Lecture 4

PostgreSQL DQL

PostgreSQL DQL (SELECT statement)

The SELECT statement is one of the most complex statements in PostgreSQL. It has many clauses that you can use to form a flexible query.

SELECT statement that retrieves data from a single table. The following illustrates the syntax of the SELECT statement:

```
SELECT
    select_list
FROM
    table_name;
```

PostgreSQL SELECT examples

1) Using PostgreSQL SELECT statement to query data from one column example:

```
SELECT first_name FROM customer;
```

2) Using PostgreSQL SELECT statement to query data from multiple columns example:

```
SELECT
        first_name,
        last_name,
        email
FROM
        customer;
```

3) Using PostgreSQL SELECT statement to query data from all columns of a table example:

```
SELECT * FROM customer;
```

PostgreSQL SELECT examples

4) Using PostgreSQL SELECT statement with expressions example:

```
SELECT
        first_name || ‘ ’ || last_name,
        email
FROM
        customer;
```

5) Using PostgreSQL SELECT statement with expressions example:

```
SELECT 5 * 3;

SELECT now();
```

PostgreSQL Column Alias

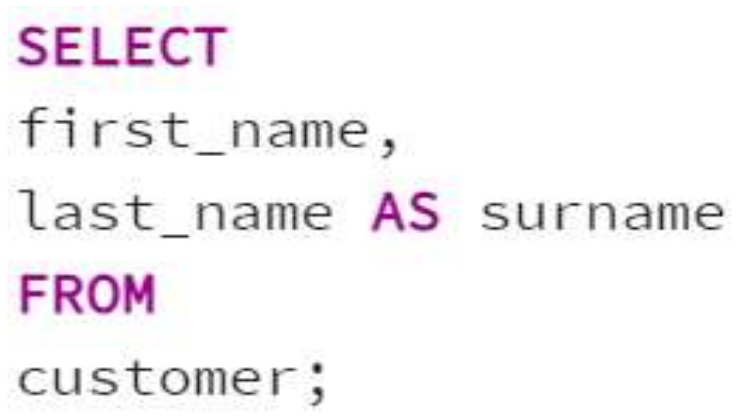
A **column alias** allows you to assign a column or an expression in the select list of a SELECT statement a temporary name. The column alias exists temporarily during the execution of the query.

The following illustrates the syntax of using a column alias:

```
SELECT  column_name AS alias_name
FROM     table_name;
```

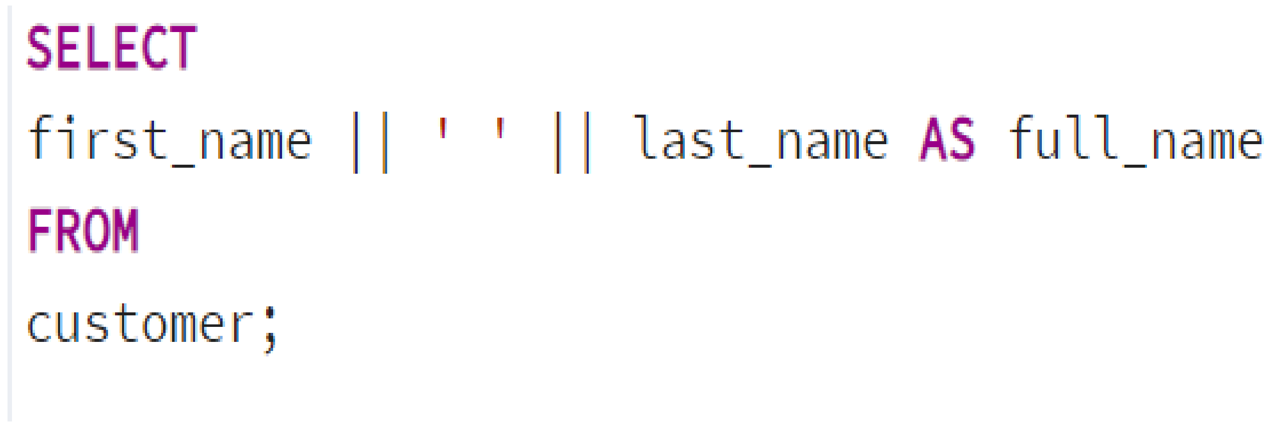
PostgreSQL column alias examples

1) Assigning a column alias to a column example:



```
SELECT
first_name,
last_name AS surname
FROM
customer;
```

2) Assigning a column alias to an expression example:



```
SELECT
first_name || ' ' || last_name AS full_name
FROM
customer;
```

PostgreSQL Table Aliases

Table aliases temporarily assign tables new names during the execution of a query.

The following illustrates the syntax of a table alias:

```
table_name AS alias_name;
```

PostgreSQL SELECT DISTINCT

The **DISTINCT** clause is used in the SELECT statement to remove duplicate rows from a result set.

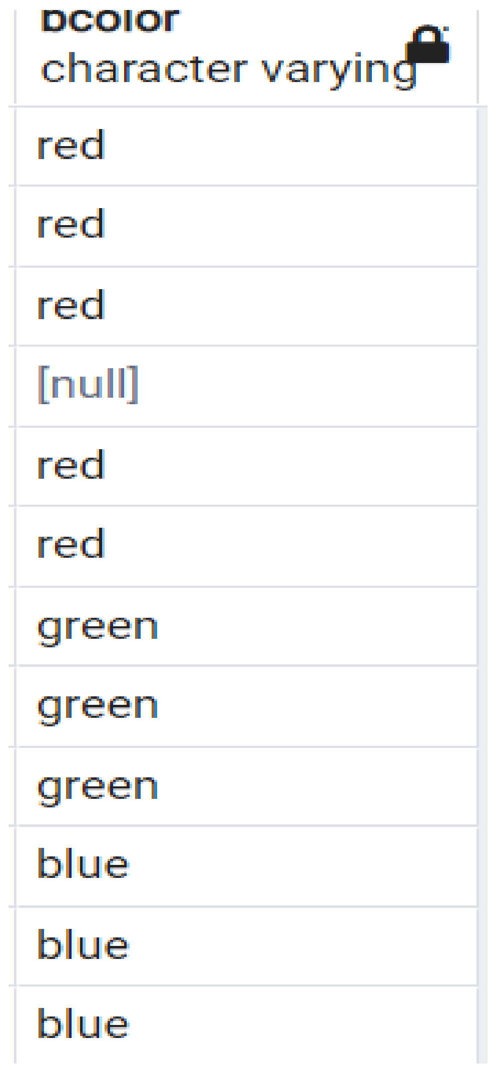
The following illustrates the syntax of the   clause:

```
SELECT
       DISTINCT column1
FROM
       table_name;
```

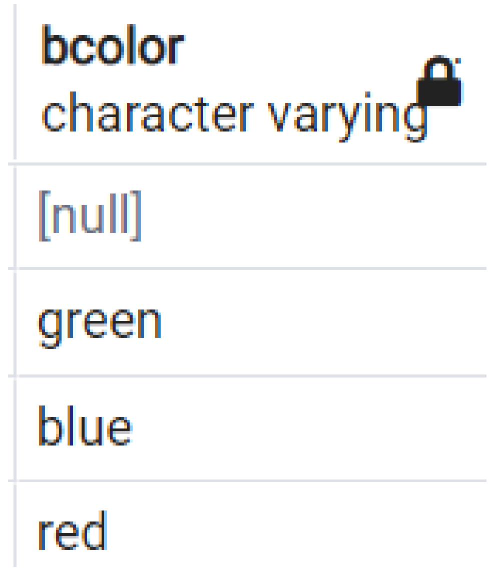
Example

```sql
select bcolor from distinct_demo
```

```sql
select distinct bcolor from distinct_demo
```



bcolor character varying
red
red
red
[null]
red
red
green
green
green
blue
blue
blue



bcolor character varying
[null]
green
blue
red

Basic PostgreSQL operators

PostgreSQL operators allow you to perform a wide variety of operations on data, making your queries more powerful and flexible. Understanding how to use these operators effectively can greatly enhance your ability to manipulate and analyze data.

Basic PostgreSQL operators

1. Arithmetic Operators
2. Comparison Operators
3. Logical Operators
4. Pattern Matching Operators
5. NULL-Related Operators
6. Other Operators

Arithmetic Operators

Used to perform mathematical operations on numeric data.
+Addition
-Substruction
*Multiplication
/Division
%Modulo

```
SELECT 5 + 4;
-- Result: 9
```

```
SELECT 5 - 3;
-- Result: 2
```

```
SELECT 5 * 3;
-- Result: 15
```

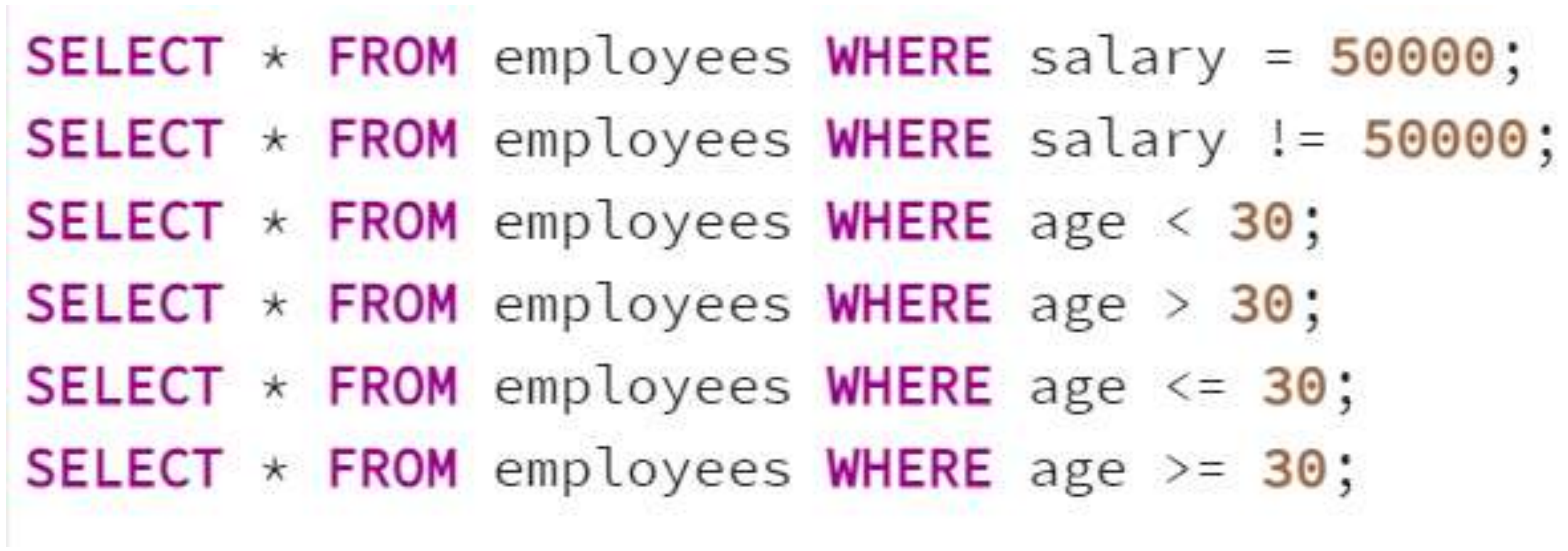
```
SELECT 10 / 2;
-- Result: 5
```

```
SELECT 10 % 3;
-- Result: 1
```

Comparison Operators

Used to compare two values and return a boolean result (TRUE, FALSE, or NULL).

- , < , >= , <= , === , and !==

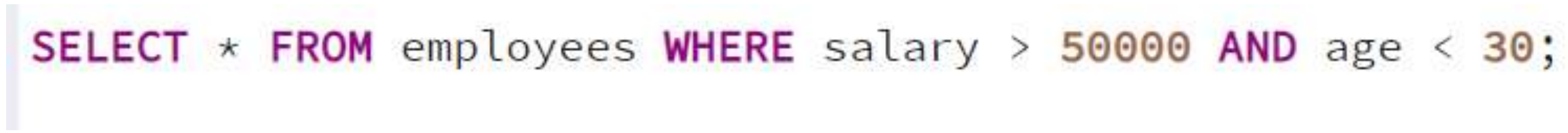


```
SELECT * FROM employees WHERE salary = 50000;
SELECT * FROM employees WHERE salary != 50000;
SELECT * FROM employees WHERE age < 30;
SELECT * FROM employees WHERE age > 30;
SELECT * FROM employees WHERE age <= 30;
SELECT * FROM employees WHERE age >= 30;
```

Logical Operators

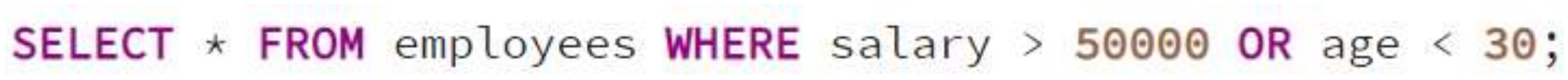
Used to combine multiple conditions in a WHERE clause.

AND: Returns TRUE if all conditions are true



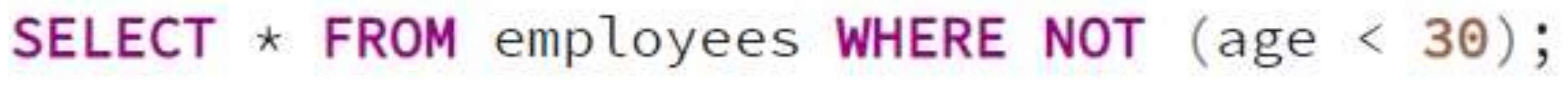
```sql
SELECT * FROM employees WHERE salary > 50000 AND age < 30;
```

OR: Returns TRUE if at least one condition is true.



```sql
SELECT * FROM employees WHERE salary > 50000 OR age < 30;
```

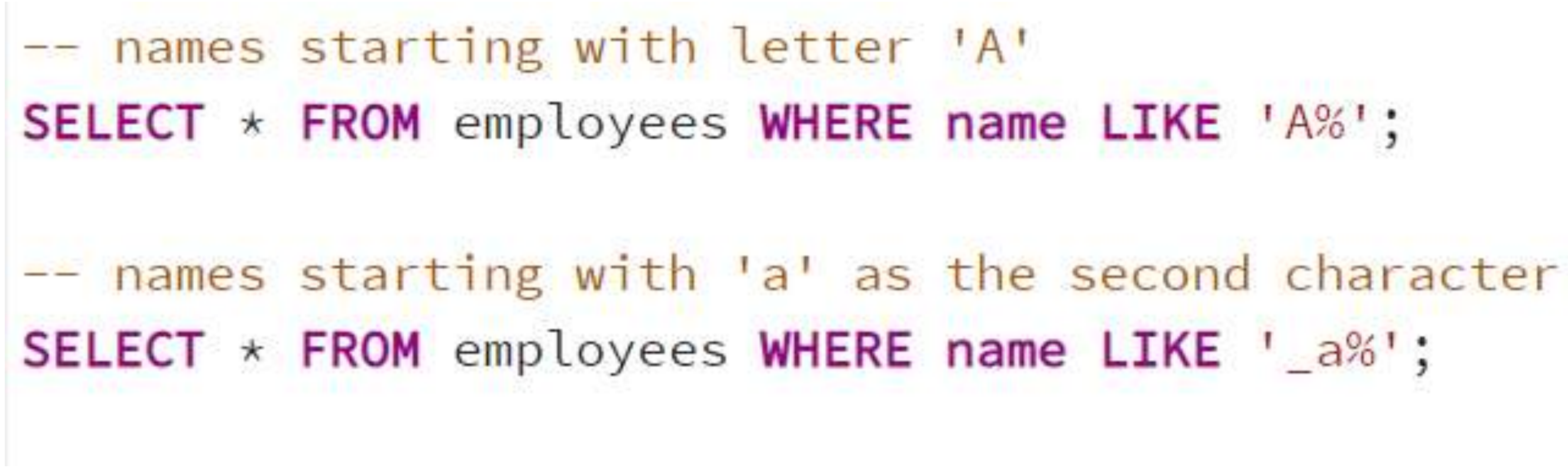
NOT: Inverts the result of a condition.



```sql
SELECT * FROM employees WHERE NOT (age < 30);
```

Pattern Matching Operators

Used to search for patterns within text.
LIKE: Searches for a specified pattern in a column.
%: Represents zero or more characters.
_: Represents a single character.

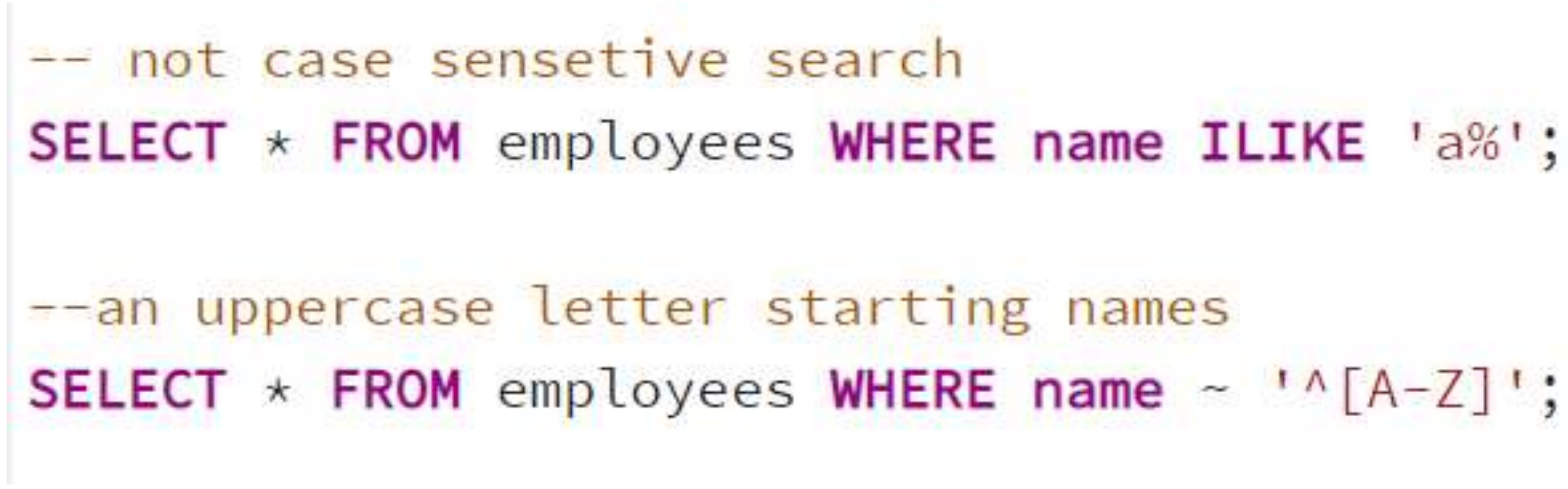


```
-- names starting with letter 'A'
SELECT * FROM employees WHERE name LIKE 'A%';

-- names starting with 'a' as the second character
SELECT * FROM employees WHERE name LIKE '_a%';
```

Pattern Matching Operators

ILIKE: Case-insensitive version of LIKE.
~ (Matches POSIX regular expression)

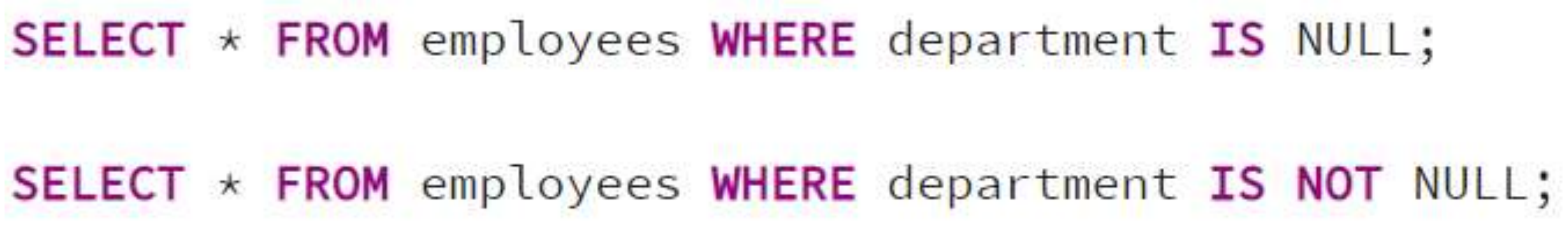


```
-- not case sensetive search
SELECT * FROM employees WHERE name ILIKE 'a%';

--an uppercase letter starting names
SELECT * FROM employees WHERE name ~ '^[A-Z]';
```

NULL-Related Operators

Used to check for NULL values.
IS NULL: Checks if a value is NULL.
IS NOT NULL: Checks if a value is not NULL.



```
SELECT * FROM employees WHERE department IS NULL;

SELECT * FROM employees WHERE department IS NOT NULL;
```

Other Operators

Concatenation (||): Concatenates two strings.

```
-- Result: 'Databases PostgreSQL'
SELECT 'Databases' || ' ' || 'PostgreSQL';
```

BETWEEN: Checks if a value is within a specified range (inclusive).

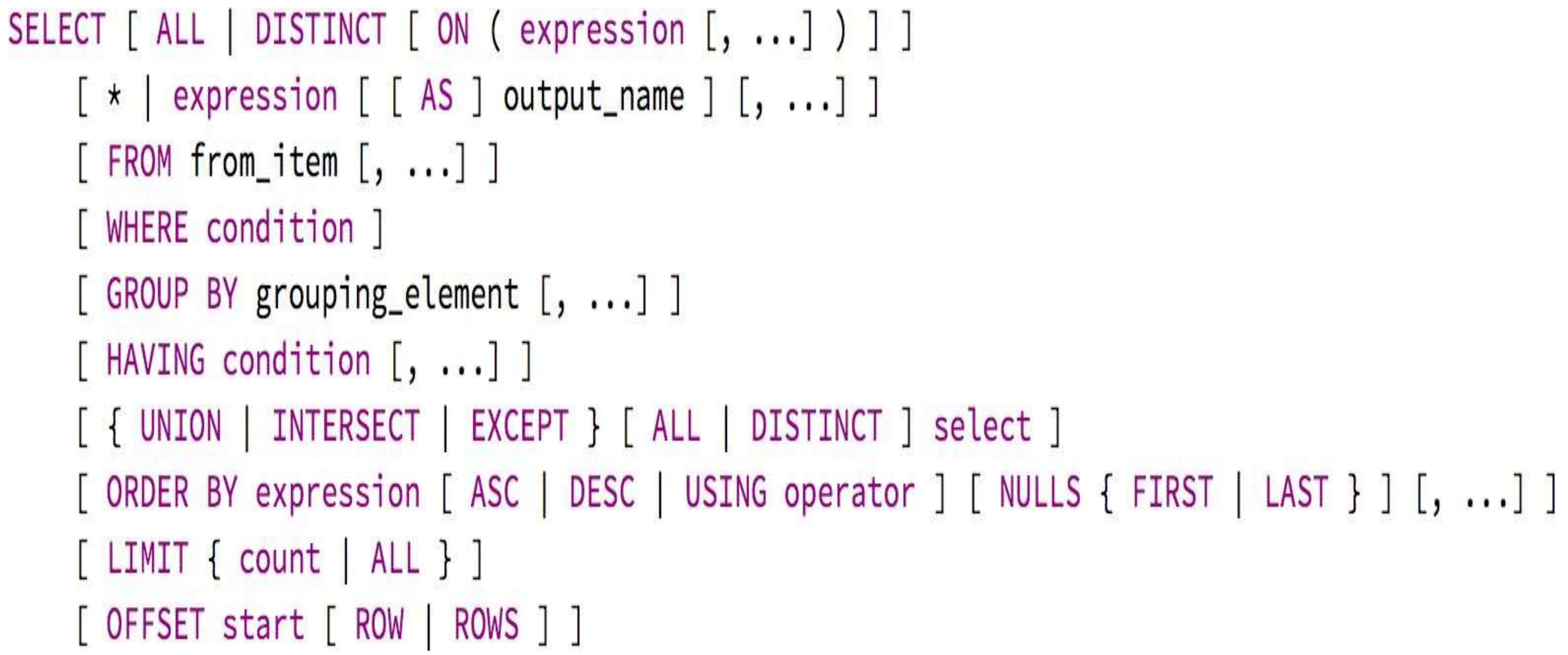
```
SELECT * FROM employees WHERE age BETWEEN 18 AND 55;
```

IN: Checks if a value matches any value in a list.

```
SELECT * FROM employees WHERE department IN ('IT dep', 'Social dep', 'Econimics dep');
```

Queries

SELECT syntax



```
SELECT [ ALL | DISTINCT [ ON ( expression [, ...] ) ] ]
    [ * | expression [ [ AS ] output_name ] [, ...] ]
    [ FROM from_item [, ...] ]
    [ WHERE condition ]
    [ GROUP BY grouping_element [, ...] ]
    [ HAVING condition [, ...] ]
    [ { UNION | INTERSECT | EXCEPT } [ ALL | DISTINCT ] select ]
    [ ORDER BY expression [ ASC | DESC | USING operator ] [ NULLS { FIRST | LAST } ] [, ...] ]
    [ LIMIT { count | ALL } ]
    [ OFFSET start [ ROW | ROWS ] ]
```

SELECT List

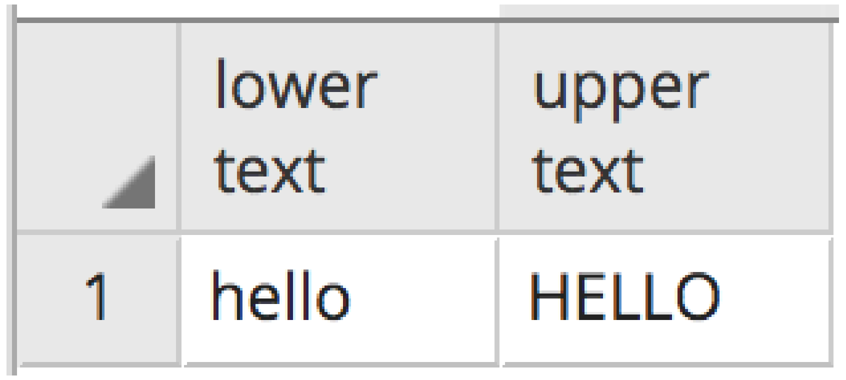
- If you do not specify a column name, a name is chosen automatically by PostgreSQL.
- If the column's expression is a simple column reference then the chosen name is the same as that column's name.

SELECT List

- If you do not specify a column name, a name is chosen automatically by PostgreSQL.
- If the column's expression is a simple column reference then the chosen name is the same as that column's name.
- In more complex cases a function or type name may be used, or the system may fall back on a generated name such as ?column?.

SELECT List

```sql
SELECT lower('HELLO'), upper('hello')
```



	lower text	upper text
1	hello	HELLO

WHERE clause

- The optional `WHERE` clause has the general form

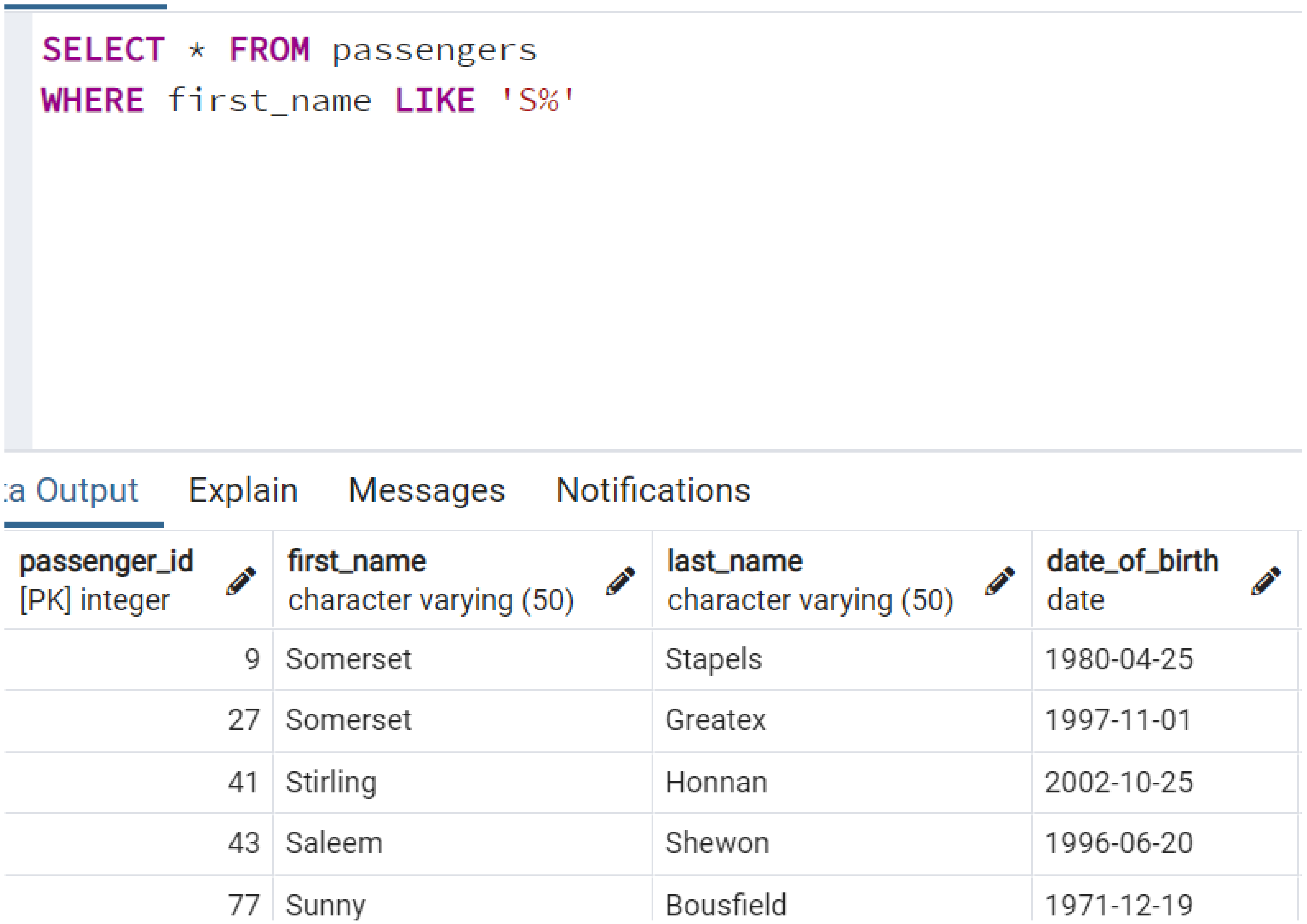
```
WHERE condition
```

WHERE clause

- The optional `WHERE` clause has the general form

```
WHERE condition
```

- Where ***condition*** is any expression that evaluates to a result of type boolean.
- Any row that does not satisfy this condition will be eliminated from the output.



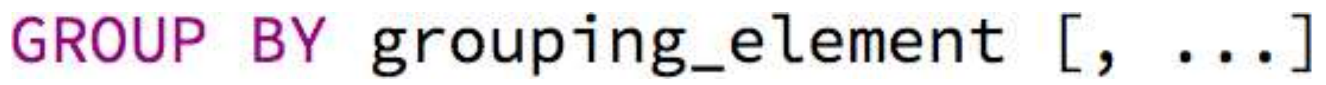
```
SELECT * FROM passengers
WHERE first_name LIKE 'S%'
```

:a Output | Explain | Messages | Notifications

passenger_id [PK] integer	first_name character varying (50)	last_name character varying (50)	date_of_birth date
9	Somerset	Stapels	1980-04-25
27	Somerset	Greatex	1997-11-01
41	Stirling	Honnan	2002-10-25
43	Saleem	Shewon	1996-06-20
77	Sunny	Bousfield	1971-12-19

GROUP BY

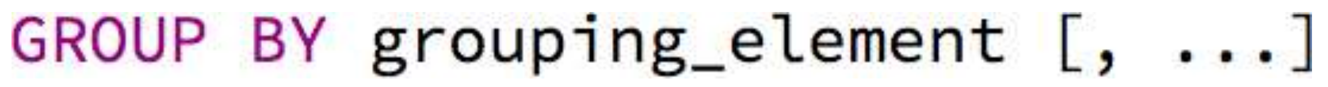
- The optional `GROUP BY` clause has the general form



```
GROUP BY grouping_element [, ...]
```

GROUP BY

- The optional `GROUP BY` clause has the general form

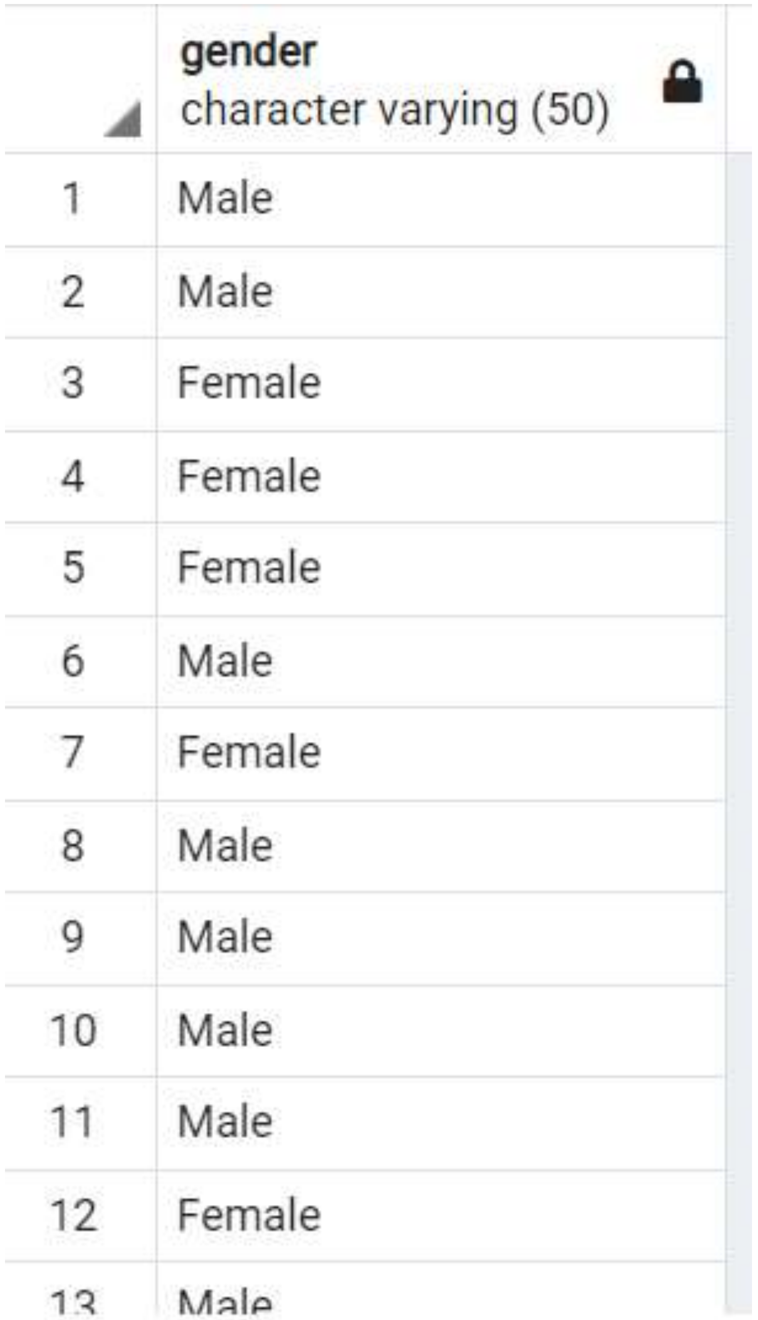


```
GROUP BY grouping_element [, ...]
```

- GROUP BY will condense into a single row all selected rows that share the same values for the grouped expressions.

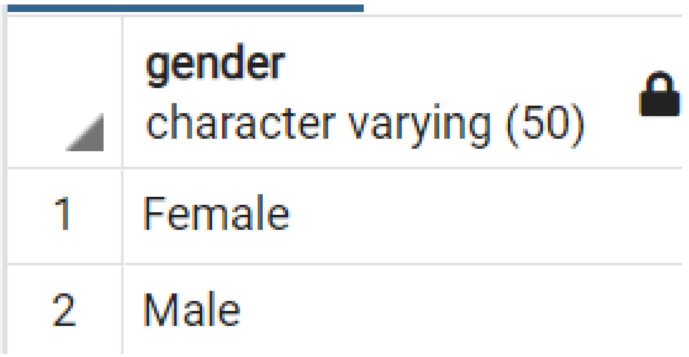
- An *`expression`* used inside a *`grouping_element`* can be an input column name, or the name or ordinal number of an output column, or an arbitrary expression formed from input-column values.

```sql
SELECT gender FROM passengers
```

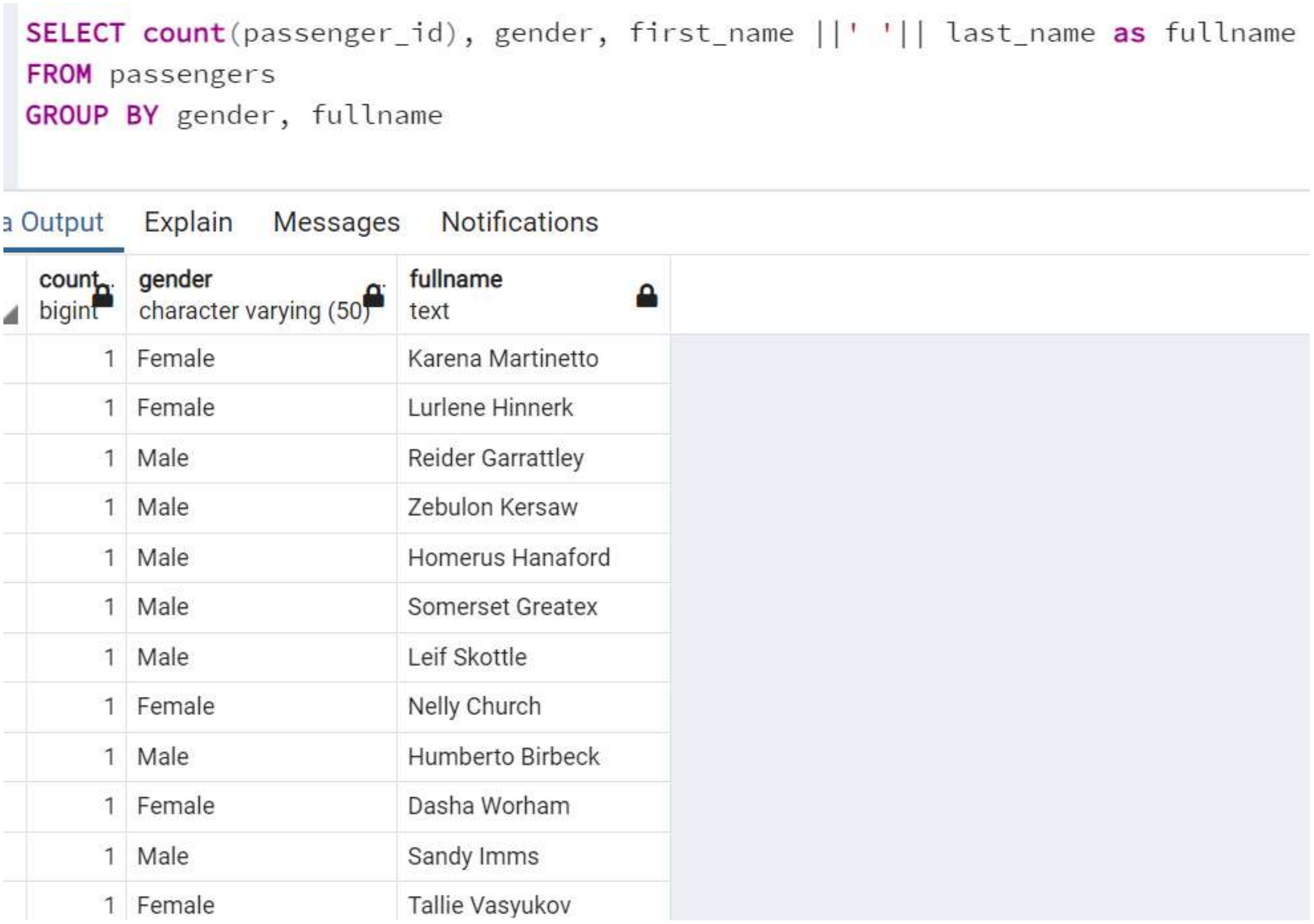


	gender character varying (50)
1	Male
2	Male
3	Female
4	Female
5	Female
6	Male
7	Female
8	Male
9	Male
10	Male
11	Male
12	Female
13	Male

```sql
SELECT gender FROM passengers
GROUP BY gender
```



	gender character varying (50)
1	Female
2	Male



```
SELECT count(passenger_id), gender, first_name ||' '|| last_name as fullname
FROM passengers
GROUP BY gender, fullname
```

count bigint	gender character varying (50)	fullname text
1	Female	Karena Martinetto
1	Female	Lurlene Hinnerk
1	Male	Reider Garrattley
1	Male	Zebulon Kersaw
1	Male	Homerus Hanaford
1	Male	Somerset Greatex
1	Male	Leif Skottle
1	Female	Nelly Church
1	Male	Humberto Birbeck
1	Female	Dasha Worham
1	Male	Sandy Imms
1	Female	Tallie Vasyukov

HAVING

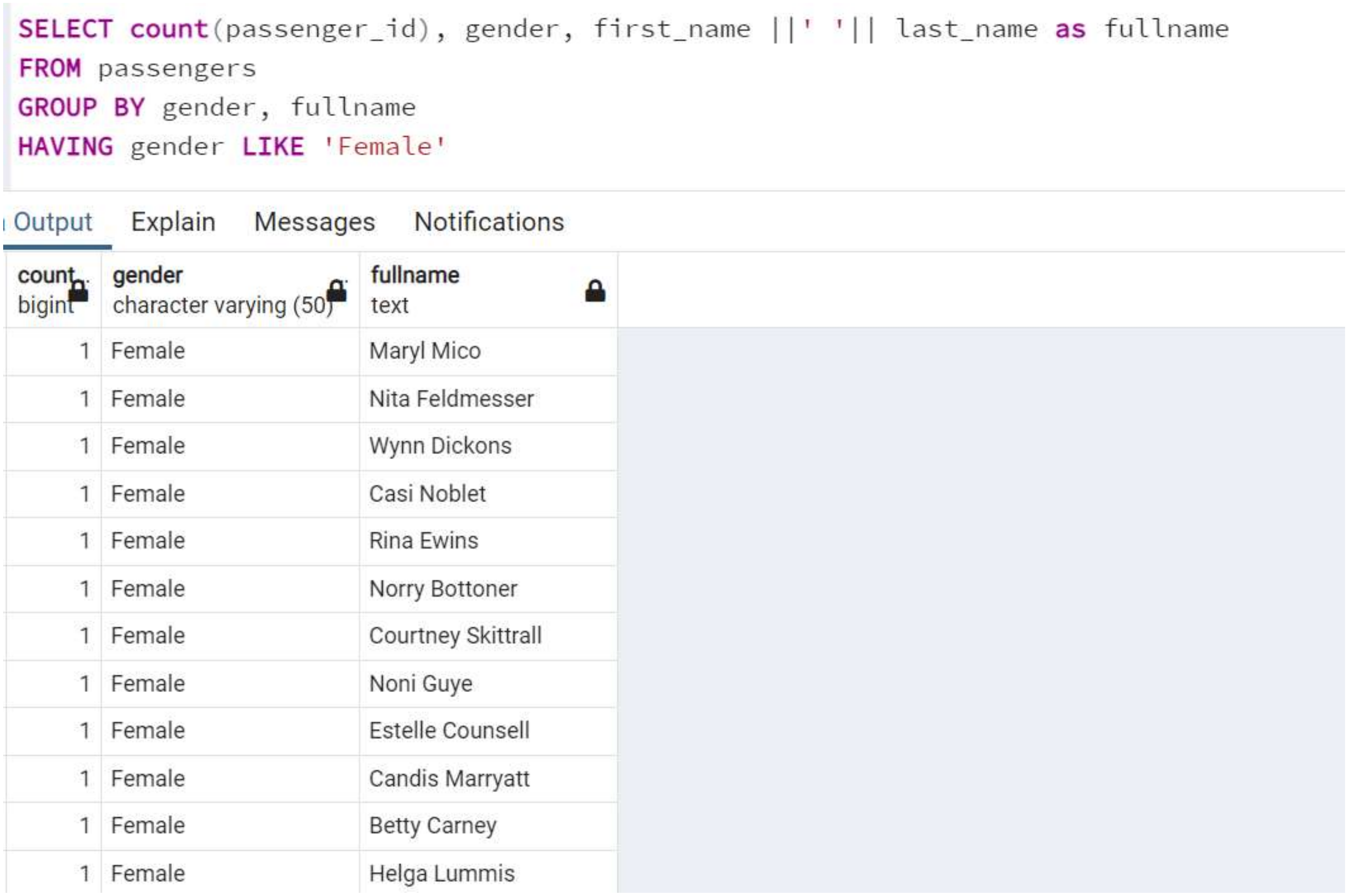
- The optional HAVING clause has the general form

```
HAVING condition
```

- where *`condition`* is the same as specified for the `WHERE` clause.
- `HAVING` eliminates group rows that do not satisfy the condition.

HAVING

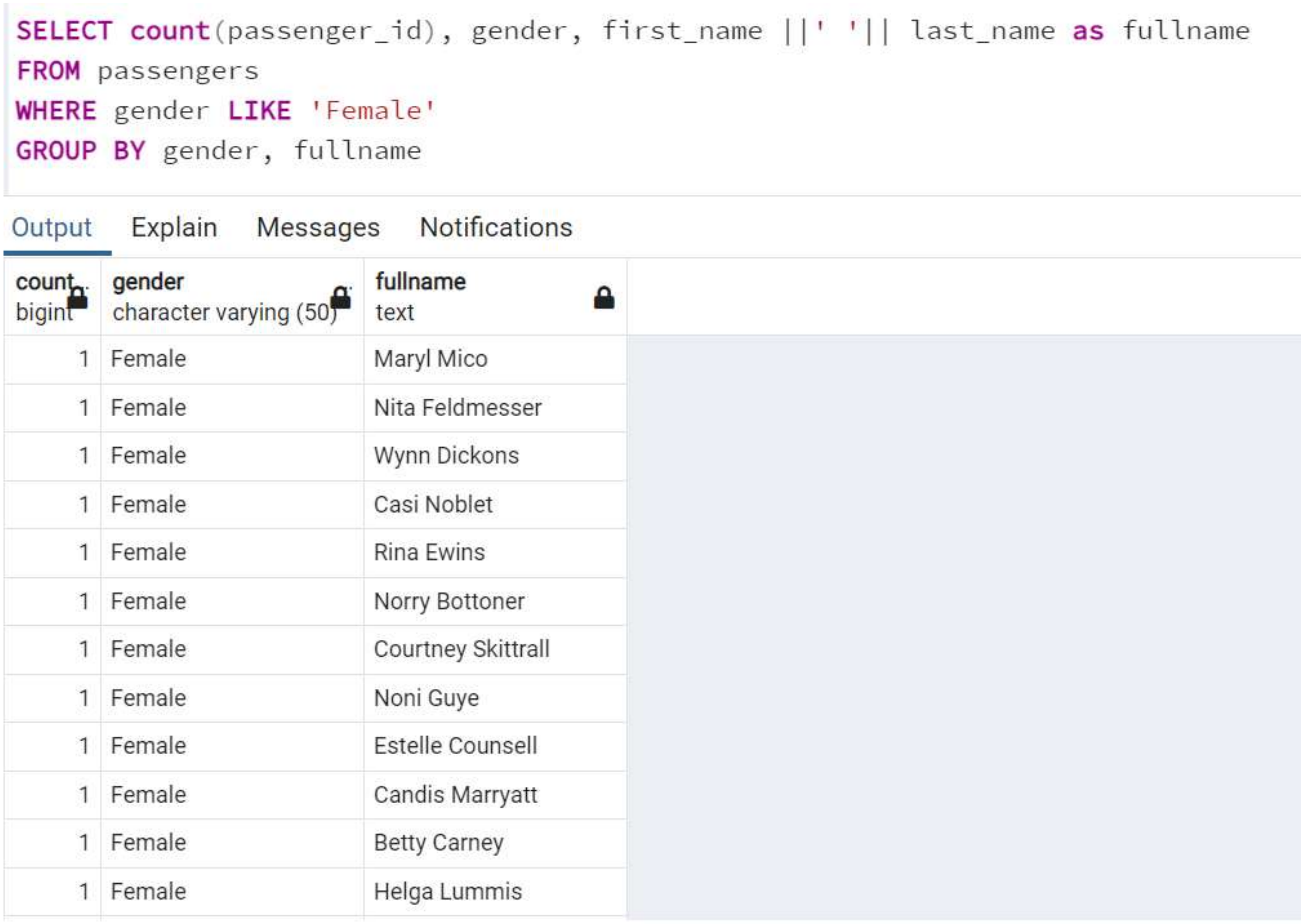
- `HAVING` is different from `WHERE`:
- `WHERE` filters individual rows before the application of `GROUP BY`
- `HAVING` filters group rows created by `GROUP BY`
- Each column referenced in *`condition`* must unambiguously reference a grouping column, unless the reference appears within an aggregate function



```sql
SELECT count(passenger_id), gender, first_name ||' '|| last_name as fullname
FROM passengers
GROUP BY gender, fullname
HAVING gender LIKE 'Female'
```

Output Explain Messages Notifications

count bigint	gender character varying (50)	fullname text
1	Female	Maryl Mico
1	Female	Nita Feldmesser
1	Female	Wynn Dickons
1	Female	Casi Noblet
1	Female	Rina Ewins
1	Female	Norry Bottoner
1	Female	Courtney Skittrall
1	Female	Noni Guye
1	Female	Estelle Counsell
1	Female	Candis Marryatt
1	Female	Betty Carney
1	Female	Helga Lummis

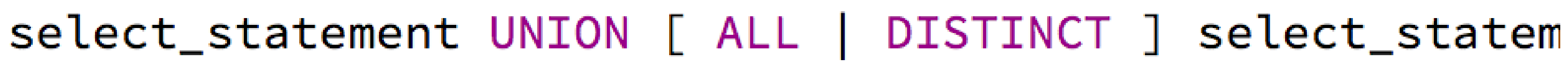


```
SELECT count(passenger_id), gender, first_name ||' '|| last_name as fullname
FROM passengers
WHERE gender LIKE 'Female'
GROUP BY gender, fullname
```

count bigint	gender character varying (50)	fullname text
1	Female	Maryl Mico
1	Female	Nita Feldmesser
1	Female	Wynn Dickons
1	Female	Casi Noblet
1	Female	Rina Ewins
1	Female	Norry Bottoner
1	Female	Courtney Skittrall
1	Female	Noni Guye
1	Female	Estelle Counsell
1	Female	Candis Marryatt
1	Female	Betty Carney
1	Female	Helga Lummis

UNION

- The `UNION` clause has this general form:



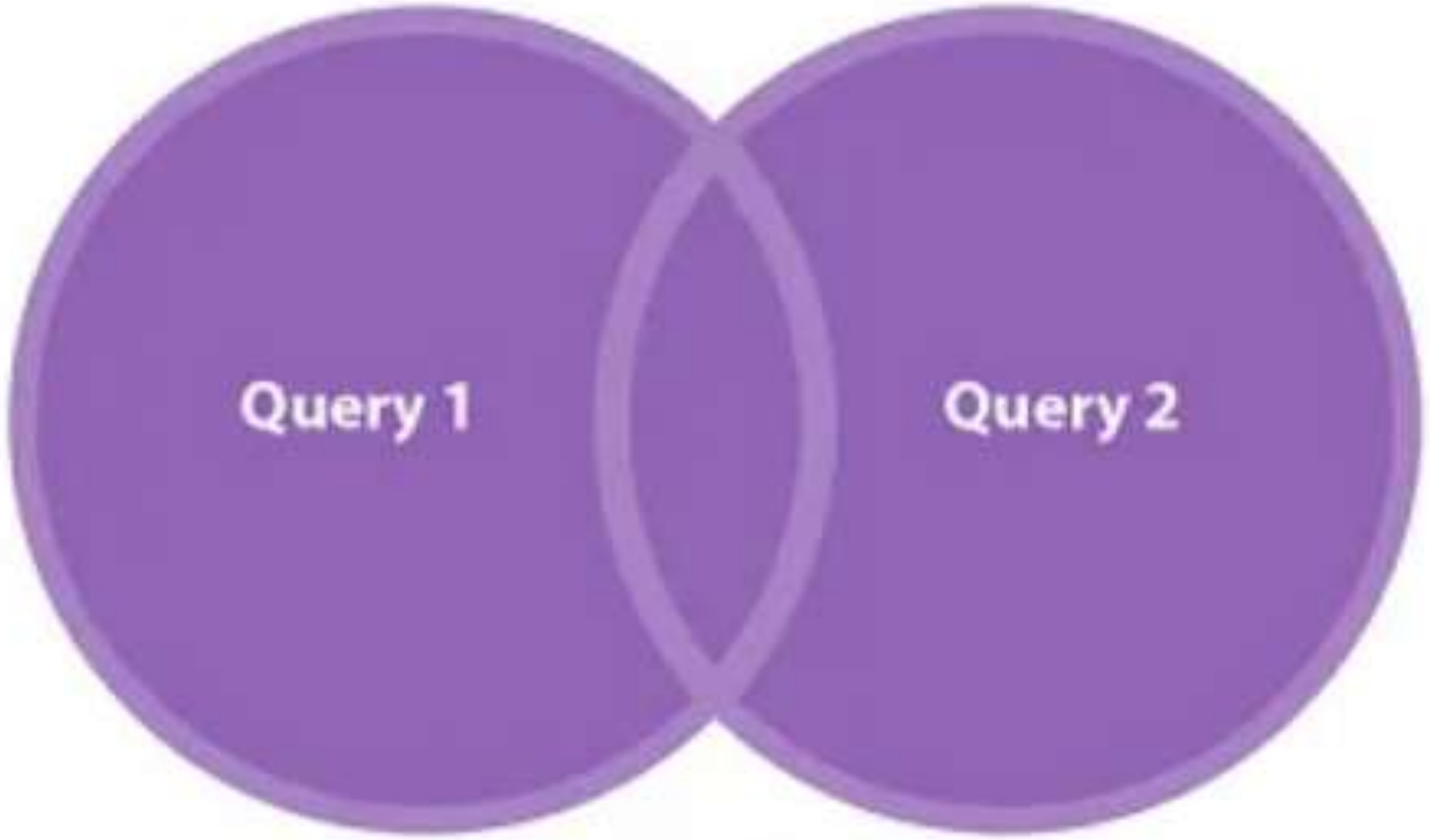
```
select_statement UNION [ ALL | DISTINCT ] select_statem
```

- *`select_statement`* is any `SELECT` statement without an `ORDER BY`, `LIMIT` clause

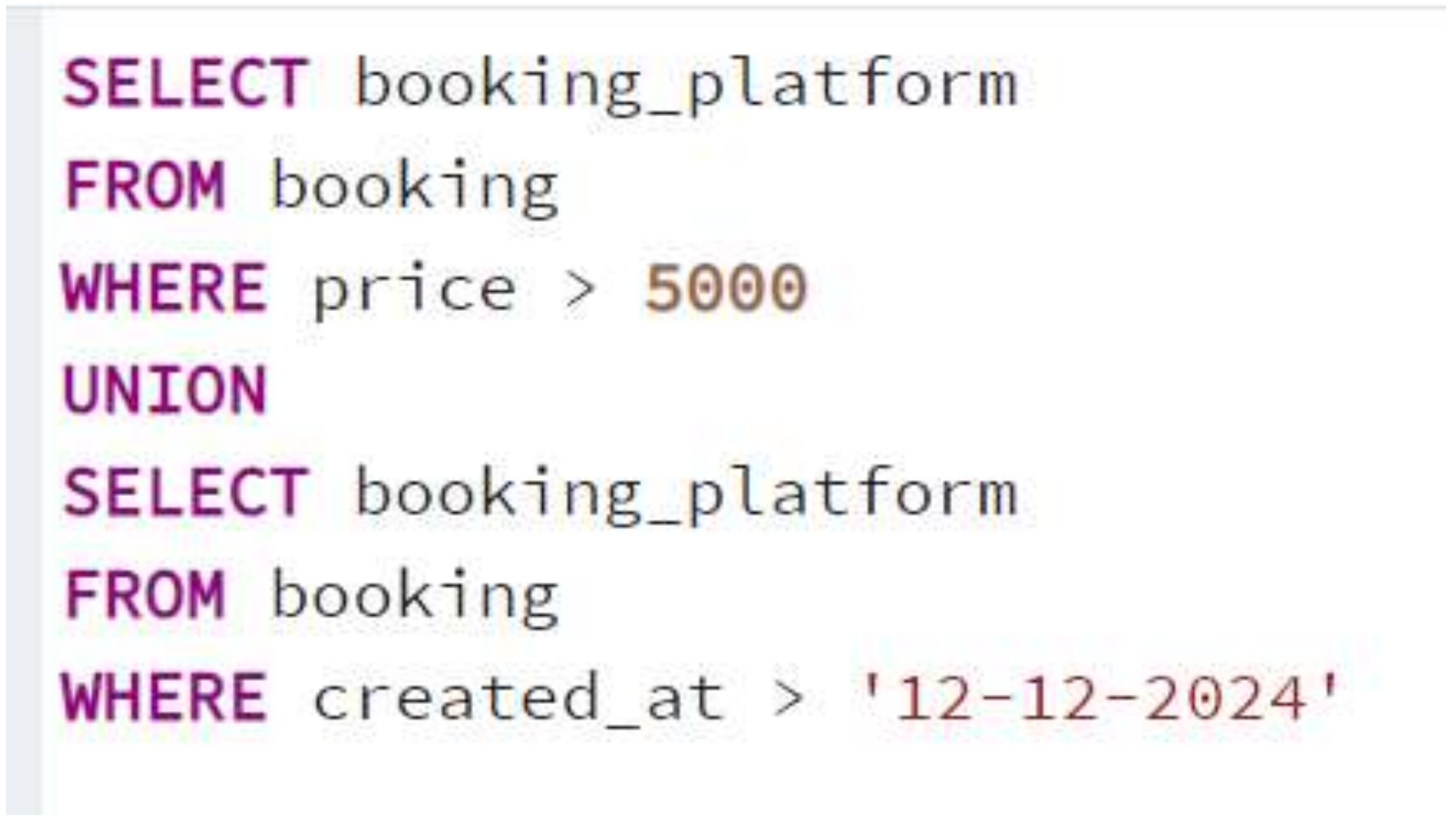
- The `UNION` operator computes the set union of the rows returned by the involved `SELECT` statements.

UNION

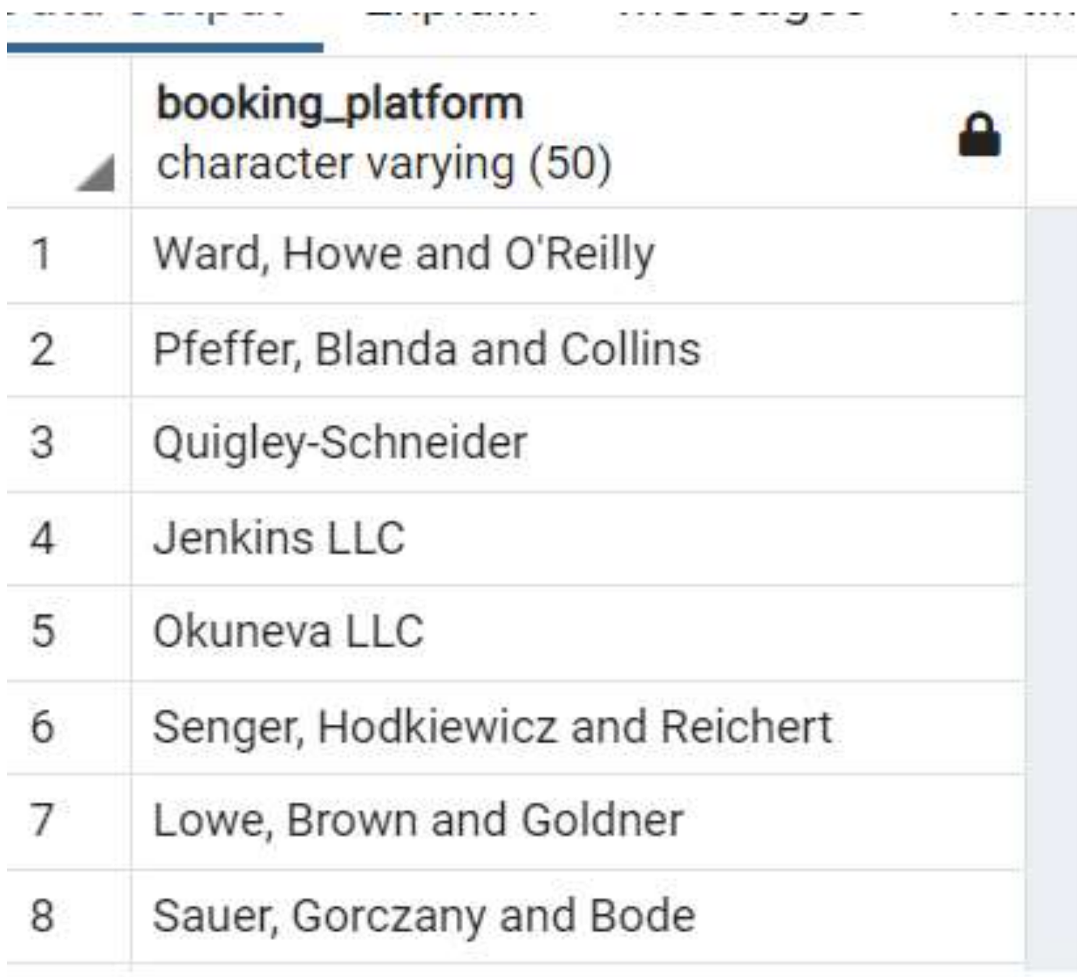
- A row is in the set union of two result sets if it appears in at least one of the result sets.
- The two SELECT statements that represent the direct operands of the UNION must produce the same number of columns
- Corresponding columns must be of compatible data types.



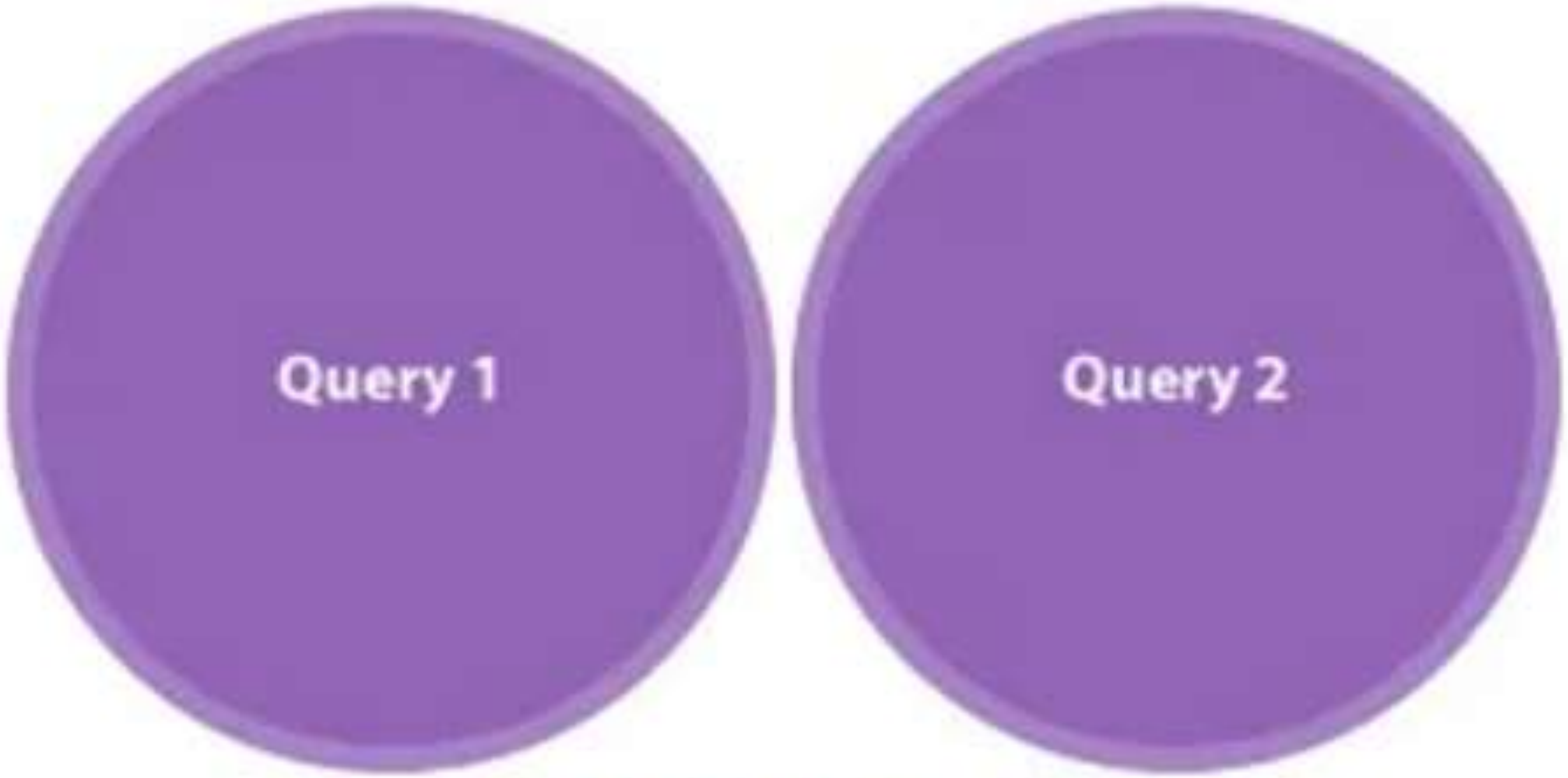
UNION



```sql
SELECT booking_platform
FROM booking
WHERE price > 5000
UNION
SELECT booking_platform
FROM booking
WHERE created_at > '12-12-2024'
```

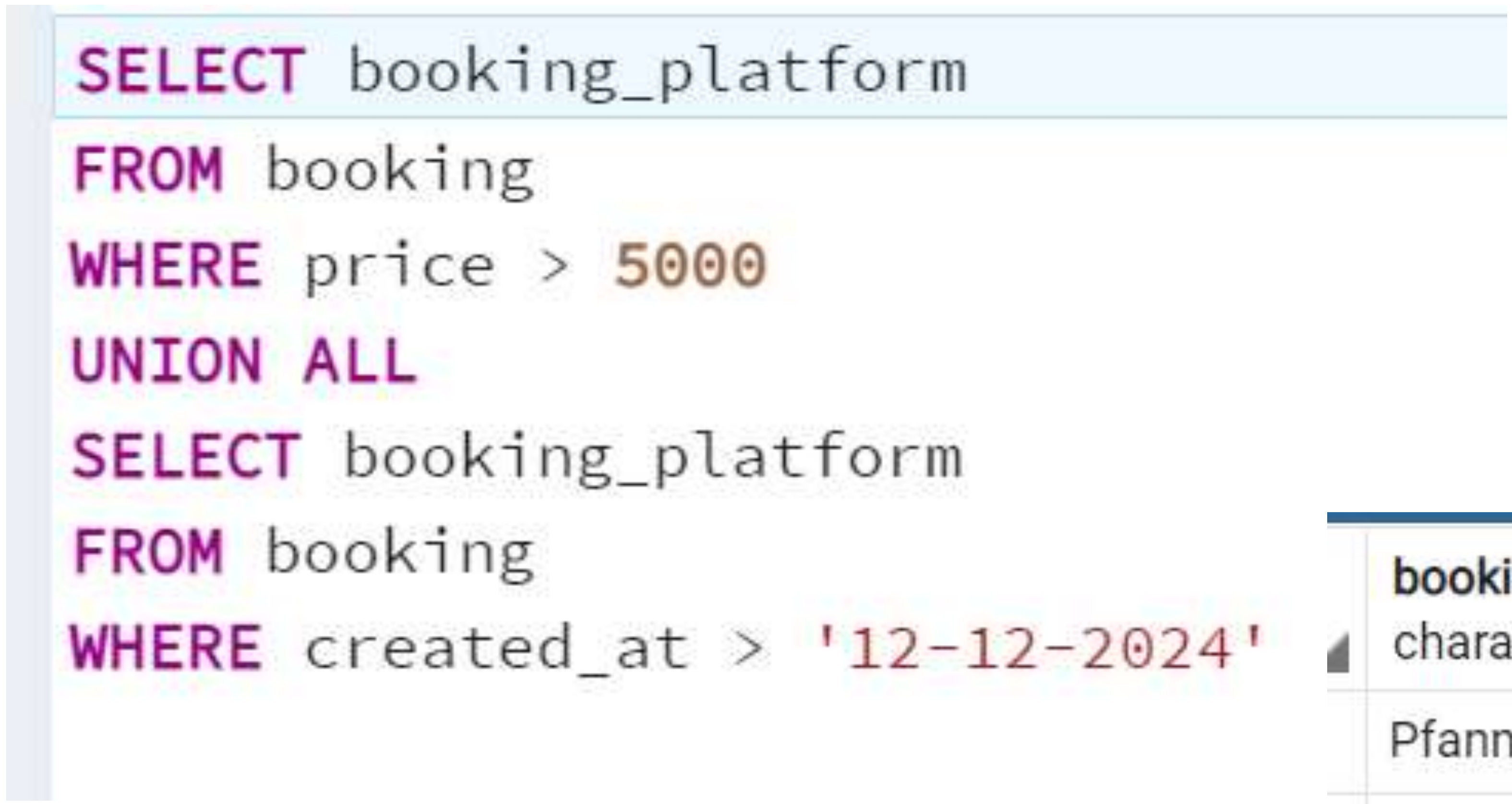


	booking_platform character varying (50)
1	Ward, Howe and O'Reilly
2	Pfeffer, Blanda and Collins
3	Quigley-Schneider
4	Jenkins LLC
5	Okuneva LLC
6	Senger, Hodkiewicz and Reichert
7	Lowe, Brown and Goldner
8	Sauer, Gorczany and Bode

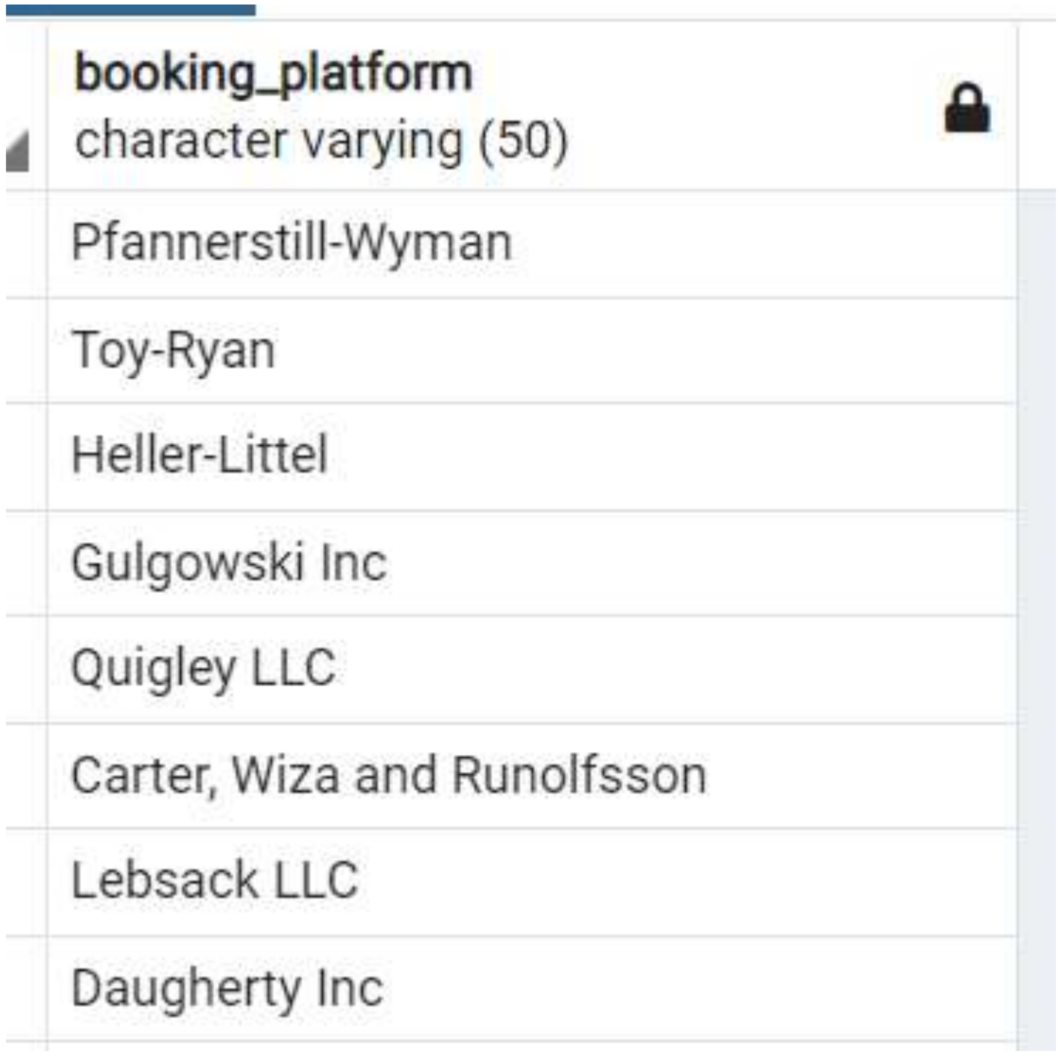


UNION ALL

UNION ALL



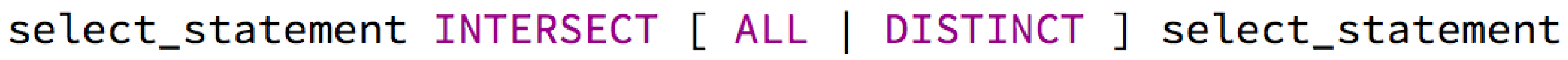
```
SELECT booking_platform
FROM booking
WHERE price > 5000
UNION ALL
SELECT booking_platform
FROM booking
WHERE created_at > '12-12-2024'
```



booking_platform character varying (50)
Pfannerstill-Wyman
Toy-Ryan
Heller-Littel
Gulgowski Inc
Quigley LLC
Carter, Wiza and Runolfsson
Lebsack LLC
Daugherty Inc

INTERSECT

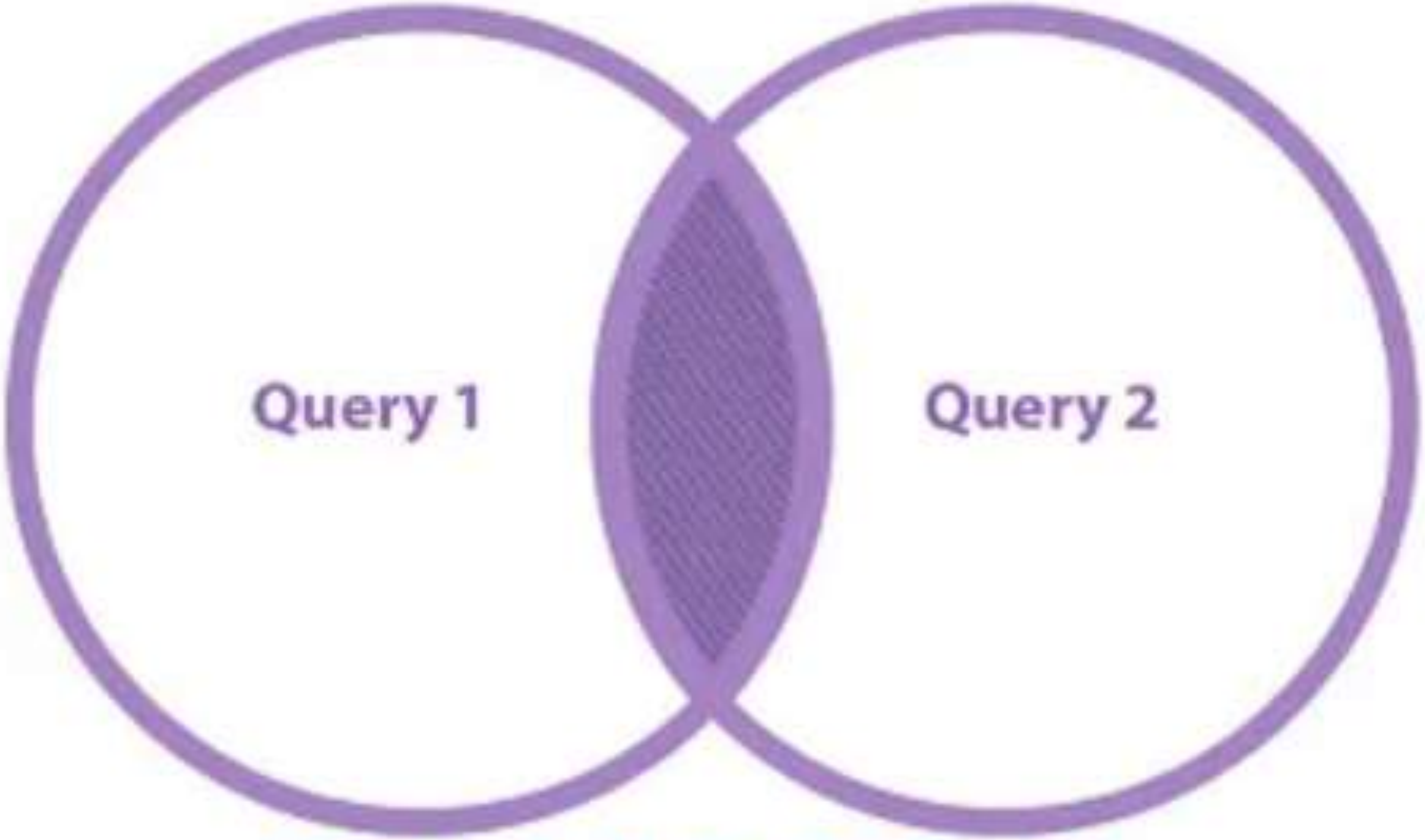
- The `INTERSECT` clause has this general form:



```
select_statement INTERSECT [ ALL | DISTINCT ] select_statement
```

- *`select_statement`* is any `SELECT` statement without an `ORDER BY`, `LIMIT` clause

- The `INTERSECT` operator computes the set intersection of the rows returned by the involved `SELECT` statements.



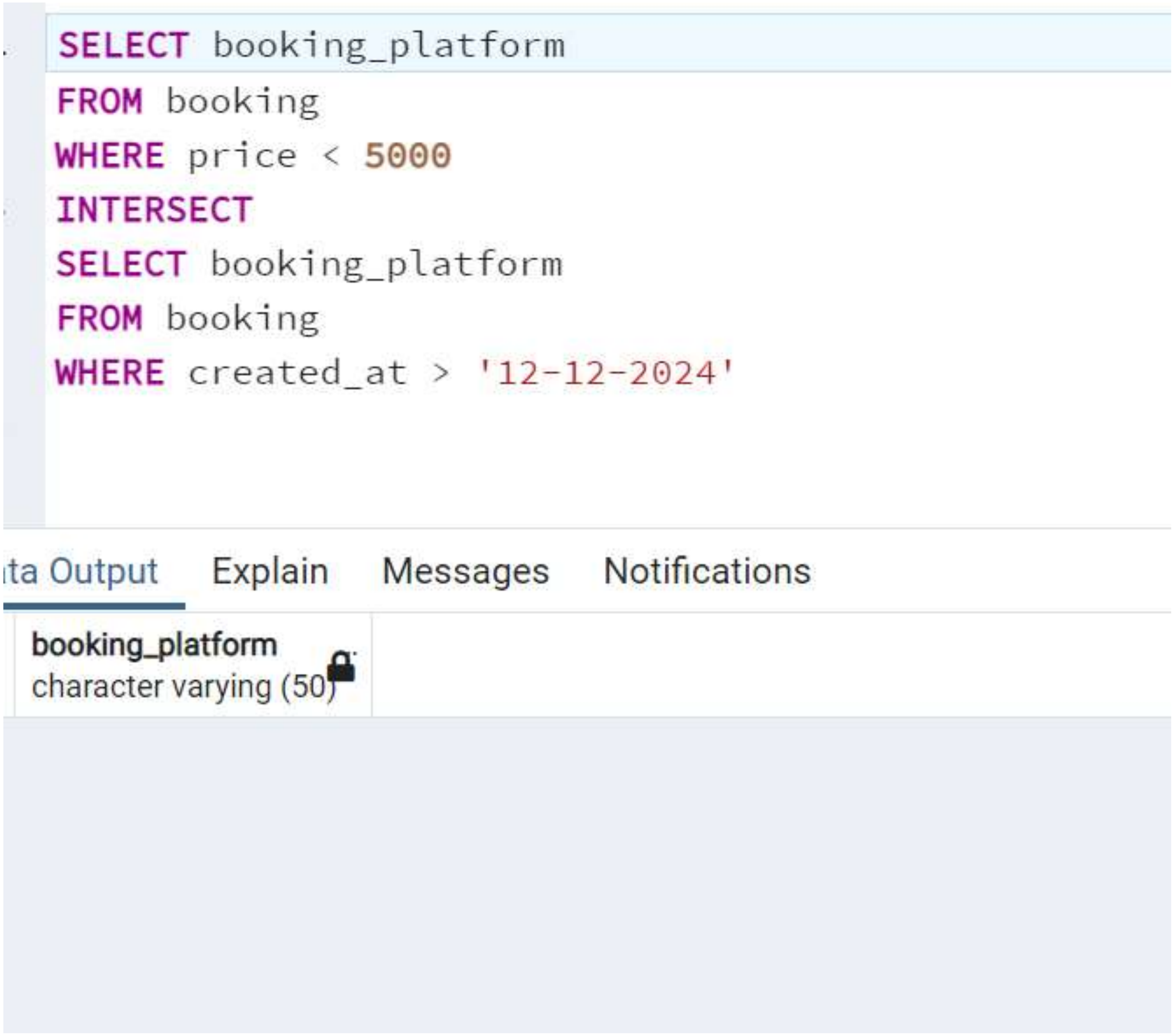
INTERSECT

INTERSECT

- A row is in the intersection of two result sets if it appears in both result sets.
- The result of `INTERSECT` does not contain any duplicate rows unless the `ALL` option is specified.
- With `ALL`, a row that has m duplicates in the left table and n duplicates in the right table will appear min(m,n) times in the result set.

INTERSECT

- `INTERSECT` binds more tightly than `UNION`.
- A `UNION B INTERSECT C` will be read as `A UNION (B INTERSECT C)`



```
SELECT booking_platform
FROM booking
WHERE price < 5000
INTERSECT
SELECT booking_platform
FROM booking
WHERE created_at > '12-12-2024'
```

```
booking_platform
character varying (50)
```

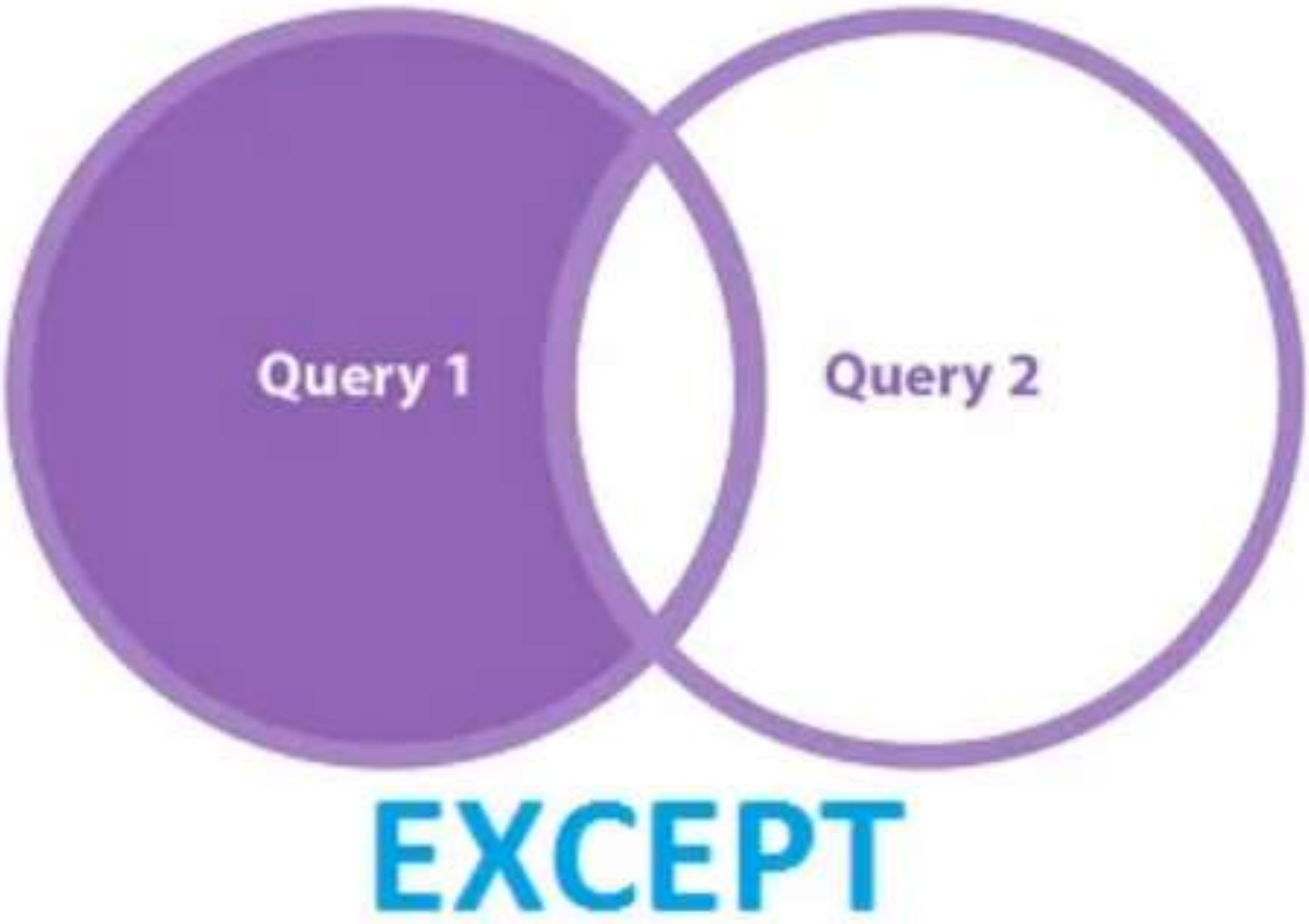
EXCEPT

- The `EXCEPT` clause has this general form:

```
select_statement EXCEPT [ ALL | DISTINCT ] select_statement
```

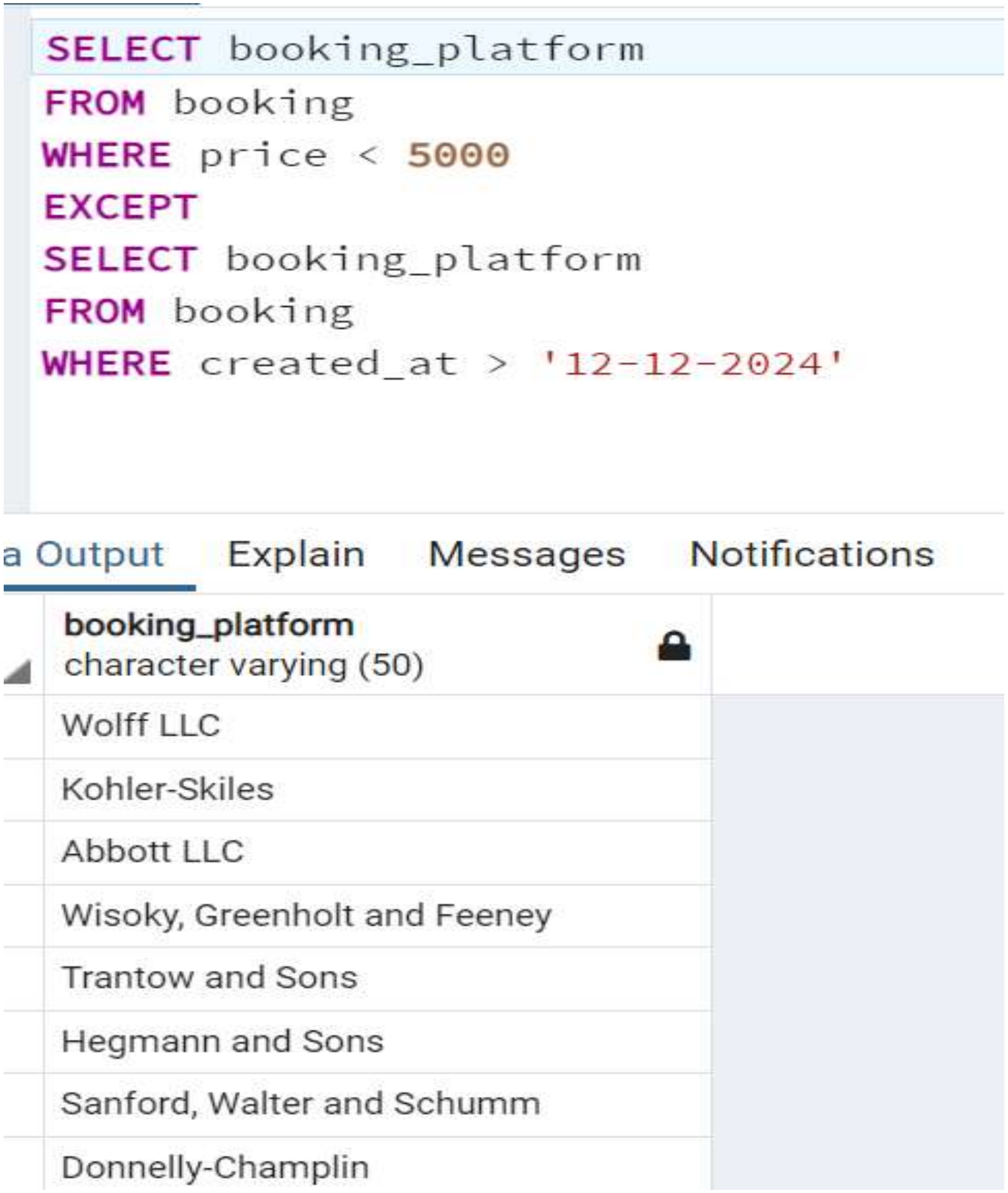
- *`select_statement`* is any `SELECT` statement without an `ORDER BY`, `LIMIT` clause

- The `EXCEPT` operator computes the set of rows that are in the result of the left `SELECT` statement but not in the result of the right one.


Query 1
Query 2
EXCEPT

EXCEPT

- The result of `EXCEPT` does not contain any duplicate rows unless the `ALL` option is specified.
- With `ALL`, a row that has *m* duplicates in the left table and *n* duplicates in the right table will appear max(*m*-*n*,0) times in the result set.
- Multiple EXCEPT operators in the same SELECT statement are evaluated left to right, unless parentheses dictate otherwise.
- EXCEPT binds at the same level as UNION.

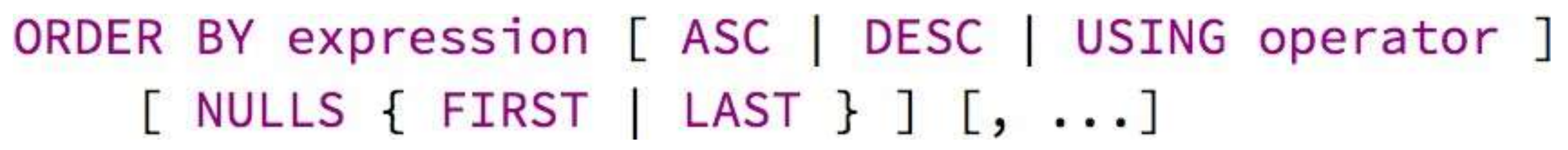


```
SELECT booking_platform
FROM booking
WHERE price < 5000
EXCEPT
SELECT booking_platform
FROM booking
WHERE created_at > '12-12-2024'
```

```
booking_platform
character varying (50)
Wolff LLC
Kohler-Skiles
Abbott LLC
Wisoky, Greenholt and Feeney
Trantow and Sons
Hegmann and Sons
Sanford, Walter and Schumm
Donnelly-Champlin
```

ORDER BY

- The optional `ORDER BY` clause has the general form



```
ORDER BY expression [ ASC | DESC | USING operator ]
    [ NULLS { FIRST | LAST } ] [, ...]
```

- The `ORDER BY` clause causes the result rows to be sorted according to the specified expression(s)

- If two rows are equal according to the leftmost expression, they are compared according to the next expression and so on.

ORDER BY

- If they are equal according to all specified expressions, they are returned in an implementation-dependent order.

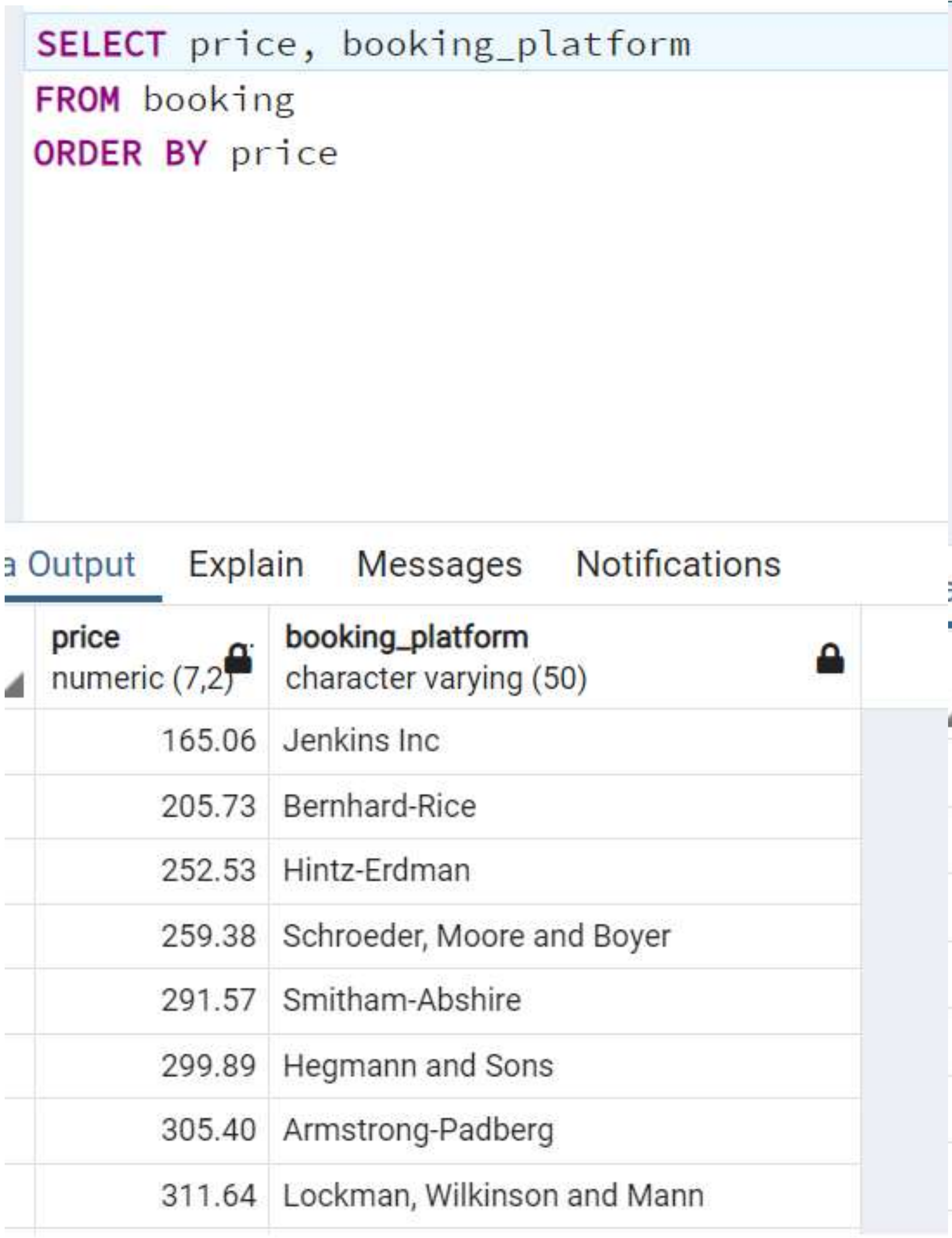
- Each *`expression`* can be the name or ordinal number of an output column (`SELECT` list item)

- or it can be an arbitrary expression formed from input-column values.

- Optionally one can add the key word `ASC` (ascending) or `DESC` (descending)

ORDER BY

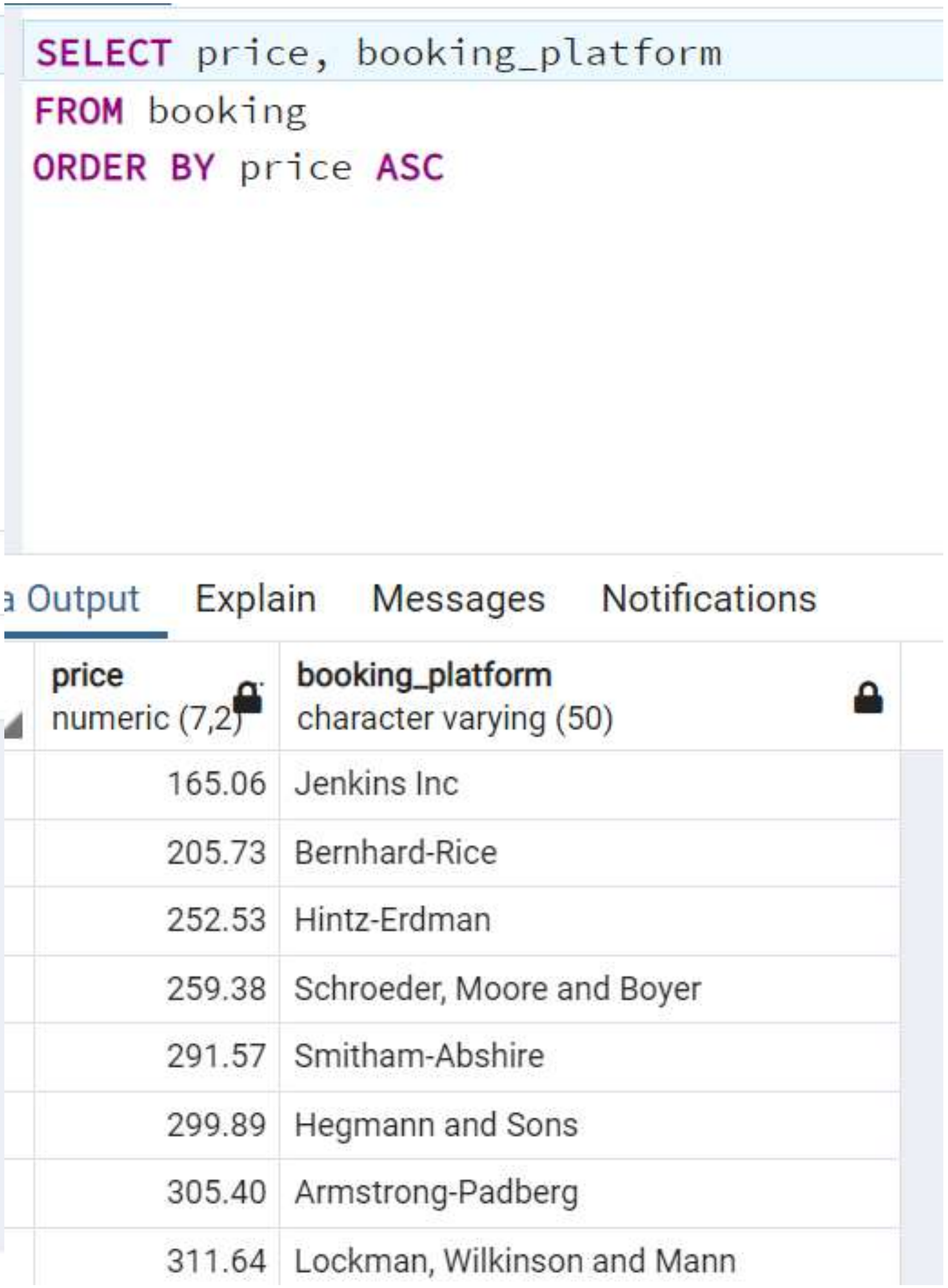
- If not specified, `ASC` is assumed by default.
- Alternatively, a specific ordering operator name can be specified in the `USING` clause.Each *`expression`* can be the name or ordinal number of an output column (`SELECT` list item)
- If `NULLS LAST` is specified, null values sort after all non-null values
- If `NULLS FIRST` is specified, null values sort before all non-null values.



```sql
SELECT price, booking_platform
FROM booking
ORDER BY price
```

a Output Explain Messages Notifications

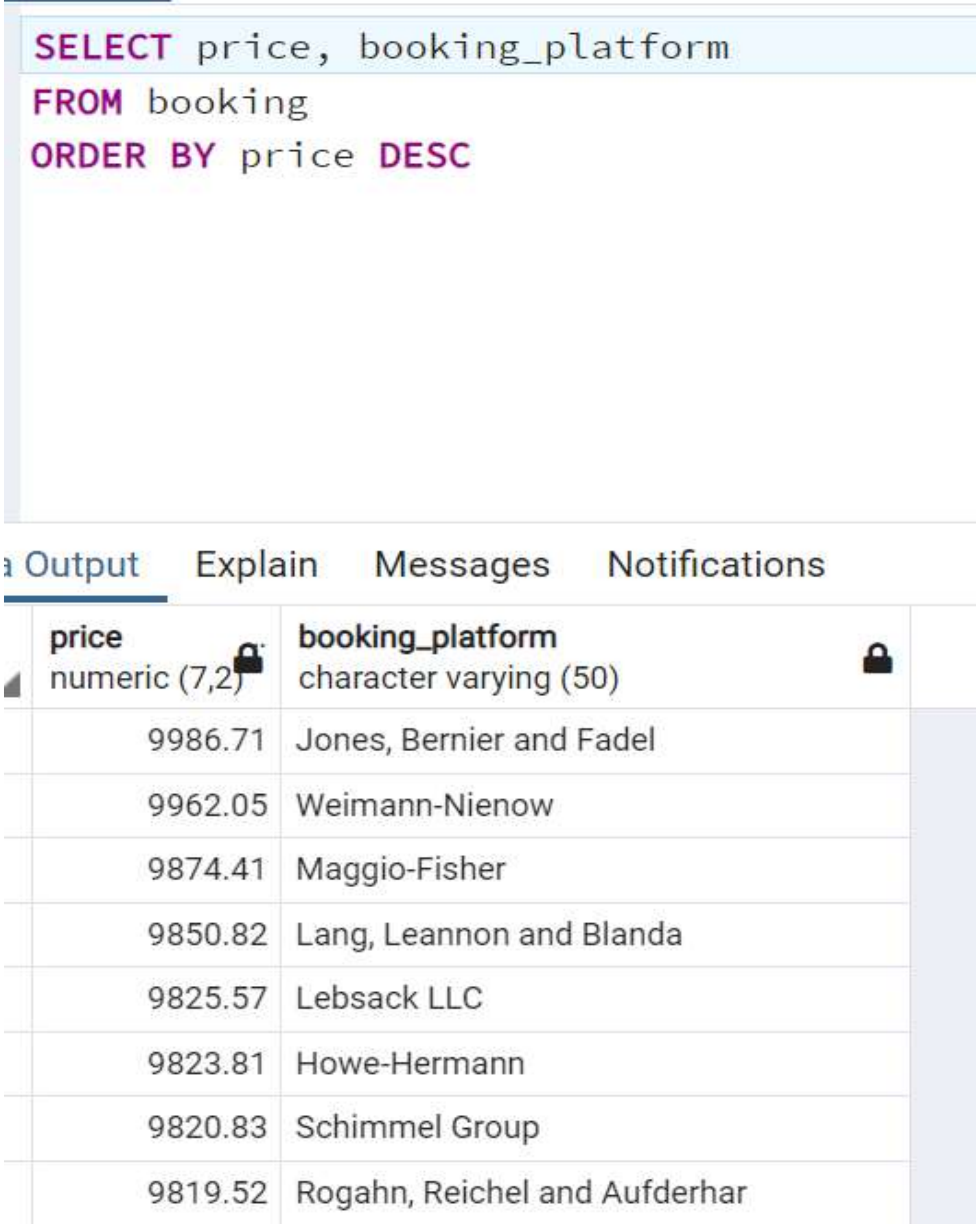
price numeric (7,2)	booking_platform character varying (50)
165.06	Jenkins Inc
205.73	Bernhard-Rice
252.53	Hintz-Erdman
259.38	Schroeder, Moore and Boyer
291.57	Smitham-Abshire
299.89	Hegmann and Sons
305.40	Armstrong-Padberg
311.64	Lockman, Wilkinson and Mann



```sql
SELECT price, booking_platform
FROM booking
ORDER BY price ASC
```

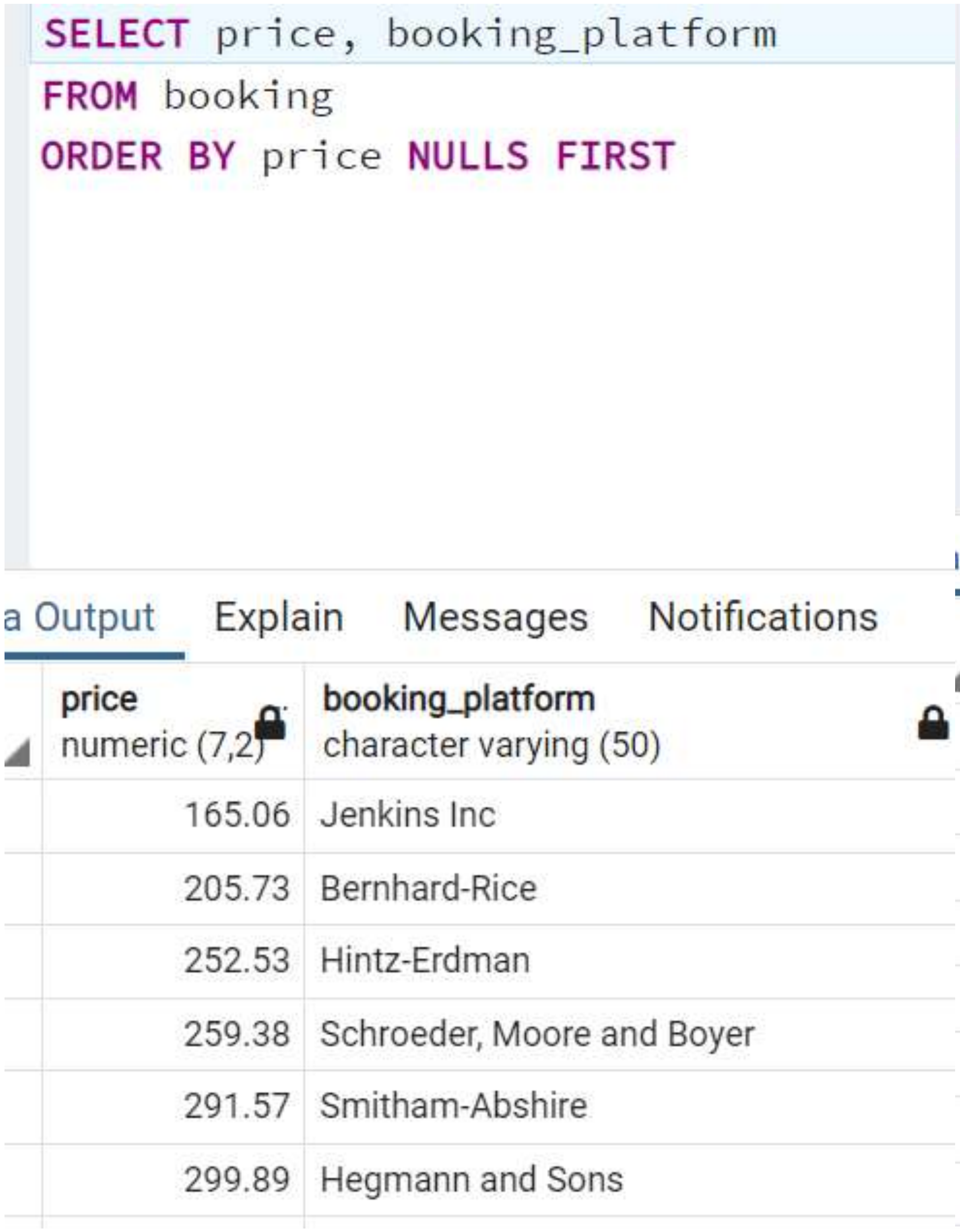
a Output Explain Messages Notifications

price numeric (7,2)	booking_platform character varying (50)
165.06	Jenkins Inc
205.73	Bernhard-Rice
252.53	Hintz-Erdman
259.38	Schroeder, Moore and Boyer
291.57	Smitham-Abshire
299.89	Hegmann and Sons
305.40	Armstrong-Padberg
311.64	Lockman, Wilkinson and Mann



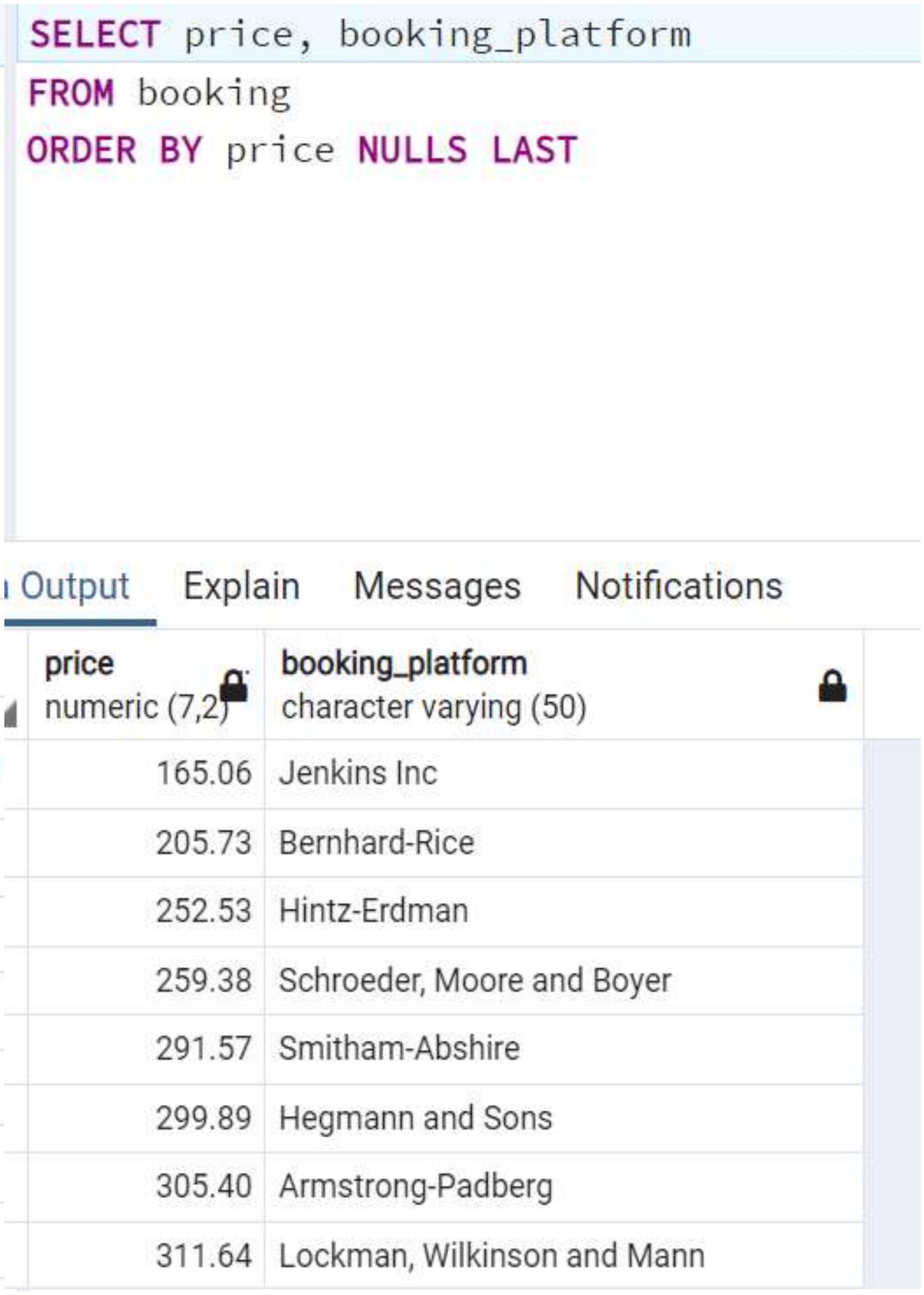
```
SELECT price, booking_platform
FROM booking
ORDER BY price DESC
```

price numeric (7,2)	booking_platform character varying (50)
9986.71	Jones, Bernier and Fadel
9962.05	Weimann-Nienow
9874.41	Maggio-Fisher
9850.82	Lang, Leannon and Blanda
9825.57	Lebsack LLC
9823.81	Howe-Hermann
9820.83	Schimmel Group
9819.52	Rogahn, Reichel and Aufderhar



```sql
SELECT price, booking_platform
FROM booking
ORDER BY price NULLS FIRST
```

price numeric (7,2)	booking_platform character varying (50)
165.06	Jenkins Inc
205.73	Bernhard-Rice
252.53	Hintz-Erdman
259.38	Schroeder, Moore and Boyer
291.57	Smitham-Abshire
299.89	Hegmann and Sons

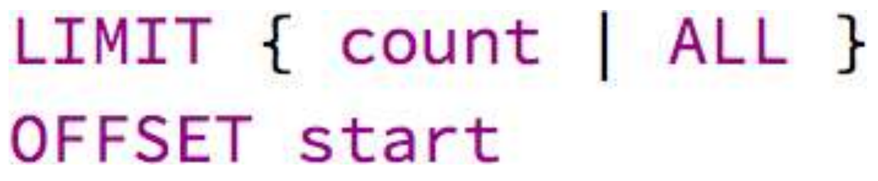


```sql
SELECT price, booking_platform
FROM booking
ORDER BY price NULLS LAST
```

price numeric (7,2)	booking_platform character varying (50)
165.06	Jenkins Inc
205.73	Bernhard-Rice
252.53	Hintz-Erdman
259.38	Schroeder, Moore and Boyer
291.57	Smitham-Abshire
299.89	Hegmann and Sons
305.40	Armstrong-Padberg
311.64	Lockman, Wilkinson and Mann

LIMIT

- The `LIMIT` clause consists of two independent sub-clauses:



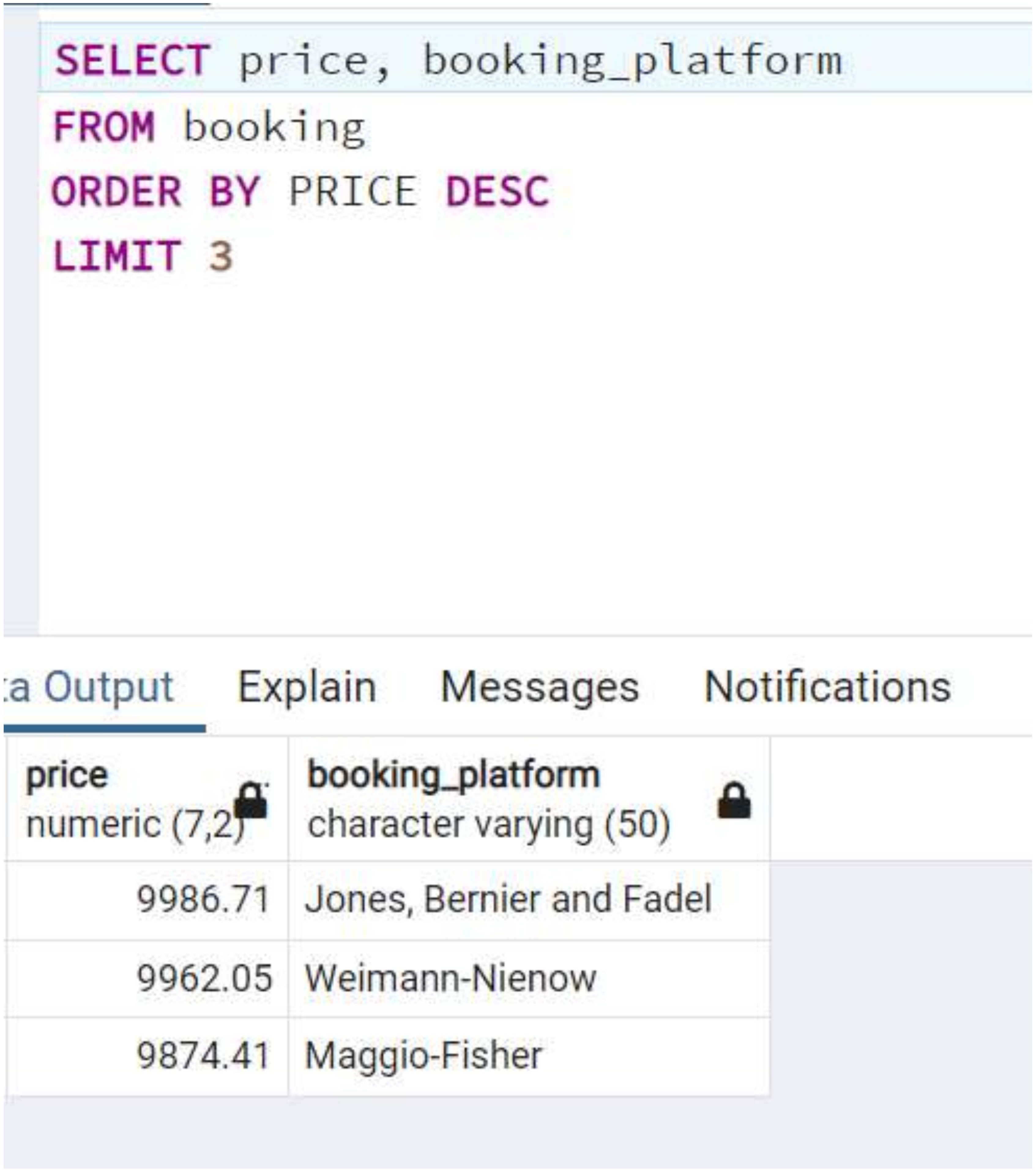
```
LIMIT { count | ALL }
OFFSET start
```

- *`count`* specifies the maximum number of rows to return,

- while *`start`* specifies the number of rows to skip before starting to return rows.

LIMIT

- When both are specified, *`start`* rows are skipped before starting to count the *`count`* rows to be returned.
- If the *`count`* expression evaluates to NULL, it is treated as `LIMIT ALL`, i.e., no limit.
- If *`start`* evaluates to NULL, it is treated the same as `OFFSET 0`.



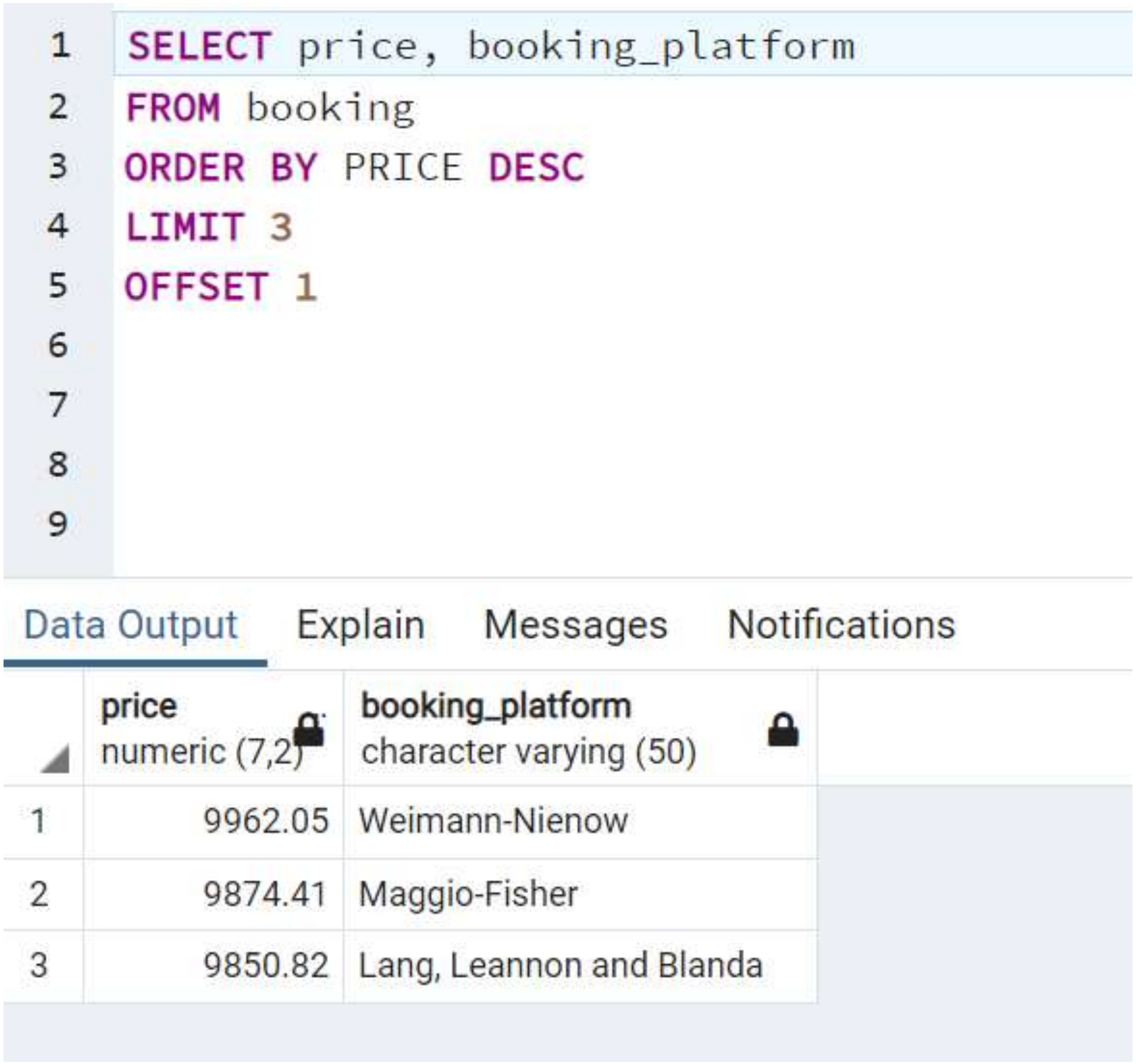
```sql
SELECT price, booking_platform
FROM booking
ORDER BY PRICE DESC
LIMIT 3
```

price numeric (7,2)	booking_platform character varying (50)
9986.71	Jones, Bernier and Fadel
9962.05	Weimann-Nienow
9874.41	Maggio-Fisher

OFFSET

The **OFFSET** clause is used to skip a specified number of rows in the result set before starting to return rows.

It is commonly used with the **LIMIT** clause for pagination or to retrieve a subset of rows from a query.



```sql
SELECT price, booking_platform
FROM booking
ORDER BY PRICE DESC
LIMIT 3
OFFSET 1
```

	price numeric (7,2)	booking_platform character varying (50)
1	9962.05	Weimann-Nienow
2	9874.41	Maggio-Fisher
3	9850.82	Lang, Leannon and Blanda