# 2. Разработка баз данных в СУБД MySQL

## 2.1. Построение EDR-модели

Существует ювелирная мастерская, производящая ремонт ювелирных изделий. Мастерская создаёт для себя цель удовлетворять спрос населения в ремонте ювелирных украшений и получение прибыли. Изделие выступает в качестве товара и как товар имеет определенные характеристики. Предприятие имеет своих клиентов, сведения о которых хранится определённое время, обговоренное руководством. Деятельность мастерской может быть описана данными, сгруппированными следующим образом: клиенты (код клиента, фамилия, имя, отчество, номер телефона); скидки (код скидки и размер скидки); ювелирное изделие (код изделия, вес, состав, код типа изделия); типы изделия (код типа изделия, тип изделия); заказы (код заказа, дата приёмки, дата выдачи, код изделия, код клиента, код услуги, цена, код скидки); услуги (код услуги, услуга).

В процессе проектирования ERD модели БД были выделены следующий сущности:

Clients, discount, jewellery, orders, product\_type, service.

Описание сущностей:

clients – хранит данные о клиентах.

discount – хранит данные о размерах скидки.

jewellery – хранит данные об ювелирных изделиях.

orders – хранит данные о заказах.

product\_type – хранит данные о типах ювелирного изделия.

service – хранит данные об услугах.

Описание атрибутов сущностей:

|  |  |  |  |
| --- | --- | --- | --- |
| **Clients** | | | |
| Поле | Тип | Атрибуты | Описание |
| idclients | INT | PK NN AI | Уникальный номер клиента |
| lastname | VARCHAR(45) | NN | Фамилия |
| firstname | VARCHAR(45) | NN | Имя |
| patronymic | VARCHAR(45) | NN | Отчество |
| phone | VARCHAR(45) | NN | Номер телефона |

|  |  |  |  |
| --- | --- | --- | --- |
| **Discount** | | | |
| Поле | Тип | Атрибуты | Описание |
| iddiscount | INT | PK NN AI | Уникальный номер скидки |
| discount | INT | NN | Размер скидки |

|  |  |  |  |
| --- | --- | --- | --- |
| **Jewellery** | | | |
| Поле | Тип | Атрибуты | Описание |
| idjewellery | INT | PK NN AI | Уникальный номер изделия |
| weight | FLOAT | NN | Вес изделия |
| material | VARCHAR(45) | NN | Состав изделия |
| fk\_product\_type | INT | NN | Номер типа изделия |

|  |  |  |  |
| --- | --- | --- | --- |
| **Orders** | | | |
| Поле | Тип | Атрибуты | Описание |
| idorder | INT | PK NN AI | Уникальный номер заказа |
| date\_of\_reseipt | DATE | NN | Дата приёмки |
| date\_of\_issue | DATE | NN | Дата выдачи |
| fk\_clients | INT | NN | Номер клиента |
| fk\_jewellery | INT | NN | Номер изделия |
| fk\_service | INT | NN | Номер услуги |
| price | INT | NN | Цена |
| fk\_discount | INT |  | Номер скидки |

|  |  |  |  |
| --- | --- | --- | --- |
| **Product\_type** | | | |
| Поле | Тип | Атрибуты | Описание |
| idproduct\_type | INT | PK NN AI | Уникальный номер типа изделия |
| nametype | VARCHAR(45) | NN | Тип изделия |

|  |  |  |  |
| --- | --- | --- | --- |
| **Service** | | | |
| Поле | Тип | Атрибуты | Описание |
| idservice | INT | PK NN AI | Уникальный номер услуги |
| nameservice | VARCHAR(45) | NN | Услуга |

Для соединения таблиц использовались следующие типы связей:

 - один ко многим вида: not-**identifying relationship**

**Данный вид связи свидетельствует о том, что данные подчинённой таблицы могут существовать без связи с главной таблицей.**

**После того как были установлены все отношения между сущностями получили следующую модель данных: см.** [Приложение 1](#_Приложение_1)**.**

На основе полученной модели была создана БД средствами мастера. По итогу работы мастера был получен следующий SQL код: см. [Приложение 2](#_Приложение_2._SQL).

## 2.2. Заполнение базы данных

Следующим шагом необходимо заполнить БД:

Таблица service

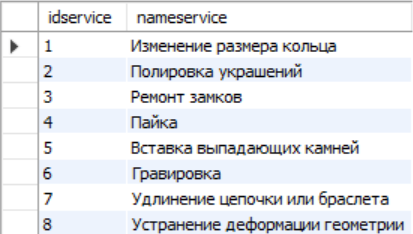


Таблица product\_type

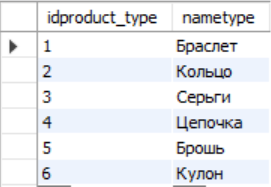


Таблица jewellery



Таблица discount

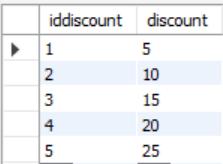


Таблица clients

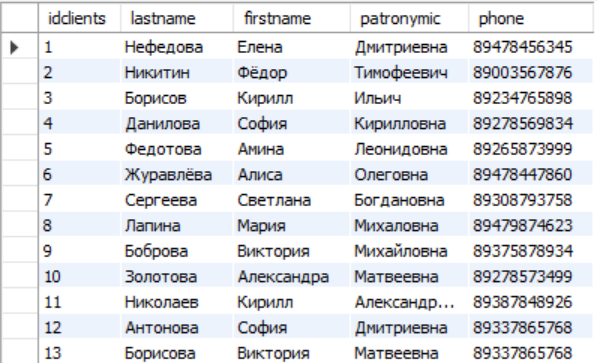
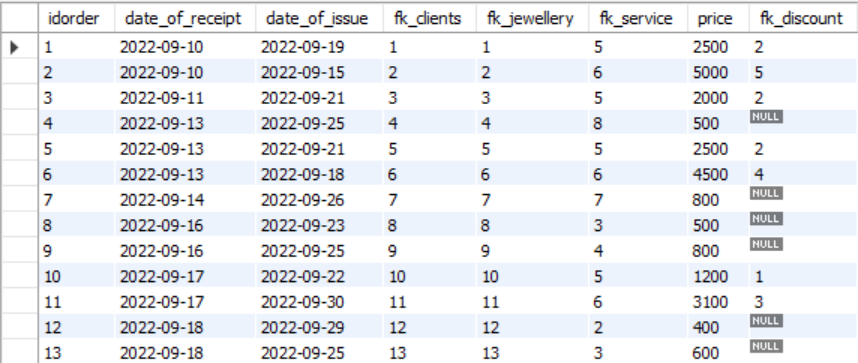


Таблица orders



При заполнении таблиц очень важна последовательность: сперва заполняются главные таблицы, затем подчиненные.

## 2.3. Написание запросов по работе с данными

### 2.3.1. Выборка и выборка с условием

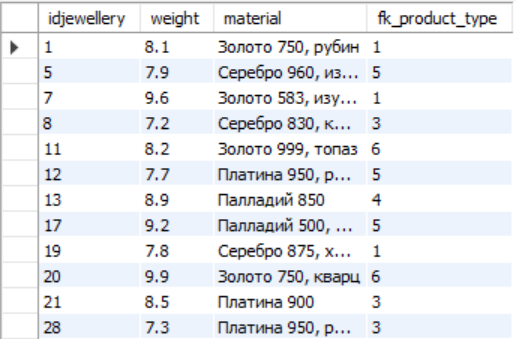
1. Выбрать данные из одной таблицы.

select \* from jewellery



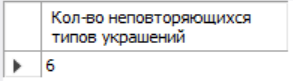
1. Выбрать данные из одной таблицы по условию WHERE

select \* from jewellery where weight > 7



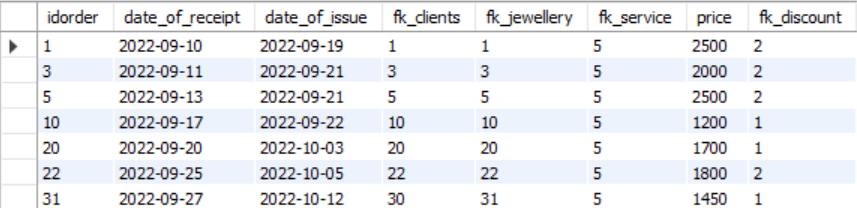
1. Выбрать данные с применением команды DISTINCT

select count(distinct fk\_product\_type) as 'Кол-во неповторяющихся типов украшений' from jewellery

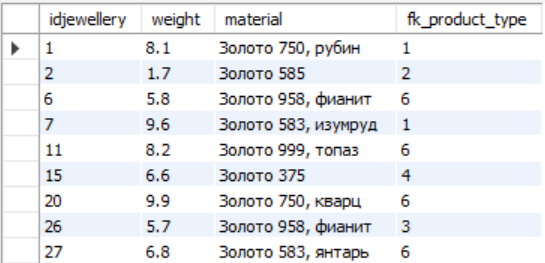


1. Выбрать данные из таблицы по условию WHERE с применением оператора вхождения в коллекцию – IN

select \* from orders where fk\_service in(5)

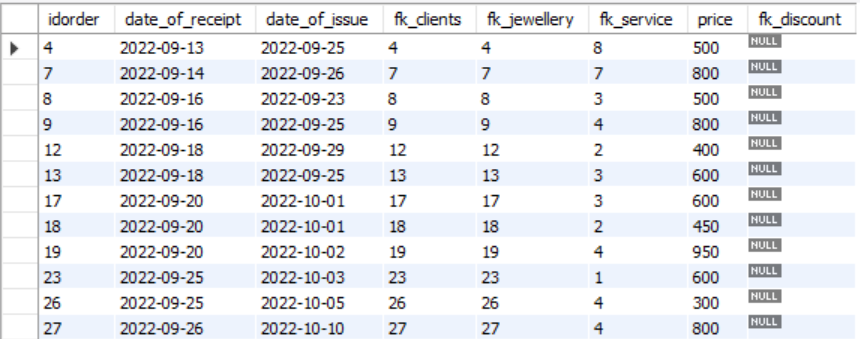


1. Выбрать данные из таблицы по условию WHERE с использованием оператора Like

select \* from jewellery where material like 'Золото%'

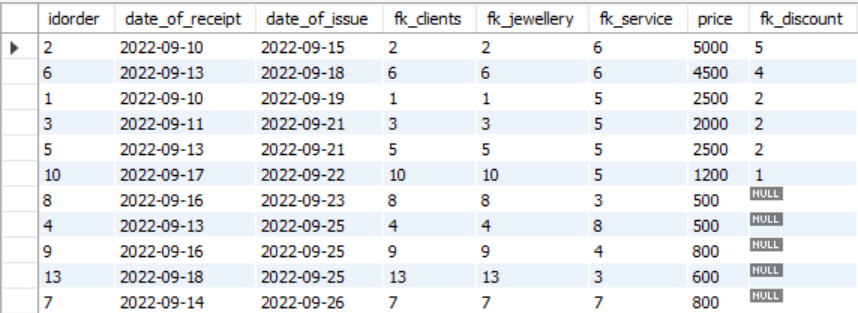
1. Выберите данные из таблицы с применением проверки данных на NULL (IS NULL, IS NOT NULL)

select \* from orders where fk\_discount is null



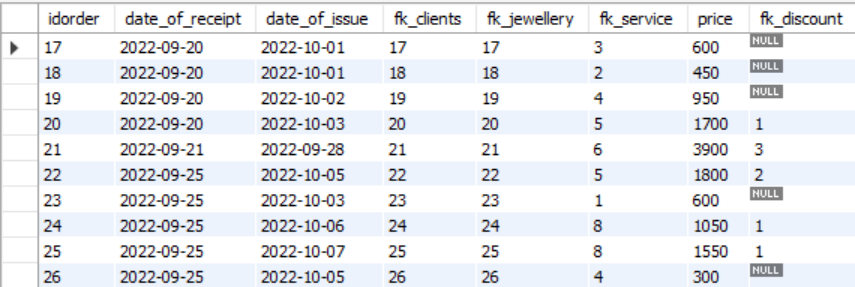
1. Выберите данные из таблицы с применением сортировки по выбранным полям

select \* from orders order by date\_of\_issue

****

1. Выбрать данные с применением оператора BETWEEN

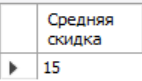
select \* from orders where date\_of\_receipt between '2022-09-20' and '2022-09-25'



### 2.3.2. Агрегатные функции

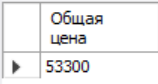
1. Avg

select round(avg(discount), 0) as 'Средняя скидка' from discount



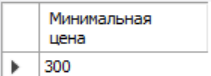
1. Sum

select sum(price) as 'Общая цена' from orders



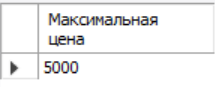
1. Min

select min(price) as 'Минимальная цена' from orders



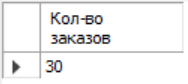
1. Max

select max(price) as 'Максимальная цена' from orders



1. Count

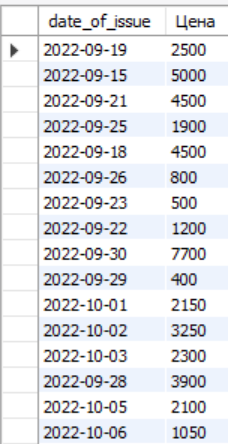
select count(idorder) as 'Кол-во заказов' from orders

****

### 2.3.3. Группировка

1. GROUP BY

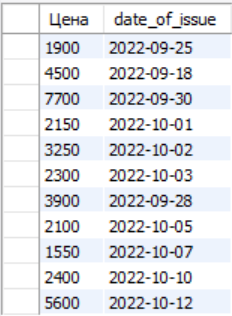
select date\_of\_issue, sum(price) as 'Цена' from orders group by date\_of\_issue



1. GROUP BY с использованием условия HAVING

SELECT sum(price) as 'Цена', date\_of\_issue FROM orders

group by date\_of\_issue having sum(price) > 1500

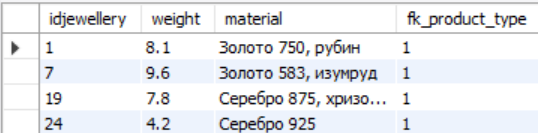


### 2.3.4. Подзапросы

1. Первый подзапрос

select \* from jewellery

where fk\_product\_type = (select idproduct\_type from product\_type where nametype = 'Браслет')

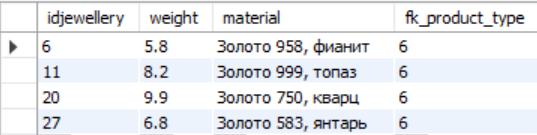


1. Второй подзапрос

select \* from jewellery j

where fk\_product\_type = (select idproduct\_type from product\_type where nametype = 'Кулон')

and j.weight > 5



1. Третий подзапрос

SELECT \* FROM orders

where fk\_service=(select idservice from service where nameservice = 'Изменение размера кольца')



### 2.3.5. Соединение таблиц

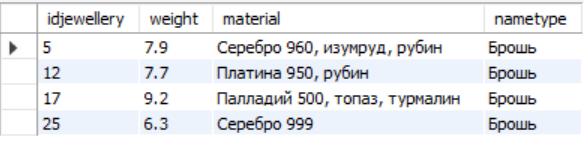
1. Inner Join

SELECT jewellery.idjewellery, jewellery.weight, jewellery.material, product\_type.nametype

FROM jewellery

JOIN product\_type ON jewellery.fk\_product\_type = product\_type.idproduct\_type

where nametype = 'Брошь'

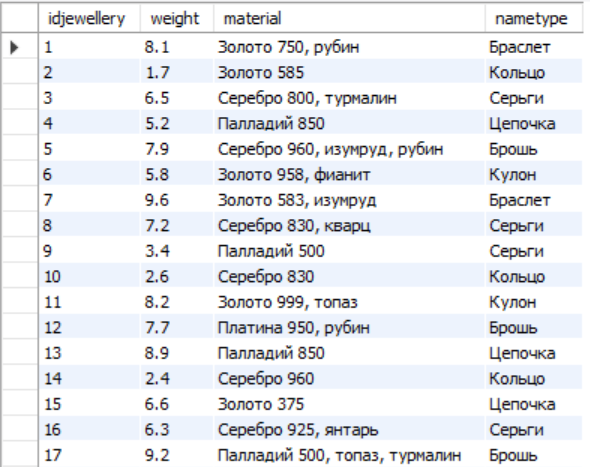


1. Left | Right Join

SELECT jewellery.idjewellery, jewellery.weight, jewellery.material, product\_type.nametype

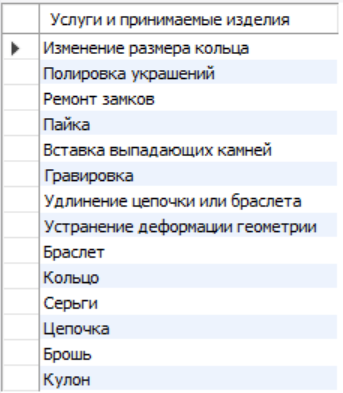
FROM jewellery

left JOIN product\_type ON jewellery.fk\_product\_type = product\_type.idproduct\_type



1. Union

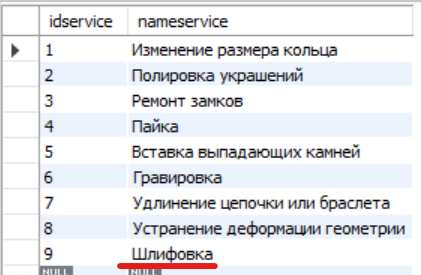
SELECT nameservice as 'Услуги и принимаемые изделия' FROM service UNION SELECT nametype FROM product\_type



### 2.3.6. Запросы на добавление, удаление, изменение данных

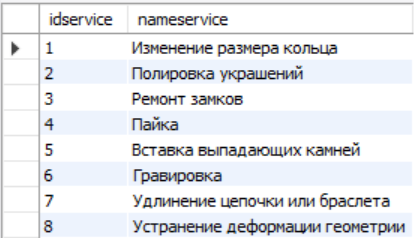
1. Запрос на добавление

insert into service (nameservice) values('Шлифовка')



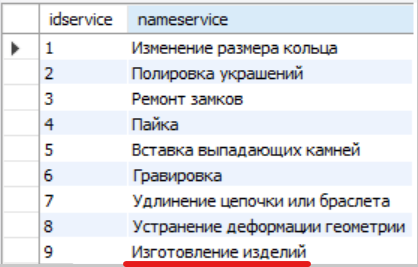
1. Запрос на удаление

delete from service where nameservice='Изготовление изделий'



1. Запрос на обновление

update service set nameservice='Изготовление изделий' where nameservice='Шлифовка'



### 2.3.7. Встроенные функции

1. Условие CASE

SELECT \*,

CASE

WHEN price < 500 THEN 'Дешёвая'

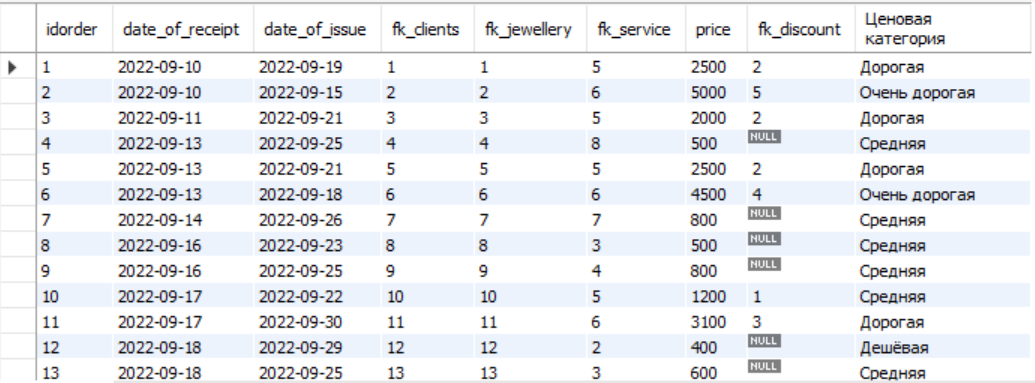
WHEN price < 2000 THEN 'Средняя'

WHEN price < 3500 THEN 'Дорогая'

else 'Очень дорогая'

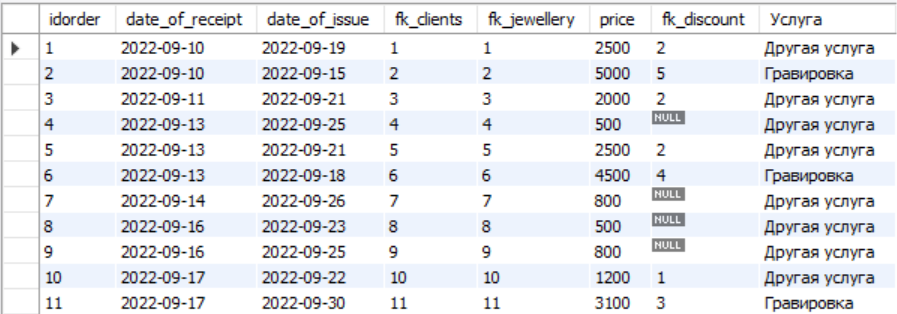
END as 'Ценовая категория'

FROM orders;



1. Условие IF

SELECT idorder, date\_of\_receipt, date\_of\_issue, fk\_clients, fk\_jewellery, price, fk\_discount, IF(fk\_service=6,"Гравировка","Другая услуга") AS 'Услуга' FROM orders



# 3. Работа со встроенными объектами СУБД MySQL

## 3.1. Работа с процедурами, функциями и представлениями

### 3.1.1. Основные понятия

Процедура - объект базы данных, представляющий собой набор SQL-инструкций, который компилируется один раз и хранится на сервере. Хранимые процедуры очень похожи на обыкновенные процедуры языков высокого уровня, у них могут быть входные и выходные параметры и локальные переменные, в них могут производиться числовые вычисления и операции над символьными данными, результаты которых могут присваиваться переменным и параметрам. В хранимых процедурах могут выполняться стандартные операции с базами данных. Кроме того, в хранимых процедурах возможны циклы и ветвления, то есть в них могут использоваться инструкции управления процессом исполнения.

Функция - представляет собой модуль, который возвращает значение командой RETURN (вместо аргументов OUT или IN OUT). В отличие от вызова процедуры, который представляет собой отдельный оператор, вызов функции всегда является частью исполняемого оператора, то есть включается в выражение или служит в качестве значения по умолчанию, присваиваемого переменной при объявлении.

Представление - это виртуальная таблица, основанная на результирующем наборе инструкции SQL. Представление содержит строки и столбцы, как и настоящая таблица. Поля в представлении - это поля из одной или нескольких реальных таблиц в базе данных. Вы можете добавить в представление инструкции SQL функций, WHERE и JOIN и представить данные так, как если бы они поступали из одной таблицы.

### 3.1.2. Практическая часть

**Процедуры.**

1. **pr\_1\_AddNewClient**

Описание процедуры.

Данная процедура добавляет в таблицу «Clients» нового клиента.

Код процедуры:

CREATE DEFINER=`root`@`localhost` PROCEDURE `pr\_1\_AddNewClient`()

BEGIN

insert into clients values (31, 'Маркова', 'Дарья', 'Матвеевна', '89455436452');

END

Вызов процедуры:

call jewelryshop.pr\_1\_AddNewClient()

Результат работы:



1. **pr\_2\_CountOrders**

Описание процедуры.

Данная процедура выводит кол-во заказов из таблицы «Orders»

Код процедуры:

CREATE DEFINER=`root`@`localhost` PROCEDURE `pr\_2\_CountOrders`()

BEGIN

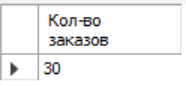
select count(idorder) as 'Кол-во заказов' from orders;

END

Вызов процедуры:

call jewelryshop.pr\_2\_CountOrders()

Результат работы:



1. **pr\_3\_FindOrderForPrice**

Описание процедуры.

Данная процедура выводит заказ из таблицы «Orders» где цена начинается от 1000 тысяч и заканчивается 2000.

Код процедуры:

CREATE DEFINER=`root`@`localhost` PROCEDURE `pr\_3\_FindOrderForPrice`(Начальная\_цена int, Конечная\_цена int)

BEGIN

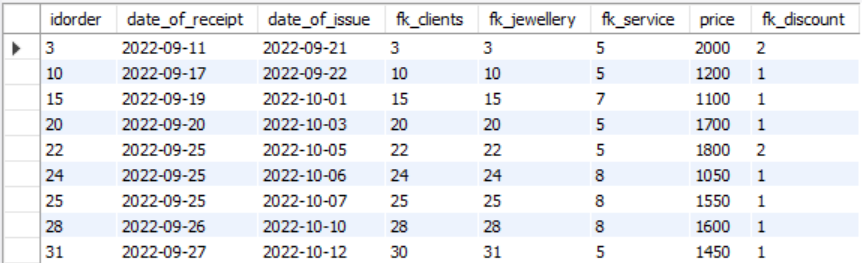
select \* from orders where price between Начальная\_цена and Конечная\_цена;

END

Вызов процедуры:

call jewelryshop.pr\_3\_FindOrderForPrice(1000, 2000)

Результат работы:



1. **pr\_4\_Service**

Описание процедуры.

Данная процедура выводит заказы из таблицы «Orders», и выводит «Изменение размера кольца», если выбрана эта услуга, и «Другая услуга», если другая услуга.

Код процедуры:

CREATE DEFINER=`root`@`localhost` PROCEDURE `pr\_4\_Service`()

BEGIN

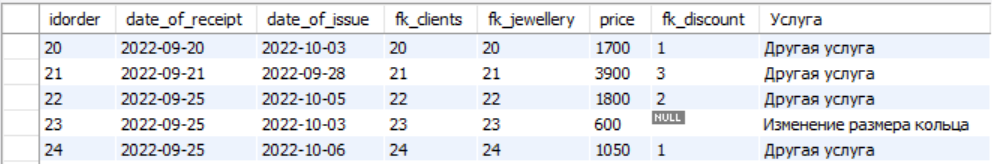
SELECT idorder, date\_of\_receipt, date\_of\_issue, fk\_clients, fk\_jewellery, price, fk\_discount, IF(fk\_service=1,"Изменение размера кольца","Другая услуга") AS 'Услуга' FROM orders;

END

Вызов процедуры:

call jewelryshop.pr\_4\_Service()

Результат работы:



1. **pr\_5\_Type**

Описание процедуры.

Данная функция выводит все кулоны из таблицы «jewellery»

Код процедуры:

CREATE DEFINER=`root`@`localhost` PROCEDURE `pr\_5\_Type`()

BEGIN

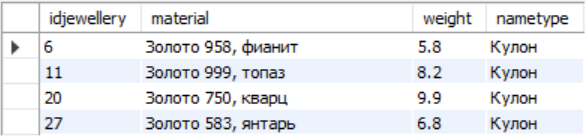
select jewellery.idjewellery, jewellery.material, jewellery.weight, product\_type.nametype from jewellery join product\_type on jewellery.fk\_product\_type = product\_type.idproduct\_type where nametype = 'Кулон';

END

Вызов процедуры:

call jewelryshop.pr\_5\_Type()

Результат работы:



**Функции**

1. **fn\_1\_CountJewelleryLikeZ**

Описание функции.

Данная функция выводит кол-во ювелирных изделий, название материала которых начинается на букву З из таблицы «jewellery»

Код функции:

CREATE DEFINER=`root`@`localhost` FUNCTION `fn\_1\_CountJewelleryLikeZ`() RETURNS int

DETERMINISTIC

BEGIN

declare count\_jewellery float;

select count(idjewellery) into count\_jewellery from jewellery where material like'З%';

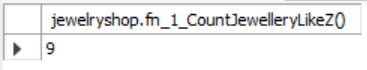
RETURN count\_jewellery;

END

Вызов функции:

select jewelryshop.fn\_1\_CountJewelleryLikeZ()

Результат работы:



1. **fn\_2\_MinWeight**

Описание функции.

Данная функция выводит наименьший вес изделия из таблицы «jewellery»

Код функции:

CREATE DEFINER=`root`@`localhost` FUNCTION `fn\_2\_MinWeight`() RETURNS float

DETERMINISTIC

BEGIN

declare min\_weight float;

select min(weight) into min\_weight from jewellery;

RETURN min\_weight;

END

Вызов функции:

select jewelryshop.fn\_2\_MinWeight()

Результат работы:



1. **fn\_3\_LOOP**

Описание функции.

Данная функция перебирает курсором даты приёмки, и коды из таблицы «orders» и где дата приёмки является «3.09.2022 меняет дату выдачи на 16.09.2022.

Код функции:

CREATE DEFINER=`root`@`localhost` FUNCTION `fn\_3\_LOOP`() RETURNS varchar(50) CHARSET utf8mb3

DETERMINISTIC

BEGIN

declare done int default false;

declare id int;

declare receipt date;

declare info varchar(45);

declare cur1 cursor for select idorder, date\_of\_receipt from orders;

declare continue handler for not found set done = true;

open cur1;

read\_loop: LOOP

fetch cur1 into id, receipt;

if done then leave read\_loop; end if;

if receipt = '2022-09-13' then update orders set date\_of\_issue = '2022-09-16' where id = idorder;

set info = 'Замена выполнена';

end if;

end loop;

close cur1;

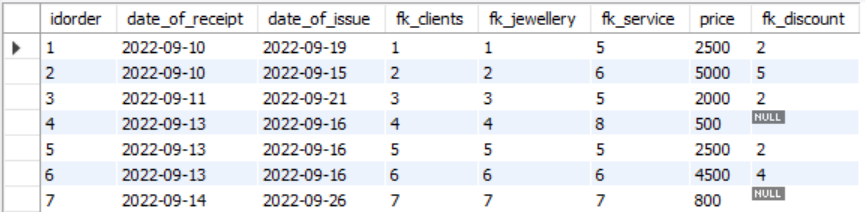
RETURN info;

END

Вызов функции:

select jewelryshop.fn\_3\_LOOP()

Результат работы:



1. **fn\_4\_DeleteClient**

Описание функции.

Данная функция удаляет запись по введенной фамилии из таблицы «clients»

Код функции:

CREATE DEFINER=`root`@`localhost` FUNCTION `fn\_4\_DeleteClient`(familia varchar(50)) RETURNS varchar(50) CHARSET utf8mb3

DETERMINISTIC

BEGIN

declare f int;

declare info varchar(100);

set f = 0;

select count(\*) into f from clients where lastname = familia;

if f = 1 then

delete from clients where lastname = familia limit 1000 ;

set info = 'Клиент удалён';

else set info = 'Такого клиента нет';

end if;

RETURN info;

END

Вызов функции:

select jewelryshop.fn\_4\_DeleteClient('Прохова')

Результат работы:

Клиент с фамилией «Прохова» удалён.



1. **fn\_5\_AddClient**

Описание функции.

Данная функция, пробует добавить новую запись в таблицу и, если запись с такими значениями уже есть выдает соответствующее сообщение.

Код функции:

CREATE DEFINER=`root`@`localhost` FUNCTION `fn\_5\_AddClient`(familia varchar(45), imya varchar(45), otchestvo varchar(45), telephone varchar(45)) RETURNS varchar(45) CHARSET utf8mb3

DETERMINISTIC

BEGIN

declare f int;

declare info varchar(100);

set f = 0;

select count(\*) into f from clients where phone = telephone;

if f = 0 then

insert into clients(lastname, firstname, patronymic, phone)

values(familia, imya, otchestvo, telephone);

set info = 'Клиент успешно добавлен';

else set info = 'Клиент с такими данными уже есть уже есть';

end if;

RETURN info;

END

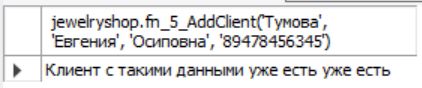
Вызов функции:

select jewelryshop.fn\_5\_AddClient('Тумова', 'Евгения', 'Осиповна', '89478456345');

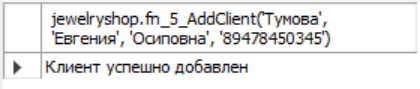
select jewelryshop.fn\_5\_AddClient('Тумова', 'Евгения', 'Осиповна', '89478450345')

Результат работы:

1. При попытке добавить уже существующего в базу



1. Добавление нового сотрудника



Представления

1. **new\_view1weight**

Описание представления.

Данное представление берет изделия из таблицы «jewellery», вес которых больше 4 грамм.

Код представления:

CREATE

ALGORITHM = UNDEFINED

DEFINER = `root`@`localhost`

SQL SECURITY DEFINER

VIEW `jewelryshop`.`new\_view1weight` AS

SELECT

`jewelryshop`.`jewellery`.`idjewellery` AS `idjewellery`,

`jewelryshop`.`jewellery`.`weight` AS `weight`,

`jewelryshop`.`jewellery`.`material` AS `material`,

`jewelryshop`.`jewellery`.`fk\_product\_type` AS `fk\_product\_type`

FROM

`jewelryshop`.`jewellery`

WHERE

(`jewelryshop`.`jewellery`.`weight` > 4)

Вызов представления:

SELECT \* FROM jewelryshop.new\_view1weight;

Результат работы:



1. **new\_view2minprice**

Описание представления.

Данное представление выводит даты приёмки и выдачи и минимальную цену заказа из таблицы «orders»

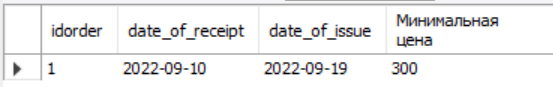
Код представления:

CREATE ALGORITHM=UNDEFINED DEFINER=`root`@`localhost` SQL SECURITY DEFINER VIEW `jewelryshop`.`new\_view2minprice` AS select `jewelryshop`.`orders`.`idorder` AS `idorder`,`jewelryshop`.`orders`.`date\_of\_receipt` AS `date\_of\_receipt`,`jewelryshop`.`orders`.`date\_of\_issue` AS `date\_of\_issue`,min(`jewelryshop`.`orders`.`price`) AS `Минимальная цена` from `jewelryshop`.`orders`

Вызов представления:

SELECT \* FROM jewelryshop.new\_view2minprice

Результат работы:



1. **v\_3\_clientpurchase**

Описание представления.

Данное представление объединяет 2 таблицы «product\_type» и «jewellery».

Выводит номер изделия, вес, материла и тип.

Код представления:

CREATE

ALGORITHM = UNDEFINED

DEFINER = `root`@`localhost`

SQL SECURITY DEFINER

VIEW `jewelryshop`.`new\_view3jewellerytype` AS

SELECT

`jewelryshop`.`jewellery`.`idjewellery` AS `idjewellery`,

`jewelryshop`.`jewellery`.`weight` AS `weight`,

`jewelryshop`.`jewellery`.`material` AS `material`,

`jewelryshop`.`product\_type`.`nametype` AS `nametype`

FROM

(`jewelryshop`.`jewellery`

JOIN `jewelryshop`.`product\_type` ON ((`jewelryshop`.`product\_type`.`idproduct\_type` = `jewelryshop`.`jewellery`.`fk\_product\_type`)))

Вызов представления:

SELECT \* FROM jewelryshop.new\_view3jewellerytype;

Результат работы:



1. **v\_4\_price**

Описание представления.

Данное представление выводит фамилию и телефон клиента, услугу и максимальную цену из таблицы «orders».

Код представления:

CREATE

ALGORITHM = UNDEFINED

DEFINER = `root`@`localhost`

SQL SECURITY DEFINER

VIEW `jewelryshop`.`new\_view4maxclientserviceprice` AS

SELECT

`jewelryshop`.`clients`.`lastname` AS `lastname`,

`jewelryshop`.`clients`.`phone` AS `phone`,

`jewelryshop`.`service`.`nameservice` AS `nameservice`,

MAX(`jewelryshop`.`orders`.`price`) AS `max(price)`

FROM

((`jewelryshop`.`orders`

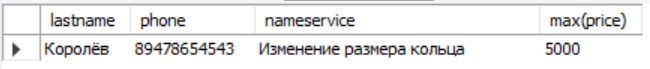
JOIN `jewelryshop`.`clients` ON ((`jewelryshop`.`orders`.`fk\_clients` = `jewelryshop`.`clients`.`idclients`)))

JOIN `jewelryshop`.`service` ON ((`jewelryshop`.`orders`.`fk\_service` = `jewelryshop`.`service`.`idservice`)))

Вызов представления:

SELECT \* FROM jewelryshop.new\_view4maxclientserviceprice

Результат работы:



1. **new\_view5between**

Описание представления.

Данное представление выводит покупки из таблицы «щквукы», где дата покупки между 17.09.2022 и 20.09.2022

Код представления:

CREATE

ALGORITHM = UNDEFINED

DEFINER = `root`@`localhost`

SQL SECURITY DEFINER

VIEW `jewelryshop`.`new\_view5between` AS

SELECT

`jewelryshop`.`orders`.`idorder` AS `idorder`,

`jewelryshop`.`orders`.`date\_of\_receipt` AS `date\_of\_receipt`,

`jewelryshop`.`orders`.`date\_of\_issue` AS `date\_of\_issue`,

`jewelryshop`.`orders`.`fk\_clients` AS `fk\_clients`,

`jewelryshop`.`orders`.`fk\_jewellery` AS `fk\_jewellery`,

`jewelryshop`.`orders`.`fk\_service` AS `fk\_service`,

`jewelryshop`.`orders`.`price` AS `price`,

`jewelryshop`.`orders`.`fk\_discount` AS `fk\_discount`

FROM

`jewelryshop`.`orders`

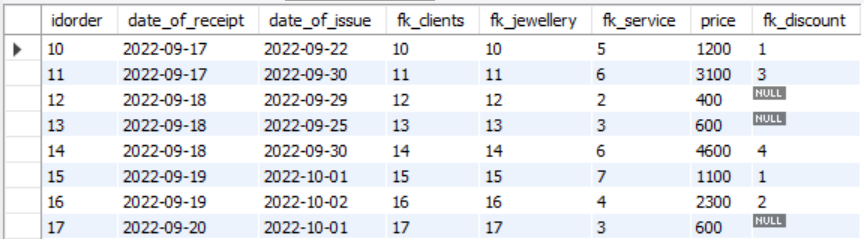
WHERE

(`jewelryshop`.`orders`.`date\_of\_receipt` BETWEEN '2022-09-17' AND '2022-09-20')

Вызов представления:

SELECT \* FROM jewelryshop.new\_view5between

Результат работы:



## 3.2. Пользователи и его права

Пользователь базы данных — это идентификатор имени входа при подключении к базе данных. Имя пользователя базы данных может совпадать с именем входа, но это не является обязательным требованием.

Права — это возможность выполнения конкретного типа SQL-оператора или доступа к объекту базы данных, принадлежащему другому пользователю.

### 3.2.1. Практическая часть. Работа с пользователями и правами через написание команд

1. Создайте нового пользователя со своим именем латиницей.

CREATE USER ekaterina@localhost IDENTIFIED BY '1234';

1. Присвойте пользователю все привилегии на вашу БД выбрав в ней одну любую таблицу.

grant all privileges on jewelryshop.clients to 'ekaterina'@'localhost';

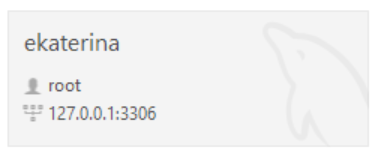
flush privileges;

1. Выведите таблицу mysql.user и убедитесь, что созданный вами пользователь в ней есть.

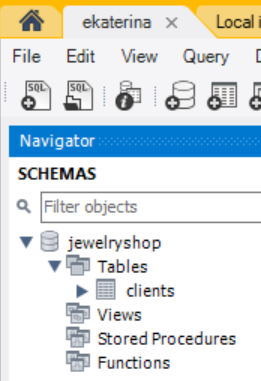
select \* from mysql.user



1. Создайте новое подключение для созданного пользователя.



1. Подключитесь используя данное подключение.



1. Посмотрите, что вам доступна именно та БД и та таблица, которой вы назначили привилегии для вашего пользователя.



1. Удалите данного пользователя.

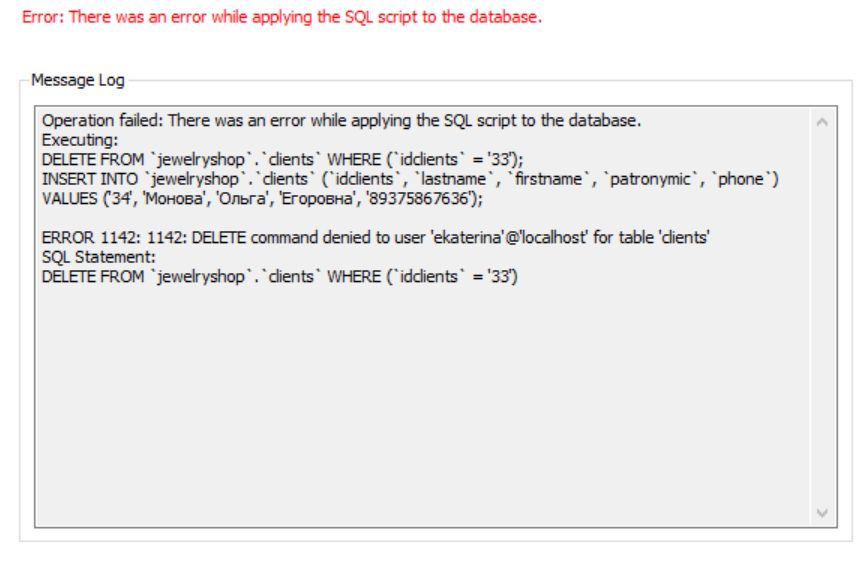
drop user 'ekaterina'@'localhost'

1. Создайте заново его-же, но уже с привилегиями: SELECT, UPDATE, INSERT.

CREATE USER ekaterina@localhost IDENTIFIED BY '1234';

grant select, update, insert on jewelryshop.clients to 'ekaterina'@'localhost'

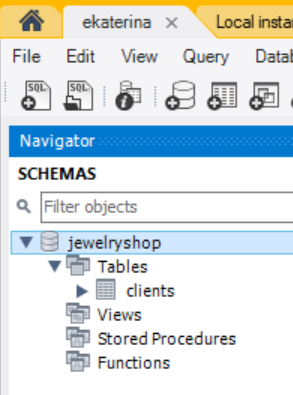
1. Убедитесь, что никакие другие действия с таблицей не разрешены.



1. Измените пароль своему пользователю на «1111»

ALTER USER 'ekaterina'@'localhost' IDENTIFIED BY '1111'

1. Перезайдите в СУБД с новым паролем.

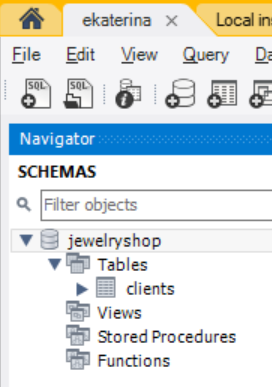


1. Удалите пароль своему пользователю.

update mysql.user set authentication\_string=null where User='ekaterina' limit 1000;

flush privileges;

1. Перезайдите заново в СУБД (*уже не вводя пароль*)



1. Добавьте пароль своему пользователю «1234»

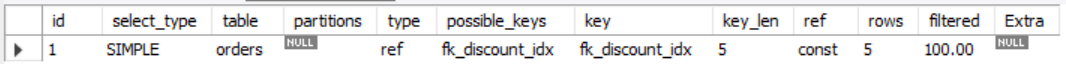
alter user 'ekaterina'@'localhost' identified with caching\_sha2\_password by '1234'

## 3.3. Индексы

**Выбор индексов в MySQL**

Выполняем запрос без индекса.

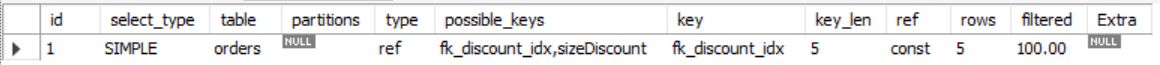
explain select \* from orders where fk\_discount = 2



Создаем индекс.

CREATE INDEX sizeDiscount ON orders(fk\_discount);

explain select \* from orders where fk\_discount = 2



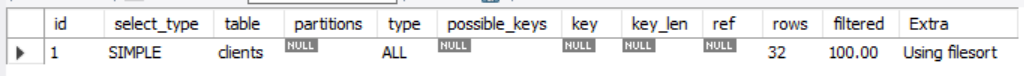
Колонка rows показывает число записей, которые пришлось прочитать базе данных для выполнения этого запроса.

**Вывод:** запрос после создания индекса «sizeDiscount» работает быстрее.

**Сортировка.**

Выполним сортировку по фамилиям.

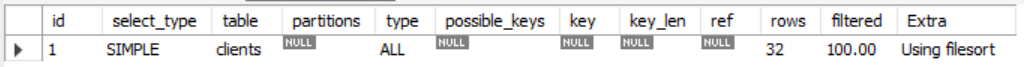
explain select \* from clients order by lastname



Создадим индекс.

CREATE INDEX familia ON clients(lastname);

explain select \* from clients order by lastname



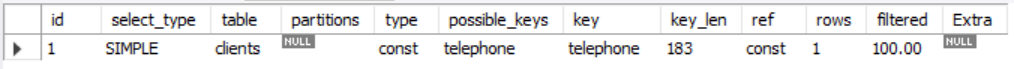
Вывод: создание индекса для поля «lastname» особо не повлияла на скорость выполнения запроса.

**Уникальные индексы**

Создаю уникальный индекс для поля «Address»

EXPLAIN SELECT \* FROM clients WHERE phone = '89478456345' 

CREATE UNIQUE INDEX telephone ON clients(phone);

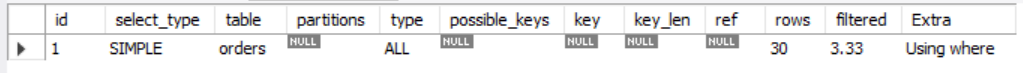
EXPLAIN SELECT \* FROM clients WHERE phone = '89337865768'; 

**Вывод:** запрос выполняется быстрее с уникальным индексом.

**Составные индексы**

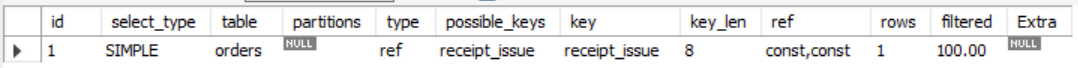
Запрос на выборку данных по дате приёмки и выдачи.

SELECT \* FROM orders WHERE date\_of\_receipt = '2022-09-18' AND date\_of\_issue = '2022-09-25';

EXPLAIN SELECT \* from orders WHERE date\_of\_receipt = '2022-09-18' AND date\_of\_issue = '2022-09-25' 

Создаю индекс «receipt\_issue»

CREATE INDEX receipt\_issue ON orders(date\_of\_receipt, date\_of\_issue);

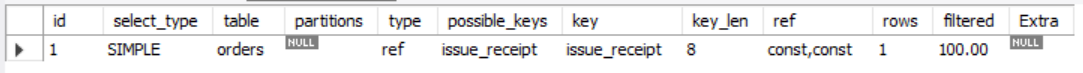
EXPLAIN SELECT \* FROM orders WHERE date\_of\_receipt = '2022-09-18' AND date\_of\_issue = '2022-09-25' 

Удаляю индекс receipt\_issue.

drop index `receipt\_issue` on orders

Создаю индекс «issue\_receipt».

CREATE INDEX issue\_receipt ON orders(date\_of\_issue, date\_of\_receipt);

EXPLAIN SELECT \* FROM orders WHERE date\_of\_receipt = '2022-09-18' AND date\_of\_issue = '2022-09-25' 

**Вывод:** в первом случае (без индексов) будет проверено все 30 записей таблицы. После создания индексов «receipt\_issue» и «issue\_receipt» будет проверена 1 строка. Т.к. при индексах «receipt\_issue» и «issue\_receipt» условие date\_of\_issue = '2022-09-25' будет найдено сразу же.

## 3.4. Исключения

### 3.4.1. Практическая часть

1. Напишите процедуру «create\_table\_clients», которая создает таблицу «clients», обработав ошибку номер 1050 (таблица уже существует)

Скрипт:

CREATE DEFINER=`root`@`localhost` PROCEDURE ` create\_table\_clients`()

BEGIN

declare exit handler for 1050

select 'Таблица clients уже существует' as msg;

create table clients(

id int primary key,

lastname varchar(45),

firstname varchar(45),

patronymic varchar(45),

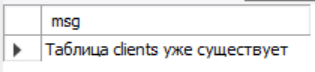
phone varchar(45));

END

Вызываем процедуру:

call jewelryshop.create\_table\_clients()

Результат:



1. Напишите процедуру «found\_table», которая выбирает данные из указанной таблицы. Если такой таблицы нет, то выводится сообщение об ошибке 1146 (Таблица не существует)

Скрипт:

CREATE DEFINER=`root`@`localhost` PROCEDURE `found\_table`(in tbl\_name varchar(45))

BEGIN

declare exit handler for 1146

select concat('Таблицы "',tbl\_name,'" не существует!') as msg;

set @qr = concat('select \* from ', tbl\_name);

prepare result from @qr;

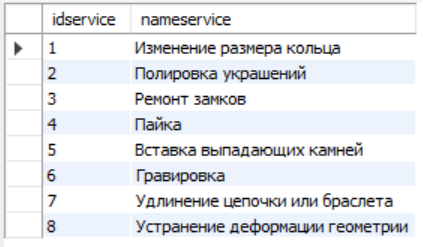
execute result;

END

Вызываем процедуру с существующей таблицей:

call jewelryshop.found\_table('service')

Результат:



Вызываем процедуру с несуществующей таблицей:

call jewelryshop.found\_table('price')

Результат:



1. Напишите процедуру «select\_column\_jewellery», которая выводит данные указанного столбца таблицы «имя таблицы», а в случае если такого столбца нет, выводит ошибку. (код ошибки 1054)

Скрипт:

CREATE DEFINER=`root`@`localhost` PROCEDURE `select\_column\_jewellery`(in namecolumn varchar(45))

BEGIN

declare exit handler for 1054

select concat('Не существующий столбец "',namecolumn,'"') as msg;

set @qr = concat('select ',namecolumn,' from jewellery');

prepare result from @qr;

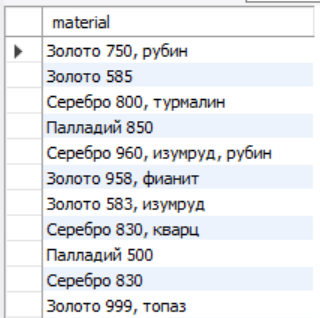
execute result;

END

Вызываем процедуру с существующим столбцом:

call jewelryshop.select\_column\_jewellery('material')

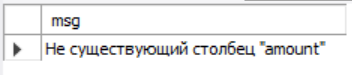
Результат:



Вызываем процедуру с несуществующим столбцом:

call jewelryshop.select\_column\_jewellery('amount')

Результат:



1. Напишите процедуру «create\_db», которая создает БД, а в случае если такая БД есть, выводит ошибку. (код ошибки 1007)

Скрипт:

CREATE DEFINER=`root`@`localhost` PROCEDURE `create\_db`(in namedb varchar(45))

BEGIN

declare exit handler for 1007

select concat('База данных "',namedb,'" уже существует') as msg;

set @qr = concat('create database ', namedb);

prepare result from @qr;

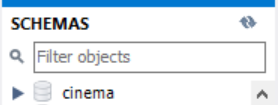
execute result;

END

Вызываем процедуру с несуществующей БД:

call jewelryshop.create\_db('cinema')

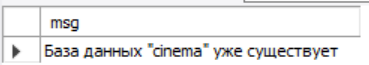
Результат:



Вызываем процедуру с существующей БД:

call jewelryshop.create\_db('cinema')

Результат:



1. Напишите процедуру «drop\_db», которая удаляет БД, а в случае если такой БД нет, выводит ошибку. (код ошибки 1008)

Скрипт:

CREATE DEFINER=`root`@`localhost` PROCEDURE `drop\_db`(in namedb varchar(45))

BEGIN

declare exit handler for 1008

select concat('Базы данных "',namedb,'" не существует') as msg;

set @qr = concat('drop database ', namedb);

prepare result from @qr;

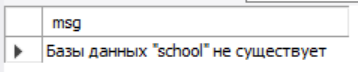
execute result;

END

Вызываем процедуру с несуществующей БД:

call jewelryshop.drop\_db('school')

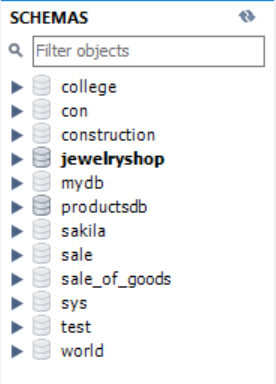
Результат:



Вызываем процедуру с существующей БД:

call jewelryshop.drop\_db('cinema')

Результат:



## 3.5. Триггеры

### 3.5.1 Практическая часть

Создаем таблицу log.

Скрипт:

CREATE TABLE `log` (

`id` int unsigned NOT NULL AUTO\_INCREMENT,

`msg` varchar(255) NOT NULL,

`time` timestamp NOT NULL DEFAULT CURRENT\_TIMESTAMP,

`row\_id` int NOT NULL,

`table\_name` varchar(45) NOT NULL,

PRIMARY KEY (`id`)

) ENGINE=MyISAM

1. Создаем триггер «insert\_service»

Скрипт:

CREATE DEFINER=`root`@`localhost` TRIGGER `insert\_service` AFTER INSERT ON `service` FOR EACH ROW BEGIN

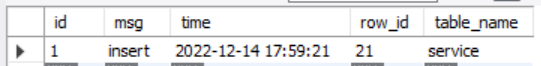
INSERT INTO log Set msg = 'insert', row\_id = NEW.idservice, table\_name = 'service';

END

Запрос:

INSERT INTO `jewelryshop`.`service`(`nameservice`) VALUES ('Изготовление изделий')

Результат:



1. Создаем триггер «clients\_BEFORE\_DELETEclient»

Скрипт:

CREATE DEFINER=`root`@`localhost` TRIGGER `clients\_BEFORE\_DELETEclient` BEFORE DELETE ON `clients` FOR EACH ROW BEGIN declare k int;

select count(\*) into k from clients C join orders O on C.idclients=O.fk\_clients where C.idclients = old.idclients;

if k > 0 then

signal sqlstate '45000'

set message\_text = 'Нельзя удалить этого клиента';

end if;

END

Запрос:

delete from clients where idclients=30

Результат:



1. Создаем триггер «update\_service»

Скрипт:

CREATE DEFINER=`root`@`localhost` TRIGGER `update\_service` AFTER UPDATE ON `service` FOR EACH ROW BEGIN

INSERT INTO log Set msg = 'update', row\_id = NEW.idservice, table\_name = 'service';

END

Запрос:

UPDATE `jewelryshop`.`service` SET `nameservice` = 'Шлифовка' WHERE (`idservice` = '21')

Результат:



1. Создаем триггер «jewellery\_BEFORE\_INSERT»

Скрипт:

CREATE DEFINER=`root`@`localhost` TRIGGER `jewellery\_BEFORE\_INSERT` BEFORE INSERT ON `jewellery` FOR EACH ROW BEGIN

IF NEW.weight < 0.1 THEN

SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'Вес не может быть отрицательным или быть равным 0';

END IF;

END

Запрос:

Insert `jewelryshop`.`jewellery`(`weight`,`material`,`fk\_product\_type`) values (-2, 'Золото 585', 3)

Результат:



1. Создаем триггер «delete\_service»

Скрипт:

CREATE DEFINER=`root`@`localhost` TRIGGER `delete\_service` AFTER DELETE ON `service` FOR EACH ROW BEGIN

INSERT INTO log Set msg = 'delete', row\_id = OLD.idservice, table\_name = 'service';

END

Запрос:

DELETE FROM `jewelryshop`.`service` WHERE (`idservice` = '21')

Результат:



1. Создаем триггер «delete \_client»

Скрипт:

CREATE DEFINER=`root`@`localhost` TRIGGER `delete\_client` AFTER DELETE ON `clients` FOR EACH ROW BEGIN

INSERT INTO log Set msg = 'delete', row\_id = OLD.idclients, table\_name = 'clients';

END

Запрос:

DELETE FROM `jewelryshop`.`clients` WHERE (`idclients` = '33')

Результат:



# Приложения

## Приложение 1. EDR - модель

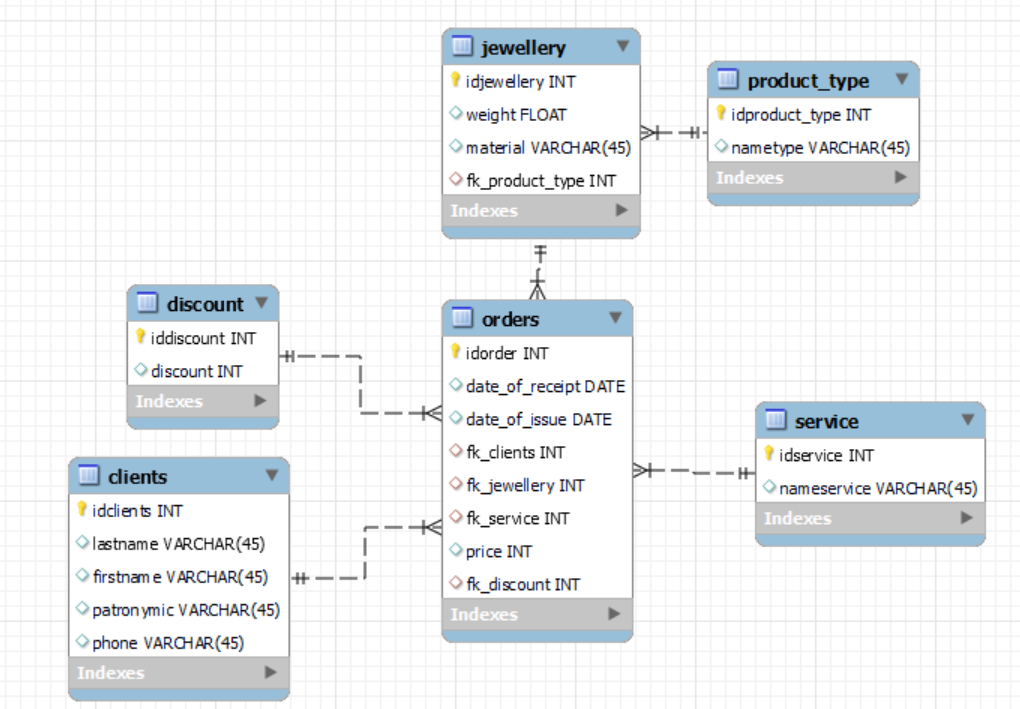


Рис. 1 Модель БД «jewelryshop»

## Приложение 2. SQL код создания БД «jewelryshop»

-- MySQL Workbench Forward Engineering

SET @OLD\_UNIQUE\_CHECKS=@@UNIQUE\_CHECKS, UNIQUE\_CHECKS=0;

SET @OLD\_FOREIGN\_KEY\_CHECKS=@@FOREIGN\_KEY\_CHECKS, FOREIGN\_KEY\_CHECKS=0;

SET @OLD\_SQL\_MODE=@@SQL\_MODE, SQL\_MODE='ONLY\_FULL\_GROUP\_BY,STRICT\_TRANS\_TABLES,NO\_ZERO\_IN\_DATE,NO\_ZERO\_DATE,ERROR\_FOR\_DIVISION\_BY\_ZERO,NO\_ENGINE\_SUBSTITUTION';

-- -----------------------------------------------------

-- Schema jewelryshop

-- -----------------------------------------------------

-- -----------------------------------------------------

-- Schema jewelryshop

-- -----------------------------------------------------

CREATE SCHEMA IF NOT EXISTS `jewelryshop` DEFAULT CHARACTER SET utf8mb4 COLLATE utf8mb4\_0900\_ai\_ci ;

USE `jewelryshop` ;

-- -----------------------------------------------------

-- Table `jewelryshop`.`clients`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `jewelryshop`.`clients` (

`idclients` INT NOT NULL AUTO\_INCREMENT,

`lastname` VARCHAR(45) NOT NULL DEFAULT NULL,

`firstname` VARCHAR(45) NOT NULL DEFAULT NULL,

`patronymic` VARCHAR(45) NOT NULL DEFAULT NULL,

`phone` VARCHAR(45) NOT NULL DEFAULT NULL,

PRIMARY KEY (`idclients`),

UNIQUE INDEX `telephone` (`phone` ASC) VISIBLE,

INDEX `familia` (`lastname` ASC) VISIBLE)

ENGINE = InnoDB

AUTO\_INCREMENT = 35

DEFAULT CHARACTER SET = utf8mb4

COLLATE = utf8mb4\_0900\_ai\_ci;

-- -----------------------------------------------------

-- Table `jewelryshop`.`discount`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `jewelryshop`.`discount` (

`iddiscount` INT NOT NULL AUTO\_INCREMENT,

`discount` INT NOT NULL DEFAULT NULL,

PRIMARY KEY (`iddiscount`))

ENGINE = InnoDB

AUTO\_INCREMENT = 6

DEFAULT CHARACTER SET = utf8mb4

COLLATE = utf8mb4\_0900\_ai\_ci;

-- -----------------------------------------------------

-- Table `jewelryshop`.`product\_type`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `jewelryshop`.`product\_type` (

`idproduct\_type` INT NOT NULL AUTO\_INCREMENT,

`nametype` VARCHAR(45) NOT NULL DEFAULT NULL,

PRIMARY KEY (`idproduct\_type`))

ENGINE = InnoDB

AUTO\_INCREMENT = 7

DEFAULT CHARACTER SET = utf8mb4

COLLATE = utf8mb4\_0900\_ai\_ci;

-- -----------------------------------------------------

-- Table `jewelryshop`.`jewellery`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `jewelryshop`.`jewellery` (

`idjewellery` INT NOT NULL AUTO\_INCREMENT,

`weight` FLOAT NOT NULL DEFAULT NULL,

`material` VARCHAR(45) NOT NULL DEFAULT NULL,

`fk\_product\_type` INT NOT NULL DEFAULT NULL,

PRIMARY KEY (`idjewellery`),

INDEX `fk\_product\_type\_idx` (`fk\_product\_type` ASC) VISIBLE,

CONSTRAINT `fk\_product\_type`

FOREIGN KEY (`fk\_product\_type`)

REFERENCES `jewelryshop`.`product\_type` (`idproduct\_type`))

ENGINE = InnoDB

AUTO\_INCREMENT = 35

DEFAULT CHARACTER SET = utf8mb4

COLLATE = utf8mb4\_0900\_ai\_ci;

-- -----------------------------------------------------

-- Table `jewelryshop`.`log`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `jewelryshop`.`log` (

`id` INT UNSIGNED NOT NULL AUTO\_INCREMENT,

`msg` VARCHAR(255) NOT NULL,

`time` TIMESTAMP NOT NULL DEFAULT CURRENT\_TIMESTAMP,

`row\_id` INT NOT NULL,

`table\_name` VARCHAR(45) NOT NULL,

PRIMARY KEY (`id`))

ENGINE = MyISAM

AUTO\_INCREMENT = 6

DEFAULT CHARACTER SET = utf8mb4

COLLATE = utf8mb4\_0900\_ai\_ci;

-- -----------------------------------------------------

-- Table `jewelryshop`.`service`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `jewelryshop`.`service` (

`idservice` INT NOT NULL AUTO\_INCREMENT,

`nameservice` VARCHAR(45) NOT NULL DEFAULT NULL,

PRIMARY KEY (`idservice`))

ENGINE = InnoDB

AUTO\_INCREMENT = 23

DEFAULT CHARACTER SET = utf8mb4

COLLATE = utf8mb4\_0900\_ai\_ci;

-- -----------------------------------------------------

-- Table `jewelryshop`.`orders`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `jewelryshop`.`orders` (

`idorder` INT NOT NULL AUTO\_INCREMENT,

`date\_of\_receipt` DATE NOT NULL DEFAULT NULL,

`date\_of\_issue` DATE NOT NULL DEFAULT NULL,

`fk\_clients` INT NOT NULL DEFAULT NULL,

`fk\_jewellery` INT NOT NULL DEFAULT NULL,

`fk\_service` INT NOT NULL DEFAULT NULL,

`price` INT NOT NULL DEFAULT NULL,

`fk\_discount` INT NULL DEFAULT NULL,

PRIMARY KEY (`idorder`),

INDEX `fk\_clients\_idx` (`fk\_clients` ASC) VISIBLE,

INDEX `fk\_jewellery\_idx` (`fk\_jewellery` ASC) VISIBLE,

INDEX `fk\_service\_idx` (`fk\_service` ASC) VISIBLE,

INDEX `fk\_discount\_idx` (`fk\_discount` ASC) VISIBLE,

INDEX `sizeDiscount` (`fk\_discount` ASC) VISIBLE,

INDEX `issue\_receipt` (`date\_of\_issue` ASC, `date\_of\_receipt` ASC) VISIBLE,

CONSTRAINT `fk\_clients`

FOREIGN KEY (`fk\_clients`)

REFERENCES `jewelryshop`.`clients` (`idclients`),

CONSTRAINT `fk\_discount`

FOREIGN KEY (`fk\_discount`)

REFERENCES `jewelryshop`.`discount` (`iddiscount`),

CONSTRAINT `fk\_jewellery`

FOREIGN KEY (`fk\_jewellery`)

REFERENCES `jewelryshop`.`jewellery` (`idjewellery`),

CONSTRAINT `fk\_service`

FOREIGN KEY (`fk\_service`)

REFERENCES `jewelryshop`.`service` (`idservice`))

ENGINE = InnoDB

AUTO\_INCREMENT = 32

DEFAULT CHARACTER SET = utf8mb4

COLLATE = utf8mb4\_0900\_ai\_ci;

SET SQL\_MODE=@OLD\_SQL\_MODE;

SET FOREIGN\_KEY\_CHECKS=@OLD\_FOREIGN\_KEY\_CHECKS;

SET UNIQUE\_CHECKS=@OLD\_UNIQUE\_CHECKS;