

Laboratory work

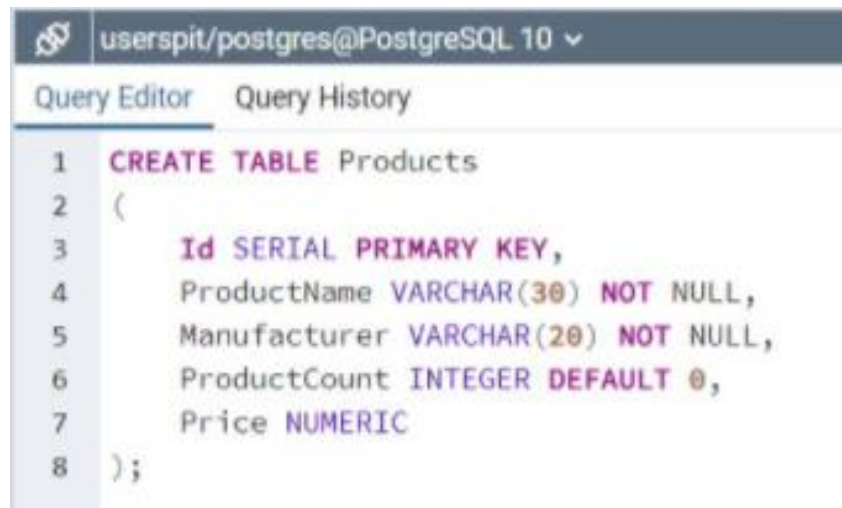
Topic: Data operations in PostgreSQL.

Purpose: Data operations.

Progress

1. Adding data. The Insert command:

1) Suppose we have the following table in the database:



```
userspit/postgres@PostgreSQL 10 v
Query Editor  Query History
1  CREATE TABLE Products
2  (
3      Id SERIAL PRIMARY KEY,
4      ProductName VARCHAR(30) NOT NULL,
5      Manufacturer VARCHAR(20) NOT NULL,
6      ProductCount INTEGER DEFAULT 0,
7      Price NUMERIC
8  );
```

Figure 1.1 - Products table

2) Add one line to it using the INSERT command:

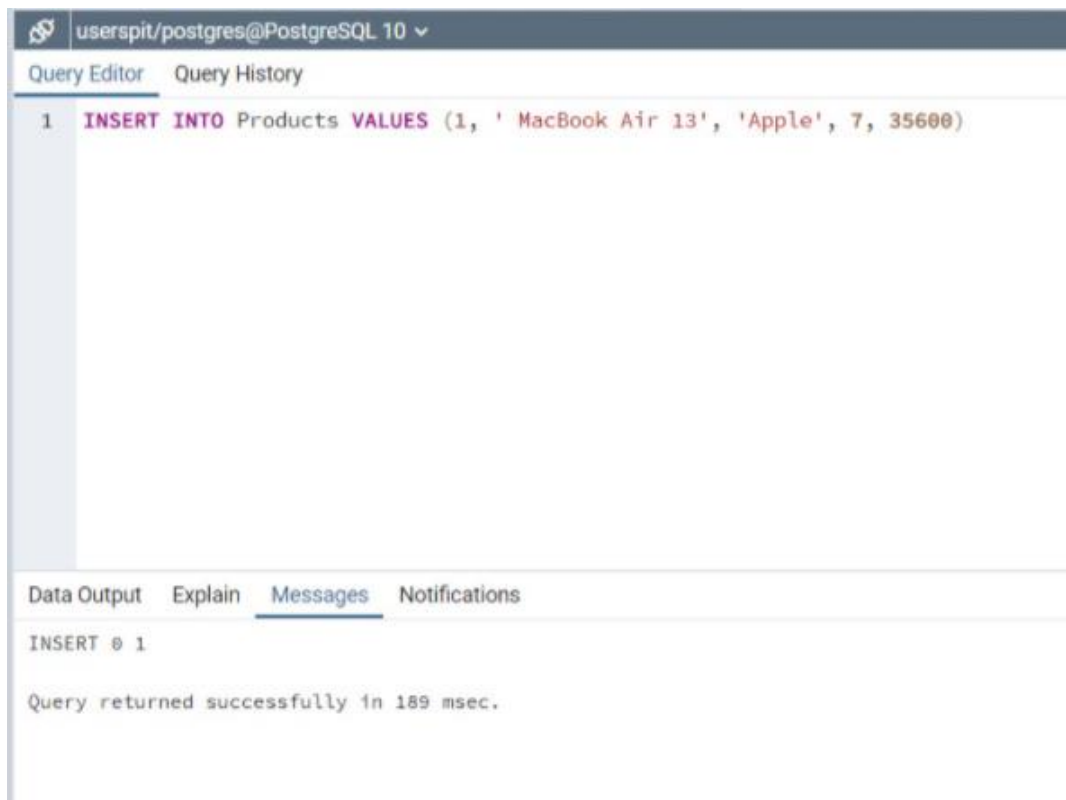


Figure 1.2 –INSERT commands

After successful execution in pgAdmin, the message "INSERT 0 1" should appear in the message field:

It should be noted that values for columns in parentheses after the VALUES keyword are passed in the order of their declaration. For example, in the CREATE TABLE statement above, you can see that the first column is Id, so 1 is passed to this column. The second column is called ProductName, so the second value - the string "MacBook Air 13" - will be passed to this column, and so on. That is, the values are transferred to the columns as follows:

- ID: 1
- Product Name: MacBook Air 13'
- Manufacturer: 'Apple'
- Products count: 7
- Price: 35600

Also, when entering values, you can specify the immediate columns to which values will be added:



Figure 1.3 - Columns

Here, the value is specified only for three columns. Moreover, now the values are transferred in the order of passing the columns:

ProductName: 'MacBook Air 13'

Manufacturer: 'Apple'

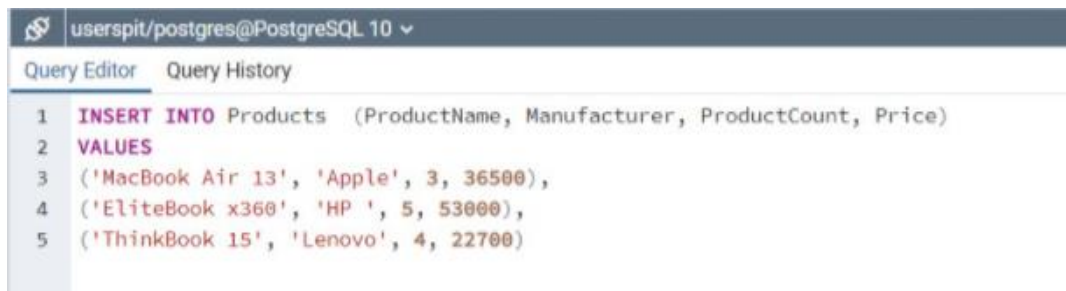
Price: 35600

For the Id column, the value will be automatically generated by the database as it represents the Serial type. That is, one will be added to the value from the last line.

For other columns, a default value will be added if the DEFAULT attribute is specified (for example, for the ProductCount column), a value of NULL. At the same time, unspecified columns (except for those of Serial type) must allow NULL values or have the DEFAULT attribute.

If specific columns are not specified, as in the first example, then we must pass values for all columns in the table.

We can also add several lines at once:



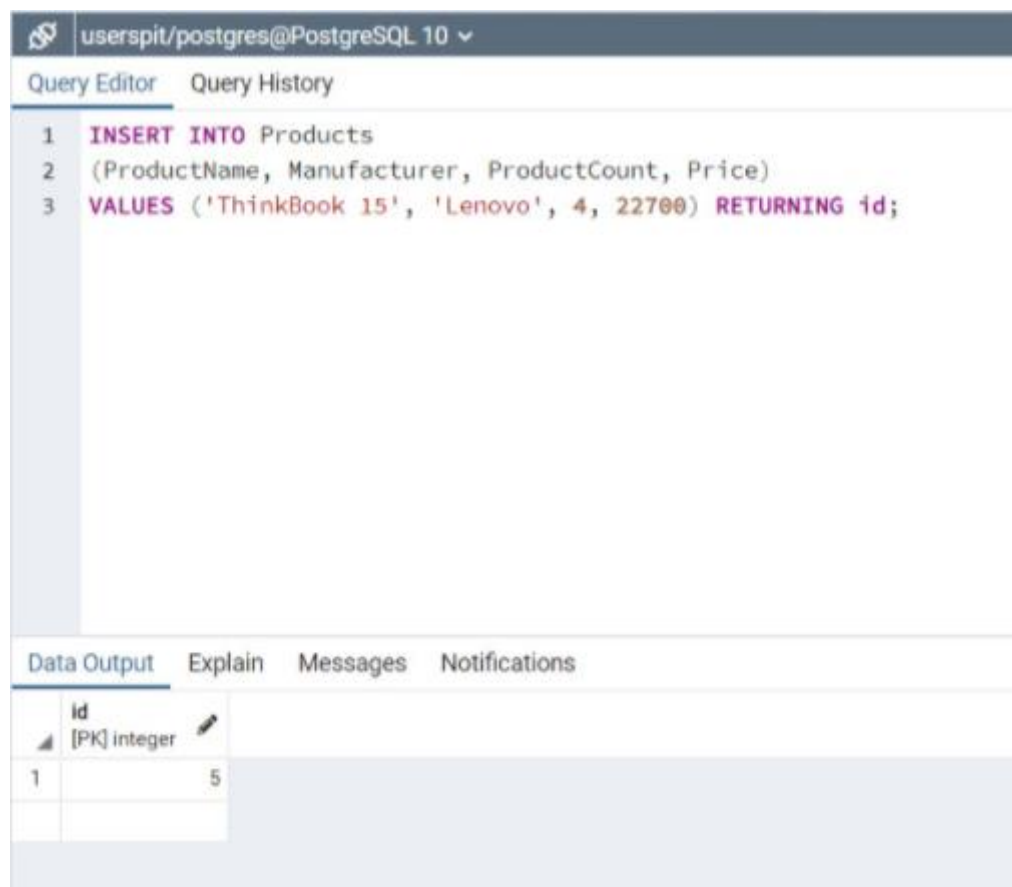
```
userspit/postgres@PostgreSQL 10
Query Editor  Query History
1  INSERT INTO Products (ProductName, Manufacturer, ProductCount, Price)
2  VALUES
3  ('MacBook Air 13', 'Apple', 3, 36500),
4  ('EliteBook x360', 'HP ', 5, 53000),
5  ('ThinkBook 15', 'Lenovo', 4, 22700)
```

Figure 1.4 - Add several lines at once

In this case, three rows will be added to the table.

3) Return values

If we add values for only part of the columns, then we may not know what values will be in other columns. For example, what value will the Id column receive in the product. Using the RETURNING statement, we can get this value:



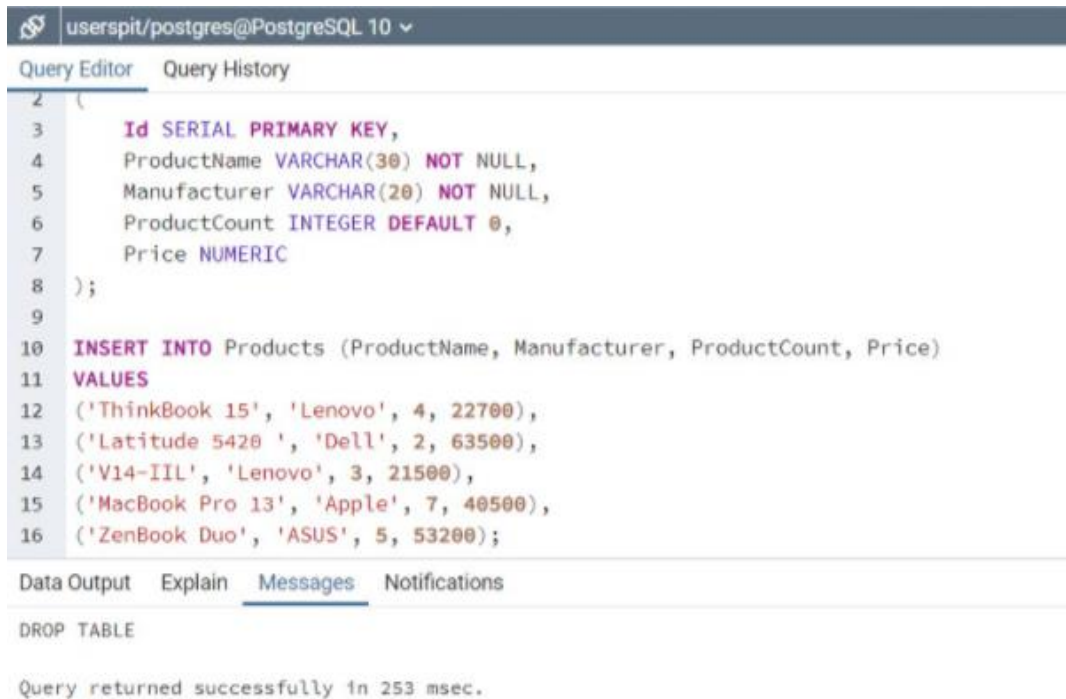
```
userspit/postgres@PostgreSQL 10
Query Editor  Query History
1  INSERT INTO Products
2  (ProductName, Manufacturer, ProductCount, Price)
3  VALUES ('ThinkBook 15', 'Lenovo', 4, 22700) RETURNING id;
```

Data Output		Explain	Messages	Notifications
	id [PK] integer			
1	5			

Figure 1.5 - Return values

2. Receiving data. The Select command

1) For example, suppose the Products table was previously created and some initial Data was added to it:



```
userspit/postgres@PostgreSQL 10
Query Editor  Query History
2  (
3      Id SERIAL PRIMARY KEY,
4      ProductName VARCHAR(30) NOT NULL,
5      Manufacturer VARCHAR(20) NOT NULL,
6      ProductCount INTEGER DEFAULT 0,
7      Price NUMERIC
8  );
9
10 INSERT INTO Products (ProductName, Manufacturer, ProductCount, Price)
11 VALUES
12 ('ThinkBook 15', 'Lenovo', 4, 22700),
13 ('Latitude 5420 ', 'Dell', 2, 63500),
14 ('V14-IIL', 'Lenovo', 3, 21500),
15 ('MacBook Pro 13', 'Apple', 7, 40500),
16 ('ZenBook Duo', 'ASUS', 5, 53200);

Data Output  Explain  Messages  Notifications
DROP TABLE

Query returned successfully in 253 msec.
```

Figure 2.1 - Products table

Let's get all objects from this table:

The screenshot shows a PostgreSQL Query Editor interface. At the top, the user is logged in as 'userspit/postgres@PostgreSQL 10'. The 'Query Editor' tab is active, displaying the SQL query: `1 SELECT * FROM Products;`. Below the query editor, the 'Data Output' tab is selected, showing the results of the query. The results are displayed in a table with 6 columns: 'id' (integer), 'productname' (character varying (30)), 'manufacturer' (character varying (20)), 'productcount' (integer), 'price' (numeric), and an empty column. The table contains 5 rows of data.

	id [PK] integer	productname character varying (30)	manufacturer character varying (20)	productcount integer	price numeric	
1	1	ThinkBook 15	Lenovo	4	22700	
2	2	Latitude 5420	Dell	2	63500	
3	3	V14-IIL	Lenovo	3	21500	
4	4	MacBook Pro 13	Apple	7	40500	
5	5	ZenBook Duo	ASUS	5	53200	

Figure 2.2 – Select command

The asterisk * indicates that we need to retrieve all columns.

However, using the asterisk * is not considered good practice, as not all columns are generally required. And a more optimal approach is to specify all the necessary columns after the word SELECT. An exception is the case when it is necessary to obtain data on absolutely all columns of the table. Also, the use of the * symbol can be preferred in situations where the names of the columns are not known exactly.

If we need to get data not for all, but for some specific columns, then all these column specifications are listed with a comma after SELECT:

userspit/postgres@PostgreSQL 10

Query Editor

Query History

1

SELECT ProductName, Price FROM Products;

Data Output

Explain

Messages

Notifications

	productname character varying (30)	price numeric
1	ThinkBook 15	22700
2	Latitude 5420	63500
3	V14-IIL	21500
4	MacBook Pro 13	40500
5	ZenBook Duo	53200

Figure 2.3 – Specific Select columns

A column specification does not necessarily have to represent its name. It can be any expression, for example, the result of an arithmetic operation. Yes, let's execute the following request:

userspit/postgres@PostgreSQL 10

Query Editor

Query History

1

2

3

4

5

SELECT

 ProductCount

AS

 Title,
Manufacturer,
Price * ProductCount

AS

 TotalSum

FROM

 Products;

Data Output

Explain

Messages

Notifications

	<div>title</div> <div>integer</div>	<div>manufacturer</div> <div>character varying (20)</div>	<div>totalsum</div> <div>numeric</div>
1	4	Lenovo	90800
2	2	Dell	127000
3	3	Lenovo	64500
4	7	Apple	283500
5	5	ASUS	266000

Figure 2.5 – AS operator

3.Filtration. WHERE

The WHERE statement is used to filter data, followed by the condition on the basis of which filtering is performed:

WHERE condition

If the condition is true, then the string is included in the resulting sample. How comparison operations can be used. These operations compare two expressions. The following comparison operations can be applied in PostgreSQL:

- = : comparison for equality
- < > : comparison for inequality
- < : less than
- : more than
- !< : not less than

- != : not more than
 - <= : less than or equal to
 - >= : greater than or equal to
- 1) For example, let's find all products manufactured by Apple:

```
1 SELECT * FROM Products
2 WHERE Manufacturer = 'Apple';
3
```

id	productname	manufacturer	productcount	price
1	MacBook Pro 13	Apple	7	40500
2	MacBook Pro 15	Apple	7	40500
3	MacBook Pro 16	Apple	7	40500
4	MacBook Pro 17	Apple	7	40500

Figure 3.1 - Apple products

It is worth noting that in this case the case of characters is of great importance, for example, the string "Apple" is not equivalent to the string "APPLE" or "apple".

Another example is to find all products whose price is less than 37,000:

userspit/postgres@PostgreSQL 10 ▾

Query Editor Query History

1 **SELECT** * **FROM** Products

2 **WHERE** Price < 37000;

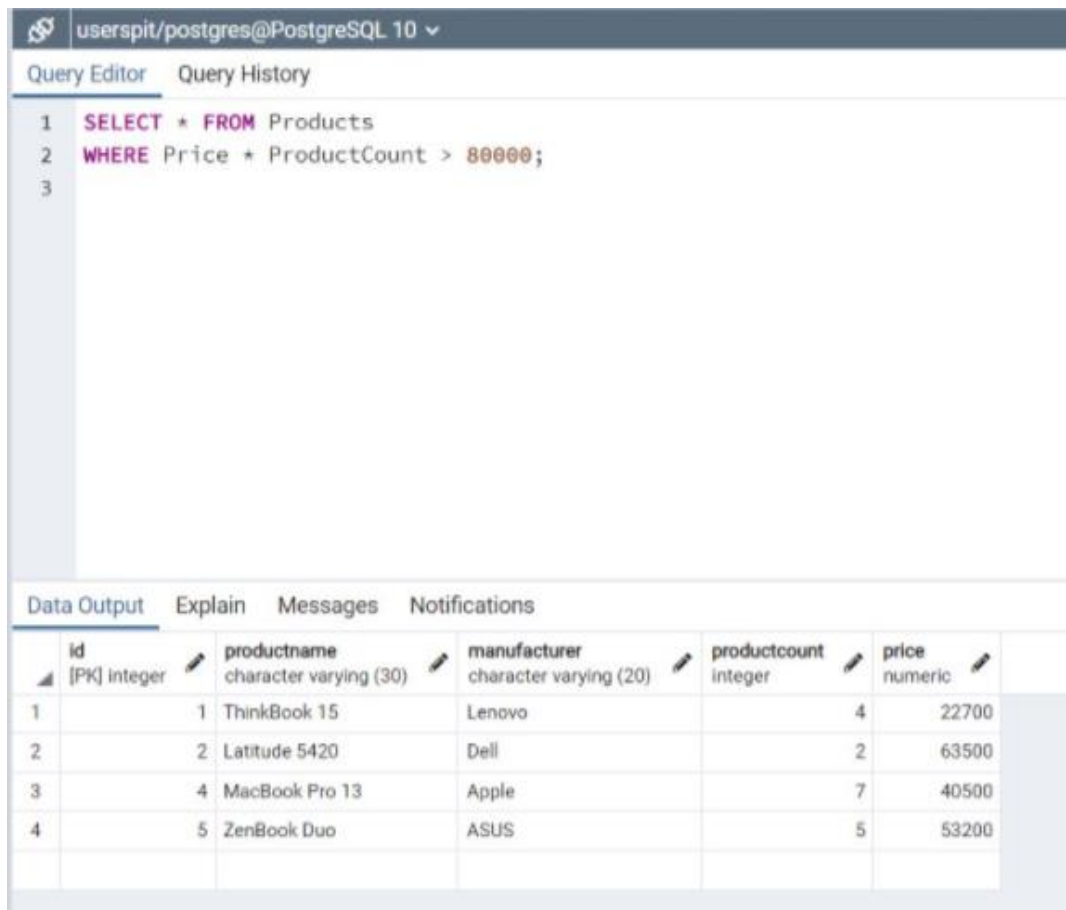
3

Data Output Explain Messages Notifications

	<div>id</div> <div>[PK] integer</div>	<div>productname</div> <div>character varying (30)</div>	<div>manufacturer</div> <div>character varying (20)</div>	<div>productcount</div> <div>integer</div>	<div>price</div> <div>numeric</div>	
1	1	ThinkBook 15	Lenovo	4	22700	
2	3	V14-IIL	Lenovo	3	21500	

Figure 3.2 – Goods whose price is less than 37,000

More complex expressions can be used as a condition. For example, let's find all products with a total value of more than 80,000:



The screenshot shows a PostgreSQL Query Editor interface. At the top, the user is logged in as 'userspit/postgres@PostgreSQL 10'. The 'Query Editor' tab is active, displaying a SQL query:

```

1 SELECT * FROM Products
2 WHERE Price * ProductCount > 80000;
3

```

Below the query editor, the 'Data Output' tab is active, showing the results of the query. The results are displayed in a table with the following columns: id (PK) integer, productname character varying (30), manufacturer character varying (20), productcount integer, and price numeric. The table contains four rows of data:

	id (PK) integer	productname character varying (30)	manufacturer character varying (20)	productcount integer	price numeric
1	1	ThinkBook 15	Lenovo	4	22700
2	2	Latitude 5420	Dell	2	63500
3	4	MacBook Pro 13	Apple	7	40500
4	5	ZenBook Duo	ASUS	5	53200

Figure 3.2 – Find all products with a total value of more than 80,000

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) Logical operator

To combine several conditions into one, you can use logical operators in PostgreSQL:

- **AND:** logical and operation. it combines two expressions:
expression1 AND expression2

Only if both of these expressions are true at the same time, then the general condition of the AND operator will also be true. That is, if both the first condition is true and the second.

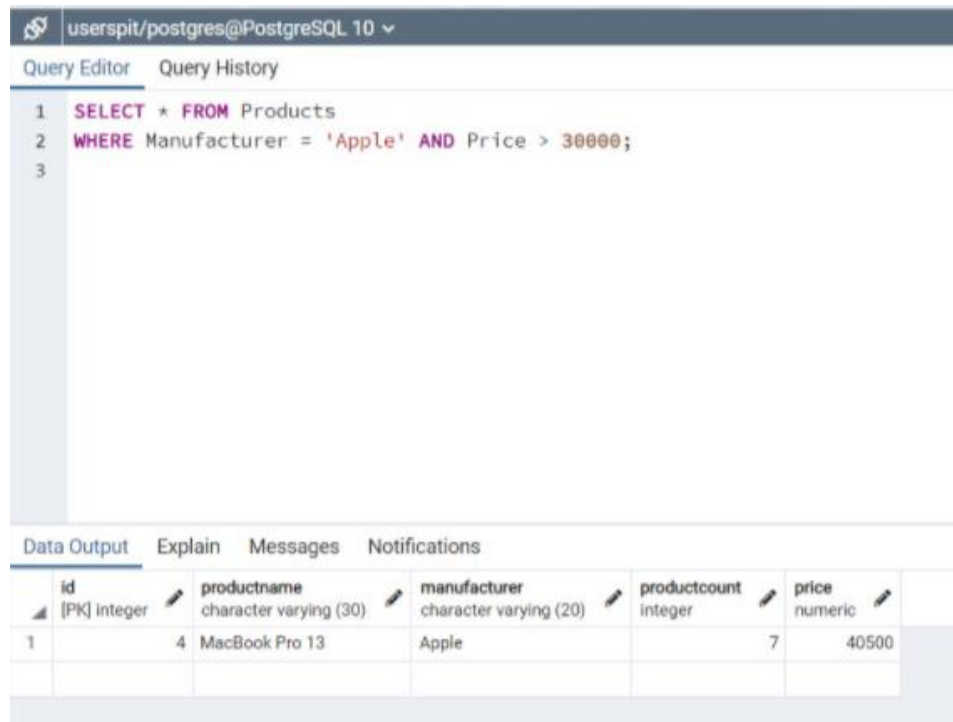
- **OR:** logical or operation. It also combines two expressions:
expression1 OR expression2

If at least one of these expressions is true, then the general condition of the OR operator will also be true. That is, if either the first condition is true or the second.

- NOT: logical negation operation. If the expression in this operation is false, then the general condition is true.

NOT expression

For example, let's select all products that are manufactured by Apple and at the same time the price is more than 30,000:



The screenshot shows a PostgreSQL query editor interface. The top bar indicates the user is 'userspit/postgres@PostgreSQL 10'. Below the bar are tabs for 'Query Editor' and 'Query History'. The 'Query Editor' tab is active, displaying a SQL query:

```
1 SELECT * FROM Products
2 WHERE Manufacturer = 'Apple' AND Price > 30000;
3
```

Below the query editor are tabs for 'Data Output', 'Explain', 'Messages', and 'Notifications'. The 'Data Output' tab is active, showing a table with the following columns: 'id' (integer, PK), 'productname' (character varying (30)), 'manufacturer' (character varying (20)), 'productcount' (integer), and 'price' (numeric). The table contains one row of data:

id	productname	manufacturer	productcount	price
1	MacBook Pro 13	Apple	7	40500

Figure 3.3 – Let's select all the products in which the manufacturer is Apple and at the same time the price is more than 30,000

Now let's change the operator to OR. That is, we will select all products that either have the Apple manufacturer or the price is more than 30,000:

userspit/postgres@PostgreSQL 10

Query Editor

Query History

1

SELECT * FROM Products

2

WHERE Manufacturer = 'Apple' OR Price > 30000;

3

Data Output

Explain

Messages

Notifications

	id [PK] integer	productname character varying (30)	manufacturer character varying (20)	productcount integer	price numeric
1	2	Latitude 5420	Dell	2	63500
2	4	MacBook Pro 13	Apple	7	40500
3	5	ZenBook Duo	ASUS	5	53200

Figure 3.4 - Products with either the manufacturer Apple or a price greater than 30,000

Using the NOT operator, we will select all products whose manufacturer is not Apple:

userspit/postgres@PostgreSQL 10

Query Editor

Query History

1

SELECT * FROM Products

2

WHERE NOT Manufacturer = 'Apple';

3

Data Output

Explain

Messages

Notifications

	id [PK] integer	productname character varying (30)	manufacturer character varying (20)	productcount integer	price numeric	
1	1	ThinkBook 15	Lenovo	4	22700	
2	2	Latitude 5420	Dell	2	63500	
3	3	V14-IIL	Lenovo	3	21500	
4	5	ZenBook Duo	ASUS	5	53200	

Figure 3.5 - Let's select all products whose manufacturer is not Apple

But in most cases, it is quite possible to do without the NOT operator. So, in the previous example, we can rewrite as follows:

userspit/postgres@PostgreSQL 10

Query Editor

Query History

1

SELECT * FROM Products

2

WHERE Manufacturer <> 'Apple';

3

Data Output

Explain

Messages

Notifications

	id [PK] integer	productname character varying (30)	manufacturer character varying (20)	productcount integer	price numeric
1	1	ThinkBook 15	Lenovo	4	22700
2	2	Latitude 5420	Dell	2	63500
3	3	V14-IIL	Lenovo	3	21500
4	5	ZenBook Duo	ASUS	5	53200

Figure 3.6 - NOT (<>) Apple

You can also use several statements in one SELECT command:

userspit/postgres@PostgreSQL 10

Query Editor

Query History

1

SELECT * FROM Products

2

WHERE Manufacturer = 'Apple' OR Price > 30000 AND ProductCount > 2;

3

Data Output

Explain

Messages

Notifications

	id [PK] integer	productname character varying (30)	manufacturer character varying (20)	productcount integer	price numeric
1	4	MacBook Pro 13	Apple	7	40500
2	5	ZenBook Duo	ASUS	5	53200

Figure 3.7 – Multiple SELECT statements

Since the AND operator has a higher priority, the subexpression Price > 30000 AND ProductCount > 2 will be executed first, and only then the OR operator. That is, the products that are in stock for more than 2 and whose price is more than 30,000 at the same time, or those products that are manufactured by Apple, are selected here.

Using parentheses, we can also override the order of operations:

userspit/postgres@PostgreSQL 10

Query Editor

Query History

1

SELECT * FROM Products

2

WHERE (Manufacturer = 'Apple' OR Price > 30000) AND ProductCount > 2;

3

Data Output

Explain

Messages

Notifications

	id [PK] integer	productname character varying (30)	manufacturer character varying (20)	productcount integer	price numeric
1	4	MacBook Pro 13	Apple	7	40500
2	5	ZenBook Duo	ASUS	5	53200

Figure 3.8 – Several statements of the SELECT parenthesis

3) IS NULL

A number of columns can accept NULL values. This value is not equivalent to the empty string". NULL represents the complete absence of any value. And the IS NULL operator is used to check for such a value.

For example, let's select all products that do not have the Product Count field set:

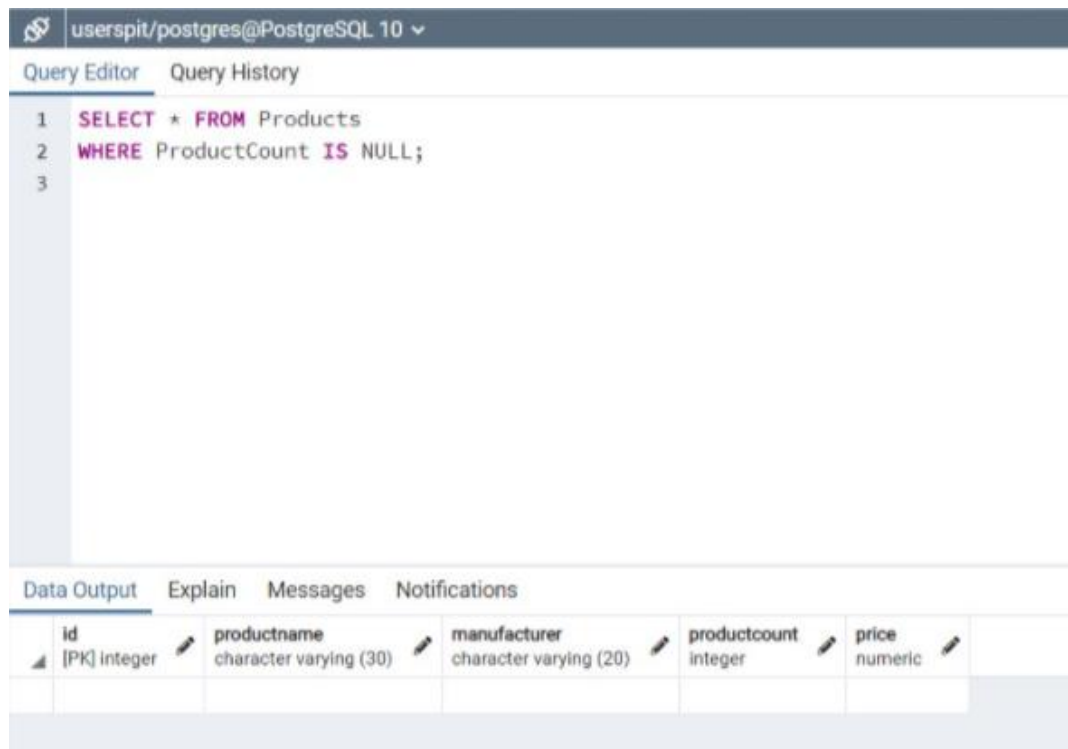
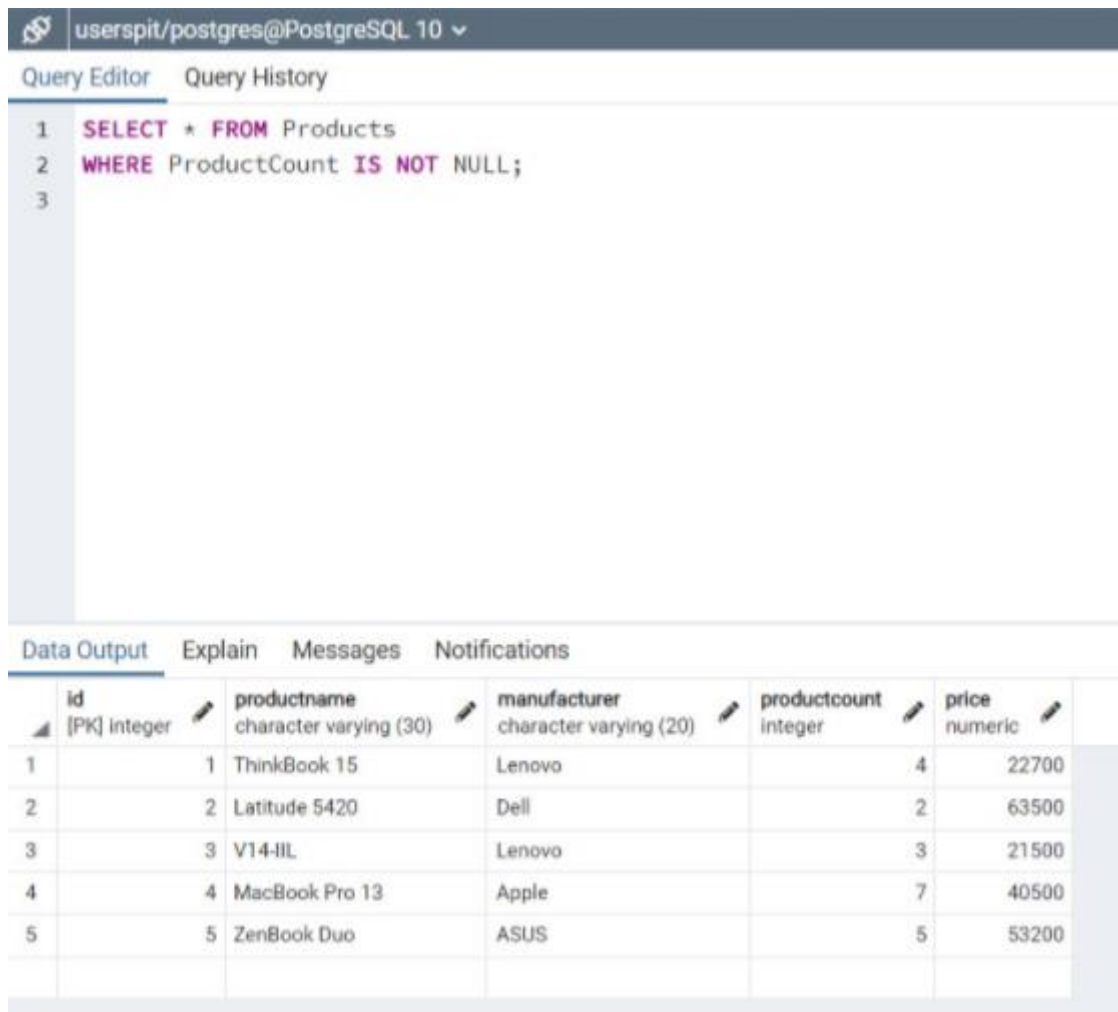


Figure 3.9 - IS NULL operator

If, on the contrary, you need to get rows in which the Product Count field is not NULL, then you can use the NOT operator:



The screenshot shows a PostgreSQL Query Editor interface. The top bar indicates the user is 'userspit/postgres@PostgreSQL 10'. Below the bar, there are two tabs: 'Query Editor' and 'Query History'. The 'Query Editor' tab is active, displaying a SQL query:

```
1 SELECT * FROM Products
2 WHERE ProductCount IS NOT NULL;
3
```

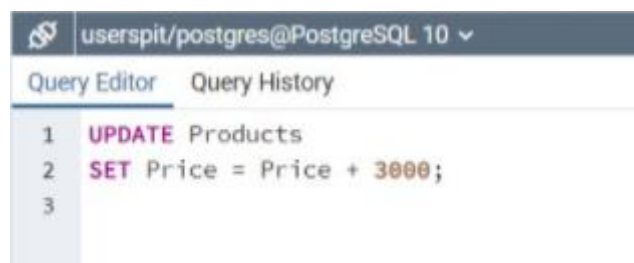
Below the query editor, there are four tabs: 'Data Output', 'Explain', 'Messages', and 'Notifications'. The 'Data Output' tab is active, showing a table of results. The table has five columns: 'id', 'productname', 'manufacturer', 'productcount', and 'price'. The data is as follows:

id	productname	manufacturer	productcount	price
1	ThinkBook 15	Lenovo	4	22700
2	Latitude 5420	Dell	2	63500
3	V14-IIL	Lenovo	3	21500
4	MacBook Pro 13	Apple	7	40500
5	ZenBook Duo	ASUS	5	53200

Figure 3.10 - IS NOT NULL operator

4. Updating data. The UPDATE command

The UPDATE command is used to update data in the PostgreSQL database. For example, let's increase the price of all products by 3000:



The screenshot shows a PostgreSQL Query Editor interface. The top bar indicates the user is 'userspit/postgres@PostgreSQL 10'. Below the bar, there are two tabs: 'Query Editor' and 'Query History'. The 'Query Editor' tab is active, displaying a SQL query:

```
1 UPDATE Products
2 SET Price = Price + 3000;
3
```

Figure 4.1 – UPDATE

userspit/postgres@PostgreSQL 10

Query Editor

Query History

1

UPDATE Products

2

SET Price = Price + 3000;

3

4

SELECT * FROM Products;

Data Output

Explain

Messages

Notifications

	id [PK] integer	productname character varying (30)	manufacturer character varying (20)	productcount integer	price numeric	
1	1	ThinkBook 15	Lenovo	4	28700	
2	2	Latitude 5420	Dell	2	69500	
3	3	V14-IIL	Lenovo	3	27500	
4	4	MacBook Pro 13	Apple	7	46500	
5	5	ZenBook Duo	ASUS	5	59200	

Figure 4.2 – UPDATE + SELECT * FROM

In this case, the update applies to all rows. Using the WHERE expression, you can use a condition to specify updated rows - if the row meets the condition, then it will be updated. For example, let's change the name of the manufacturer from "Lenovo" to "Lenovo Inc.":

userspit/postgres@PostgreSQL 10 ▾

Query Editor Query History

```

1 UPDATE Products
2 SET Manufacturer = 'Lenovo',
3     ProductCount = ProductCount + 3
4 WHERE Manufacturer = 'Lenovo Ins';
5
6
7 SELECT * FROM Products;

```

Data Output Explain Messages Notifications

	id [PK] integer	productname character varying (30)	manufacturer character varying (20)	productcount integer	price numeric
1	2	Latitude 5420	Dell	2	69500
2	4	MacBook Pro 13	Apple	7	46500
3	5	ZenBook Duo	ASUS	5	59200
4	1	ThinkBook 15	Lenovo	7	28700
5	3	V14-IIL	Lenovo	6	27500

Figure 4.4 –We change the name of the manufacturer in several columns

5.Deleting data. The DELETE command

1)To delete data in PostgreSQL, the DELETE command is used.

For example, let's delete the lines in which the manufacturer is Dell:

userspit/postgres@PostgreSQL 10

Query Editor Query History

1 **DELETE FROM** Products

2 **WHERE** Manufacturer='Dell';

3

4

5 **SELECT * FROM** Products;

Data Output Explain Messages Notifications

	id [PK] Integer	productname character varying (30)	manufacturer character varying (20)	productcount Integer	price numeric
1	4	MacBook Pro 13	Apple	7	46500
2	5	ZenBook Duo	ASUS	5	59200
3	1	ThinkBook 15	Lenovo	7	28700
4	3	V14-IIL	Lenovo	6	27500

Figure 5.1 –We change the name of the manufacturer in several columns

Or delete all products manufactured by Lenovo and priced below 28,000:

userspit/postgres@PostgreSQL 10

Query Editor

Query History

1

DELETE FROM Products

2

WHERE Manufacturer='Lenovo' AND Price < 28000;

3

4

5

SELECT * FROM Products;

Data Output

Explain

Messages

Notifications

	id [PK] integer	productname character varying (30)	manufacturer character varying (20)	productcount integer	price numeric
1	4	MacBook Pro 13	Apple	7	46500
2	5	ZenBook Duo	ASUS	5	59200
3	1	ThinkBook 15	Lenovo	7	28700

Figure 5.2 –Removal of goods

If it is necessary to completely delete all rows regardless of the condition, then the condition can be omitted:

userspit/postgres@PostgreSQL 10

Query Editor Query History

1

2

3

4

DELETE FROM Products;

SELECT * FROM Products;

Data Output Explain Messages Notifications

id	productname	manufacturer	productcount	price
[PK] integer	character varying (30)	character varying (20)	integer	numeric

Figure 5.3 –Let's delete all lines