

## Лабораторна робота

Тема: Визначення структури даних в PostgreSQL.

Мета: Визначення структури даних.

### Хід роботи

#### 1. Створення та видалення бази даних:

1) Щоб створити нову базу даних (БД) відкриємо pgAdmin. У лівій частині програми виберемо якусь базу даних, наприклад, стандартну БД postgres, і натиснемо на неї правою кнопкою миші.

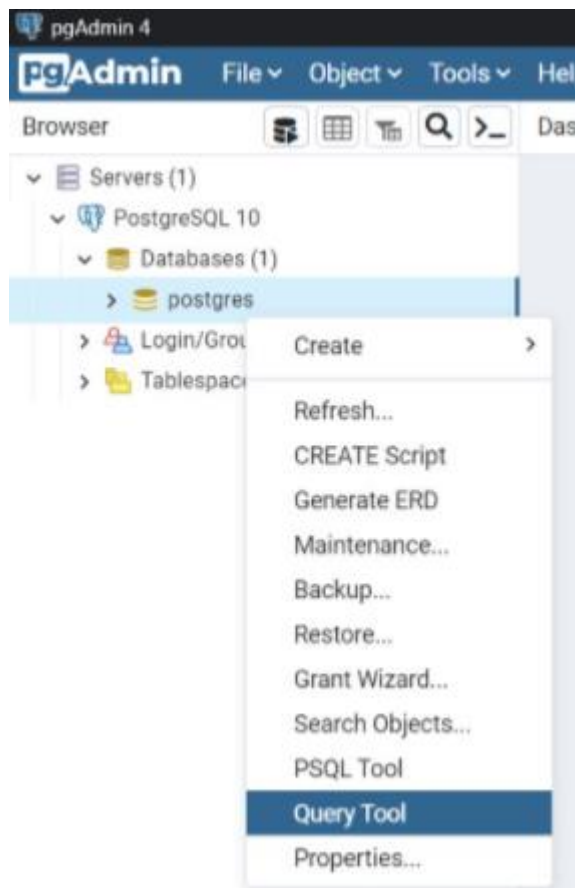


Рисунок 1 – Інтерфейс пакета pgAdmin

2) У меню вибираємо пункт Query Tool, і в центральній частині програми відкриється поле для введення коду SQL. В це поле введемо наступний код: `CREATE DATABASE usersplit;`

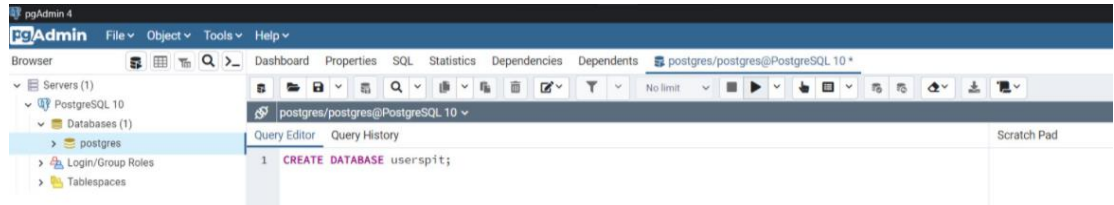


Рисунок 1.2 – Створення бази даних

3) Щоб побачити нашу базу даних, натиснемо в лівій частині на вузол Databases правою кнопкою миші і в контекстному меню вибираємо Refresh.

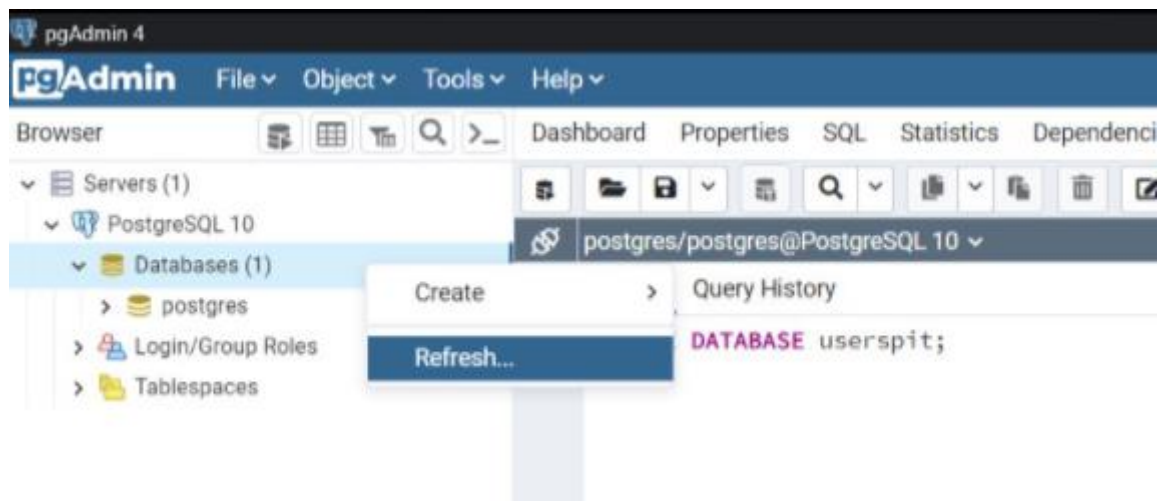


Рисунок 1.3 – Оновлення баз даних

4) Відбудеться оновлення, і ми побачимо створену базу даних.

За замовчуванням база є неактивною, тому її значок має сірий колір. Але щоб до неї підключитися, досить натиснути на неї і розкрити її вузол.

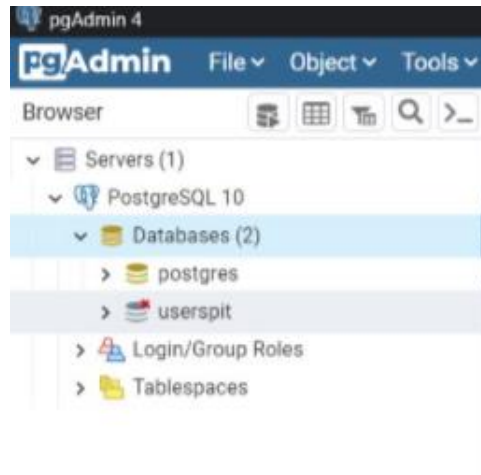


Рисунок 1.4 – база даних

#### 5) Видалення бази даних:

Для видалення бази даних застосовується команда `DROP DATABASE`, після якої вказується назва бази даних.

Видаляється база даних повинна бути неактивною, тобто підключення до неї повинно бути закрито.

Наприклад, видалення бази даних `usersplit`:

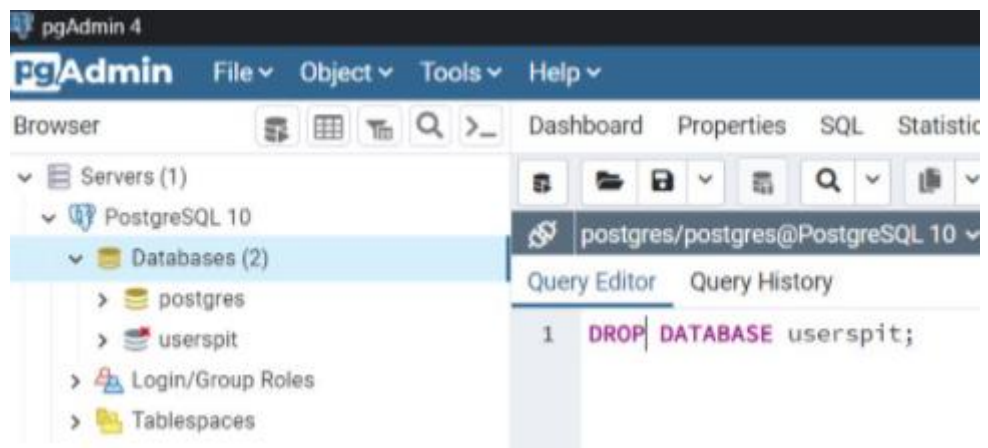


Рисунок 1.5 – Видалення бази даних

#### 2. Створення та видалення таблиць:

Для створення таблиць застосовується команда `CREATE TABLE`, після якої вказується назва таблиці. Також з цією командою можна використовувати

ряд операторів, які визначають стовпці таблиці і їх атрибути. Загальний синтаксис створення таблиці виглядає наступним чином:

```
CREATE TABLE table_name  
(  
    column_name1 data_type column_attributes1,  
    column_name2 data_type column_attributes2,  
    .....  
    column_nameN data_type column_attributesN,  
    table_attributes  
);
```

1) Створимо таблицю в базі даних через pgAdmin. Для цього спочатку виберемо в pgAdmin цільову базу даних, натиснемо на неї правою кнопкою миші і в контекстному меню виберемо пункт Query Tool. Після цього відкриється поле для введення коду на SQL. Причому таблиця буде створюватися саме для тієї бази даних, для якої ми відкриємо це поле для введення SQL.

Далі в відкрилося в центральній частині програми поле введемо наступний набір виразів:

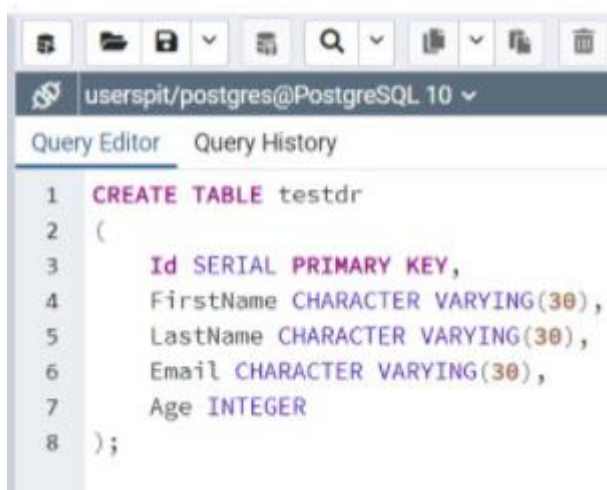


Рисунок 2 – Створення таблиці

В даному випадку в таблиці testdr визначаються п'ять стовпців: Id, FirstName, LastName, Age, Email. Перший стовпець-Id представляє ідентифікатор клієнта, він служить первинним ключем і тому має тип SERIAL. Фактично даний стовпець буде зберігати Числове значення 1, 2, 3 і т.д., яке для кожного нового рядка буде автоматично збільшуватися на одиницю.

Наступні три стовпці представляють ім'я, прізвище клієнта та його електронну адресу і мають тип CHARACTER VARYING(30), тобто представляють рядок довжиною не більше 30 символів.

Останній стовпець - Age представляє вік користувача і має тип INTEGER, тобто зберігає числа.

## 2) Видалення таблиць

Для видалення таблиць використовується команда DROP TABLE, яка має наступний синтаксис:



Рисунок 2.1 – Видалення таблиць

## 3. Типи даних в PostgreSQL:

При визначенні таблиці для всіх її стовпців необхідно вказати тип даних. Тип даних визначає діапазон значень, які можуть зберігатися в стовпці, скільки вони будуть займати місця в пам'яті. PostgreSQL підтримує багату палітру різних типів даних, серед які умовно можна розділити на підгрупи: Числові, символьні, логічні, дата і час, бінарні і ряд інших.

### 1) Числові типи даних

- serial: представляє автоінкрементне числове значення, яке займає 4 байти і може зберігати числа від 1 до 2147483647. Значення даного типу

утворюється шляхом автоінкременту значення попереднього рядка. Тому, як правило, даний тип використовується для визначення ідентифікаторів рядка.

- `smallserial`: представляє автоінкрементне числове значення, яке займає 2 байти і може зберігати числа від 1 до 32767. Аналог типу `serial` для невеликих чисел.

- `bigserial`: представляє автоінкрементне числове значення, яке займає 8 байт і може зберігати числа від 1 до 9223372036854775807. Аналог типу `serial` для великих чисел.

- `smallint`: зберігає числа від -32768 до +32767. Займає 2 байти. Має псевдонім `int2`.

- `integer`: зберігає числа від -2147483648 до +2147483647. Займає 4 байти. Має псевдоніми `int` і `int4`.

- `bigint`: зберігає числа від -9223372036854775808 до +9223372036854775807. Займає 8 байт. Має псевдонім `int8`.

- `numeric`: зберігає числа з фіксованою точністю, які можуть мати до 131072 знаків в цілій частині і до 16383 знаків після коми.

Даний тип може приймати два параметри `precision` і `scale`: `numeric (precision, scale)`.

Параметр `precision` вказує на максимальну кількість цифр, які може зберігати число.

Параметр `scale` представляє максимальну кількість цифр, які може містити число після коми. Це значення повинно знаходитися в діапазоні від 0 до значення параметра `precision`. За замовчуванням воно дорівнює 0.

Наприклад, для числа 23.5141 `precision` дорівнює 6, а `scale`-4.

- `decimal`: зберігає числа з фіксованою точністю, які можуть мати до 131072 знаків в цілій частині і до 16383 знаків в дробовій частині. Те ж саме, що і `numeric`.

- `real`: зберігає числа з плаваючою точкою з діапазону від  $1E-37$  до  $1E+37$ . Займає 4 байти. Має псевдонім `float4`.

- **double precision:** зберігає числа з плаваючою точкою з діапазону від  $1E - 307$  до  $1E + 308$ . Займає 8 байт. Має псевдонім float8.

#### Приклади використання:

```
Id SERIAL,
TotalWeight NUMERIC(9,2),
Age INTEGER,
Surplus REAL
```

#### 2) Типи для роботи з валютою (грошовими одиницями)

Для роботи з грошовими одиницями визначено тип money, який може приймати значення в діапазоні від -92233720368547758.08 до +92233720368547758.07 і займає 8 байт.

#### 3) Символьний тип

- **character (n):** представляє рядок з фіксованої кількості символів. За допомогою параметра задається кількість символів в рядку. Має псевдонім char (n).
- **character varying( n):** представляє рядок з фіксованої кількості символів. За допомогою параметра задається кількість символів в рядку. Має псевдонім varchar (n).
- **text:** представляє текст довільної довжини.

#### 4) Бінарні дані

Для зберігання бінарних даних визначено тип bytea. Він зберігає дані у вигляді бінарних рядків, які представляють послідовність октетів або байт.

#### 5) Типи для роботи з датами і часом

- **timestamp:** зберігає дату і час. Займає 8 байт. Для дат найнижче значення - 4713 р До н. е., саме верхнє значення - 294276 р н. е.
- **timestamp with time zone:** те ж саме, що і timestamp, тільки додає дані про часовий пояс.
- **date:** представляє дату від 4713 р. До н. е. до 5874897 р. н. е. займає 4 байти.
- **time:** зберігає час з точністю до 1 мікросекунди без вказівки часового поясу. Приймає значення від 00:00:00 до 24: 00: 00. Займає 8 байт.
- **time with time zone:** зберігає час з точністю до 1 мікросекунди із зазначенням часового поясу. Приймає значення від 00:00:00+1459 до 24: 00: 00-1459. Займає 12 байт.
- **interval:** представляє часовий інтервал. Займає 16 байт.

Поширені формати дат:

yyyy-mm-dd - 1999-01-08

Month dd, yyyy - January 8, 1999

mm/dd/yyyy - 1/8/1999

Поширені формати часу:

hh:mi - 13:21

hh:mi am/pm - 1:21 pm

hh:mi:ss - 1:21:34

#### 6) Логічний тип

Тип `boolean` може зберігати одне з двох значень: `true` або `false`.

Замість `true` можна вказувати наступні значення: `TRUE`, `'t'`, `'true'`, `'y'`, `'yes'`, `'on'`, `'1'`.

Замість `false` можна вказувати наступні значення: `FALSE`, `'f'`, `'false'`, `'n'`, `'no'`, `'off'`, `'0'`.

#### 7) Типи для представлення інтернет-адрес

- `cidr`: інтернет-адреса в форматі IPv4 і IPv6. Наприклад, 192.168.0.1. Займає від 7 до 19 байт.

- `inet`: інтернет-адреса в форматі `cidr / у`, де `cidr` це адреса в форматі IPv4 або IPv6, а `/у` - кількість біт в адресі (якщо цей параметр не вказано, то використовується 34 для IPv4, 128 для IPv6). Наприклад, 192.168.0.1 / 24 або 2001:4 f8:3:ba:2E0:81 ff:fe 22: d1f1/128. Займає від 7 до 19 байт.

- `macaddr`: зберігає MAC-адресу. Займає 6 байт.

- `macaddr8`: зберігає MAC-адресу у форматі EUI-64. Займає 8 байт.



#### 8) Геометричні типи

- point: представляє точку на площині у форматі (x,y). Займає 16 байт.
- line: представляє лінію невизначеної довжини у форматі {A,B,C}. Займає 32 байти.
- lseg: представляє відрізок у форматі ((x1,y1), (x2,y2)). Займає 32 байти.
- box: представляє прямокутник у форматі ((x1,y1), (x2,y2)). Займає 32 байти.
- path: представляє набір з'єднаних точок. У форматі ((x1,y1),...) шлях є закритим (перша і остання точка з'єднуються лінією) і фактично представляє багатокутник. У форматі [(x1,y1),...] шлях є відкритим займає 16 + 16N байт.
- polygon: представляє багатокутник у форматі ((x1,y1),...). Займає 40 + 16N байт.
- circle: представляє окружність у форматі <(x,y),r>. Займає 24 байти.

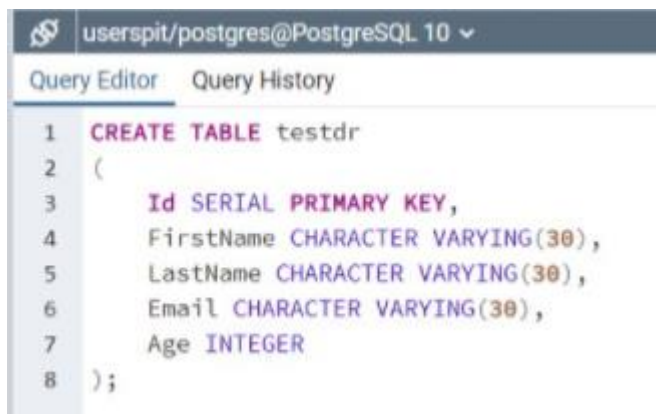
#### 9) Інші типи даних

- json: зберігати дані json в текстовому вигляді.
- json: зберігати дані json в бінарному форматі.
- uuid: зберігає універсальний унікальний ідентифікатор (UUID), наприклад, a0eebc99-9c0b-4ef8-bb6d-6bb9bd380a11. Займає 32 байти.
- xml: зберігає дані у форматі XML.

### 4.Обмеження стовпців і таблиць

#### 1) PRIMARY KEY

За допомогою виразу PRIMARY KEY стовпець можна зробити первинним ключем.



```
1 CREATE TABLE testdr
2 (
3     Id SERIAL PRIMARY KEY,
4     FirstName CHARACTER VARYING(30),
5     LastName CHARACTER VARYING(30),
6     Email CHARACTER VARYING(30),
7     Age INTEGER
8 );
```

Рисунок 4.1 – Первинним ключем PRIMARY KEY

Первинний ключ унікально ідентифікує рядок у таблиці. В якості первинного ключа необов'язково повинні виступати стовпці з типом SERIAL, вони можуть представляти будь-який інший тип.

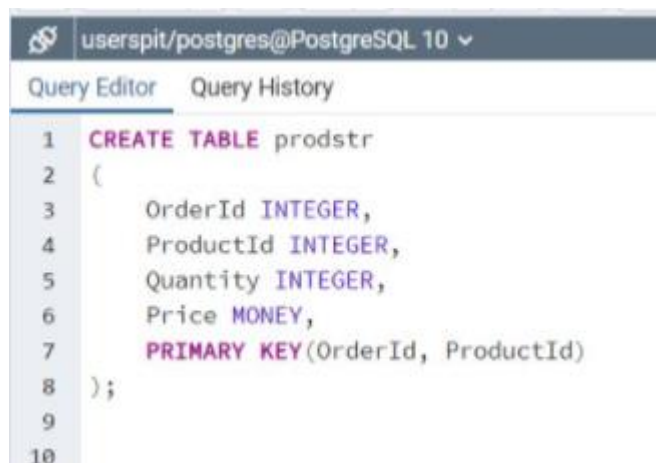
## 2) Встановлення первинного ключа на рівні таблиці:



```
1 CREATE TABLE testdr
2 (
3     Id SERIAL,
4     FirstName CHARACTER VARYING(30),
5     LastName CHARACTER VARYING(30),
6     Email CHARACTER VARYING(30),
7     Age INTEGER
8     PRIMARY KEY (Id)
9 );
```

Рисунок 4.2 –PRIMARY KEY на рівні таблиці

3) Первинний ключ може бути складовим (compound key). Такий ключ може знадобитися, якщо у нас відразу два стовпці повинні унікально ідентифікувати рядок в таблиці. Наприклад:



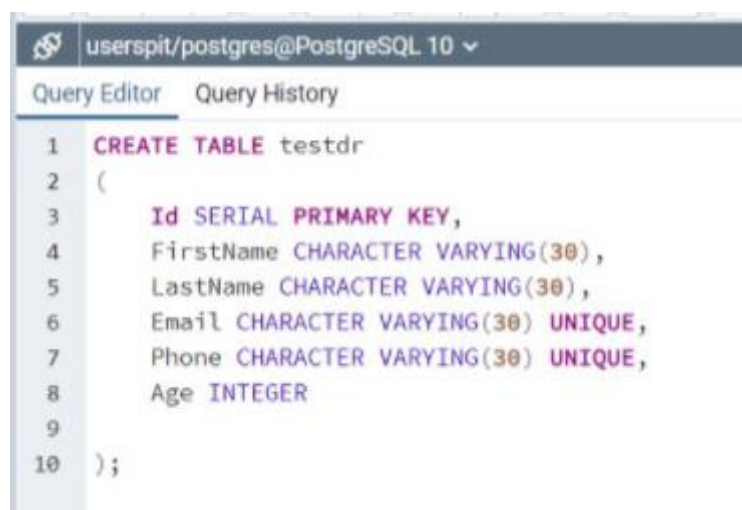
```
1 CREATE TABLE prodstr
2 (
3     OrderId INTEGER,
4     ProductId INTEGER,
5     Quantity INTEGER,
6     Price MONEY,
7     PRIMARY KEY (OrderId, ProductId)
8 );
9
10
```

Рисунок 4.3 – compound key

Тут поля OrderId і ProductId разом виступають як складовою первинний ключ. Тобто в таблиці OrderLines не може бути двох рядків, де для обох з цих полів одночасно були б одні й ті ж значення.

#### 4) UNIQUE

якщо ми хочемо, щоб стовпець мав тільки унікальні значення, то для нього можна визначити атрибут UNIQUE.



```
1 CREATE TABLE testdr
2 (
3     Id SERIAL PRIMARY KEY,
4     FirstName CHARACTER VARYING(30),
5     LastName CHARACTER VARYING(30),
6     Email CHARACTER VARYING(30) UNIQUE,
7     Phone CHARACTER VARYING(30) UNIQUE,
8     Age INTEGER
9
10 );
```

Рисунок 4.4 – UNIQUE

В даному випадку стовпці, які представляють електронну адресу і телефон, будуть мати унікальні значення. І ми не зможемо додати в таблицю два рядки, у яких значення для цих стовпців буде збігатися.

### 5) NULL і NOT NULL

Щоб вказати, чи може стовпець приймати значення NULL, при визначенні стовпця йому можна задати атрибут NULL або NOT NULL. Якщо цей атрибут явно не буде використаний, то за замовчуванням стовпець буде допускати значення NULL. Винятком є той випадок, коли стовпець виступає в ролі первинного ключа – в цьому випадку за замовчуванням стовпець має значення NOT NULL.




```
1 CREATE TABLE testdr
2 (
3     Id SERIAL PRIMARY KEY,
4     FirstName CHARACTER VARYING(20) NOT NULL,
5     LastName CHARACTER VARYING(20) NOT NULL,
6     Age INTEGER
7 );
8
```

Рисунок 4.5 – NULL і NOT NULL

### 6) DEFAULT

Атрибут DEFAULT визначає значення за замовчуванням для стовпця. Якщо при додаванні даних для стовпця не буде передбачено значення, то для нього буде використовуватися значення за замовчуванням.



```
1 CREATE TABLE testdr
2 (
3     Id SERIAL PRIMARY KEY,
4     FirstName CHARACTER VARYING(20),
5     LastName CHARACTER VARYING(20),
6     Age INTEGER DEFAULT 18
7 );
8
```

Рисунок 4.6 – DEFAULT

Тут для стовпця Age передбачено значення за замовчуванням 18.

## 7) CHECK

Ключове слово CHECK задає обмеження для діапазону значень, які можуть зберігатися в стовпці. Для цього після слова CHECK вказується в дужках умова, якій повинен відповідати стовпець або кілька стовпців. Наприклад, вік клієнтів не може бути менше 0 або більше 100:



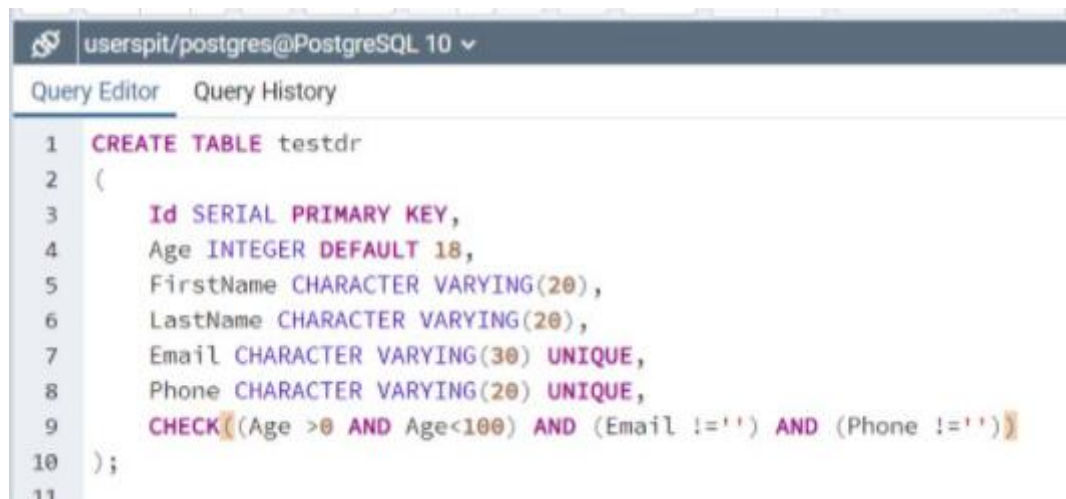
```
1 CREATE TABLE testdr
2 (
3     Id SERIAL PRIMARY KEY,
4     FirstName CHARACTER VARYING(20),
5     LastName CHARACTER VARYING(20),
6     Age INTEGER DEFAULT 18 CHECK(Age > 0 AND Age < 100),
7     Email CHARACTER VARYING(30) UNIQUE CHECK>Email != ''),
8     Phone CHARACTER VARYING(20) UNIQUE CHECK>Phone != ''))
9 );
10
```

Рисунок 4.7 – CHECK

Тут також вказується, що стовпці Email і Phone не можуть мати порожній рядок як значення (порожній рядок не еквівалентна значенню NULL).

Для з'єднання умов використовується ключове слово AND. Умови можна задати у вигляді операцій порівняння більше ( > ), менше ( < ), не дорівнює (!=).

Також за допомогою CHECK можна створити обмеження в цілому для таблиці:



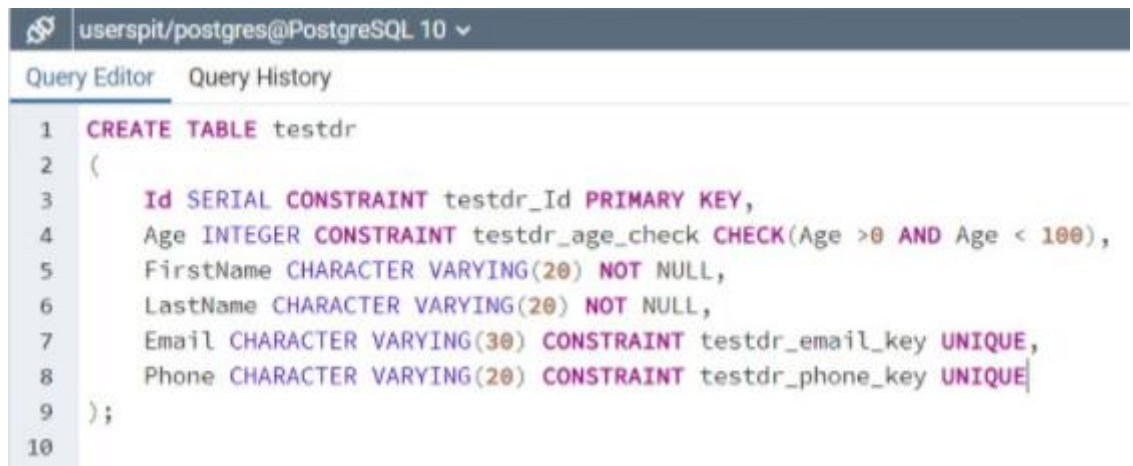
```
1 CREATE TABLE testdr
2 (
3     Id SERIAL PRIMARY KEY,
4     Age INTEGER DEFAULT 18,
5     FirstName CHARACTER VARYING(20),
6     LastName CHARACTER VARYING(20),
7     Email CHARACTER VARYING(30) UNIQUE,
8     Phone CHARACTER VARYING(20) UNIQUE,
9     CHECK((Age > 0 AND Age < 100) AND (Email != '') AND (Phone != ''))
10 );
11
```

Рисунок 4.8 – ключове слово AND

## 8) Оператор CONSTRAINT

За допомогою ключового слова CONSTRAINT можна задати ім'я для обмежень. В якості обмежень можуть використовуватися PRIMARY KEY, UNIQUE, CHECK.

Імена обмежень можна задати на рівні стовпців. Вони вказуються після CONSTRAINT перед атрибутами:

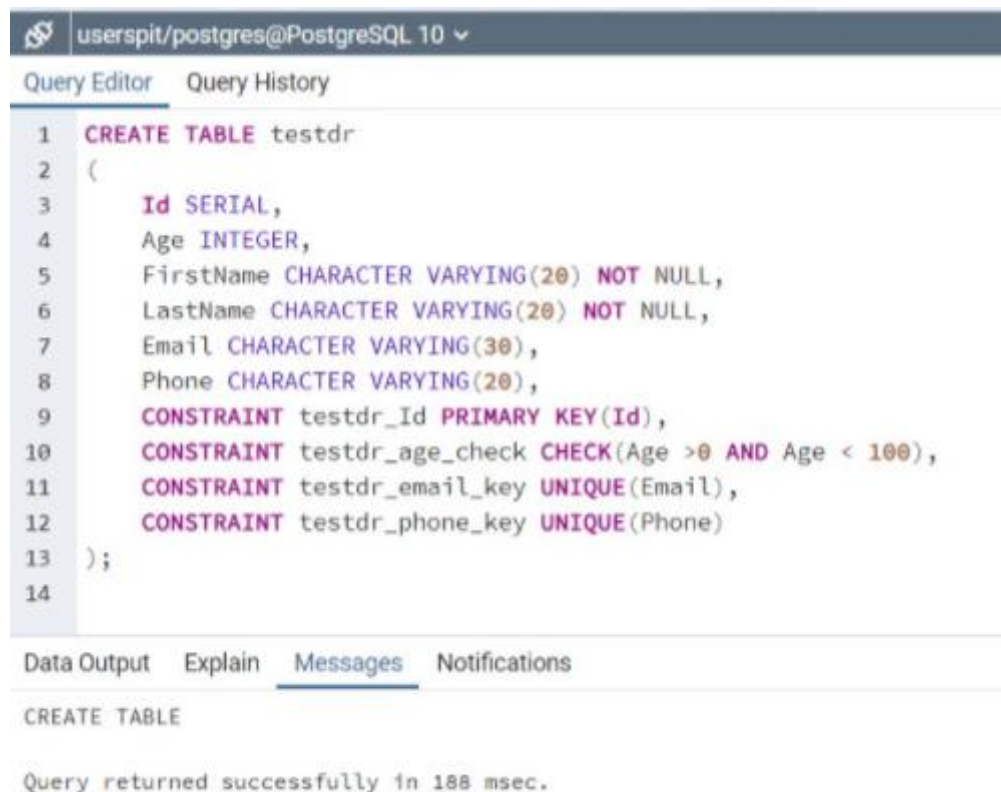


```
1 CREATE TABLE testdr
2 (
3     Id SERIAL CONSTRAINT testdr_Id PRIMARY KEY,
4     Age INTEGER CONSTRAINT testdr_age_check CHECK(Age >0 AND Age < 100),
5     FirstName CHARACTER VARYING(20) NOT NULL,
6     LastName CHARACTER VARYING(20) NOT NULL,
7     Email CHARACTER VARYING(30) CONSTRAINT testdr_email_key UNIQUE,
8     Phone CHARACTER VARYING(20) CONSTRAINT testdr_phone_key UNIQUE
9 );
10
```

Рисунок 4.9 – Оператор CONSTRAINT

В принципі необов'язково задавати імена обмежень, при установці відповідних атрибутів SQL Server автоматично визначає їх імена. Але, знаючи ім'я обмеження, ми можемо до нього звертатися, наприклад, для його видалення.

І також можна задати всі імена обмежень через атрибути таблиці:



```
1 CREATE TABLE testdr
2 (
3     Id SERIAL,
4     Age INTEGER,
5     FirstName CHARACTER VARYING(20) NOT NULL,
6     LastName CHARACTER VARYING(20) NOT NULL,
7     Email CHARACTER VARYING(30),
8     Phone CHARACTER VARYING(20),
9     CONSTRAINT testdr_Id PRIMARY KEY(Id),
10    CONSTRAINT testdr_age_check CHECK(Age >0 AND Age < 100),
11    CONSTRAINT testdr_email_key UNIQUE(Email),
12    CONSTRAINT testdr_phone_key UNIQUE(Phone)
13 );
14
```

Data Output Explain Messages Notifications

CREATE TABLE

Query returned successfully in 188 msec.

Рисунок 4.10 – CONSTRAINT атрибути таблиці

Незалежно від того, використовується оператор CONSTRAINT для створення обмежень чи ні (в цьому випадку при установці обмежень PostgreSQL сам дає їм імена), ми можемо переглянути всі обмеження в pgAdmin в вузлі бази даних в підвузлі:

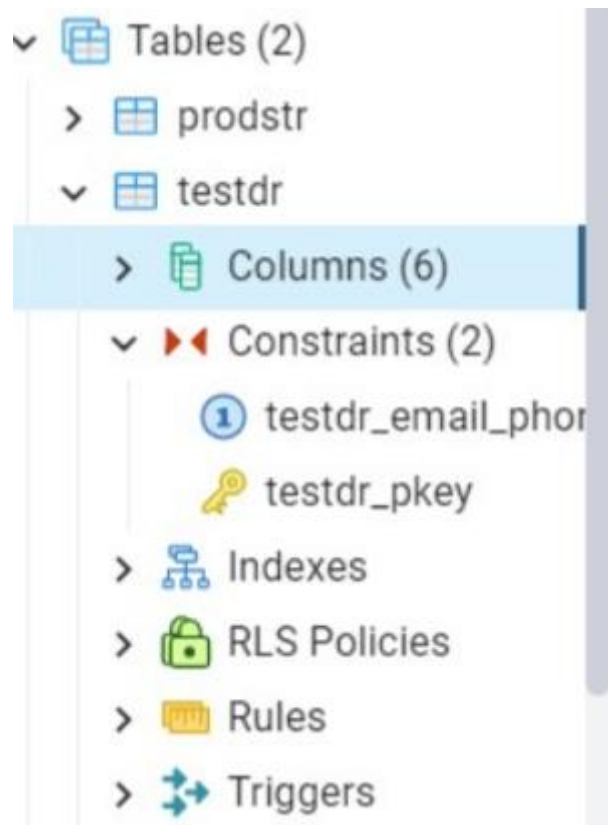


Рисунок 4.11 – Вузли бази даних

### 5.Зовнішні ключі

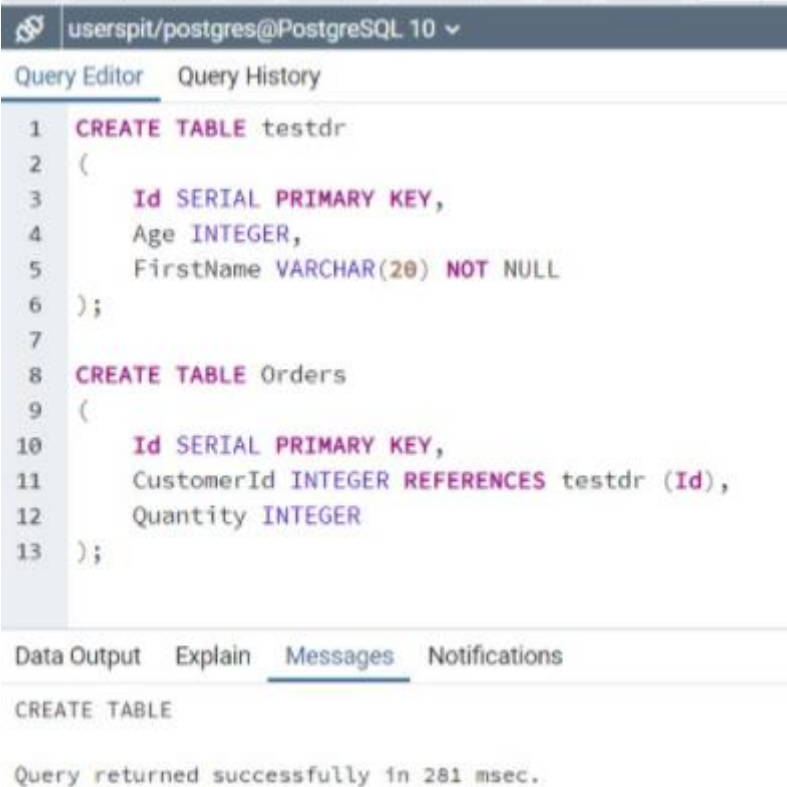
1) Для зв'язку між таблицями застосовуються зовнішні ключі. Зовнішній ключ встановлюється для стовпця з залежної, підпорядкованої таблиці (referencing table), і вказує на один зі стовпців з головної таблиці (referenced table). Як правило, зовнішній ключ вказує на первинний ключ з пов'язаної головної таблиці.

Щоб встановити зв'язок між таблицями, після ключового слова REFERENCES вказується ім'я пов'язаної таблиці і далі в дужках ім'я стовпця з



цієї таблиці, на який буде вказувати зовнішній ключ. Після виразу REFERENCES може йти вираз ON DELETE і ON UPDATE, які уточнюють поведінку при видаленні або оновленні даних.

Загальний синтаксис установки зовнішнього ключа на рівні таблиці:



```
1 CREATE TABLE testdr
2 (
3     Id SERIAL PRIMARY KEY,
4     Age INTEGER,
5     FirstName VARCHAR(20) NOT NULL
6 );
7
8 CREATE TABLE Orders
9 (
10    Id SERIAL PRIMARY KEY,
11    CustomerId INTEGER REFERENCES testdr (Id),
12    Quantity INTEGER
13 );
```

Data Output Explain Messages Notifications

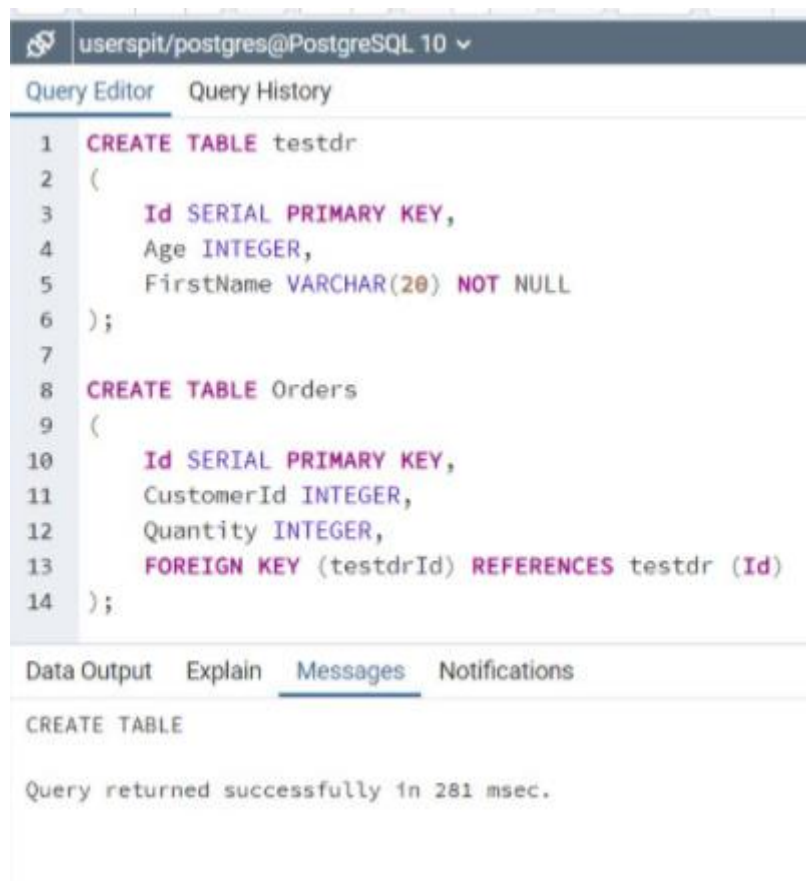
CREATE TABLE

Query returned successfully in 281 msec.

Рисунок 5.1 – Зовнішні ключі на рівні таблиці

Тут визначені таблиці testdr і Orders. testdr є головною і представляє клієнта. Orders є залежною і представляє замовлення, зроблене клієнтом. Ця таблиця через стовпець CustomerId пов'язана з таблицею Customers і її стовпцем Id. Тобто стовпець testdrId є зовнішнім ключем, який вказує на стовпець Id з таблиці testdr.

Визначення зовнішнього ключа на рівні таблиці виглядало б наступним чином:



```
userspit/postgres@PostgreSQL 10 ▾
Query Editor  Query History

1  CREATE TABLE testdr
2  (
3      Id SERIAL PRIMARY KEY,
4      Age INTEGER,
5      FirstName VARCHAR(20) NOT NULL
6  );
7
8  CREATE TABLE Orders
9  (
10     Id SERIAL PRIMARY KEY,
11     CustomerId INTEGER,
12     Quantity INTEGER,
13     FOREIGN KEY (testdrId) REFERENCES testdr (Id)
14 );

Data Output  Explain  Messages  Notifications
CREATE TABLE

Query returned successfully in 281 msec.
```

Рисунок 5.2 – Визначення зовнішнього ключа на рівні таблиці

## 2) ON DELETE і ON UPDATE

За допомогою виразів ON DELETE і ON UPDATE можна встановити дії, які виконуються відповідно при видаленні і зміні пов'язаного рядка з головної таблиці. Для установки подібної дії можна використовувати наступні опції:

- **CASCADE:** автоматично видаляє або змінює рядки з залежної таблиці при видаленні або зміні пов'язаних рядків в головній таблиці.
- **RESTRICT:** запобігає будь-які дії в залежній таблиці при видаленні або зміні пов'язаних рядків в головній таблиці. Тобто фактично будь-які дії відсутні.
- **NO ACTION:** дія за замовчуванням, запобігає будь-які дії в залежній таблиці при видаленні або зміні пов'язаних рядків в головній таблиці. І генерує помилку. На відміну від RESTRICT виконує відкладену перевірку на зв'язаність між таблицями.

- SET NULL: при видаленні зв'язаного рядка з головної таблиці встановлює для стовпця зовнішнього ключа значення NULL.
- SET DEFAULT: при видаленні пов'язаного рядка з головної таблиці встановлює для стовпця зовнішнього ключа значення за замовчуванням, яке задається за допомогою атрибуту DEFAULT. Якщо для стовпця не задано значення за замовчуванням, то в якості нього застосовується значення NULL.

### 3) Каскадне видалення

За замовчуванням, якщо на рядок з головної таблиці за зовнішнім ключом посилається будь-яка рядок із залежної таблиці, то ми не зможемо видалити цей рядок з головної таблиці. Спочатку нам необхідно буде видалити всі пов'язані рядки з залежної таблиці. І якщо при видаленні рядка з головної таблиці необхідно, щоб були видалені всі пов'язані рядки з залежної таблиці, то застосовується каскадне видалення, тобто опція CASCADE:

```
1 CREATE TABLE Orders
2 (
3     Id SERIAL PRIMARY KEY,
4     CustomerId INTEGER,
5     Quantity INTEGER,
6     FOREIGN KEY (CustomerId) REFERENCES Customers (Id) ON DELETE CASCADE
7 );
```

Рисунок 5.3 – Каскадне видалення

### 4) Установка NULL

При установці для зовнішнього ключа опції SET NULL необхідно, щоб стовпець зовнішнього ключа допускав значення NULL:



```
1 CREATE TABLE Orders
2 (
3     Id SERIAL PRIMARY KEY,
4     CustomerId INTEGER,
5     Quantity INTEGER,
6     FOREIGN KEY (testdrId) REFERENCES testdr (Id) ON DELETE SET NULL
7 );
```

Рисунок 5.4 – Установка NULL

##### 5) Встановлення значення за замовчуванням

Якщо для стовпця значення за замовчуванням не задано через параметр DEFAULT, то в якості такого використовується значення NULL (якщо стовпець допускає NULL).



```
1 CREATE TABLE Orders
2 (
3     Id SERIAL PRIMARY KEY,
4     CustomerId INTEGER DEFAULT 1,
5     Quantity INTEGER,
6     FOREIGN KEY (testdrId) REFERENCES testdr (Id) ON DELETE SET DEFAULT
7 );
```

Рисунок 5.5 – Встановлення значення за замовчуванням

## 6. Зміна таблиць

### 1) Додавання нового стовпця

Додамо в таблицю Customers новий стовпець Phone:



Рисунок 6.1 – Додавання нового стовпця

Тут стовпець Phone має тип CHARACTER VARYING (20), і для нього визначено атрибут NULL, тобто стовпець допускає відсутність значення.

2) Але що якщо нам треба додати стовпець, який не повинен приймати значення NULL? Якщо в таблиці є дані, то наступна команда не буде виконана:

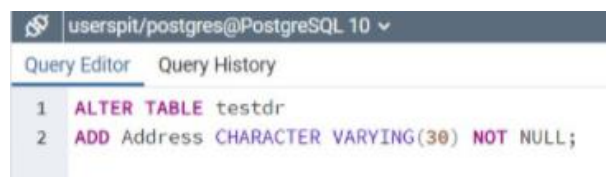


Рисунок 6.2 – Address NOT NULL

3) Тому в даному випадку рішення полягає в установці значення за замовчуванням через атрибут DEFAULT:



Рисунок 6.3 – DEFAULT

### 4) Видалення стовпця

Видалимо стовпець Address з таблиці testdr:

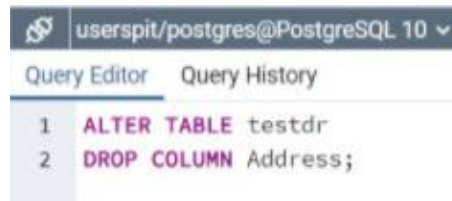


Рисунок 6.4 – Видалення стовпця

#### 5) Зміна типу стовпця

Для зміни типу застосовується ключове слово TYPE. Змінимо в таблиці customers тип даних у стовпця FirstName на VARCHAR (50) (він же VARYING CHARACTER (50)):



Рисунок 6.5 – Зміна типу стовпця

#### 6) Зміна обмежень стовпця

Для додавання обмеження застосовується оператор SET, після якого вказується обмеження. Наприклад, встановимо для стовпця FirstName обмеження NOT NULL:

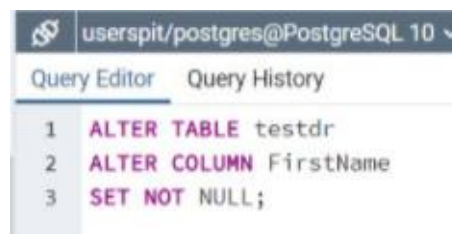


Рисунок 6.6 – Зміна обмежень стовпця

Для видалення обмеження застосовується оператор DROP, після якого вказується обмеження. Наприклад, видалимо вище встановлене обмеження:



Рисунок 6.7 – оператор DROP

#### 7) Зміна обмежень таблиці

Додавання обмеження CHECK:



Рисунок 6.8 – обмеження CHECK

Додавання первинного ключа PRIMARY KEY:



Рисунок 6.9 – Додавання первинного ключа PRIMARY KEY

В даному випадку передбачається, що в таблиці вже є стовпець Id, який не має обмеження PRIMARY KEY. А за допомогою вищевказаного скрипта встановлюється обмеження PRIMARY KEY.

8) Додавання обмеження UNIQUE-визначимо для стовпця email унікальні значення:



Рисунок 6.10 – Додавання обмеження UNIQUE

При додаванні обмеження кожному з них дається певне ім'я. Наприклад, вище додане обмеження для CHECK буде називатися customers\_age\_check. Імена обмежень можна подивитися в таблиці через pgAdmin.

9) Також ми можемо явним чином призначити обмеження при додаванні ім'я за допомогою оператора CONSTRAINT.



Рисунок 6.11 – Додавання ім'я за допомогою оператора CONSTRAINT

10) В даному випадку обмеження буде називатися "iphone\_unique".

Щоб видалити обмеження, Треба знати його ім'я, яке вказується після виразу DROP CONSTRAINT. Наприклад, видалимо вище додане обмеження:



Рисунок 6.12 – DROP CONSTRAINT

11) Перейменування стовпця і таблиці

Перейменуємо стовпець Address в City:





Рисунок 6.13 – Перейменування стовпця

Перейменуємо таблицю testdr в Users:



Рисунок 6.14 – Перейменування таблиці