**Создание таблиц**

*CREATE* [TEMPORARY] TABLE [IF NOT EXISTS] *tbl\_name*

(*create\_definition*,...)

[*table\_options*]

[*partition\_options*]

CREATE [TEMPORARY] TABLE [IF NOT EXISTS] *tbl\_name*

[(*create\_definition*,...)]

[*table\_options*]

[*partition\_options*]

[AS] *query\_expression*

CREATE [TEMPORARY] TABLE [IF NOT EXISTS] *tbl\_name*

{ LIKE *old\_tbl\_name*| (LIKE *old\_tbl\_name*) }

**Имя таблицы**

* ***tbl\_name***

Имя таблицы можно указать ***db\_name.tbl\_name*** для создания таблицы в определенной базе данных.

* **IF NOT EXISTS**

Предотвращает возникновение ошибки, если таблица существует. Однако нет никакой проверки того, что существующая таблица имеет структуру, идентичную структуре, указанной в [CREATE TABLE](https://dev.mysql.com/doc/refman/8.0/en/create-table.html) заявлении.

create table IF NOT EXISTS

primer

(kod int,

name varchar(10));

describe primer;

insert into primer values (1, 'Хлеб', 56.20);

select \* from primer;

create **TEMPORARY** table

temp\_t

(kod int,

name varchar(10),

price decimal(5,2)

);

show tables;

*Временные таблицы*

Ключевое слово TEMPORARYиспользуется при создании таблицы. TEMPORARY таблица видна только в рамках текущего сеанса и автоматически удаляется при закрытии сеанса.

***Временные таблицы в раскладке не видны.***

***\q***

#### Клонирование и копирование таблиц

* LIKE

Используется CREATE TABLE ... LIKE для создания пустой таблицы на основе определения другой таблицы, включая любые атрибуты столбцов и индексы, определенные в исходной таблице:

CREATE TABLE new\_tbl LIKE orig\_tbl;

CREATE TABLE copy LIKE primer;

* [AS] ***query\_expression***

Чтобы создать одну таблицу из другой, добавьте [SELECT](https://dev.mysql.com/doc/refman/8.0/en/select.html) инструкцию в конце [CREATE TABLE](https://dev.mysql.com/doc/refman/8.0/en/create-table.html) инструкции:

CREATE TABLE *new\_tbl* AS SELECT \* FROM *orig\_tbl*;

CREATE TABLE *primer1* SELECT \* FROM *primer*;

CREATE TABLE *primer2* SELECT name FROM *primer*;

**Ограничения целостности**

**Типы данных** — это один из способов ограничивать данные, которые могут быть сохранены в таблицы. Однако, для многих приложений, такое ограничение является слишком грубым. Например, колонка содержащая цену товара должна хранить только положительные числа. Но стандартного типа данных, который бы допускал только положительные числа не существует. В других случаях, вам может понадобится ограничить данные в колонке в соответствии с значениями других колонок или строк.

Чтобы разрешить вышеописанные ситуации, SQL позволяет вам определять ограничения для колонок и таблиц. Ограничения дают вам настолько глубокий контроль над данными в ваших таблицах, насколько вы захотите. Если пользователь пытается сохранить в какой-либо колонке данные, которые попадают под ограничение, возникнет ошибка. Она возникнет даже если сохраняемое значение является значением по умолчанию.

**Ограничения Check**

Ограничение целостности check является наиболее часто используемым видом ограничения. Оно позволяет задать для определённой колонки, выражение, которое будет осуществлять проверку, помещаемого в эту колонку значения. Если значение удовлетворяет, заданному ограничению, то выражение должно возвращать Логическое значение (истина). Например, если требуется, чтобы в колонке были только положительные цены товаров:

**CREATE TABLE products (**

**product\_no int,**

**name varchar(10),**

**price decimal (5,2) check (price>0));**

**show create table products;** - смотрим название созданного ограничения!

**Проверяем**

1. ***insert into products values (1, 'хлеб', 50);***
2. ***insert into products values (2, 'молоко', 0);***
3. ***insert into products values (1, 'кефир', -50);***

Как видите, определение ограничения следует после указания типа данных. Ограничение **check** состоит из ключевого слова **CHECK**, за которым следует выражение в круглых скобках. Выражение в ограничении check должно использовать ту колонку, на которую оно накладывается, в противном случае это ограничение не будет иметь смысла.

Вы можете дать ограничению конкретное имя. Это сделает более понятными сообщения об ошибках и позволит вам ссылаться на это ограничение, когда вам понадобится его изменить. Синтаксис такой:

drop table products;

**CREATE TABLE products (**

**product\_no int,**

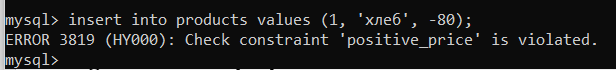
**name varchar(10),**

**price decimal (5,2) CONSTRAINT positive\_price check (price>0));**

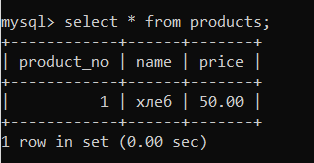
**showcreatetableproducts;** - смотрим название созданного ограничения!

**Проверяем**

1. **insert into products values (1, 'хлеб', -80);**



1. **insert into products values (1, 'хлеб', 50);**



Таким образом, для создания именованного ограничения, используйте ключевое слово **CONSTRAINT**, за которым следует идентификатор (имя), после которого следует, собственно, определение ограничения. (Если при использовании этого синтаксиса вы не укажете имя ограничения, **СУБД выберет имя за вас**.)

Ограничение **check** может также накладываться на несколько колонок. Допустим, что вы храните обычную цену и цену со скидкой и хотите иметь уверенность в том, что цена со скидкой меньше, чем обычная цена.

**CREATE TABLE products (**

**product\_no int,**

**name varchar(10),**

**price decimal (5,2) CONSTRAINT positive\_price check (price>0),**

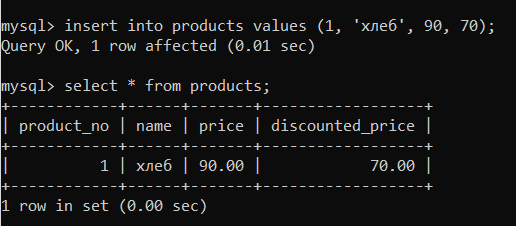
**discounted\_price decimal (5,2) CONSTRAINT dis\_price CHECK (discounted\_price> 0),**

**CONSTRAINT sravn\_price CHECK (price > discounted\_price));**

**show create table products;** - смотрим название созданного ограничения!

**Проверяем**

1. **insert into products values (1, 'хлеб', 0,0);**
2. **insert into products values (1, 'хлеб', 0, 30);**
3. **insert into products values (1, 'хлеб', 0, -30);**
4. **insert into products values (1, 'хлеб', 70, 90);**
5. **insert into products values (1, 'хлеб', 90, 70);**



Первые два ограничения нам уже знакомы. Третье использует новый синтаксис. Оно не привязано к определённой колонке, а появляется как отдельный элемент в списке колонок, разделённых запятой. Определения колонок и такие определения ограничений могут следовать в смешанном порядке.

**CREATE TABLE products (**

**product\_no int,**

**name varchar(10),**

**price decimal (5,2) CONSTRAINT positive\_pricecheck (price>0),**

**discounted\_price decimal (5,2),**

**CONSTRAINT sravn\_priceCHECK (discounted\_price> 0 AND price > discounted\_price));**

**Проверяем**

1. **insert into products values (1, 'хлеб', 0,-60);**
2. **insert into products values (1, 'хлеб', 0, 30);**
3. **insert into products values (1, 'хлеб', 70, 90);**
4. **insert into products values (1, 'хлеб', 90, 70);**

Необходимо заметить, что ограничение check удовлетворяется, если выражение, указанное в ограничении возвращает значение истина или null. Поскольку большинство выражений возвращает значение null, если любой из операндов имеет значение null, то использование выражений не предотвратит появление значений null в колонках, на которые наложены ограничения. Чтобы иметь уверенность в том, что колонка не будет содержать значений null, можно использовать ограничение не-null, которое описывается в следующей секции.

**Ограничения не-Null**

Ограничение не-null просто говорит, что колонка не должна содержать значение null.

Недостатком является то, что вы не можете явно задавать имена ограничениям не-null, которые созданы.

На колонку может накладываться более одного ограничения. Вы просто пишете другое ограничение, в след за первым:

**CREATE TABLE products**

**(**

**product\_no int NOT NULL,**

**name varchar(10) NOT NULL,**

**price decimal (5,2) NOT NULL CONSTRAINT positive\_price check (price>0));**

**insert into products**

**values (1, ‘Масло’, null);**

**insert into products (name, product\_no, price) values (‘Масло’, 1, 50);**

**select \* from products;**



Порядок записи не имеет значения. Также нет необходимости определять в каком порядке выполняется проверка ограничений.

**Ограничения уникальности**

Ограничения уникальности дают уверенность в том, что данные, содержащиеся в колонке или группе колонок являются уникальными по отношению к другим строкам в той же таблице. Синтаксис:

**CREATE TABLE products (**

**product\_no int,**

**name varchar(10) UNIQUE,**

**price decimal (5,2)**

**);**

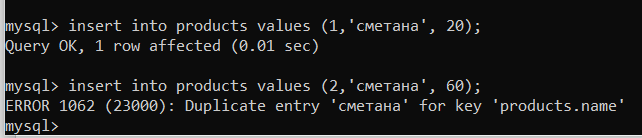
**CREATE TABLE clients**

**(id\_c INT NOT NULL UNIQUE ,**

**name varchar(20) not null,**

**inn varchar(12) UNIQUE );**

**insert into clients values (1, ‘Саша’, ‘111111111111’);**



Если ограничение уникальности ссылается на группу колонок, эти колонки перечисляются через запятую:

**CREATE TABLE passport (**

**series varchar(5),**

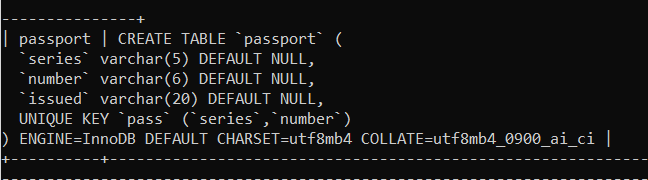
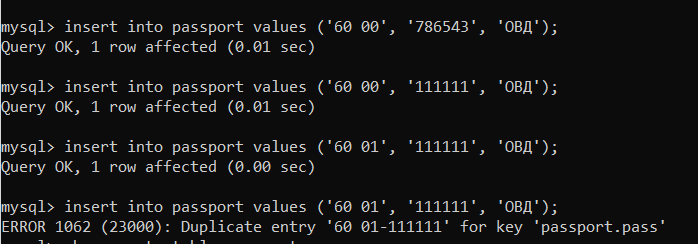
**number varchar(6),**

**issued varchar(20),**

**CONSTRAINT pass UNIQUE (series, number)**

**);**

Такая запись говорит, что комбинация значений в указанных колонках является уникальной для всей таблицы, хотя любое одно из значений этих колонок может не быть (и обычно не является) уникальным.



**Первичные ключи**

CREATE TABLE products (

product\_no integer **PRIMARY KEY**,

name varchar(10),

price decimal(4,2)

);

CREATE TABLE products (

product\_no integer **PRIMARY KEY AUTO\_INCREMENT**,

name varchar(10),

price decimal(4,2)

);

Проверяем

insert into products (name, price) values (‘Молоко’, 57.90);

Первичные ключи также могут ограничивать более чем одну колонку; синтаксис сходен с ограничением уникальности:

CREATE TABLE passport (

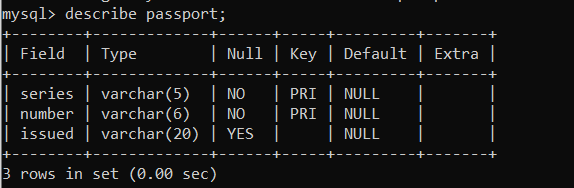
series varchar(5),

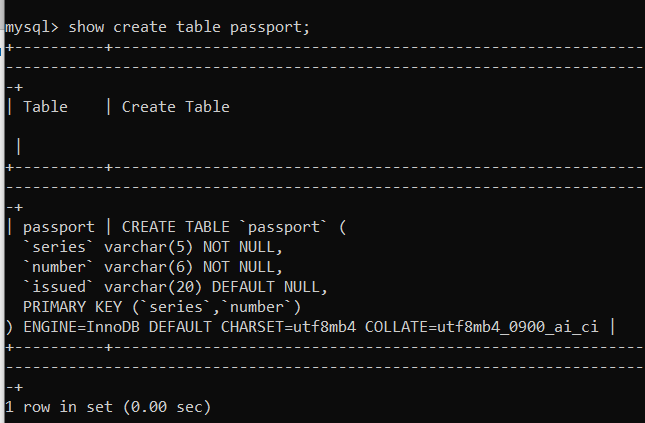
number varchar(6),

issued varchar(20),

PRIMARY KEY (series, number**)**

);





Первичный ключ говорит, что колонка или группа колонок могут быть использованы как уникальный идентификатор строки в таблице. Это прямое назначение первичного ключа. Такой идентификатор полезен как для целей документирования, так и для клиентских приложений. Например, приложение с графическим интерфейсом, которое позволяет изменять значения строк наверняка будет нуждаться в первичном ключе таблицы, чтобы идентифицировать нужные строки.

Таблица может иметь не более одного первичного ключа. (В ней может быть любое количество ограничений уникальности и ограничений не-null, которые функционально являются тем же, но только одно из этих ограничений может быть индентифицировано как первичный ключ.) Теория реляционных баз данных требует наличия первичного ключа для каждой таблицы.

Проверяем

insert into passport values (’60 00’, ‘666666’, ‘ОВД’);

insert into passport values (’60 01’, ‘666666’, ‘ОВД’);

insert into passport values (’60 00’, ‘755555’, ‘ОВД’);

insert into passport values (’60 00’, ‘755555’, ‘ОВД’);

**Внешние ключи**

Ограничение внешнего ключа говорит, что значения в колонке (или группе колонок) должно совпадать с значениями, которые существуют в некоторых строках другой таблицы. Мы говорим, что внешний ключ обслуживает ссылочную целостность между двумя относящимися друг к другу таблицами.

Например, пусть у нас есть таблица товаров products, которую мы уже использовали несколько раз:

CREATE TABLE products

(

product\_no integer auto\_increment PRIMARY KEY,

name varchar(10) not null,

price decimal(4,2) not null

);

Также пусть у нас есть таблица orders.

CREATE TABLE orders

(

order\_id integer PRIMARY KEY,

date\_o date NOT NULL

);

С помощью оператора **CONSTRAINT** можно задать имя для ограничения внешнего ключа:

CREATE TABLE ord\_prod

(

order\_id integer not null ,

product\_no integer NOT NULL ,

quantity integer NOT NULL check (quantity >0),

CONSTRAINT ord\_prod\_fk1 FOREIGN KEY (order\_id) REFERENCES orders (order\_id),

CONSTRAINT ord\_prod\_fk2 FOREIGN KEY(product\_no) REFERENCES products (product\_no),

PRIMARY KEY (order\_id, product\_no)

);

SELECT order\_id ‘Номер заказа’, name, price, quantity,

(price \* quantity) Сумма FROM products

INNER JOIN ord\_prod

ON products.product\_no=ord\_prod.product\_no WHERE order\_id=1;

Проверяем удаление товара из таблицы products с номером 1.

* delete from products where product\_no=1;

### ON DELETE

* С помощью выражений ON DELETE и ON UPDATE можно установить действия, которые выполняются соответственно при удалении и изменении связанной строки из главной таблицы. В качестве действия могут использоваться следующие опции:
* **CASCADE**: автоматически удаляет или изменяет строки из зависимой таблицы при удалении или изменении связанных строк в главной таблице.
* **SET NULL**: при удалении или обновлении связанной строки из главной таблицы устанавливает для столбца внешнего ключа значение **NULL**. (В этом случае столбец внешнего ключа должен поддерживать установку NULL)
* **RESTRICT**: отклоняет удаление или изменение строк в главной таблице при наличии связанных строк в зависимой таблице.
* **NO ACTION**: то же самое, что и RESTRICT.

### Каскадное удаление

Каскадное удаление позволяет при удалении строки из главной таблицы автоматически удалить все связанные строки из зависимой таблицы. Для этого применяется опция **CASCADE**:

|  |  |
| --- | --- |
|  |  |

CREATE TABLE ord\_prod (

order\_id integer not null,

product\_no integer NOT NULL,

quantity integer NOT NULL check (quantity >0),

CONSTRAINT ord\_prod\_fk1 FOREIGN KEY(order\_id) REFERENCES orders (order\_id) ON DELETE CASCADE,

CONSTRAINT ord\_prod\_fk2 FOREIGN KEY(product\_no) REFERENCES products (product\_no),

PRIMARY KEY (order\_id, product\_no)

);

Проверяем удаление заказа из таблицы orders с номером 1.

* delete from orders where order\_id=1;

create table clients

(id\_c int primary key auto\_increment,

name varchar(10) not null );

create table adress (

id\_a int primary key auto\_increment,

city varchar(30) not null,

street varchar(50) not null,

id\_c int,

constraint adress\_cl\_fk1 FOREIGN KEY (id\_c) REFERENCES clients (id\_c)

ON DELETE SET NULL);

### Установка NULL

При установки для внешнего ключа опции **SET NULL** необходимо, чтобы столбец внешнего ключа допускал значение NULL:

CREATE TABLE orders (

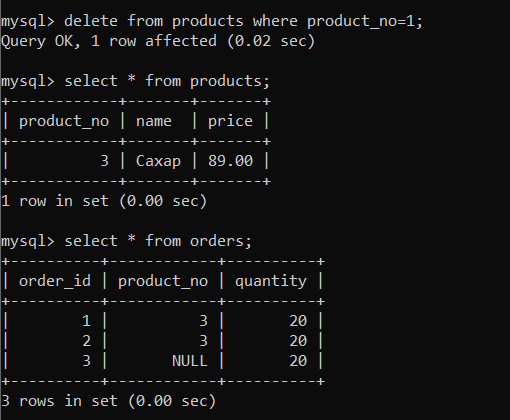
order\_id integer PRIMARY KEY AUTO\_INCREMENT,

product\_no integer,

quantity integer NOT NULL,

CONSTRAINT ord\_prod\_fk FOREIGN KEY (product\_no) REFERENCES products (product\_no) ON DELETE SET NULL

);



CREATE TABLE orders (

order\_id integer PRIMARY KEY AUTO\_INCREMENT,

product\_no integer,

quantity integer NOT NULL,

CONSTRAINT ord\_prod\_fk FOREIGN KEY (product\_no) REFERENCES products (product\_no) ON DELETE **RESTRICT**

);

Проверяем удаление товара из таблицы products с номером 1.

* delete from products where product\_no=1;

CREATE TABLE orders (

order\_id integer PRIMARY KEY AUTO\_INCREMENT,

product\_no integer,

quantity integer NOT NULL,

CONSTRAINT ord\_prod\_fk FOREIGN KEY (product\_no) REFERENCES products (product\_no) ON DELETE **NO ACTION**

);

Проверяем удаление товара из таблицы products с номером 1.

* delete from products where product\_no=1;

Внешний ключ может также ограничивать и ссылаться на группу колонок. Такое ограничение необходимо записывать как ограничение на таблицу. Вот пример:

**Изменение таблиц**

Когда вы создали таблицу и видите, что допустили ошибку или что требования приложения изменились, вы можете удалить эту таблицу и создать её заново. Но это неудобно, если таблица уже содержит данные или если на таблицу ссылаются другие объекты базы данных (например, внешний ключ). Поэтому PostgreSQL предоставляет набор команд для внесения изменений в существующие таблицы. Заметим, что эти действия концептуально отличаются от изменений самих данных, содержащихся в таблице: здесь мы говорим о изменении структуры таблицы.

ALTER TABLE название\_таблицы

{ ADD название\_столбца тип\_данных\_столбца [атрибуты\_столбца] |

DROP COLUMN название\_столбца |

MODIFY COLUMN название\_столбца тип\_данных\_столбца [атрибуты\_столбца] |

ALTER COLUMN название\_столбца SET DEFAULT значение\_по\_умолчанию |

ADD[CONSTRAINT] определение\_ограничения |

DROP[CONSTRAINT] имя\_ограничения}

Вы можете:

* Добавить колонки
* Удалить колонки
* Добавить ограничения
* Удалить ограничения
* Изменить значения по умолчанию
* Изменить типы данных колонок
* Переименовать колонки
* Переименовать таблицы

Все эти действия выполняются с помощью команды [ALTER TABLE](http://postgresql.ru.net/manual/sql-altertable.html), на странице описания которой, можно найти подробности перечисленных здесь действий.

**Добавление колонки**

Чтобы добавить колонку, используйте команду вида:

**ALTER TABLE products**

**ADD COLUMN actikul char(6)**

**NOT NULL**

**AFTER product\_no;**

Новая колонка изначально заполняется указанным значением по умолчанию (значением null, если вы не задали другое значение в предложении DEFAULT).

**Удаление колонки**

Чтобы удалить колонку, используйте команду вида:

**ALTER TABLE products DROP COLUMN description;**

**Добавление ограничения**

Чтобы добавить какое-либо ограничение, используется табличный синтаксис определения этого ограничения. Например:

create table items

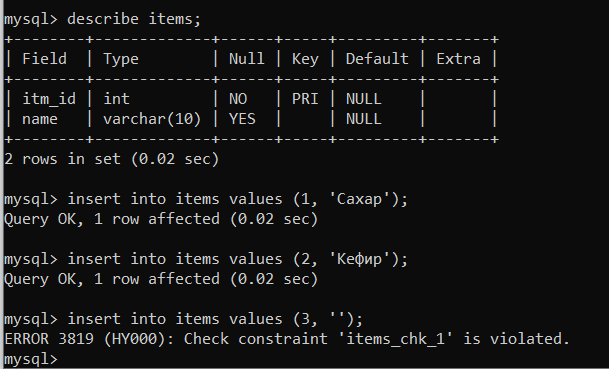
(

itm\_id int primary key auto\_increment,

name varchar(10) not null

);

**ALTER TABLE items ADD CHECK (name <> '');**

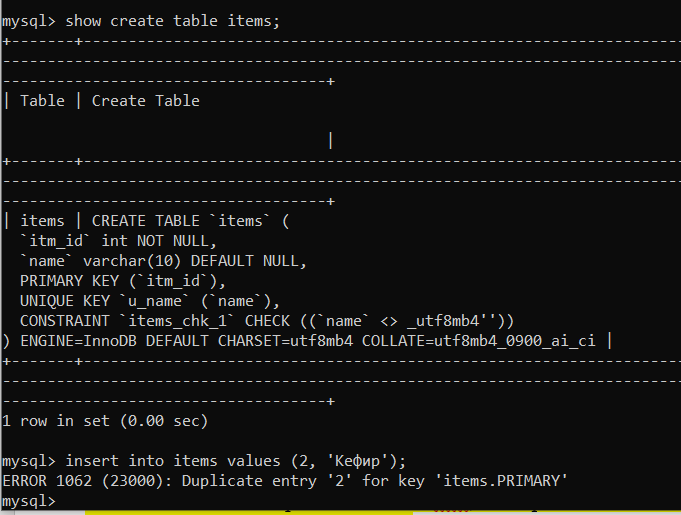


**ALTER TABLE items ADD CONSTRAINT u\_name UNIQUE (name);**

**ALTER TABLE items**

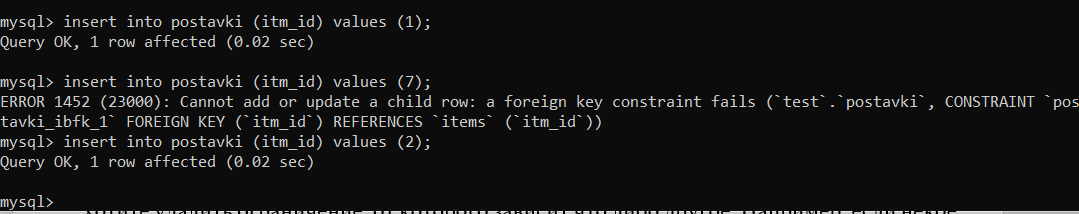
**ADD CONSTRAINT u\_artikul**

**UNIQUE (artikul);**



Create table postavki (id\_p int primary key auto\_increment, itm\_id int);

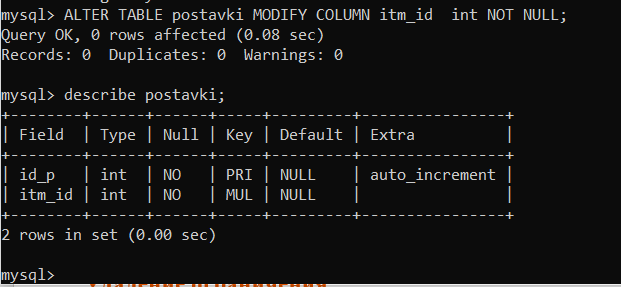
ALTER TABLE postavki ADD FOREIGN KEY (itm\_id) REFERENCES items (itm\_id);

****

Чтобы добавить ограничение не-null, которое нельзя записать как ограничение на таблицу, используйте синтаксис:

**ALTER TABLE items MODIFY COLUMN price decimal(7,2) NOT NULL;**

**update items set price=500;**



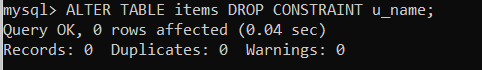
Указанное ограничение будет задействовано немедленно, так что данные в таблице перед добавлением ограничения должны ему удовлетворять.

**Удаление ограничения**

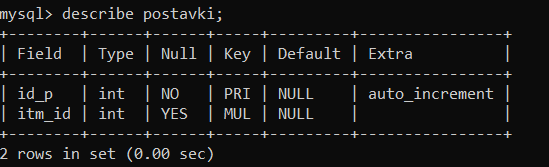
**ИС 25**

Чтобы удалить ограничение, вам необходимо **знать его имя**.

**ALTER TABLE items DROP CONSTRAINT u\_artikul;**



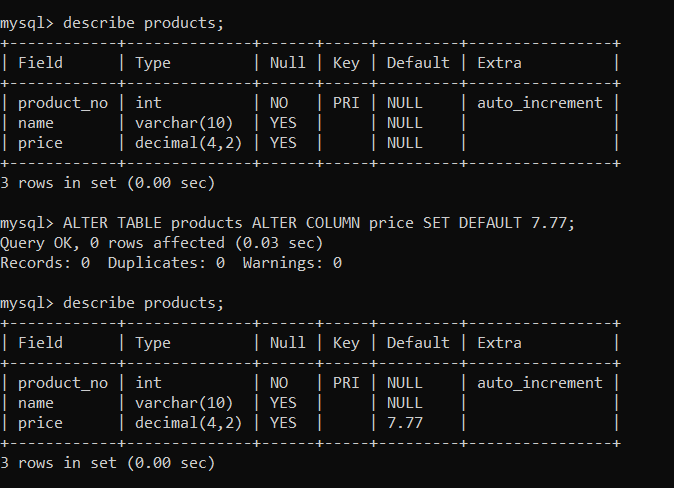
**ALTER TABLE postavki MODIFY COLUMN itm\_id int NULL;**



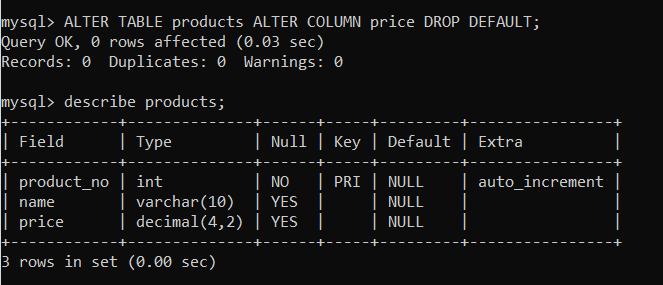
**Изменение значения по умолчанию для колонки**

Чтобы установить новое значение по умолчанию для колонки, используйте команду вида:

**ALTER TABLE products ALTER COLUMN price SET DEFAULT 0;**



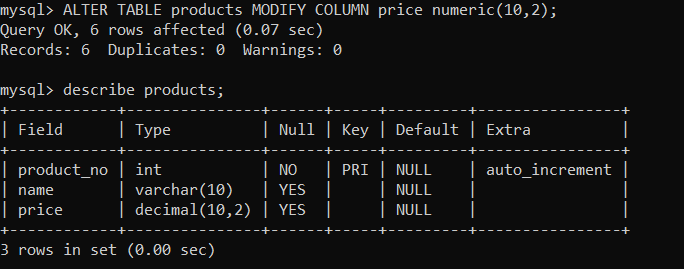
**ALTER TABLE products ALTER COLUMN price DROP DEFAULT;**



**Изменение типа данных колонки**

Чтобы преобразовать колонку к другому типу данных, используйте команду вида:

ALTER TABLE products MODIFY COLUMN price decimal(10,2);

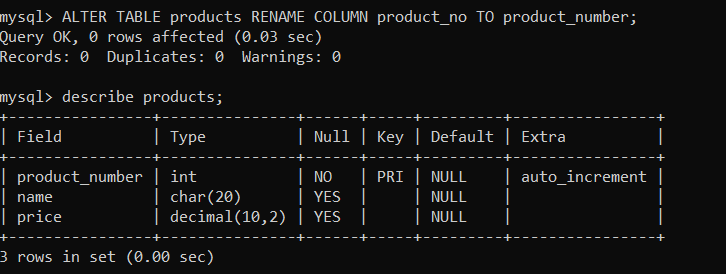


ALTER TABLE products MODIFY COLUMN name char (20);

**Переименование колонки**

Чтобы переименовать колонку, используйте команду вида:

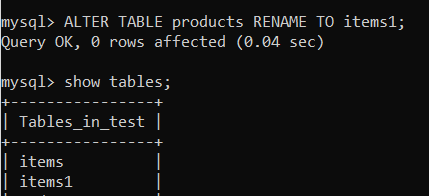
**ALTER TABLE products RENAME COLUMN product\_no TO product\_number;**



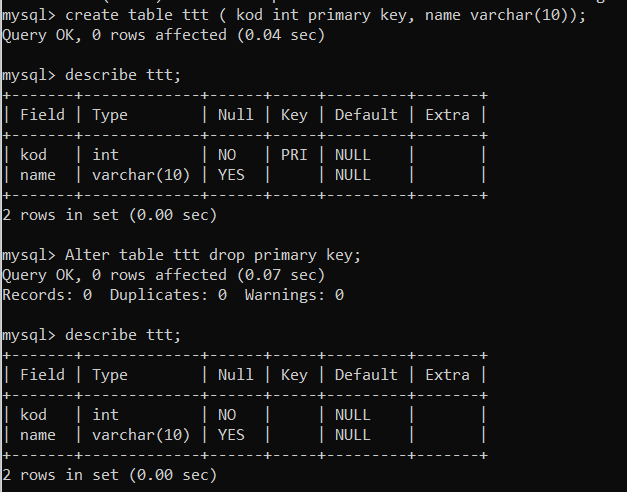
**Переименование таблицы**

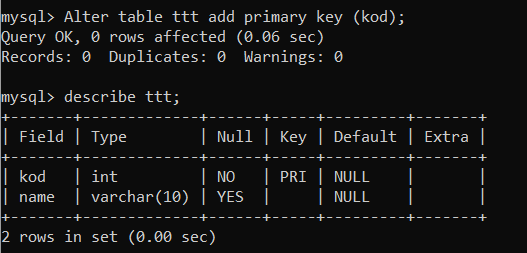
Чтобы переименовать таблицу, используйте команду вида:

**ALTER TABLE products RENAME TO items1;**



**Удаление и добавление PRIMARY KEY**





**ALTER TABLE** table\_name

**ADD FOREIGN KEY** (column\_name)

**REFERENCE** table\_name(Referencing column\_name in table\_name);