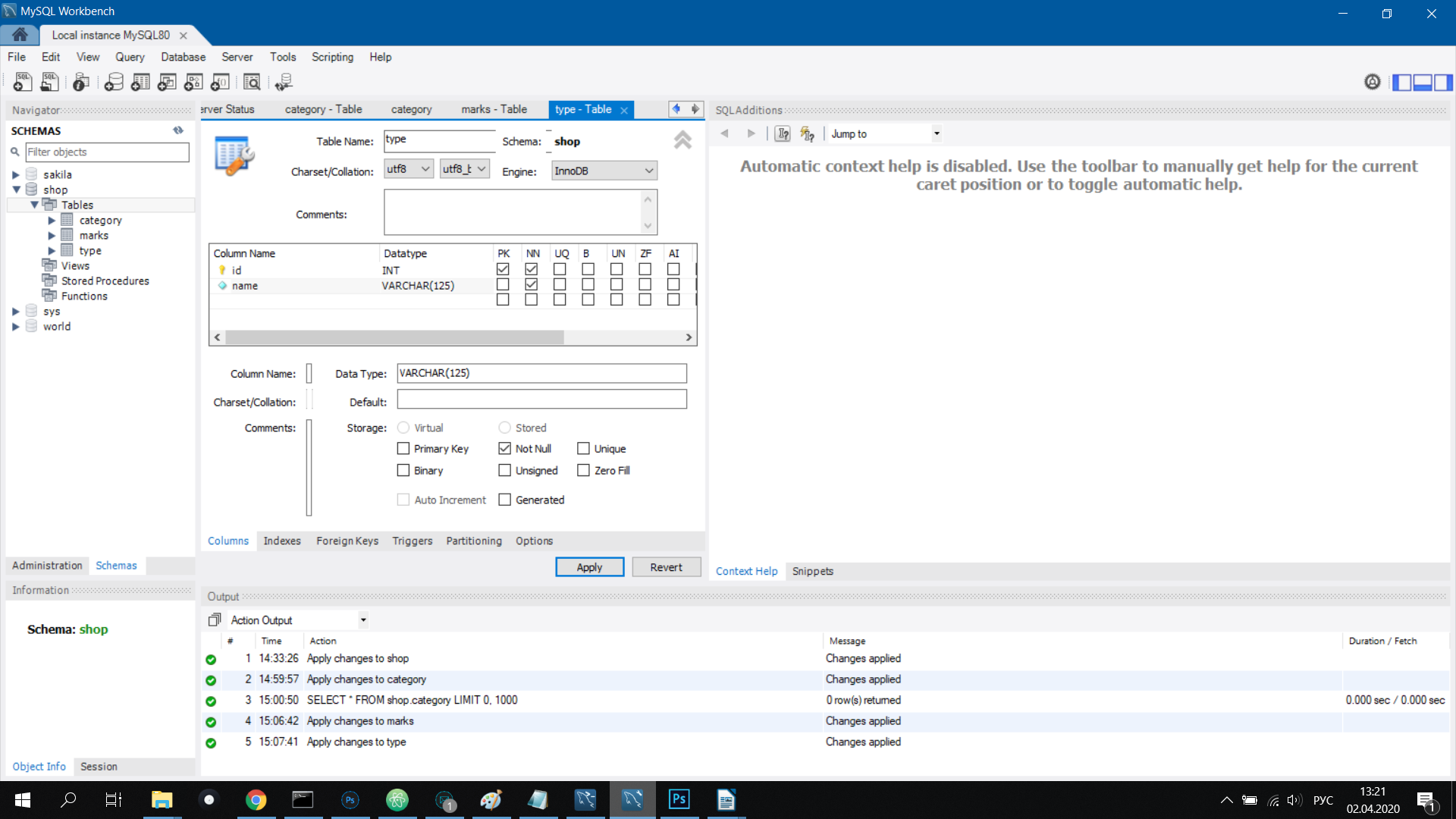
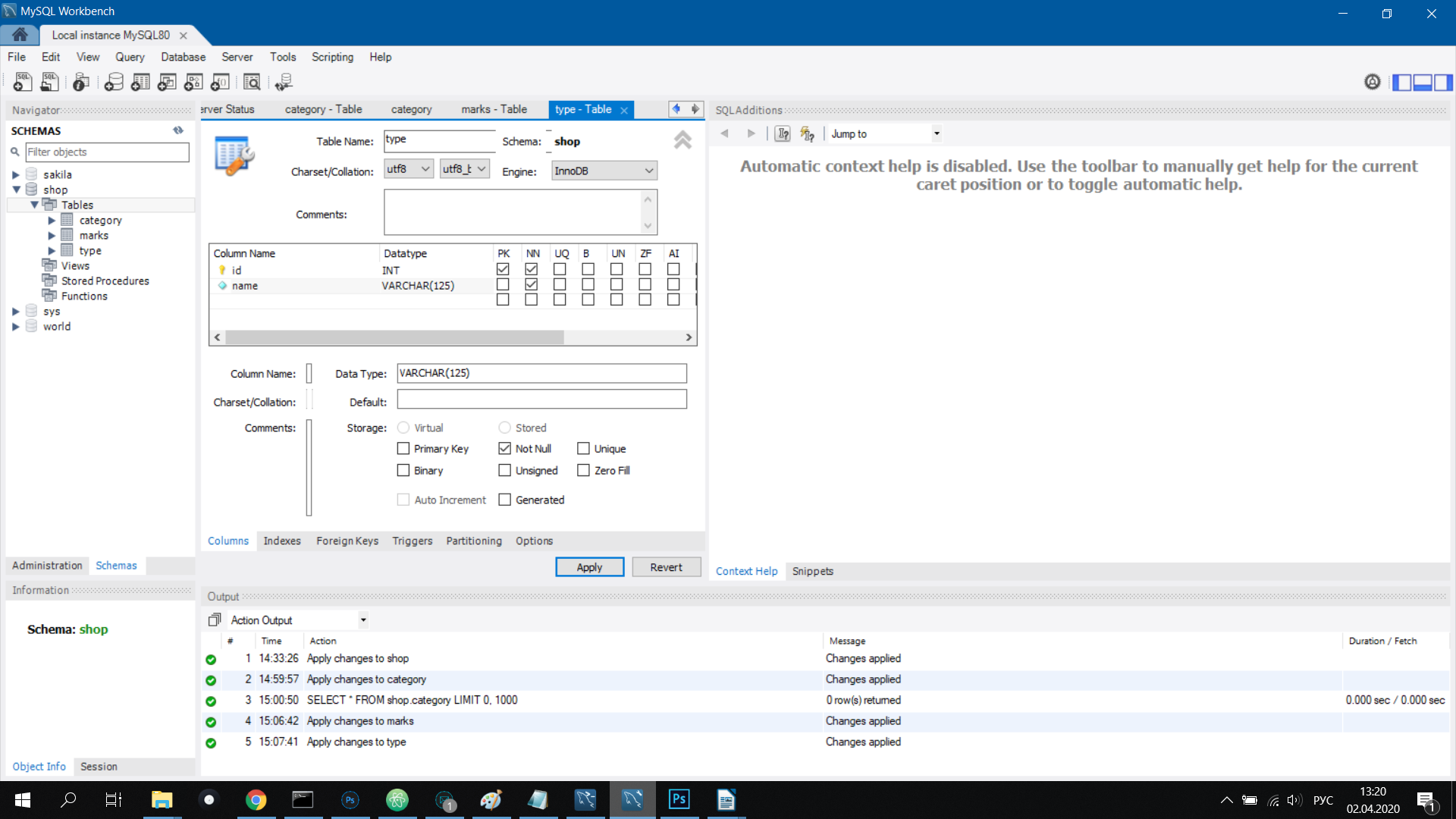
**Во время создания базы данных важно создавать отдельные таблицы для тех характеристик, к которым будет привязана какая-то другая характеристика: скидка может зависеть от того, мужская это вещь или женская. Также отдельная таблица нужна для тех характеристик, которые будут часто повторяться: марка одежды. В основную таблицу нужно вносить эти характеристики по их id. Также, можно вносить характеристики прямиком, если они не зависят ни от каких условий и не повторяются: цена на вещь.**

**Создание проекта и таблиц:**

**Графический интерфейс позволяет создавать sql** **запросы в более простом режиме. Но также это можно сделать и через консоль.**

**Первым делом создаем в базе новый проект. В нем задаем кодировку**

**В tables создаем новую таблицу. Создаем колонки в таблице и добавляем им параметры**

****

**PK - первичный ключ**

**NN - запрет на нулевое значение**

**AI - автоматическая генерация значения для поля (подходит для pk)**

**В Datatype указывается тип данных:**

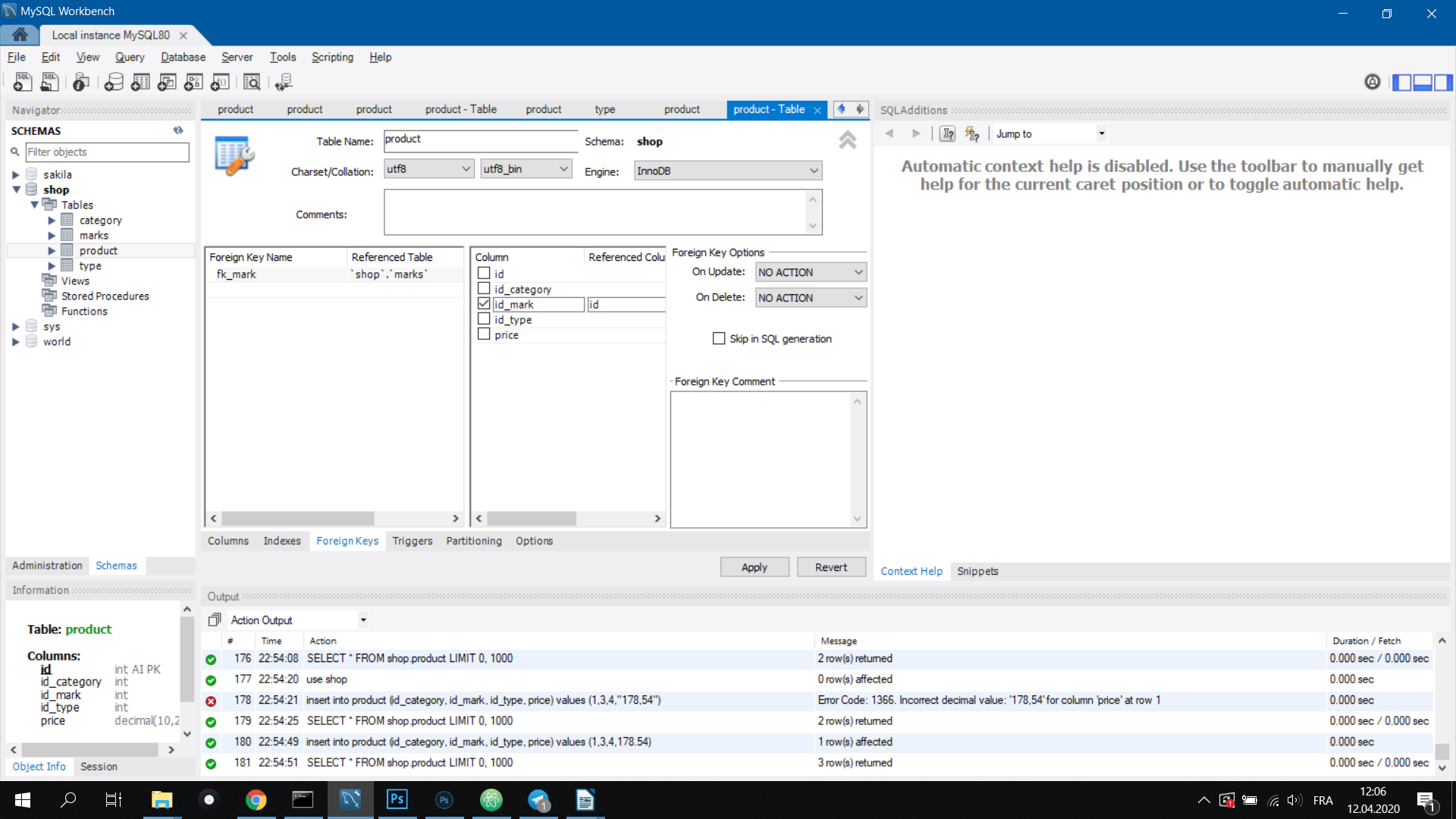
**VARCHAR(45) - текстовое поле ограниченной длины. В скобках указывается предел**

**NUMERIC и DECIMAL (знаков до запятой, знаков после запятой) - подходит для хранения цен благодаря возможности ставить запятую между числами. При это значение хранится в виде строки, а не числа с плавающей точкой**

**Таблица с пояснением диапазона разных числовых типов данных**

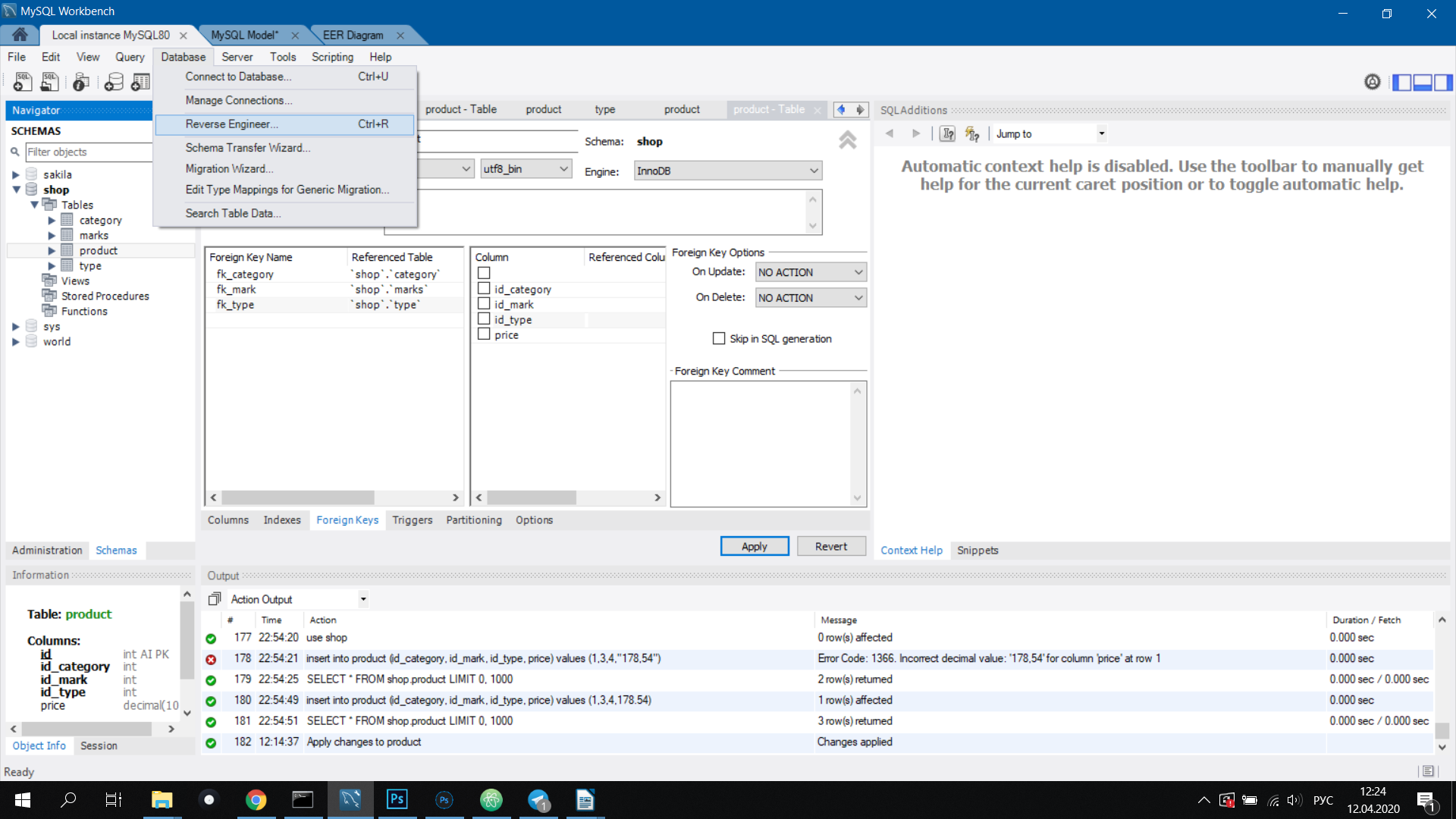
**Созданную таблицу всегда можно отредактировать, нажав на нее правой кнопкой мыши - Alter Table**

**Связь таблиц:**

****

**В редактировании таблицы заходим во вкладку Foreign Keys. В ней создаем связи: указываем имя связи, затем связь с таблицей проекта, после указываем для какого столбца и через какой столбец. Также, можно указать поведение при удалении связанного экземпляра, например: каскадное удаление**

**Визуальная схема связей:**

****

**Далее выбираем нашу бд, затем наш проект и создаем схему связей**

**Работа через консоль:**

**use shop; - после не нужно указывать каждый раз, что работаешь именно с этим проектом**

**CREATE SCHEMA `имя проекта`; - создание нового проекта**

**CREATE TABLE `имя проекта`.`имя таблицы` (  
 `id` INT NOT NULL AUTO\_INCREMENT,  
 `name` VARCHAR(45) NOT NULL,  
 PRIMARY KEY (`id`)); - создание новой таблицы в проекте. Параметры полей указываются без запятых**

**ALTER TABLE `имя проекта`.`имя таблицы`   
CHANGE COLUMN `имя поля` `id` INT NOT NULL AUTO\_INCREMENT; - для обновления поля в таблице, после имени поля идет запись будто при создании поля заново**

**INSERT INTO `название проекта`.`название таблицы` (`имя поля 1`, ..) VALUES ('значение поля 1', ..); - добавление значения в таблицу**

**Выборка:  
  
select \* FROM type; - достать все строки из таблицы  
  
select \* from type where id = 3; - достать только те строки, которые удовлетворяют условие после where**

**id <> 3 - не равен**

**(id > 3) and (id < 5) - оператор "и", много условий записываются в скобках**

**(id > 3) or (id < 5) - "или"**

**Также, доступны операторы not и >=**

**IS NULL / IS NOT NULL - со значением NULL используется другой синтаксис**

**select \* from type where id in (1, 2, 5); - более компактная запись длинного условия. Подойдет любое значение в скобках**

**where name like 'А%' - подойдут все значения, которые начинаются с А**

**select имя столбца 1, имя столбца 2 from type where id = 3; - достает только значение указанных столбцов, если строка удовлетворяет условие**

**select distinct name from type; - выводит только если значение столбца name уникально**

**select \* from type order by имя столбца (desc) - сортировка найденных значений по значению столбца. Команда в скобках для обратной сортировки**

**select \* from type where id = 3 limit 2; - задает лимит необходимых записей**

**update имя таблицы set имя столбца = новое значение where id = .. - обновление значения для таблицы. Обязательно указывать where, иначе обновление будет для всех строк в столбце**

**Важно: субд не позволит использовать в качестве условия для where не pk. Это нужно для того, чтобы избежать массового изменения данных**

**delete from имя таблицы where .. - удаление записей из бд**

**having sum(count) > 5**

**having total\_count > 5 - работает как where, но для значений, полученных в результате расчета каких-то функций. Можно указать как саму функцию, так и имя столбца, если расчет был выведен в таблице**

**Объединение таблиц:**

**SELECT product.id, type.name, marks.name, category.name, price FROM product  
 inner join category on product.id\_category = category.id  
 inner join marks on product.id\_mark = marks.id  
 inner join type on product.id\_type = type.id;**

**inner join - создает общую таблицу, но только из тех элементов, которые входят во все объединения. То есть, если соединить все товары с их типом (две разные таблицы), то общая таблица не будет содержать типы, к которым не принадлежит ни один товар. Все условия и сортировку необходимо писать после объединения таблиц.**

**Также, можно давать альтернативное имя столбцу product.id as id\_product**

**left join - создает общую таблицу, но в нее обязательно войдут все значения из главной таблицы + все подходящие из объединяемой. Пропуски будут заполнены null**

**right join - противоположный эффект**

**select product.\* from product - достает только значения из указанной таблицы при объединении**

**select \* from ..**

**union**

**select \* from ..**

**Объединяет два запроса в одну таблицу**

**Подсчет данных:**

**select count(\*) from product - возвращает количество записей**

**select sum(price) from product - возвращает сумму**

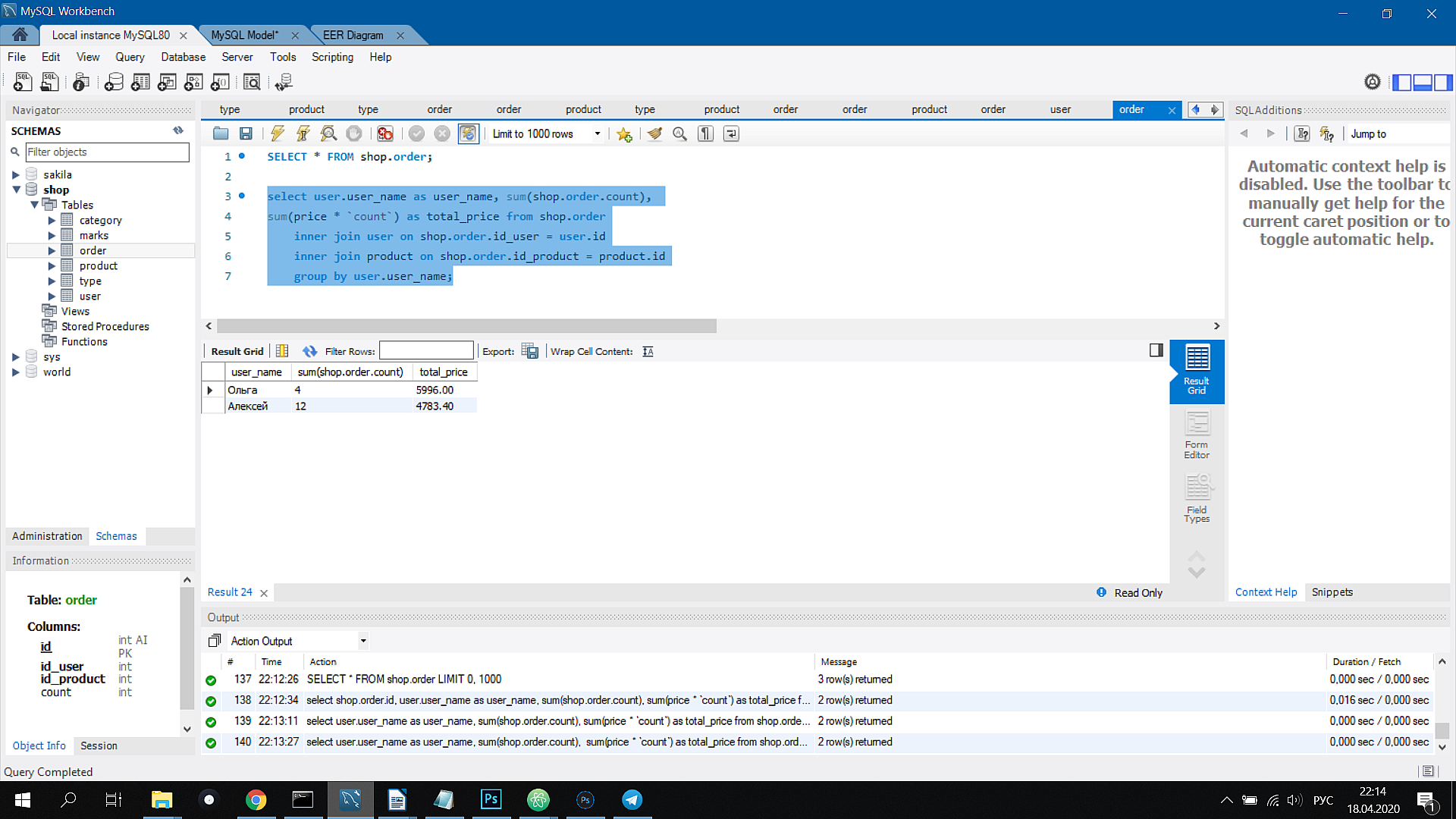
**select min/max(price) from product - возвращает минимальное/максимальное значение**

**sum(price \* `count`) as total\_price - сумма всех столбцов, которые подсчитываются с расчетом цены умноженной на количество**

**group by user.user\_name - группирует записи в одну по заданному столбцу. Также, указываем функцию, которая должна что-то подсчитать. Таким образом, возможно вывести множество записей с расчетом**

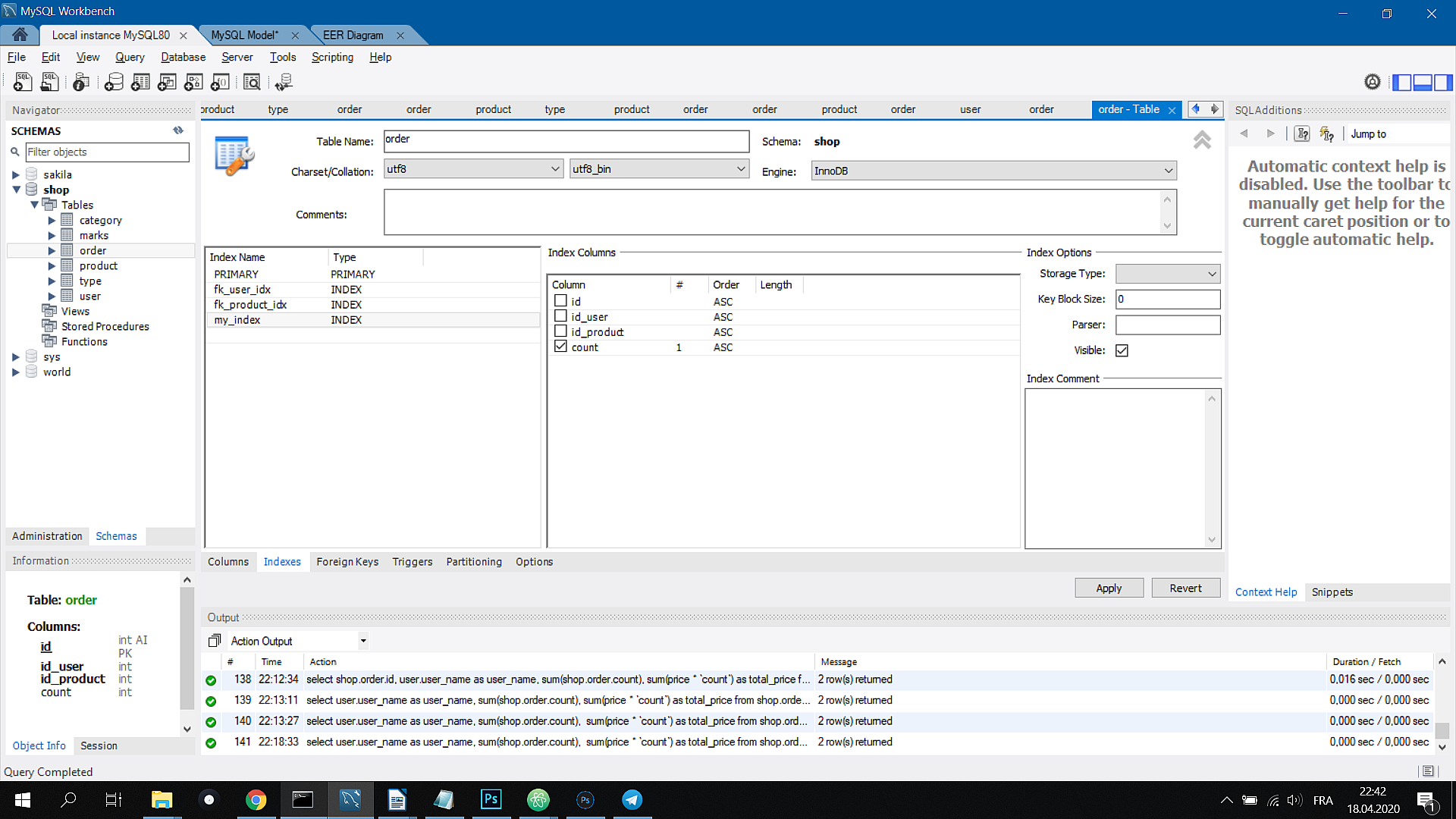
**Пример:**

**select user.user\_name as user\_name, sum(shop.order.count),   
sum(price \* `count`) as total\_price from shop.order  
 inner join user on shop.order.id\_user = user.id  
 inner join product on shop.order.id\_product = product.id  
group by user.user\_name;**

****

**Получаем общую сумму всех заказов каждого человека + общее количество товаров**

**Индексация:**

****

**Стоит применять к тем столбцам, которые часто запрашиваются для поиска и процесс занимает много времени. Это ускоряет его**

**Транзакции:**

**start transaction;**

**sql запрос ..**

**sql запрос ..**

**..**

**commit**

**Таким образом, будет выполнен весь блок команд, либо не выполнена никакая**