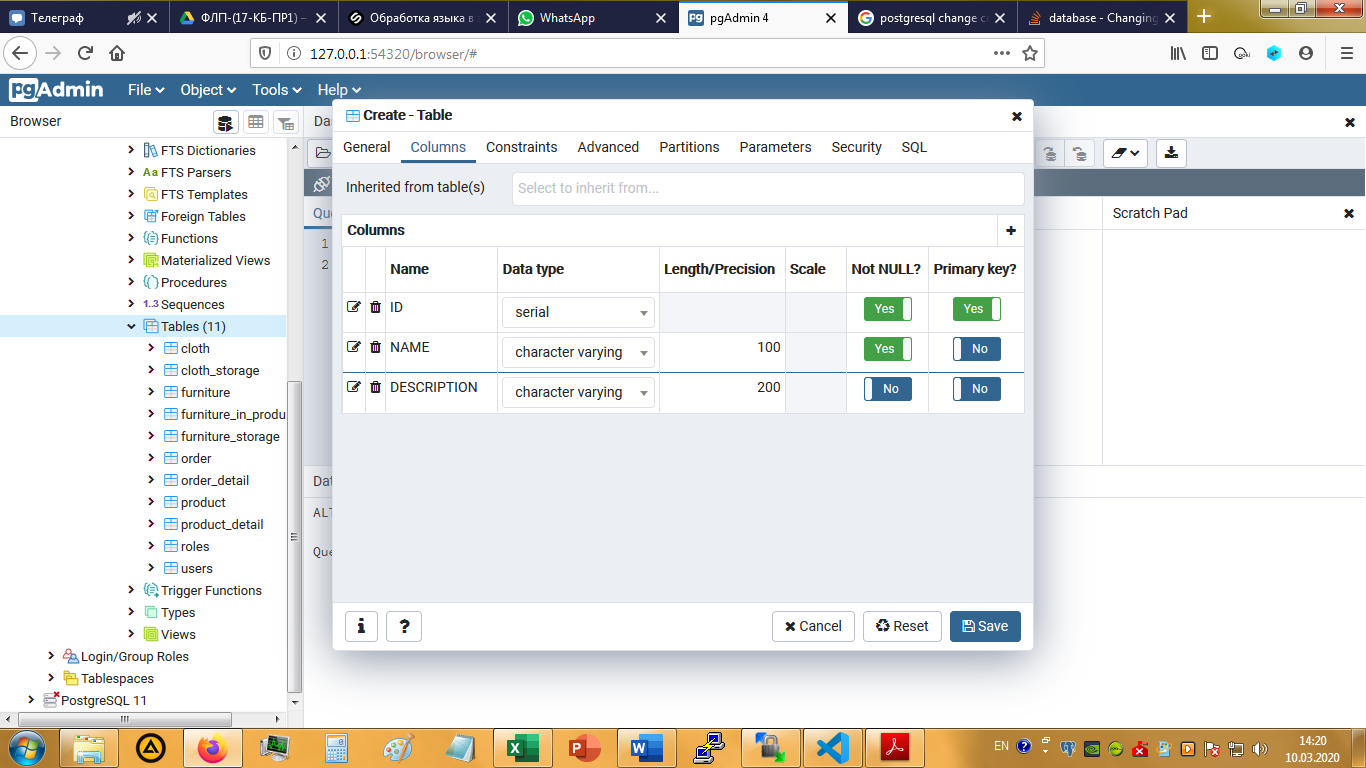


Рисунок 9 – Список полей для таблицы «colors»

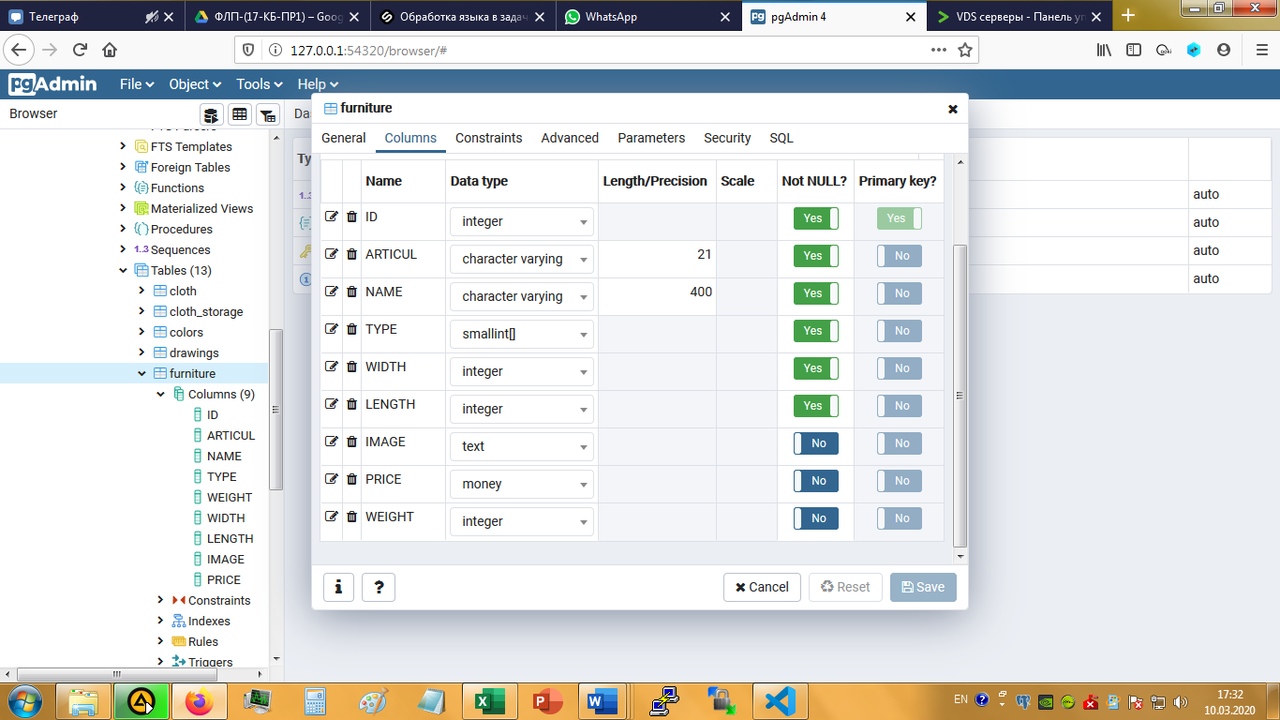


Рисунок 10 – Список полей для таблицы «furniture»

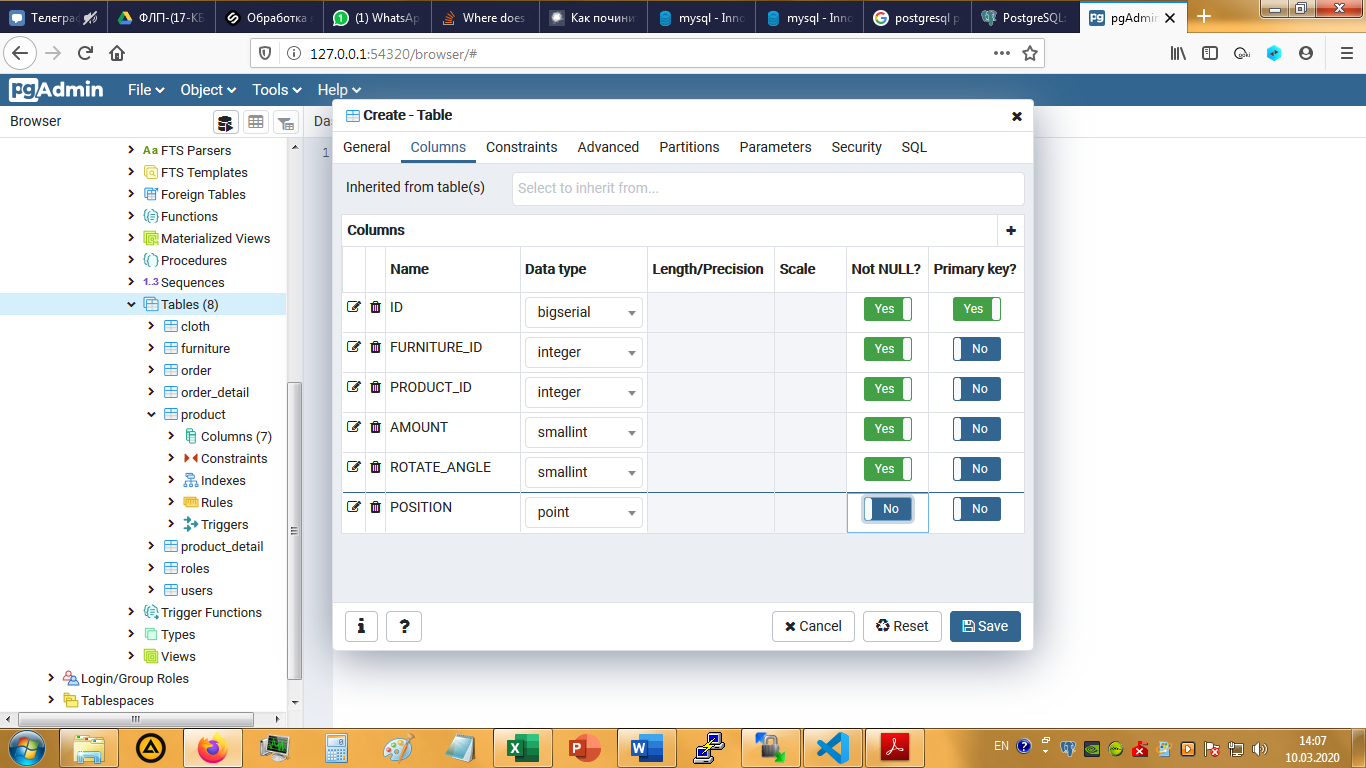


Рисунок 11 – Список полей для таблицы «furniture\_in\_product»

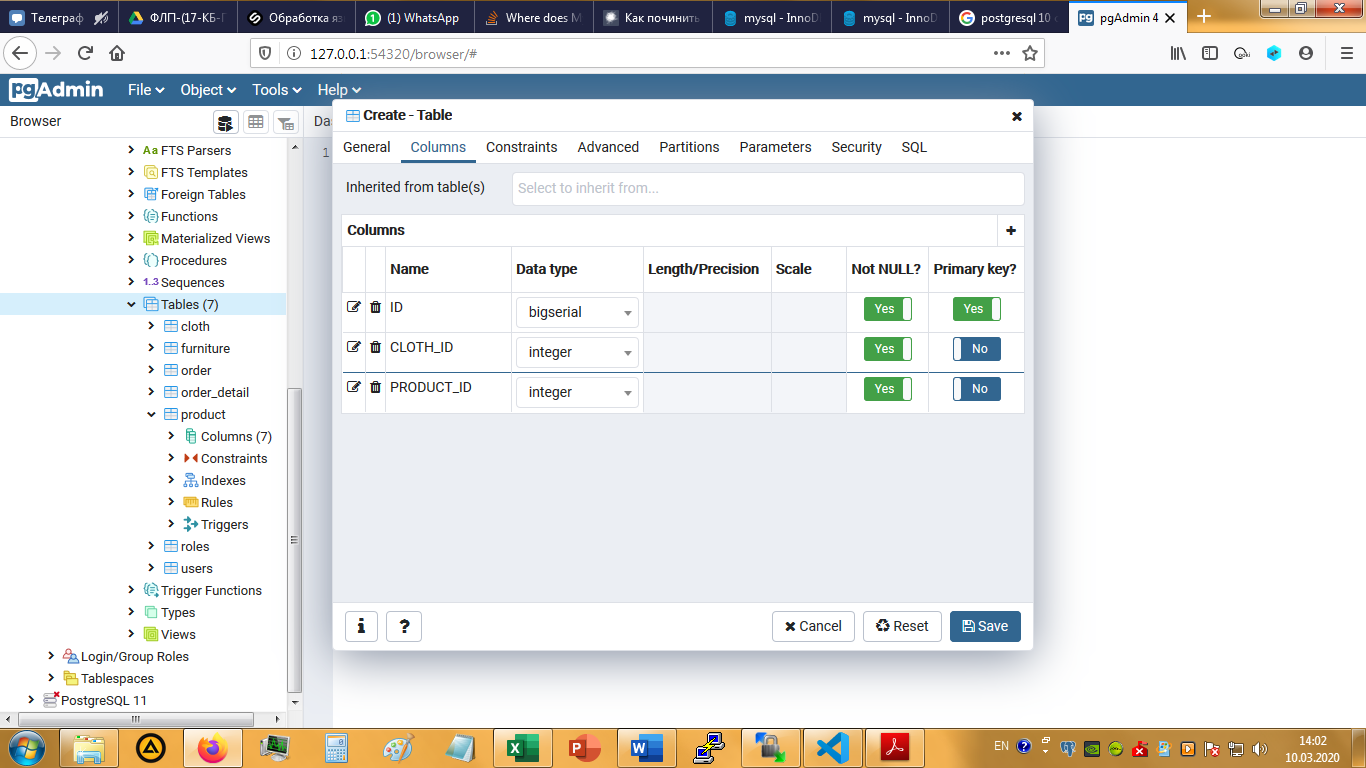


Рисунок 12 – Список полей для таблицы «product\_detail»

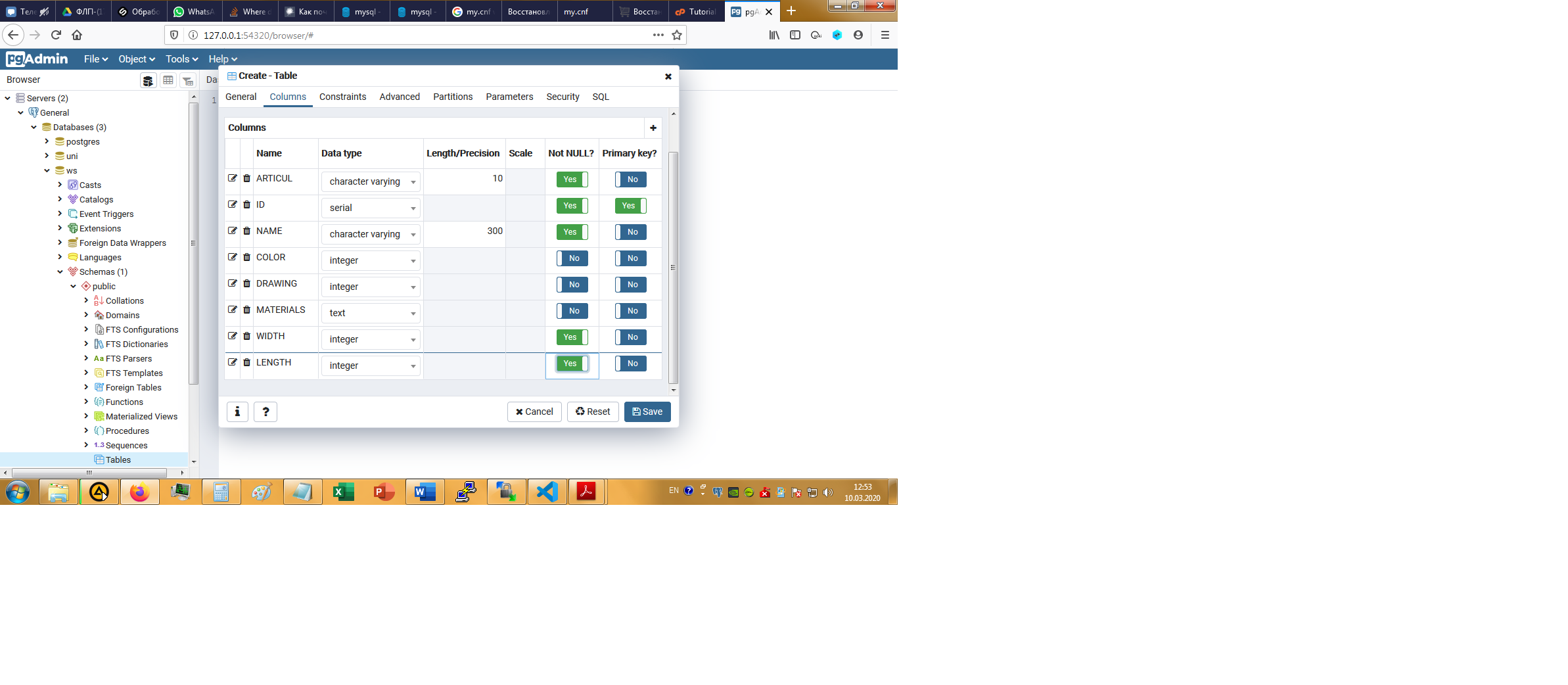


Рисунок 13 – Список полей для таблицы «cloth»

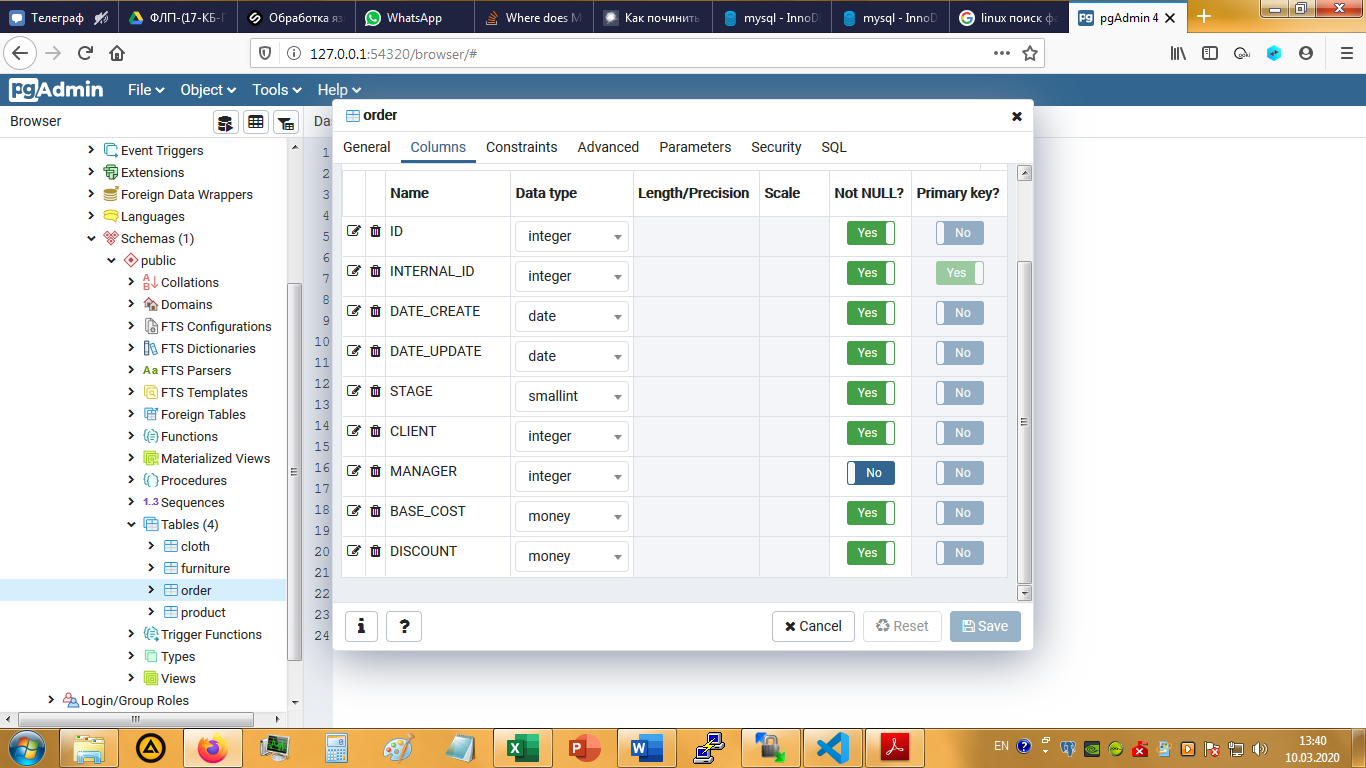


Рисунок 14 – Список полей для таблицы «order»

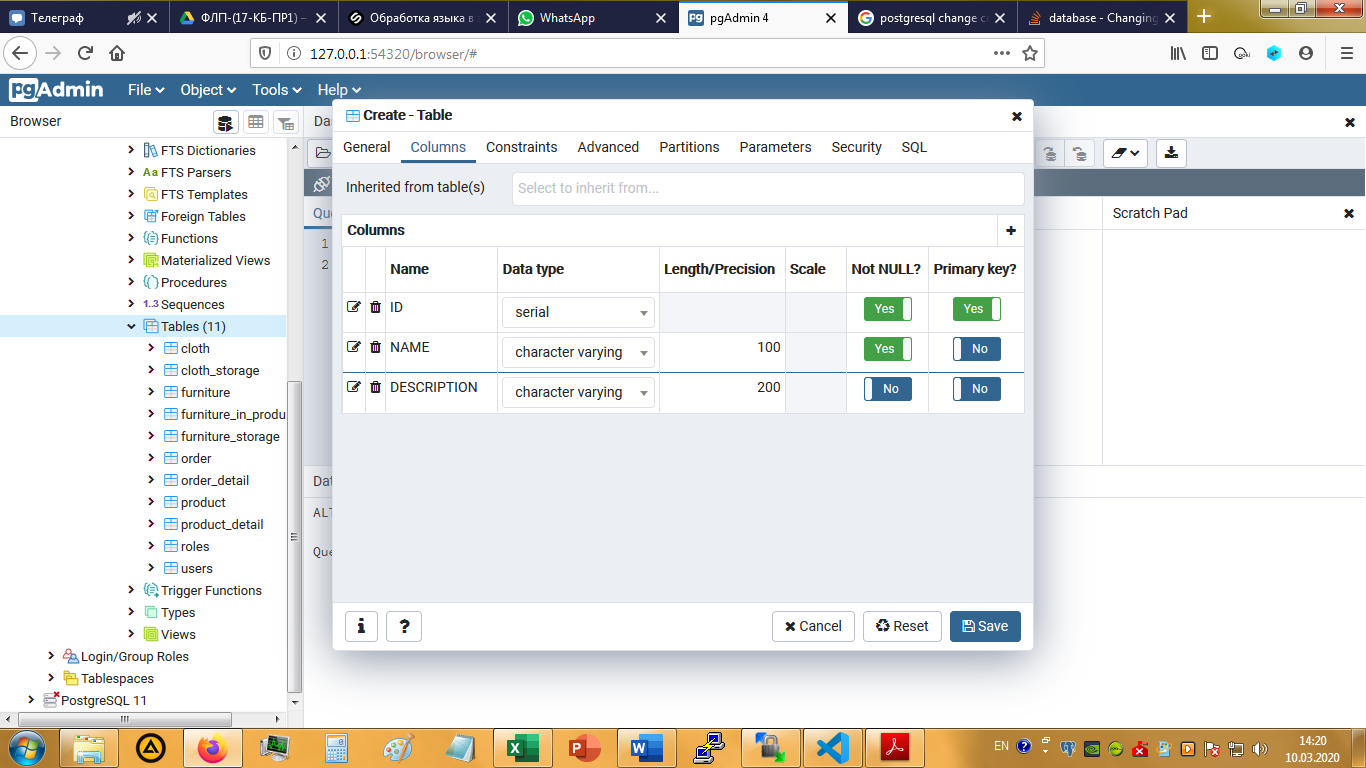


Рисунок 15 – Список полей для таблицы «drawings»

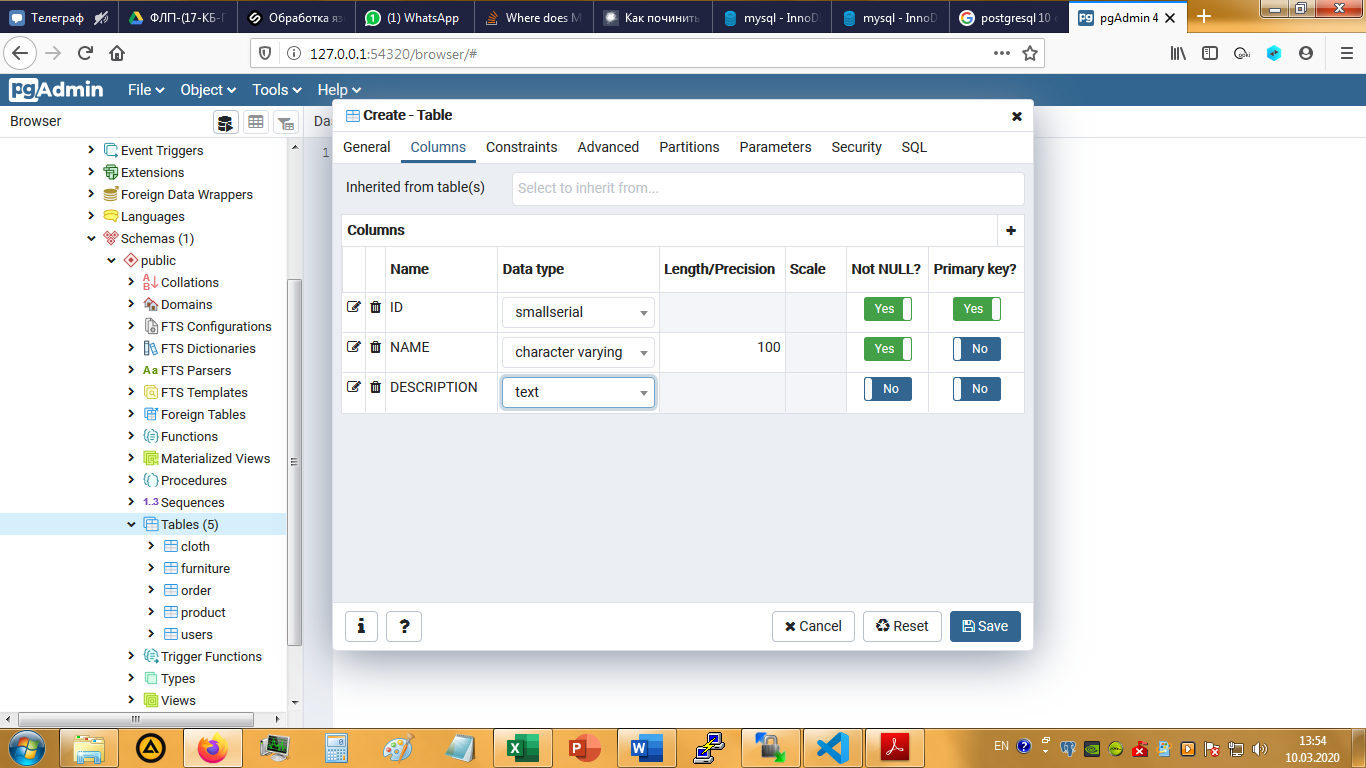


Рисунок 16 – Список полей для таблицы «roles»

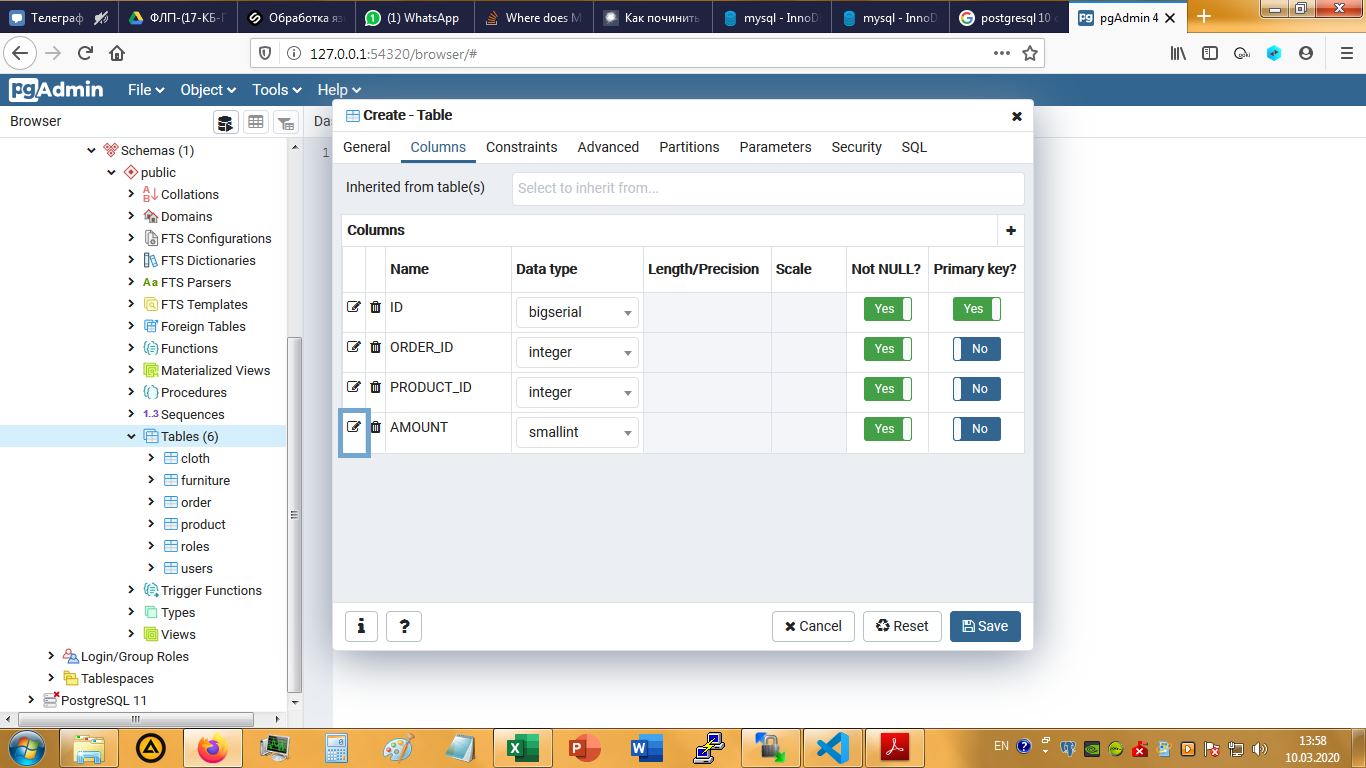


Рисунок 17 – Список полей для таблицы «order\_detail»

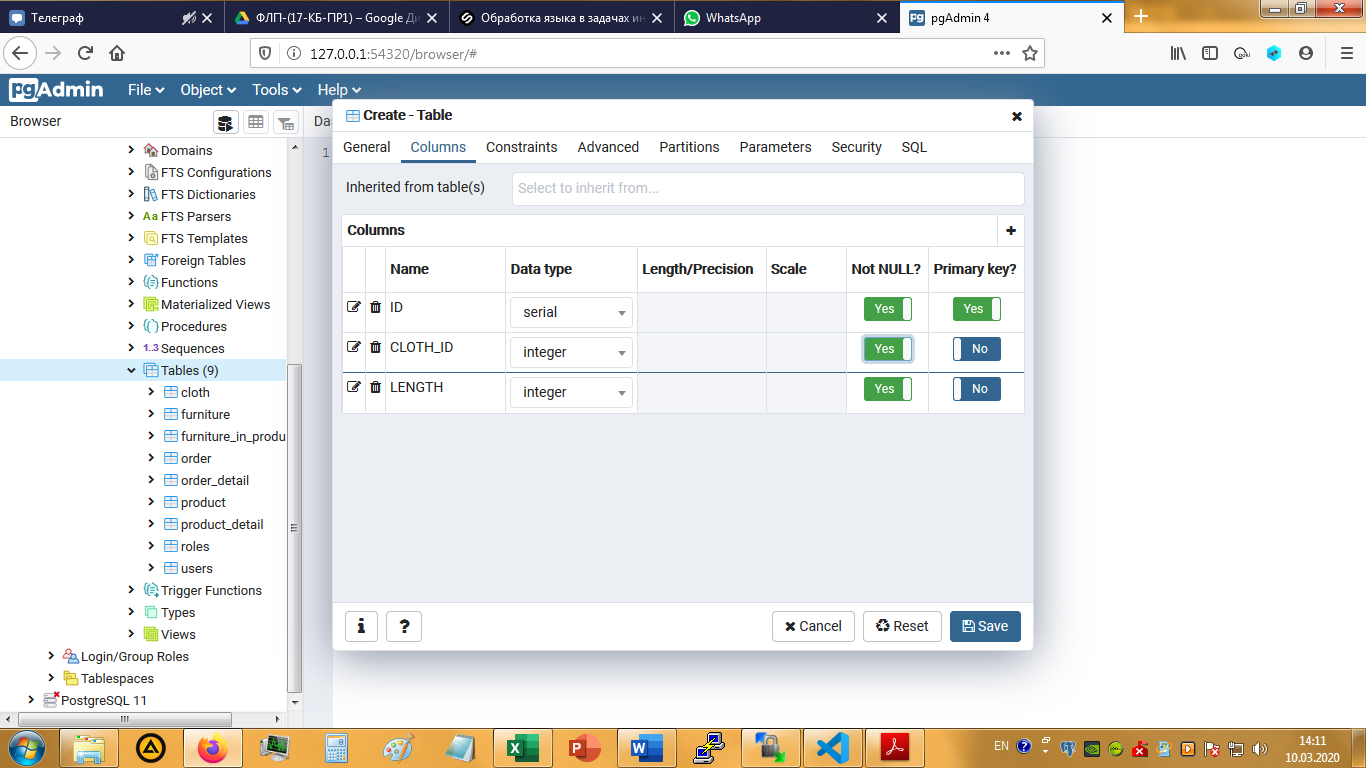


Рисунок 18 – Список полей для таблицы «cloth\_storage»

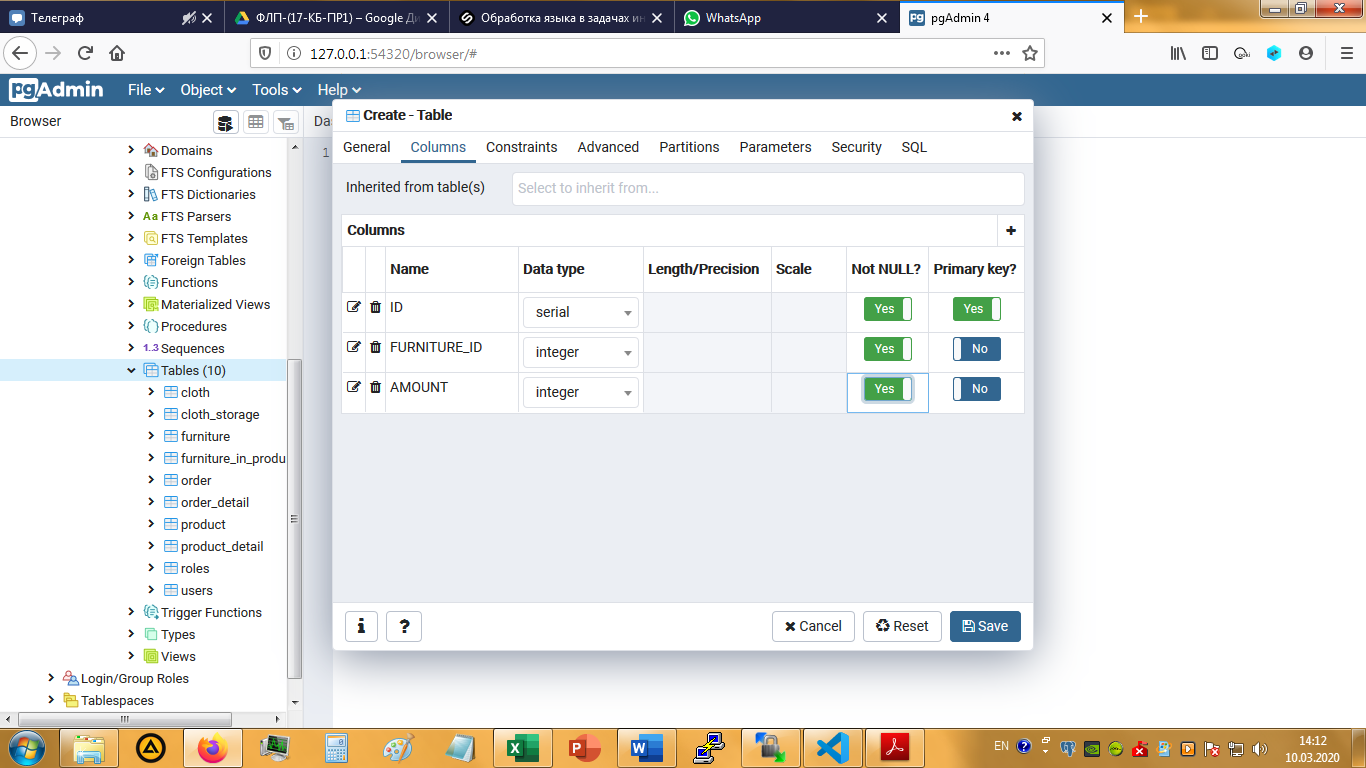


Рисунок 19 – Список полей для таблицы «furniture\_storage»

## Исправление ошибок

Если при создании колонки (поля) произошла ошибка, то сменить тип может быть весьма затруднительно. Однако можно попытаться исправить ситуацию. Если необходимый тип совпадает по размерности с текущим, то его можно изменить на вкладке «Свойства» поля (рисунок 20).

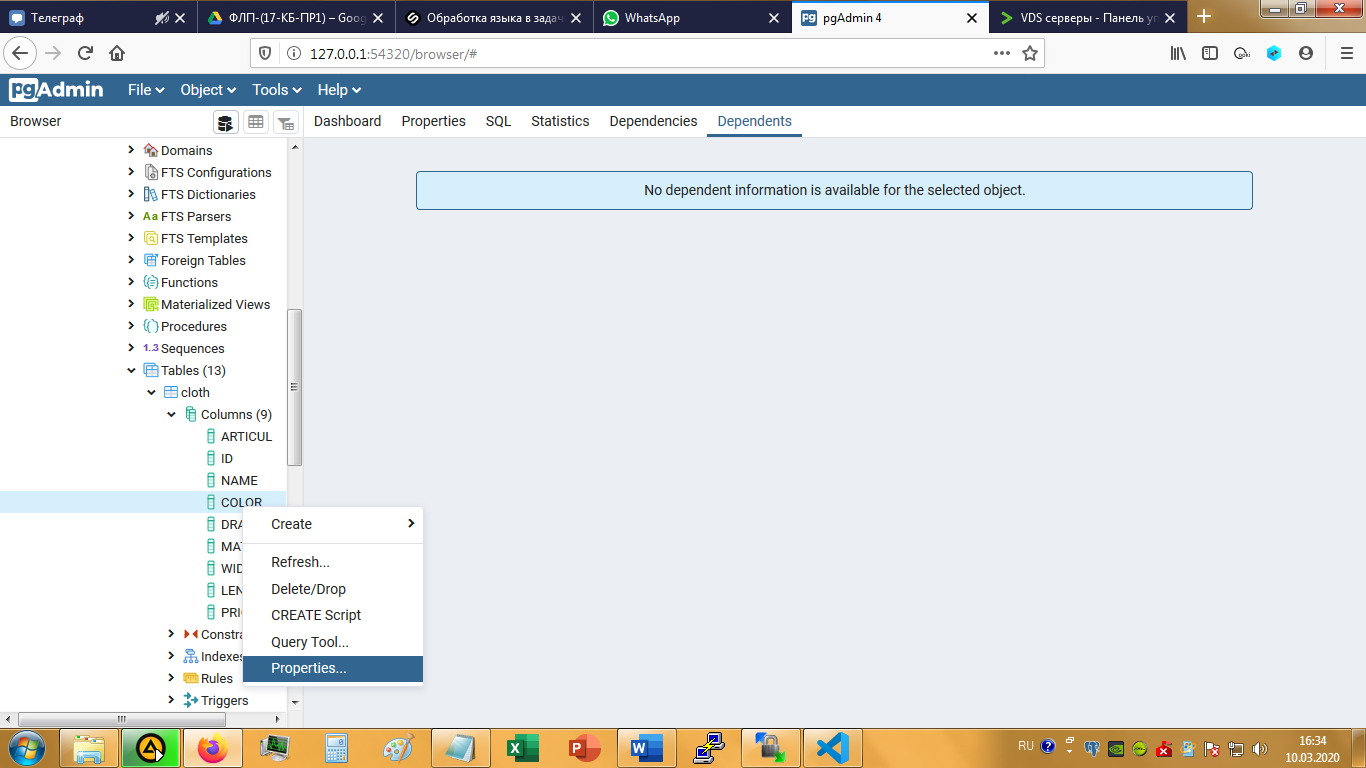


Рисунок 20 – Переход на вкладку свойств колонки

Иначе можно попробовать провести запрос ‘ALTER’. Для этого необходимо выбрать инструмент запросов. Делается это при помощи нажатия ПКМ по одной из таблиц в списке слева (рисунок 21), в открывшемся меню выбираем пункт «Query tool». В открывшемся поле пишем необходимый запрос, нажимаем на стрелочку вверху для выполнения. Пример смены типа integer на integer[] (массив целых 4-байтовых) показан на рисунке 22.

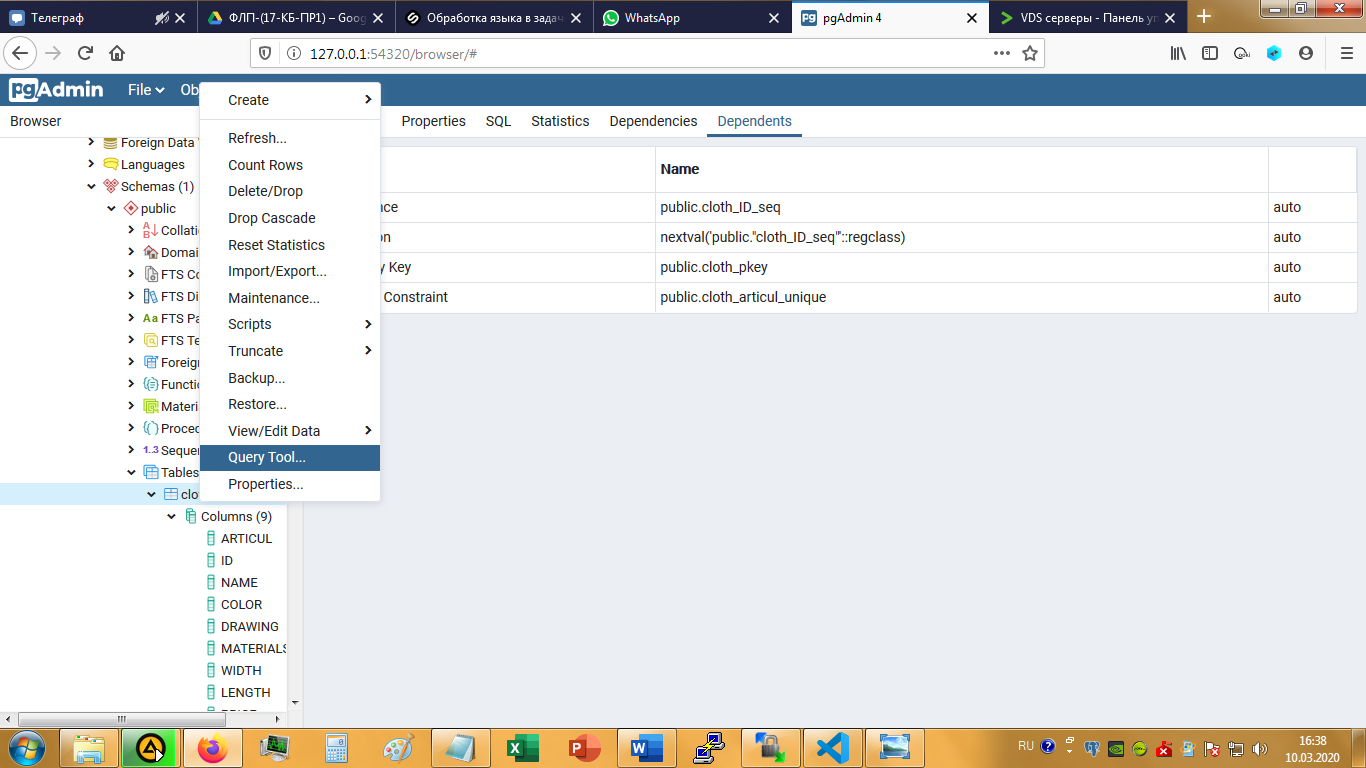


Рисунок 21 – Переход к написанию запросов

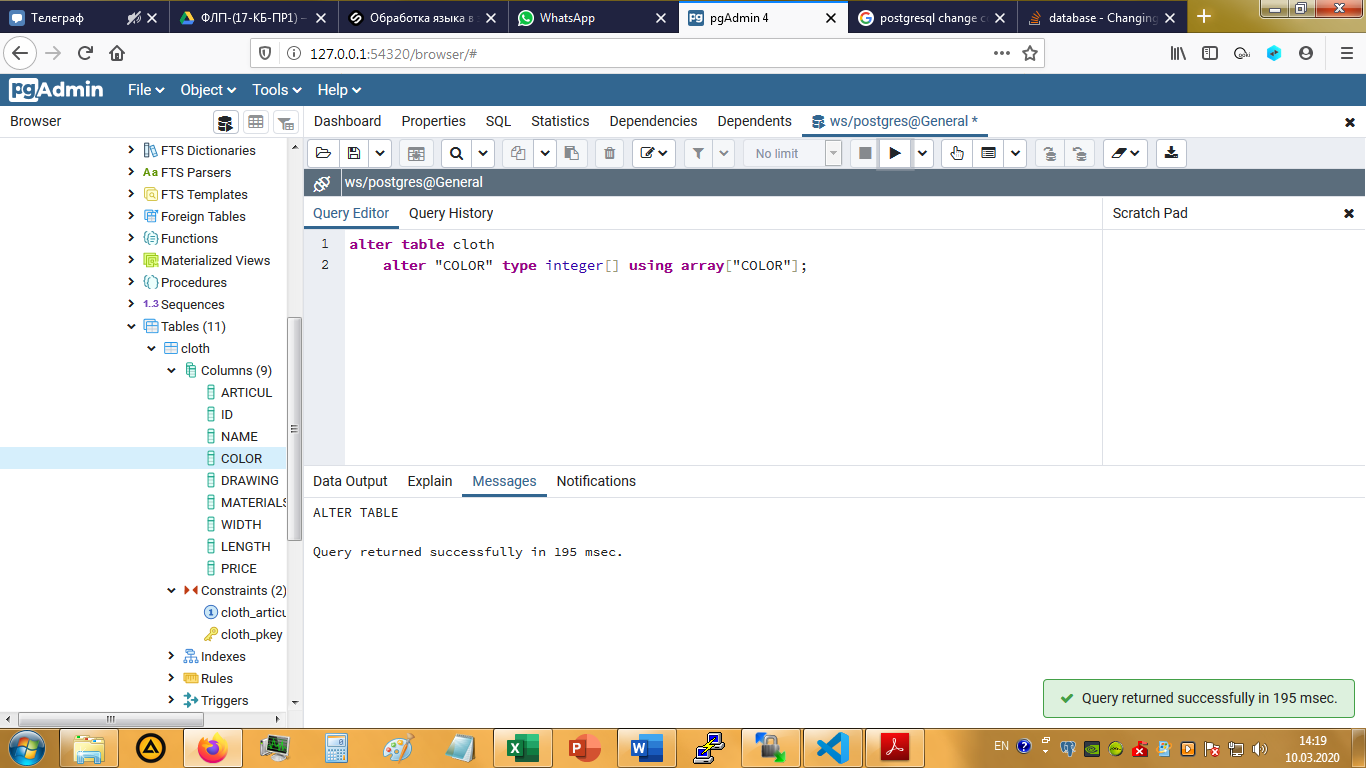


Рисунок 22 – Смена типа поля запросом

При создании таблицы «Пользователи» помимо колонки пароля мы указывали так называемую «соль» (колонка с именем SALT). Это необходимо для безопасного хранения паролей в таблицах. Подход следующий: мы храним не просто пароль, а результат хеш-функции (например md5), примененной к хеш-функции к указанному паролю с добавлением этой «соли», формируемой каждый раз случайным образом. Псевдокодом можно описать так

md5( md5(password) + salt )

Таким образом, мы значительно повышаем безопасность хранимых данных, так как данная хеш-функция необратима, а наличие «соли» и двойное шифрование уберегают данные от взлома простым перебором. Учтите, что пароль должен быть известен пользователю во время регистрации, в дальнейшем его можно будет только сбросить.

## Создание связей между таблицами

После создания таблиц нужно создать связи между ними. Следует обратить ваше внимание на то, что для установления связей межу таблицами в каждой из них необходимо наличие однотипных полей. Например, таблица «Ткани» содержит поле «Цвет» в качестве зависимого от таблицы «Цвета», содержащей поле id цвета, в качестве ключевого. Таким образом, соединение этих двух таблиц по полю id цвета позволяет установить между ними тип связи один ко многим.

Перейдём непосредственно к созданию связей, называемых внешними ключами (foreign keys). Во вкладке с нужной таблицей найдём пункт «Constraints», нажмём ПКМ и выберем пункт «Создать −> Внешний ключ». Дадим ему имя, как правило для этого отношения имя формируется так: fk\_<имя\_таблицы\_зависимой>\_<имя\_таблицы\_главной>\_<имя\_связываемого\_поля>. Для примера создадим отношение между цветами по полю ID цвета – выберем описанный выше пункт в таблице тканей. Дадим связи имя fk\_cloth\_colors\_id. Во вкладке «Колонки» укажем поле из текущей таблицы, таблицу, с которой производим связь и поле в той таблице. Когда всё указано, нажмём иконку «Плюс» вверху окна, как показано на рисунке 23.

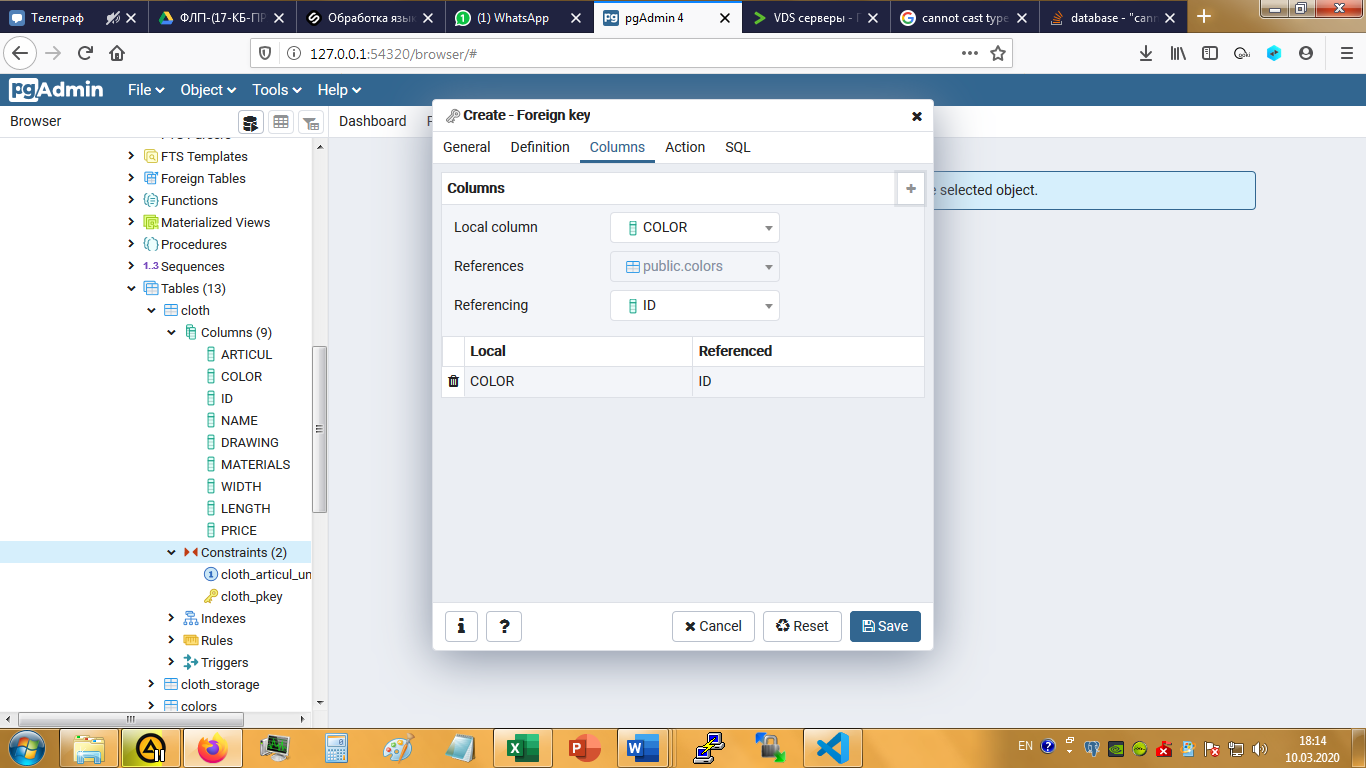


Рисунок 23– Создание внешнего ключа

После этого сохраним связь. Аналогичным образом устанавливаем связи между остальными таблицами согласно ER-диаграмме, изображенной на рисунке 24.

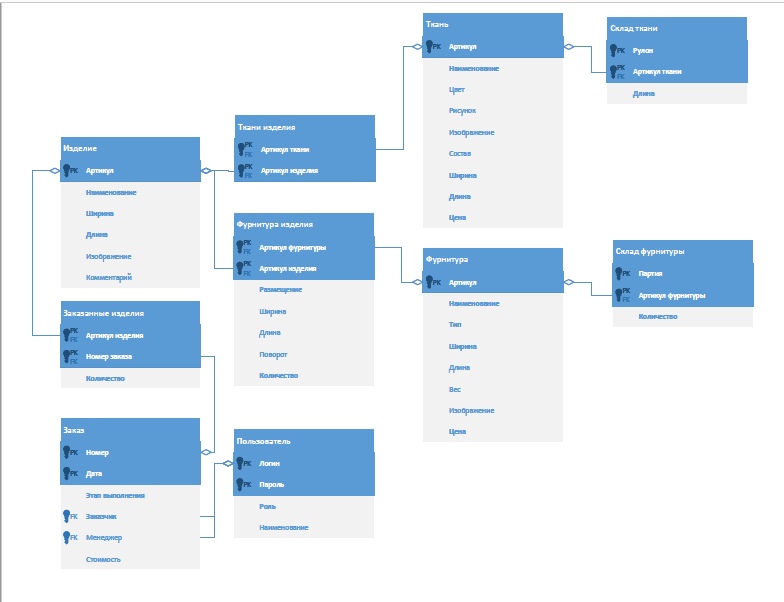


Рисунок 24 – ER-Диаграмма необходимой таблицы

В таблице №1 приведены основные типы данных, необходимые при проектировании полей таблиц базы данных.

Таблица №1 - Встроенные типы данных PostgreSQL 11

| **№** | **Тип** | **Псевдонимы** | **Описание** |
| --- | --- | --- | --- |
| 1 | bigint | int8 | знаковое целое из 8 байт |
| 2 | bigserial | serial8 | восьмибайтное целое с автоувеличением |
| 3 | bit [ (n) ] | varbit [ (*n*) ] | битовая строка фиксированной длины |
| 4 | bit varying [ (n) ] | bool | битовая строка переменной длины |
| 5 | boolean |  | логическое значение (true/false) |
| 6 | bytea |  | двоичные данные («массив байт») |
| 7 | character [ (n) ] | char [ (n) ] | символьная строка фиксированной длины |
| 8 | character varying [ (n) ] | varchar [ (n) ] | символьная строка переменной длины |
| 9 | date |  | календарная дата (год, месяц, день) |
| 10 | double precision | float8 | число двойной точности с плавающей точкой (8 байт) |
| 11 | integer | *Int*, *int4* | знаковое четырёхбайтное целое |
| 12 | Interval [ *поля* ] [ (*p*) ] |  | интервал времени |
| 13 | json |  | текстовые данные JSON |
| 14 | jsonb |  | двоичные данные JSON, разобранные |
| 15 | money |  | денежная сумма |
| 16 | numeric [ (*p*, *s*) ] | decimal [ (p, s) ] | вещественное число заданной точности |
| 17 | real | float4 | число одинарной точности с плавающей точкой (4 байта) |
| 18 | smallint | int2 | знаковое двухбайтное целое |
| 19 | smallserial | serial2 | двухбайтное целое с автоувеличением |
| 20 | serial | serial4 | четырёхбайтное целое с автоувеличением |
| 21 | text |  | символьная строка переменной длины |
| 22 | *timestamp [ (p) ] [ without time zone ]* |  | дата и время (без часового пояса) |
| 23 | *timestamp [ (p) ] with time zone* | timestamptz | дата и время с учётом часового пояса |
| 24 | uuid |  | универсальный уникальный идентификатор |
| 25 | xml |  | XML-данные |

Следует отметить, что одна из задач проектирования базы состоит в обеспечения способа идентификации различных объектов, другими словами система должна отличать друг от друга отдельные строки таблицы. Первичный ключ это столбец, однозначно определяющий строку в таблице. В случае, когда в таблице существует несколько одинаковых значений, уникальный первичный ключ поможет их различить.

Для связи данных между таблицами необходимо создать отношения. Отношения бывают:

* **Один к одному** - определение связи «один к одному» полностью соответствует ее названию. Связью «один к одному» называется такая связь, из наличия которой следует, что если имеется какая-то одна строка в одной таблице, то должна быть точно одна соответствующая ей строка в другой таблице.
* **Один ко многим** - Связью «один ко многим» называется такая связь, из наличия которой следует, что если имеется какая-то одна строка в одной таблице, то должны быть несколько соответствующих ей строк в другой таблице.
* **Многие ко многим** - Связь «многие ко многим» характеризуется тем, что на обеих сторонах связи может присутствовать несколько согласующихся строк, а не только одна.

Связь «Один ко многим» является типичной связью между таблицами, где одна таблица по вторичному ключу ссылается на вторую таблицу с индексным ключом.

Связь «Многие ко многим» характеризуется тем, что на обеих сторонах связи может присутствовать несколько согласующих строк.

## Контрольные вопросы

1. Что такое ядро базы данных?
2. Что такое система управления баз данных?
3. Перечислите основные достоинства языка SQL?
4. Дайте определение логическому имени.
5. Приведите пример команды создания базы данных.
6. Какие системные и числовые типы данных PostgreSQL вам знакомы?
7. Приведите пример команды создания таблицы базы данных.
8. Дайте определения первичному и внешнему ключам.