Lab

Topic: Definition of data structure in PostgreSQL.

Purpose: Definition of data structure.

Progress

1. Creating and deleting a database:

1) To create a new database (DB), open pgAdmin. In the left part of the program, select a database, for example, a standard postgres database, and click on it with the right mouse button.

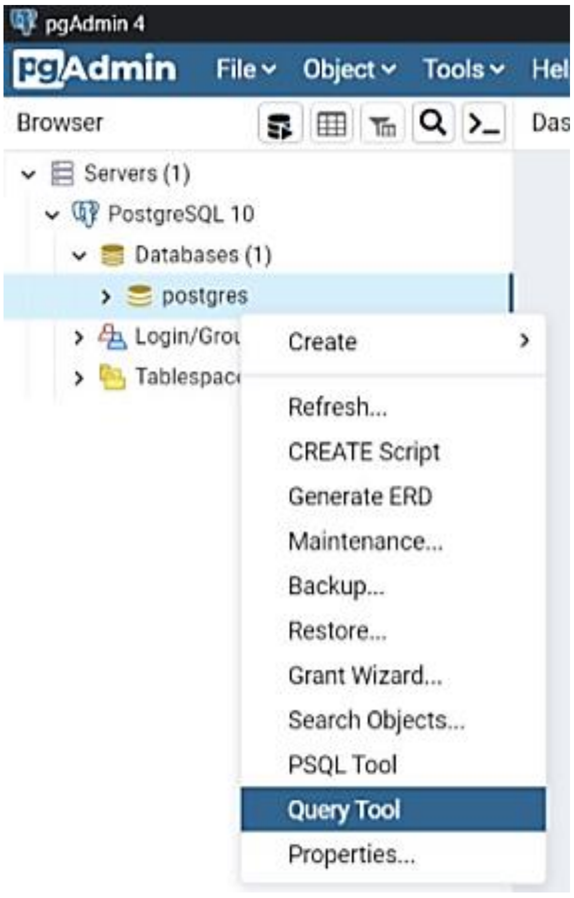
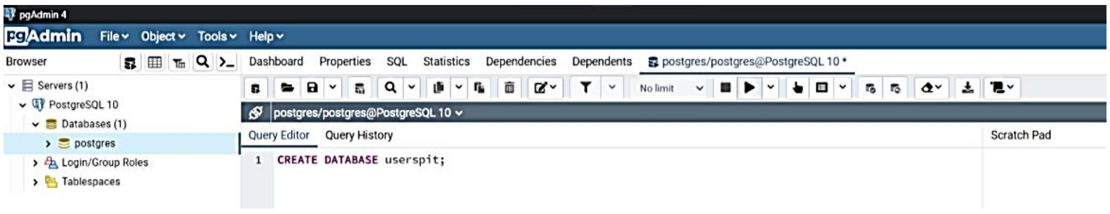


Figure 1 – The interface of the pgAdmin package

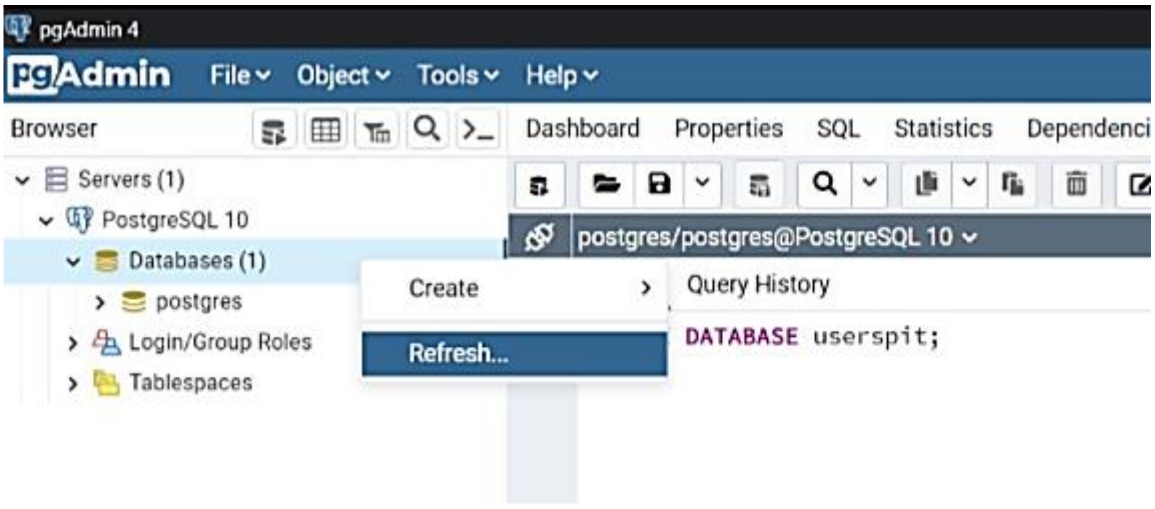
2) In the menu, select the Query Tool item, and a field for entering the SQL code will open in the central part of the program. Enter the following code into this field: CREATE `DATABASE usersplit;`



```
CREATE DATABASE userspit;
```

Figure 1.2 – Creating a database

3) To see our database, right-click on the Databases node in the left part and select Refresh in the context menu.



```
DATABASE userspit;
```

Figure 1.3 – Updating databases

4) An update will occur and we will see the database created.

By default, the base is inactive, so its icon is gray. But to connect to it, just click on it and open its node.

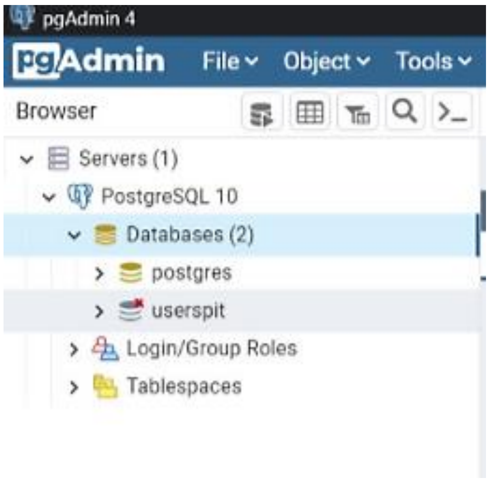


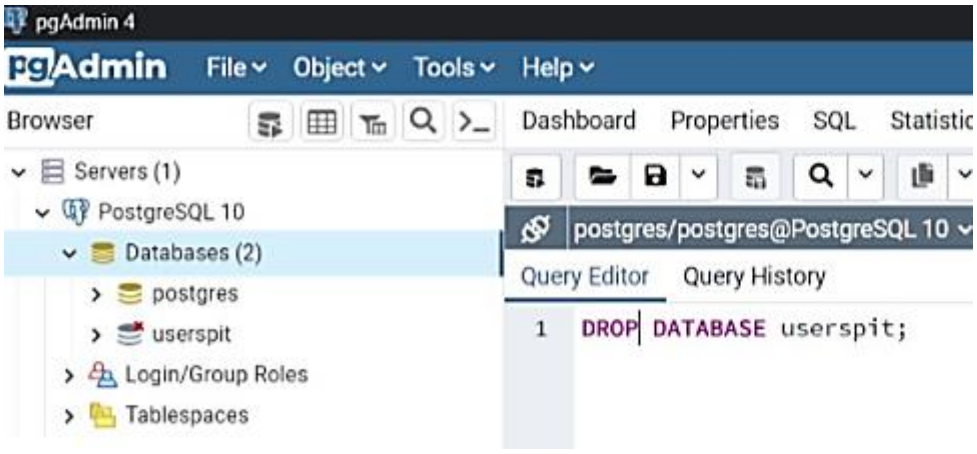
Figure 1.4 - database

5) Deleting the database:

To delete a database, use the DROP DATABASE command followed by the name of the database.

The database to be deleted must be inactive, that is, the connection to it must be closed.

For example, deleting the userspiit database:


pgAdmin 4
pgAdmin File Object Tools Help
Browser
Dashboard Properties SQL Statistic
Servers (1)
PostgreSQL 10
Databases (2)
postgres
userspit
Login/Group Roles
Tablespaces
postgres/postgres@PostgreSQL 10
Query Editor Query History

```
DROP DATABASE userspit;
```

Figure 1.5 - Deleting a database

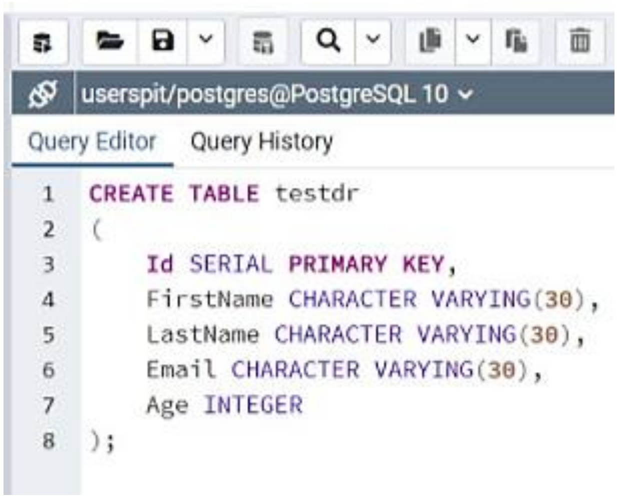
2. Creating and deleting tables:

To create tables, the CREATE TABLE command is used, followed by the name of the table. Also, with this command, you can use a number of operators that define table columns and their attributes. The general syntax for creating a table is as follows:

```
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) Let's create a table in the database using pgAdmin. To do this, first select the target database in pgAdmin, right-click on it and select Query Tool in the context menu. After that, a field for entering the SQL code will open. Moreover, the table will be created precisely for the database for which we will open this field for entering SQL.

Next, enter the following set of expressions in the field opened in the central part of the program:



```
CREATE TABLE testdr
(
    Id SERIAL PRIMARY KEY,
    FirstName CHARACTER VARYING(30),
    LastName CHARACTER VARYING(30),
    Email CHARACTER VARYING(30),
    Age INTEGER
);
```

Figure 2 – Creating a table

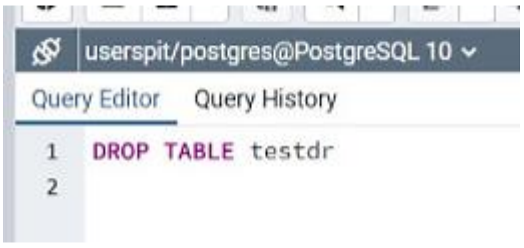
In this case, five columns are defined in the testdr table: Id, FirstName, LastName, Age, Email. The first column, Id, represents the customer ID, it serves as the primary key and is therefore of type SERIAL. In fact, this column will store the Numeric value 1, 2, 3, etc., which will automatically increase by one for each new row.

The next three columns represent the customer's first name, last name, and email address and are of type CHARACTER VARYING(30), i.e., represent a string of no more than 30 characters.

The last column - Age represents the age of the user and has type INTEGER, that is, it stores numbers.

2) Deleting tables

To delete tables, use the DROP TABLE command, which has the following syntax:



```
DROP TABLE testdr
```

Figure 2.1 – Deleting tables

3. Data types in PostgreSQL:

When defining a table, you must specify the data type for all its columns. The data type determines the range of values that can be stored in a column, how much space they will occupy in memory. PostgreSQL supports a rich palette of various data types, among which can be conditionally divided into subgroups: Numeric, symbolic, logical, date and time, binary and a number of others.

1) Numeric data types

- serial: represents an auto-incrementing numeric value that occupies 4 bytes and can store numbers from 1 to 2147483647. A value of this type is formed by auto-incrementing the value of the previous string. Therefore, as a rule, this type is used to define string identifiers.

- smallserial: Represents an auto-incrementing numeric value that occupies 2 bytes and can store numbers from 1 to 32767. Analogous to serial for small numbers.
- bigserial: Represents an auto-incrementing numeric value that occupies 8 bytes and can store numbers from 1 to 9223372036854775807. Similar to serial for large numbers.
- smallint: stores numbers from -32768 to +32767. Occupies 2 bytes. Has the alias int2.
- integer: stores numbers from -2147483648 to +2147483647. Occupies 4 bytes. Has aliases int and int4.
- bigint: stores numbers from -9223372036854775808 to +9223372036854775807. Takes 8 bytes. Has the alias int8.
- numeric: stores fixed-precision numbers that can have up to 131072 integer digits and up to 16383 decimal digits.

This type can accept two parameters precision and scale: numeric (precision, scale).

The precision parameter specifies the maximum number of digits that a number can store.

The scale parameter represents the maximum number of digits that a number can contain after the decimal point. This value must be in the range from 0 to the value of the precision parameter. By default, it is 0.

For example, for the number 23.5141, the precision is 6, and the scale is 4.

- decimal: stores fixed-precision numbers that can have up to 131072 digits in the integer part and up to 16383 digits in the fractional part. Same as numeric.
- real: Stores floating-point numbers in the range 1E-37 to 1E+37. Occupies 4 bytes. Has the alias float4.

- double precision: stores floating-point numbers from the range 1E - 307 to 1E + 308. Occupies 8 bytes. Has the alias float8.

Examples of use:

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

2) Types for working with currency (monetary units)

To work with monetary units, the money type is defined, which can take values in the range from -92233720368547758.08 to +92233720368547758.07 and occupies 8 bytes.

3) Character type

- character (n): represents a string of a fixed number of characters. The parameter specifies the specified number of characters in a line. Has the alias char (n).
- character varying(n): represents a string of a fixed number of characters. The parameter specifies the specified number of characters in a line. Has alias varchar(n).
- text: represents text of arbitrary length.

4) Binary data

The bytea type is defined for storing binary data. It stores data as binary strings that represent a sequence of octets or bytes.

5) Types for working with dates and times

- timestamp: stores the date and time. Takes 8 bytes. For dates, the lowest value is 4713 BC. e., the very top value - 294276 BC. is.
- timestamp with time zone: Same as timestamp, only adds time zone data.
- date: represents the date from 4713 B.C. e. to 5874897 e. occupies 4 bytes.
- time: Stores the time to the nearest 1 microsecond without specifying a time zone. Takes values from 00:00:00 to 24:00:00. Occupies 8 bytes.
- time with time zone: stores the time to the nearest 1 microsecond with the time zone specified. Accepts values from 00:00:00+1459 to 24:00:00-1459. Takes 12 bytes.
- interval: represents a time interval. Takes 16 bytes.

Common date formats:

yyyy-mm-dd - 1999-01-08

Month dd, yyyy - January 8, 1999

mm/dd/yyyy - 1/8/1999

Common time formats:

hh:mi - 13:21

hh:mi am/pm - 1:21 pm

hh:mi:ss - 1:21:34

6) Logical type

The boolean type can store one of two values: true or false.

Instead of true, you can specify the following values: TRUE, 't', 'true', 'y', 'yes', 'on', '1'.

The following values can be specified instead of false: FALSE, 'f', 'false', 'n', 'no', 'off', '0'.

7) Types for representing Internet addresses

- cidr: Internet address in IPv4 and IPv6 format. For example, 192.168.0.1. Occupies 7 to 19 bytes.
- inet: Internet address in the format cidr / y, where cidr is an address in IPv4 or IPv6 format, and /y is the number of bits in the address (if this option is not specified, 34 is used for IPv4, 128 for IPv6). For example, 192.168.0.1 / 24 or 2001:4 f8:3:ba:2E0:81 ff:fe 22: d1f1/128. Occupies 7 to 19 bytes.
- macaddr: Stores the MAC address. Takes 6 bytes.
- macaddr8: Stores the MAC address in EUI-64 format. Takes 8 bytes.

8) Geometric types

- point: represents a point on the plane in (x,y) format. Takes 16 bytes.
- line: represents a line of undefined length in the format {A,B,C}. Occupies 32 bytes.
- lseg: represents a segment in the format ((x1,y1), (x2,y2)). Occupies 32 bytes.
- box: represents a rectangle in the format ((x1,y1), (x2,y2)). Occupies 32 bytes.
- path: represents a set of connected points. In the format ((x1,y1),...) the path is closed (the first and last point are connected by a line) and actually represents a polygon. In the format [(x1,y1),...] the path is open takes 16 + 16N bytes.
- polygon: represents a polygon in the format ((x1,y1),...). Takes 40 + 16N bytes.
- circle: represents a circle in <(x,y),r> format. Occupies 24 bytes.

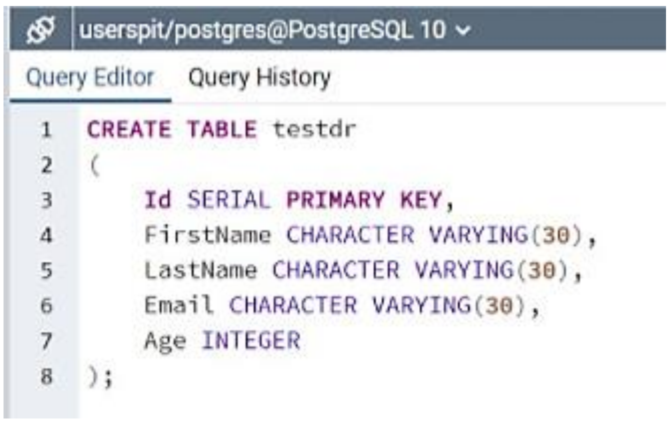
9) Other types of data

- json: store json data in text form.
- json: store json data in binary format.
- uuid: Stores a Universally Unique Identifier (UUID), such as a0eebc99-9c0b-4ef8-bb6d-6bb9bd380a11. Occupies 32 bytes.
- xml: stores data in XML format.

4. Limitation of columns and tables

1)PRIMARY KEY

Using the PRIMARY KEY expression, a column can be made a primary key.

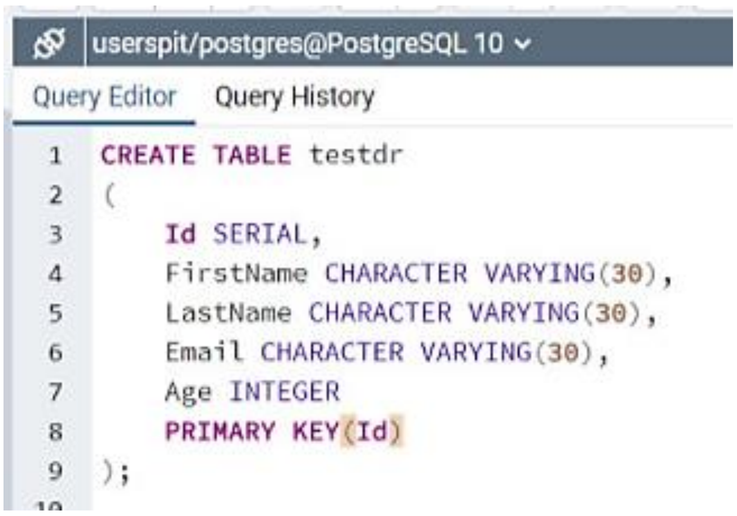


```
CREATE TABLE testdr
(
    Id SERIAL PRIMARY KEY,
    FirstName CHARACTER VARYING(30),
    LastName CHARACTER VARYING(30),
    Email CHARACTER VARYING(30),
    Age INTEGER
);
```

Figure 4.1 - Primary key PRIMARY KEY

A primary key uniquely identifies a row in a table. Columns of type SERIAL do not necessarily have to be primary keys, they can represent any other type.

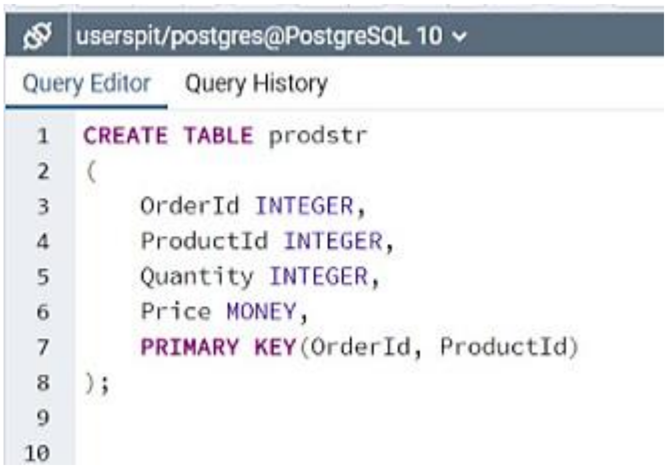
2)Setting the primary key at the table level:



```
CREATE TABLE testdr
(
    Id SERIAL,
    FirstName CHARACTER VARYING(30),
    LastName CHARACTER VARYING(30),
    Email CHARACTER VARYING(30),
    Age INTEGER
    PRIMARY KEY(Id)
);
```

Figure 4.2 – PRIMARY KEY at the table level

3) The primary key can be a compound key. Such a key may be needed if we have two columns that must uniquely identify a row in the table. Example:



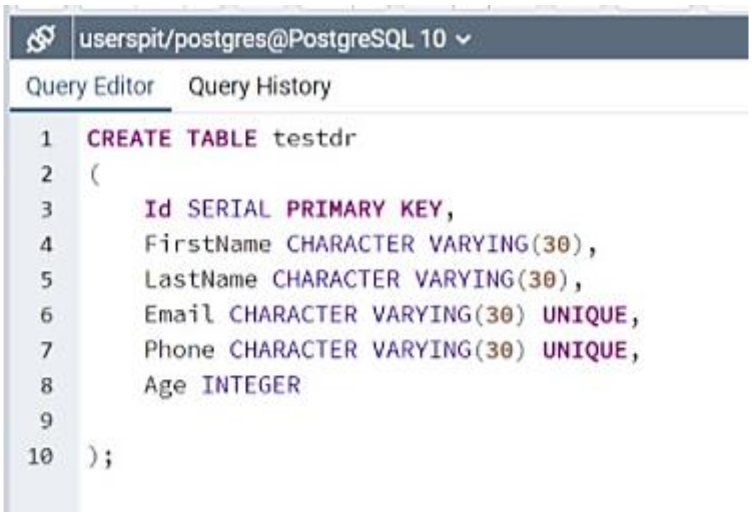
```
CREATE TABLE prodstr
(
    OrderId INTEGER,
    ProductId INTEGER,
    Quantity INTEGER,
    Price MONEY,
    PRIMARY KEY(OrderId, ProductId)
);
```

Figure 4.3 – compound key

Here, the fields OrderId and ProductId together act as a component of the primary key. That is, there cannot be two rows in the OrderLines table where both of these fields would have the same values at the same time.

4) UNIQUE

if we want the column to have only unique values, then the UNIQUE attribute can be defined for it.



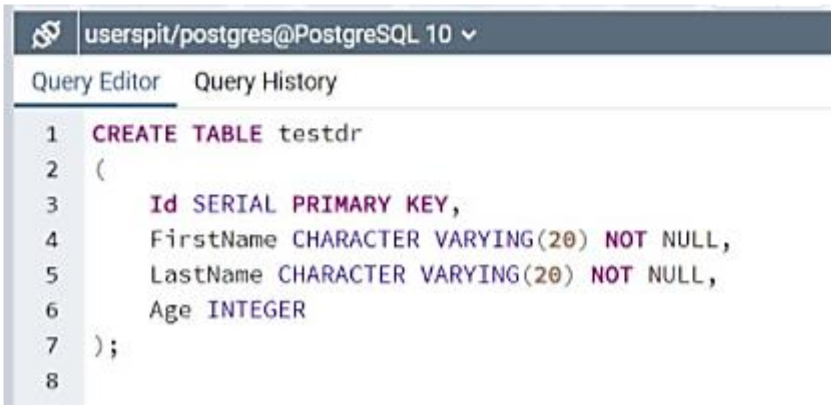
```
CREATE TABLE testdr
(
    Id SERIAL PRIMARY KEY,
    FirstName CHARACTER VARYING(30),
    LastName CHARACTER VARYING(30),
    Email CHARACTER VARYING(30) UNIQUE,
    Phone CHARACTER VARYING(30) UNIQUE,
    Age INTEGER

);
```

Figure 4.4 – UNIQUE

In this case, the columns that represent the email address and the phone number will have unique values. And we will not be able to add two rows to the table in which the value for these columns will match.

5) NULL and NOT NULL

To specify whether a column can accept the value NULL, you can specify the NULL or NOT NULL attribute when defining the column. If this attribute is not explicitly used, the column will default to NULL. The exception is the case when the column acts as a primary key - in this case, the column defaults to NOT NULL.

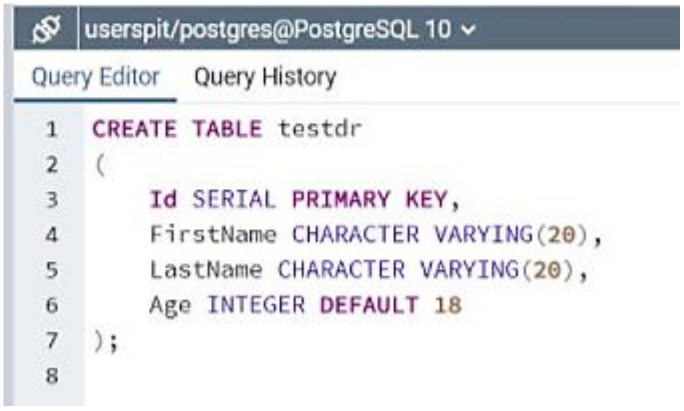


```sql
CREATE TABLE testdr
(
    Id SERIAL PRIMARY KEY,
    FirstName CHARACTER VARYING(20) NOT NULL,
    LastName CHARACTER VARYING(20) NOT NULL,
    Age INTEGER
);
```

Figure 4.5 - NULL and NOT NULL

6) DEFAULT

The DEFAULT attribute specifies the default value for the column. If no value is provided for a column when adding data, the default value will be used.



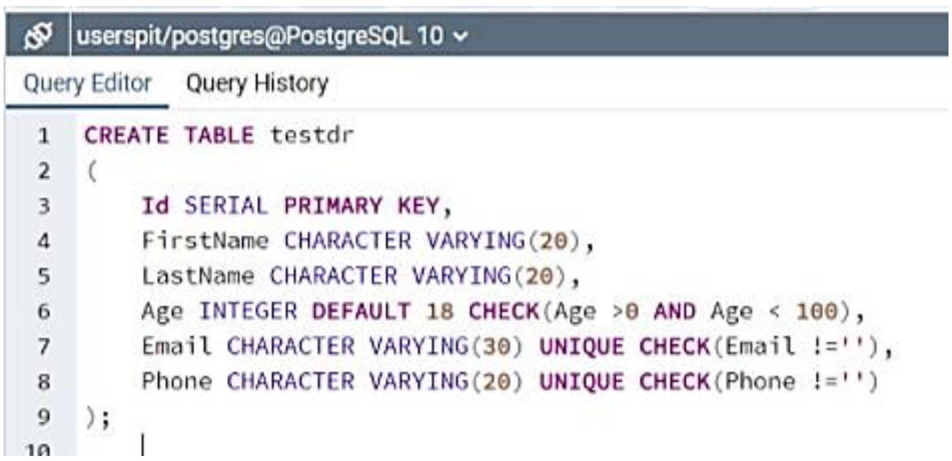
```sql
CREATE TABLE testdr
(
    Id SERIAL PRIMARY KEY,
    FirstName CHARACTER VARYING(20),
    LastName CHARACTER VARYING(20),
    Age INTEGER DEFAULT 18
);
```

Figure 4.6 - DEFAULT

Here, the Age column has a default value of 18.

7) CHECK

The CHECK keyword sets a limit on the range of values that can be stored in a column. To do this, after the word CHECK, a condition is indicated in parentheses, which must be met by a column or several columns. For example, the age of customers cannot be less than 0 or greater than 100:



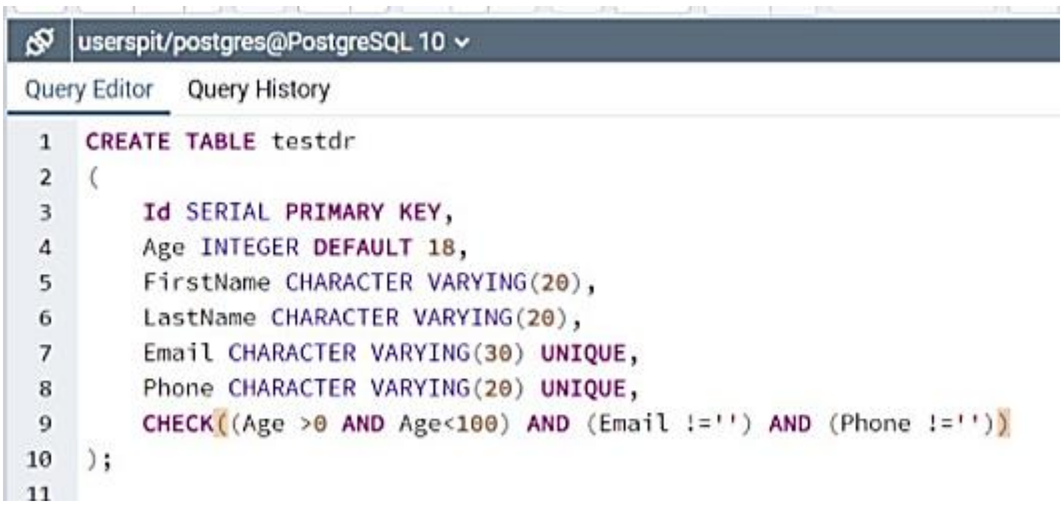
```
CREATE TABLE testdr
(
    Id SERIAL PRIMARY KEY,
    FirstName CHARACTER VARYING(20),
    LastName CHARACTER VARYING(20),
    Age INTEGER DEFAULT 18 CHECK(Age >0 AND Age < 100),
    Email CHARACTER VARYING(30) UNIQUE CHECK(Email !=''),
    Phone CHARACTER VARYING(20) UNIQUE CHECK(Phone !='')
);
```

Figure 4.7 - CHECK

It also specifies that the Email and Phone columns cannot have an empty string as a value (an empty string is not equivalent to a NULL value).

The AND keyword is used to connect conditions. Conditions can be set in the form of comparison operations greater than (>), less than (<), not equal to (!=).

You can also use CHECK to create restrictions for the table as a whole:



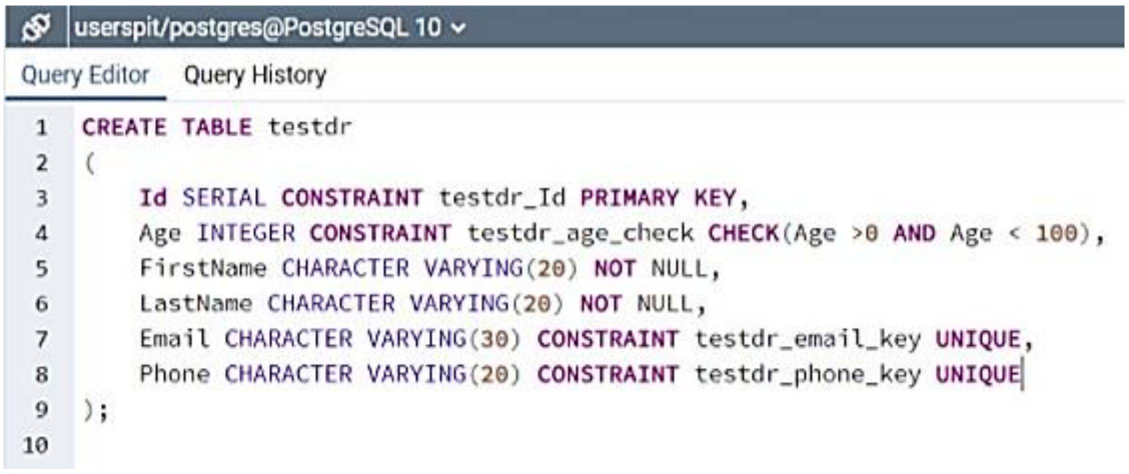
```
CREATE TABLE testdr
(
    Id SERIAL PRIMARY KEY,
    Age INTEGER DEFAULT 18,
    FirstName CHARACTER VARYING(20),
    LastName CHARACTER VARYING(20),
    Email CHARACTER VARYING(30) UNIQUE,
    Phone CHARACTER VARYING(20) UNIQUE,
    CHECK((Age >0 AND Age<100) AND (Email !='') AND (Phone !=''))
);
```

Figure 4.8 – keyword AND

8) CONSTRAINT operator

You can specify a name for the constraints using the CONSTRAINT keyword. PRIMARY KEY, UNIQUE, CHECK can be used as restrictions.

Constraint names can be specified at the column level. They are specified after CONSTRAINT before attributes:

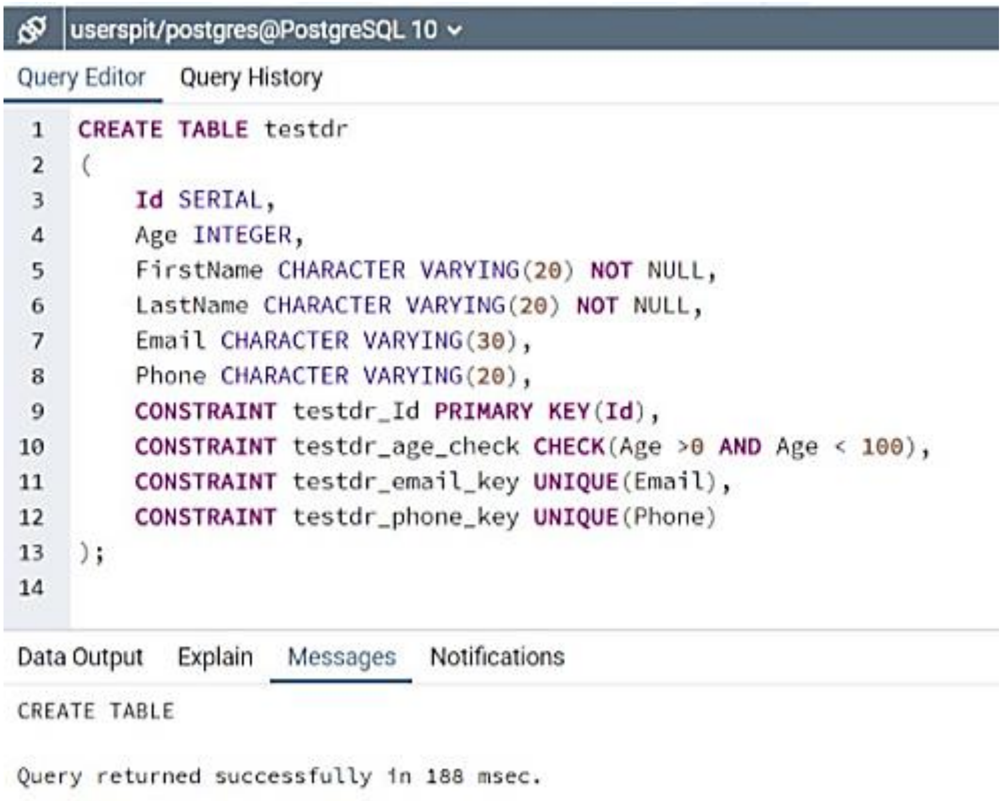


```
CREATE TABLE testdr
(
    Id SERIAL CONSTRAINT testdr_Id PRIMARY KEY,
    Age INTEGER CONSTRAINT testdr_age_check CHECK(Age >0 AND Age < 100),
    FirstName CHARACTER VARYING(20) NOT NULL,
    LastName CHARACTER VARYING(20) NOT NULL,
    Email CHARACTER VARYING(30) CONSTRAINT testdr_email_key UNIQUE,
    Phone CHARACTER VARYING(20) CONSTRAINT testdr_phone_key UNIQUE
);
```

Figure 4.9 – CONSTRAINT operator

In principle, it is not necessary to specify the names of the restrictions, when setting the corresponding attributes, SQL Server automatically determines their names. But, knowing the name of the restriction, we can refer to it, for example, to remove it.

And you can also set all the names of the constraints through table attributes:



```
CREATE TABLE testdr
(
    Id SERIAL,
    Age INTEGER,
    FirstName CHARACTER VARYING(20) NOT NULL,
    LastName CHARACTER VARYING(20) NOT NULL,
    Email CHARACTER VARYING(30),
    Phone CHARACTER VARYING(20),
    CONSTRAINT testdr_Id PRIMARY KEY(Id),
    CONSTRAINT testdr_age_check CHECK(Age >0 AND Age < 100),
    CONSTRAINT testdr_email_key UNIQUE(Email),
    CONSTRAINT testdr_phone_key UNIQUE(Phone)
);
```

```
CREATE TABLE

Query returned successfully in 188 msec.
```

Figure 4.10 – CONSTRAINT table attributes

Regardless of whether the CONSTRAINT statement is used to create the constraints or not (in which case PostgreSQL names them itself when setting the constraints), we can view all the constraints in pgAdmin on the database node in the subnode:

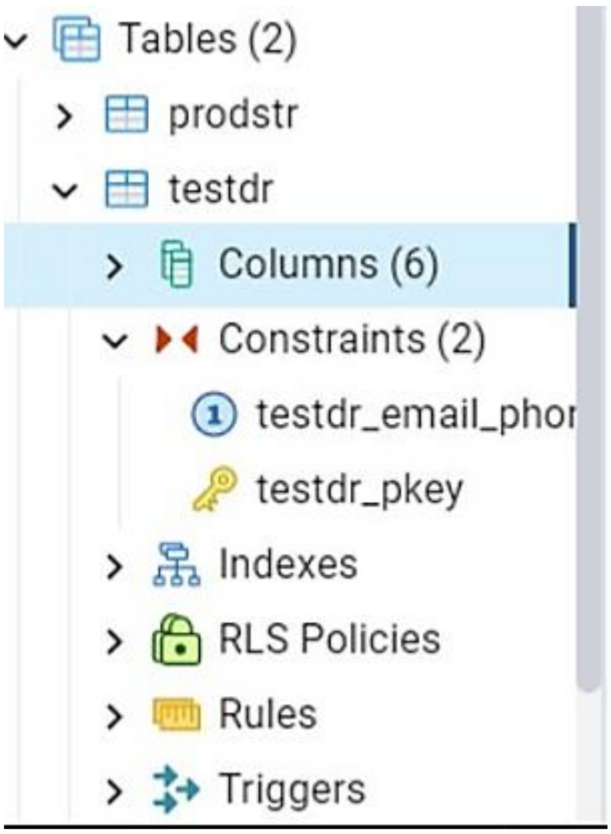


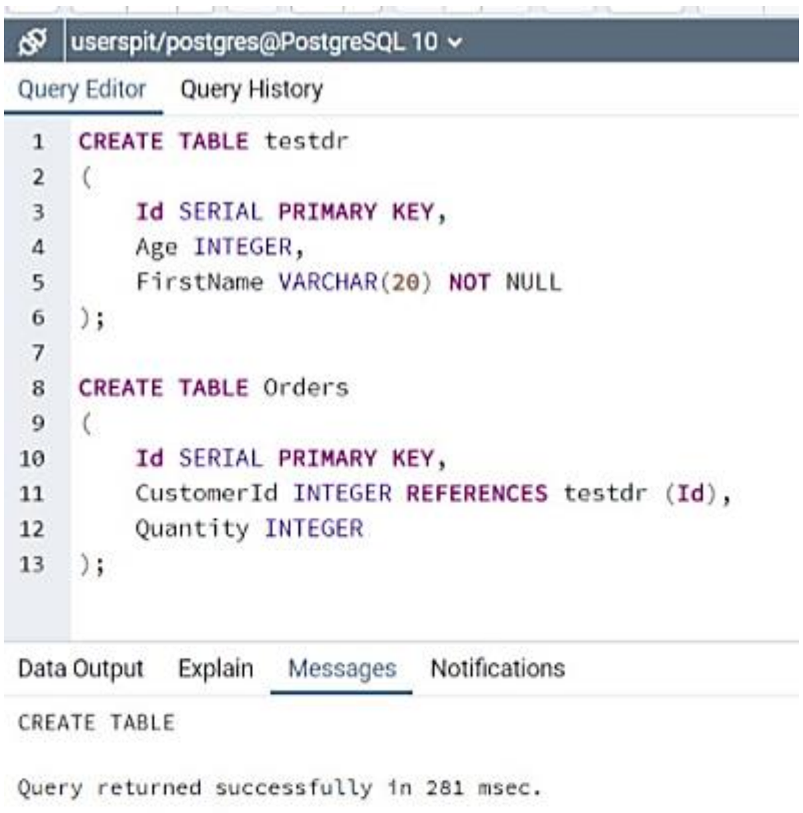
Figure 4.11 - Database nodes

5. External keys

1)Foreign keys are used to connect tables. The foreign key is set for a column from a dependent, subordinate table (referencing table) and points to one of the columns from the main table (referenced table). Typically, a foreign key points to a primary key from the associated master table.

To establish a relationship between tables, after the keyword REFERENCES, the name of the related table is indicated, and then in parentheses, the name of the column from this table, which will be pointed to by the foreign key. The REFERENCES expression can be followed by the ON DELETE and ON UPDATE expressions, which specify the behavior when data is deleted or updated.

The general syntax for setting a table-level foreign key is:



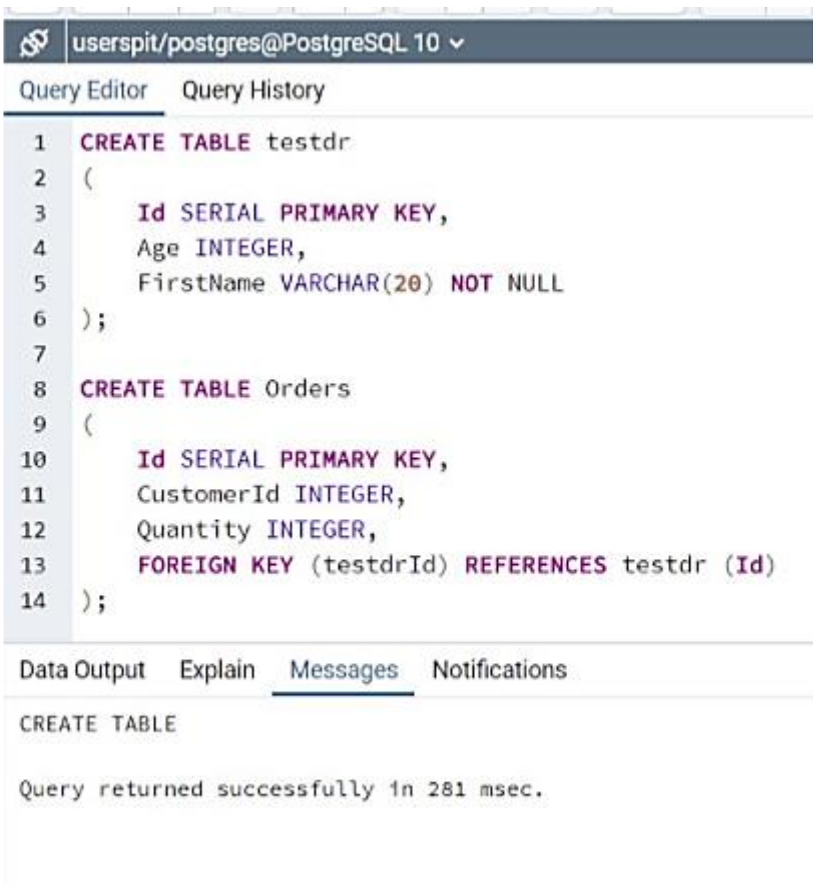
```sql
CREATE TABLE testdr
(
    Id SERIAL PRIMARY KEY,
    Age INTEGER,
    FirstName VARCHAR(20) NOT NULL
);

CREATE TABLE Orders
(
    Id SERIAL PRIMARY KEY,
    CustomerId INTEGER REFERENCES testdr (Id),
    Quantity INTEGER
);
```

```
CREATE TABLE

Query returned successfully in 281 msec.
```

Figure 5.1 - Foreign keys at the table level

The tables testdr and Orders are defined here. testdr is the main one and represents the client. Orders is dependent and represents the order placed by the customer. This table is related to the Customers table and its Id column through the CustomerId column. That is, the testdrId column is a foreign key that points to the Id column from the testdr table.

A table-level foreign key definition would look like this:



```
CREATE TABLE testdr
(
    Id SERIAL PRIMARY KEY,
    Age INTEGER,
    FirstName VARCHAR(20) NOT NULL
);

CREATE TABLE Orders
(
    Id SERIAL PRIMARY KEY,
    CustomerId INTEGER,
    Quantity INTEGER,
    FOREIGN KEY (testdrId) REFERENCES testdr (Id)
);
```

```
CREATE TABLE

Query returned successfully in 281 msec.
```

Figure 5.2 – Definition of a foreign key at the table level

2) ON DELETE and ON UPDATE

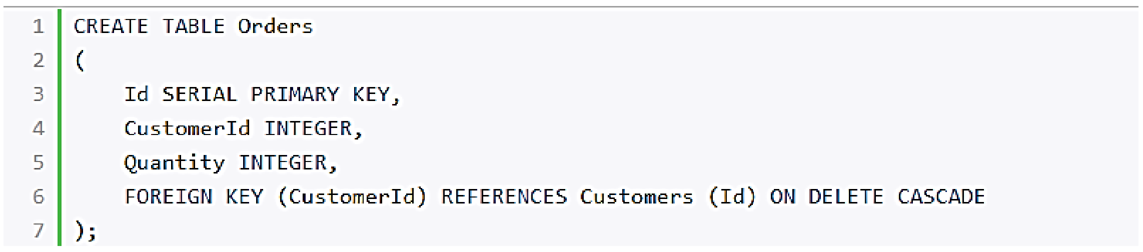
With the help of expressions ON DELETE and ON UPDATE, you can set the actions that are performed, respectively, when deleting and changing the related row from the main table. To set such an action, you can use the following options:

- CASCADE: Automatically deletes or changes rows from a dependent table when related rows in the main table are deleted or changed.
- RESTRICT: Prevents any actions in the dependent table when deleting or changing related rows in the main table. That is, in fact, there are no actions.
- NO ACTION: default action, prevents any actions in the dependent table when deleting or changing related rows in the main table. And generates an error. Unlike RESTRICT, it performs a delayed check for connectivity between tables.
- SET NULL: when deleting a related row from the main table, sets the foreign key column to NULL.

- SET DEFAULT: when deleting a related row from the main table, sets the foreign key column to the default value, which is specified using the DEFAULT attribute. If no default value is set for a column, then NULL is used as the default value.

3)Cascading delete

By default, if a row from the main table is referenced by a foreign key from any row from a dependent table, then we will not be able to delete that row from the main table. First, we will need to delete all related rows from the dependent table. And if when deleting a row from the main table, it is necessary to delete all related rows from the dependent table, then cascade deletion is used, that is, the CASCADE option:

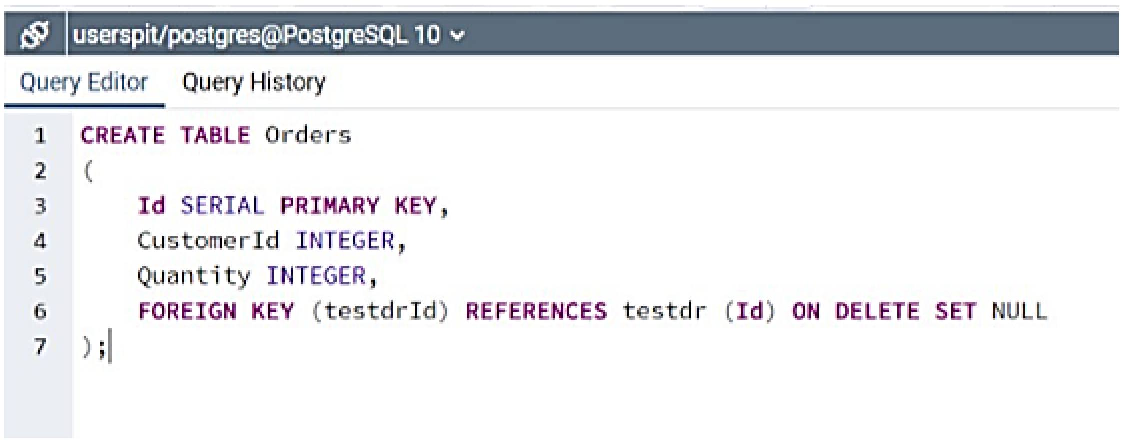


```
CREATE TABLE Orders
(
    Id SERIAL PRIMARY KEY,
    CustomerId INTEGER,
    Quantity INTEGER,
    FOREIGN KEY (CustomerId) REFERENCES Customers (Id) ON DELETE CASCADE
);
```

Figure 5.3 - Cascading deletion

4)Setting NULL

When setting the SET NULL option for a foreign key, it is necessary that the foreign key column allows NULL values:

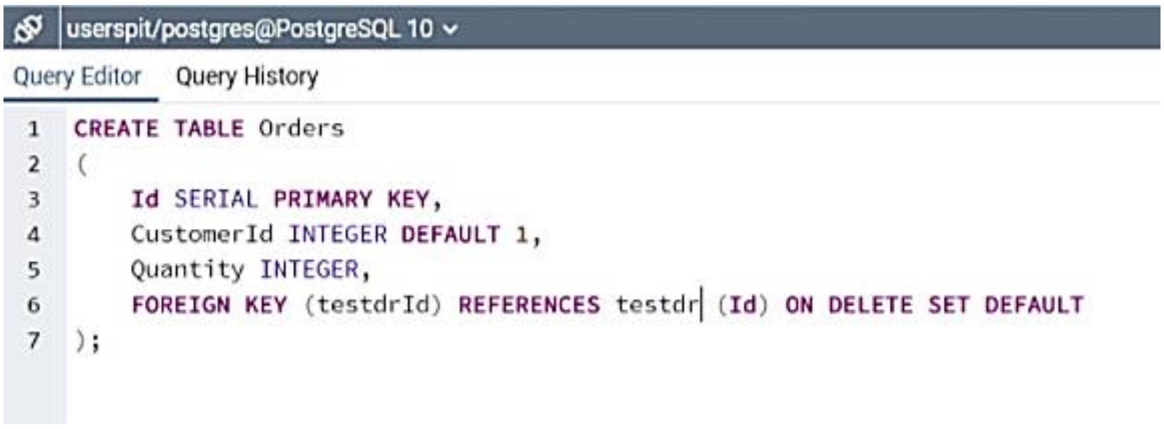


```
CREATE TABLE Orders
(
    Id SERIAL PRIMARY KEY,
    CustomerId INTEGER,
    Quantity INTEGER,
    FOREIGN KEY (testdrId) REFERENCES testdr (Id) ON DELETE SET NULL
);
```

Figure 5.4 - Setting NULL

5) Setting the default value

If the column has no default value set via the DEFAULT parameter, then NULL is used as such (if the column allows NULL).



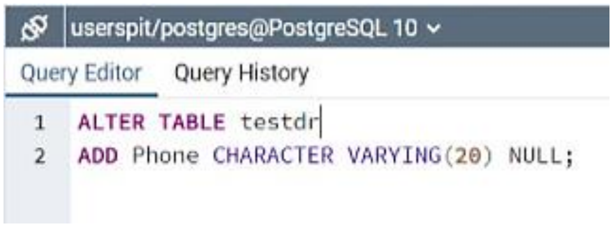
```
CREATE TABLE Orders
(
    Id SERIAL PRIMARY KEY,
    CustomerId INTEGER DEFAULT 1,
    Quantity INTEGER,
    FOREIGN KEY (testdrId) REFERENCES testdr (Id) ON DELETE SET DEFAULT
);
```

Figure 5.5 - Setting the default value

6. Change of tables

1)Adding a new column

Let's add a new Phone column to the Customers table:

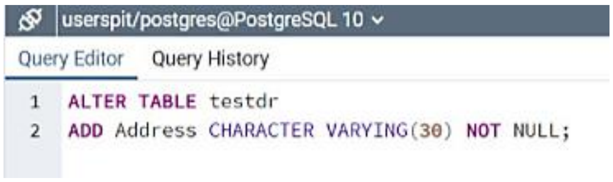


```
ALTER TABLE testdr
ADD Phone CHARACTER VARYING(20) NULL;
```

Figure 6.1 - Adding a new column

Here, the Phone column is of type CHARACTER VARYING (20) and has a NULL attribute defined for it, meaning the column accepts no value.

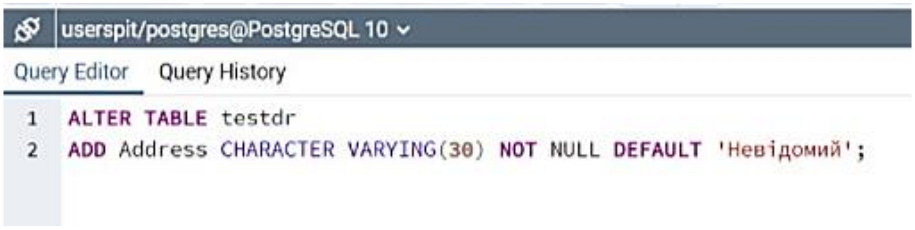
2) But what if we need to add a non-nullable column? If there is data in the table, then the following command will not be executed:



```
ALTER TABLE testdr
ADD Address CHARACTER VARYING(30) NOT NULL;
```

Figure 6.2 – Address NOT NULL

3)Therefore, in this case, the solution is to set the default value via the DEFAULT attribute:

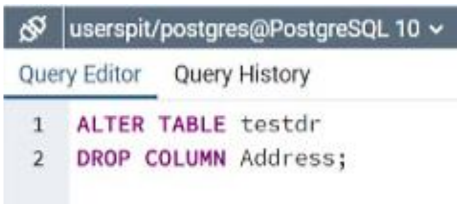


```
ALTER TABLE testdr
ADD Address CHARACTER VARYING(30) NOT NULL DEFAULT 'Невідомий';
```

Figure 6.3 - DEFAULT

4) Deleting a column

Let's delete the Address column from the testdr table:

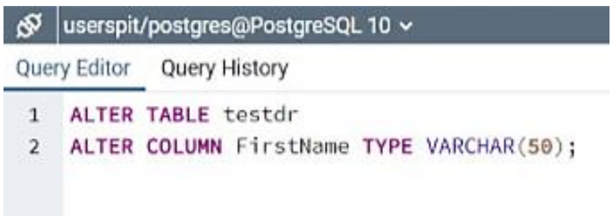


```
ALTER TABLE testdr
DROP COLUMN Address;
```

Figure 6.4 - Deleting a column

5) Changing the column type

The TYPE keyword is used to change the type. Let's change the data type in the FirstName column to VARCHAR (50) (aka VARYING CHARACTER (50)) in the customers table:

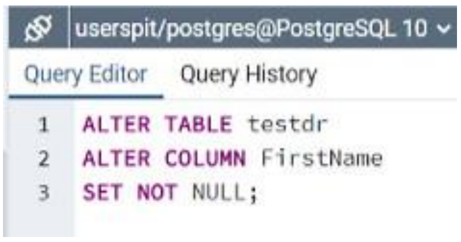


```
ALTER TABLE testdr
ALTER COLUMN FirstName TYPE VARCHAR(50);
```

Figure 6.5 – Changing the column type

6)Change column constraints

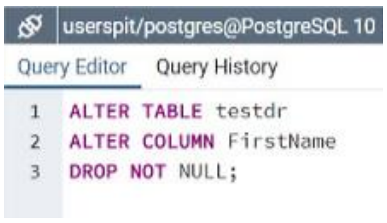
To add a constraint, the SET statement is used, followed by the constraint. For example, let's set the NOT NULL constraint for the FirstName column:



```
ALTER TABLE testdr
ALTER COLUMN FirstName
SET NOT NULL;
```

Figure 6.6 – Changing column constraints

To remove a constraint, the DROP statement is used, followed by the constraint. For example, let's remove the restriction set above:

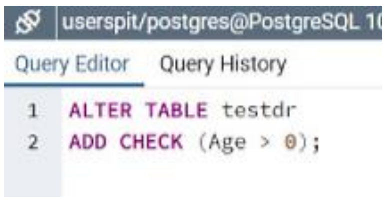


```
ALTER TABLE testdr
ALTER COLUMN FirstName
DROP NOT NULL;
```

Figure 6.7 – DROP operator

7) Change table restrictions

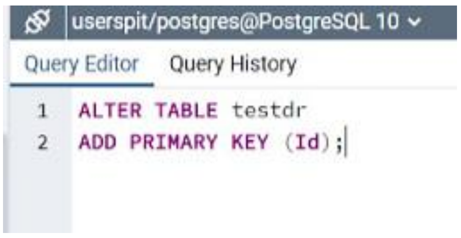
Adding a CHECK constraint:



```
ALTER TABLE testdr
ADD CHECK (Age > 0);
```

Figure 6.8 - CHECK restrictions

Adding the primary key PRIMARY KEY:

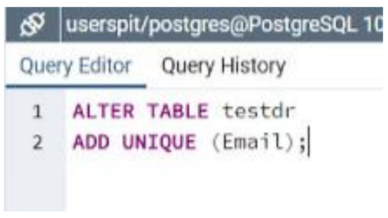


```
ALTER TABLE testdr
ADD PRIMARY KEY (Id);
```

Figure 6.9 - Adding the primary key PRIMARY KEY

In this case, it is assumed that the table already has an Id column that does not have a PRIMARY KEY constraint. And with the help of the above script, the PRIMARY KEY restriction is set.

8) Adding the UNIQUE constraint - define unique values for the email column:

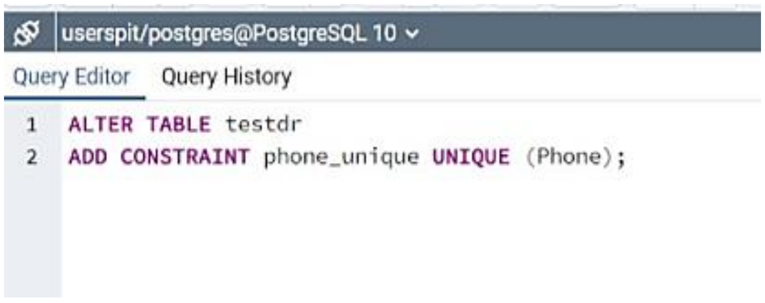


```sql
ALTER TABLE testdr
ADD UNIQUE (Email);
```

Figure 6.10 - Adding a UNIQUE constraint

When adding a constraint, each of them is given a specific name. For example, the CHECK constraint added above would be called customers_age_check. The names of the restrictions can be viewed in the table via pgAdmin.

9) We can also explicitly assign a constraint when adding a name using the CONSTRAINT operator.

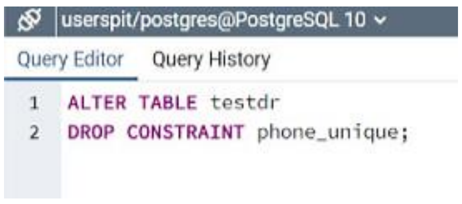


```sql
ALTER TABLE testdr
ADD CONSTRAINT phone_unique UNIQUE (Phone);
```

Figure 6.11 – Adding a name using the CONSTRAINT operator

10) In this case, the restriction will be called "iphone_unique".

To remove a constraint, you need to know its name, which is specified after the DROP CONSTRAINT expression. For example, let's remove the constraint added above:

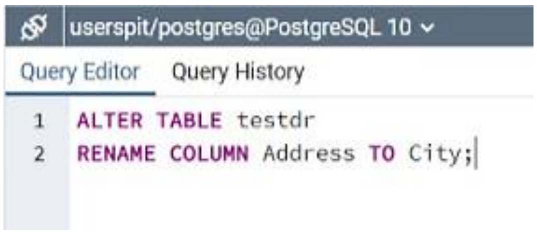


```sql
ALTER TABLE testdr
DROP CONSTRAINT phone_unique;
```

Figure 6.12 – DROP CONSTRAINT

11) Column and table renaming

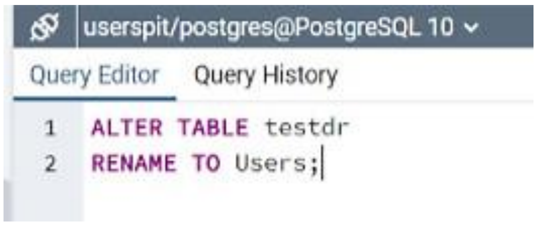
Rename the Address column to City:



```sql
ALTER TABLE testdr
RENAME COLUMN Address TO City;
```

Figure 6.13 - Renaming a column

Rename the testdr table to Users:



```sql
ALTER TABLE testdr
RENAME TO Users;
```

Figure 6.14 – Renaming the table