Introduction to PostgreSQL UNION operator

The UNION operator combines result sets of two or more SELECT statements into a single result set.

The following illustrates the syntax of the UNION operator that combines result sets from two queries:

```
SELECT select_list_1
FROM table_expresssion_1
UNION
SELECT select_list_2
FROM table_expression_2
```

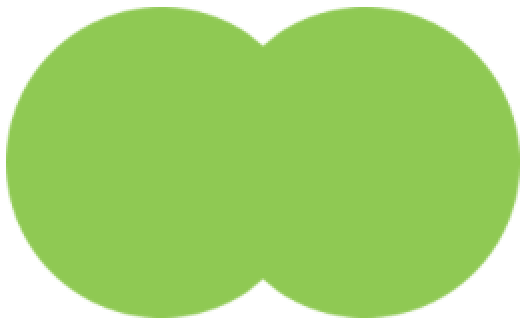
To combine the result sets of two queries using the UNION operator, the queries must conform to the following rules:

The number and the order of the columns in the select list of both queries must be the same.

The data types must be compatible.

The UNION operator removes all duplicate rows from the combined data set. To retain the duplicate rows, you use the UNION ALL instead.

The following Venn diagram illustrates how to the UNION works:



PostgreSQL UNION with ORDER BY clause

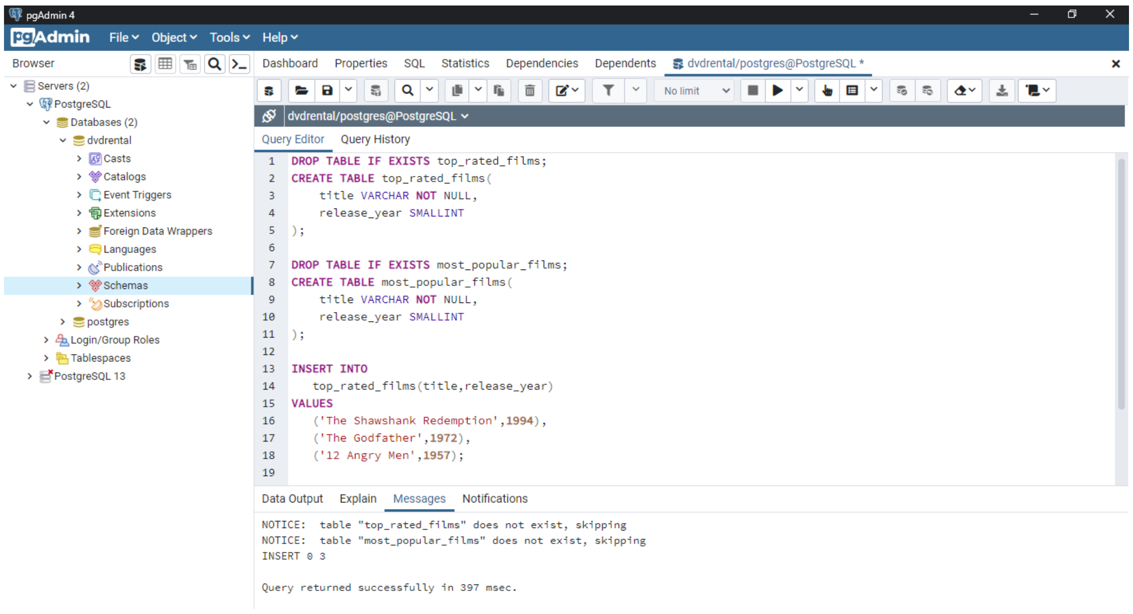
The UNION operator may place the rows from the result set of the first query before, after, or between the rows from the result set of the second query.

To sort rows in the final result set, you use the ORDER BY clause in the second query.

In practice, you often use the UNION operator to combine data from similar tables, which are not perfectly normalized, in the data warehouse or business intelligence systems.

Setting up sample tables

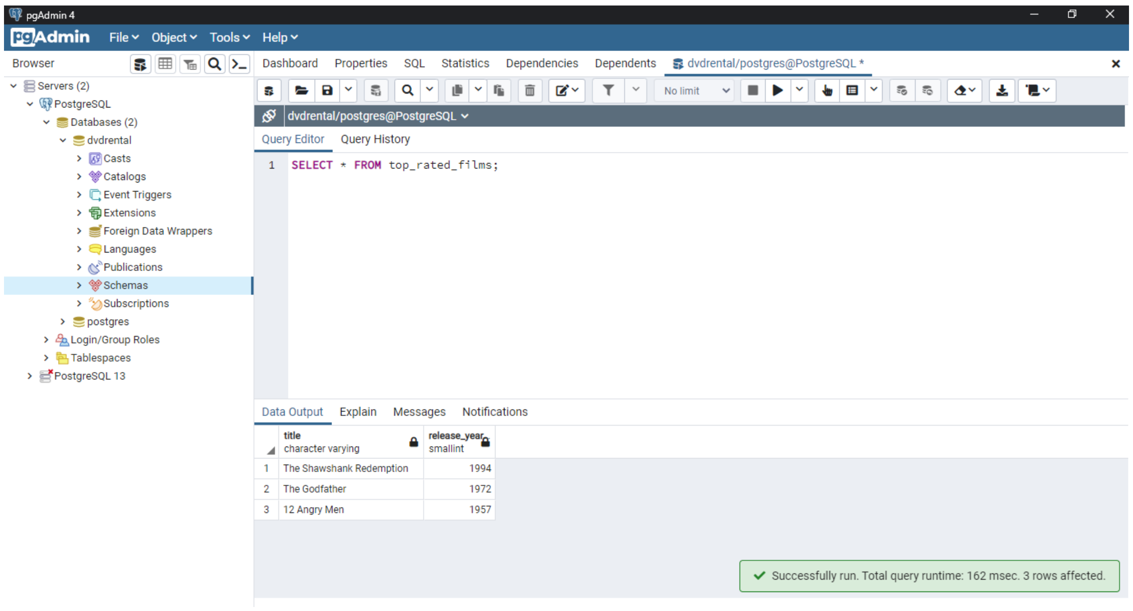
The following statements create two tables: top_rated_films and most_popular_films, and insert data into these tables:



```sql
DROP TABLE IF EXISTS top_rated_films;
CREATE TABLE top_rated_films(
    title VARCHAR NOT NULL,
    release_year SMALLINT
);

DROP TABLE IF EXISTS most_popular_films;
CREATE TABLE most_popular_films(
    title VARCHAR NOT NULL,
    release_year SMALLINT
);

INSERT INTO
   top_rated_films(title,release_year)
VALUES
   ('The Shawshank Redemption',1994),
   ('The Godfather',1972),
   ('12 Angry Men',1957);
```

```
NOTICE:  table "top_rated_films" does not exist, skipping
NOTICE:  table "most_popular_films" does not exist, skipping
INSERT 0 3

Query returned successfully in 397 msec.
```

The following shows the data from the top_rated_films table:

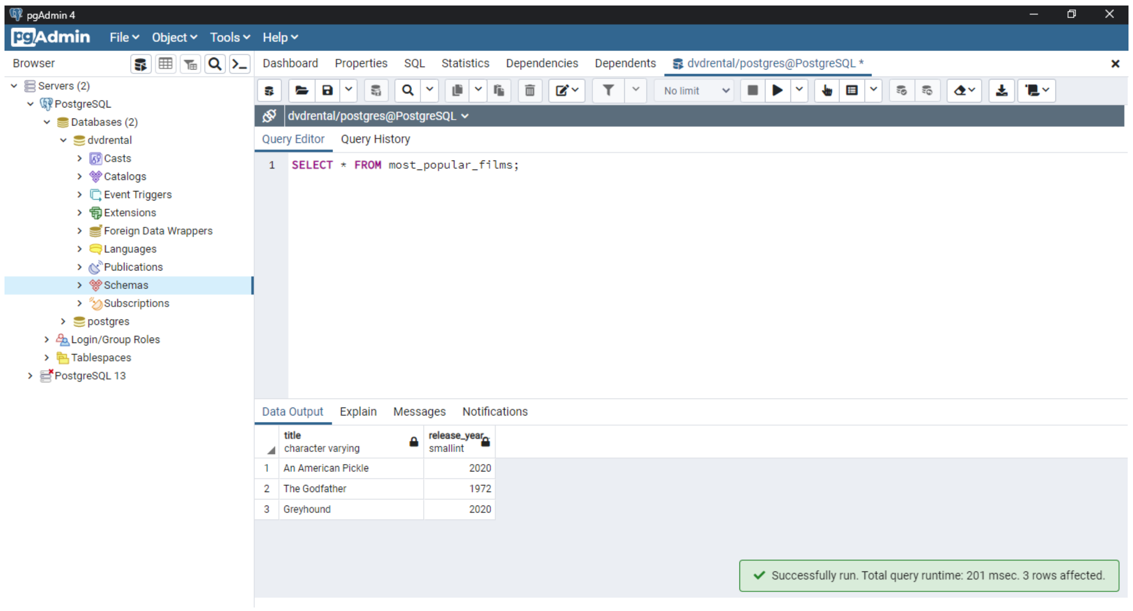


```sql
SELECT * FROM top_rated_films;
```

	title character varying	release_year smallint
1	The Shawshank Redemption	1994
2	The Godfather	1972
3	12 Angry Men	1957

Successfully run. Total query runtime: 162 msec. 3 rows affected.

The following statement returns the data from the most_popular_films table:



```sql
SELECT * FROM most_popular_films;
```

	title character varying	release_year smallint
1	An American Pickle	2020
2	The Godfather	1972
3	Greyhound	2020

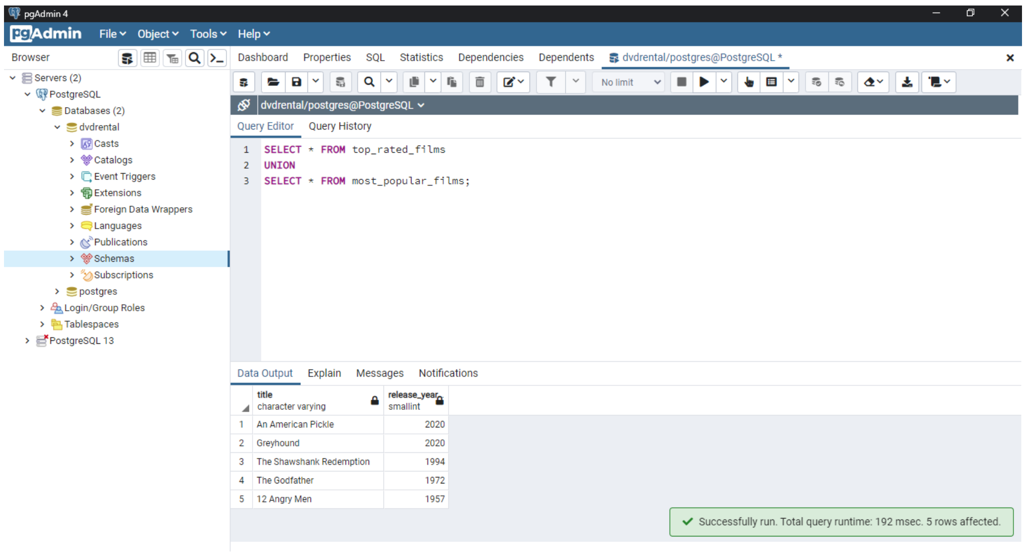
Successfully run. Total query runtime: 201 msec. 3 rows affected.

PostgreSQL UNION examples

Let's take some examples of using the PostgreSQL UNION operator.

1) Simple PostgreSQL UNION example

The following statement uses the UNION operator to combine data from both tables:



```sql
SELECT * FROM top_rated_films
UNION
SELECT * FROM most_popular_films;
```

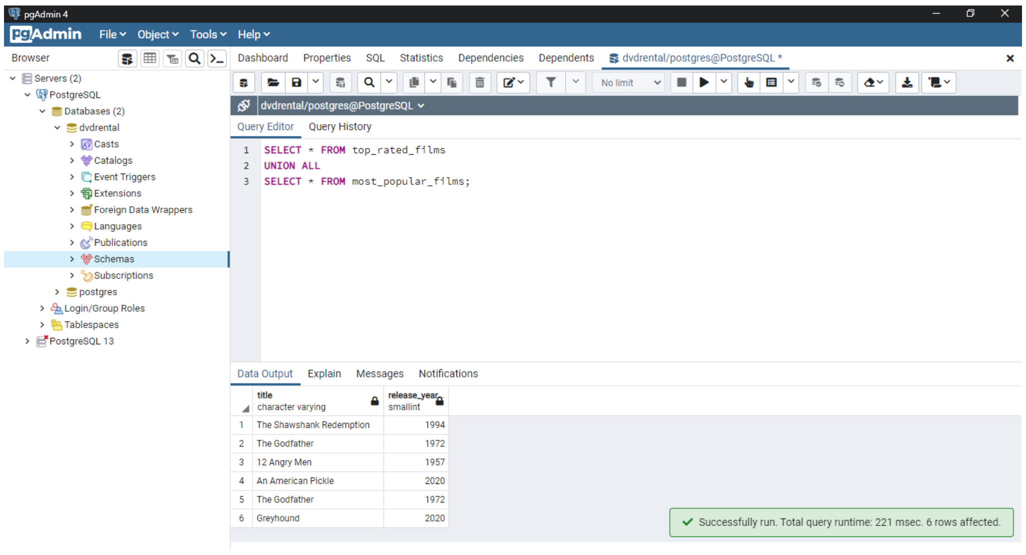
	title character varying	release_year smallint
1	An American Pickle	2020
2	Greyhound	2020
3	The Shawshank Redemption	1994
4	The Godfather	1972
5	12 Angry Men	1957

Successfully run. Total query runtime: 192 msec. 5 rows affected.

The result set includes five rows in the result set because the UNION operator removes one duplicate row.

2) PostgreSQL UNION ALL example

The following statement uses the UNION ALL operator to combine result sets from the top_rated_films and most_popular_films tables:



```sql
SELECT * FROM top_rated_films
UNION ALL
SELECT * FROM most_popular_films;
```

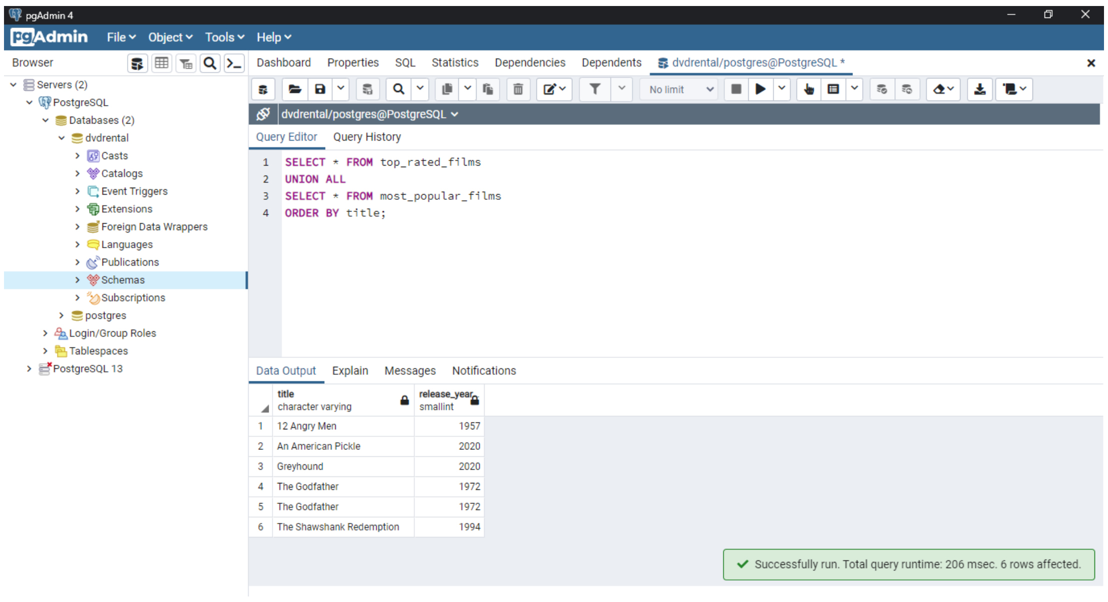
	title character varying	release_year smallint
1	The Shawshank Redemption	1994
2	The Godfather	1972
3	12 Angry Men	1957
4	An American Pickle	2020
5	The Godfather	1972
6	Greyhound	2020

Successfully run. Total query runtime: 221 msec. 6 rows affected.

In this example, the duplicate row is retained in the result set.

3) PostgreSQL UNION ALL with ORDER BY clause example

To sort the result returned by the UNION operator, you place the ORDER BY clause end of the last query like this:



```sql
SELECT * FROM top_rated_films
UNION ALL
SELECT * FROM most_popular_films
ORDER BY title;
```

	title character varying	release_year smallint
1	12 Angry Men	1957
2	An American Pickle	2020
3	Greyhound	2020
4	The Godfather	1972
5	The Godfather	1972
6	The Shawshank Redemption	1994

```
Successfully run. Total query runtime: 206 msec. 6 rows affected.
```

If you place the ORDER BY clause at the end of each query, the combined result set will not be sorted as you expected.

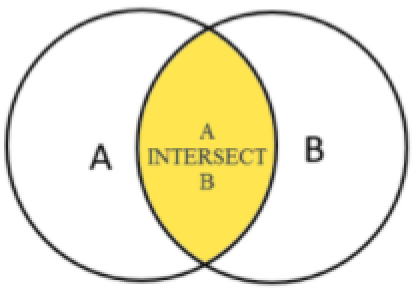
Because when UNION operator combines the sorted result sets from each query, it does not guarantee the order of rows in the final result set.

Introduction to PostgreSQL INTERSECT operator

Like the UNION and EXCEPT operators, the PostgreSQL INTERSECT operator combines result sets of two or more SELECT statements into a single result set.

The INTERSECT operator returns any rows that are available in both result sets.

The following illustration shows the final result set produced by the INTERSECT operator.



The final result set is represented by the yellow area where circle A intersects with circle B.

The following illustrates the syntax of the INTERSECT operator:

```
SELECT select_list
FROM A
INTERSECT
SELECT select_list
FROM B;
```

To use the INTERSECT operator, the columns that appear in the SELECT statements must follow the following rules:

The number of columns and their order in the SELECT clauses must be the same.

The data types of the columns must be compatible.

PostgreSQL INTERSECT with ORDER BY clause

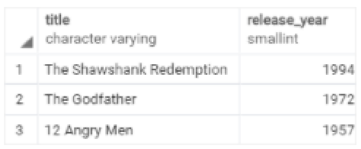
If you want to sort the result set returned by the INTERSECT operator, you place the ORDER BY at the final query in the query list like this:

```
SELECT select_list
FROM A
INTERSECT
SELECT select_list
FROM B
ORDER BY sort_expression;
```

PostgreSQL INTERSECT operator examples

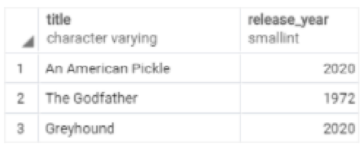
We'll use the top_rated_films and most_popular_films tables created in the UNION tutorial:

The top_rated_films table:



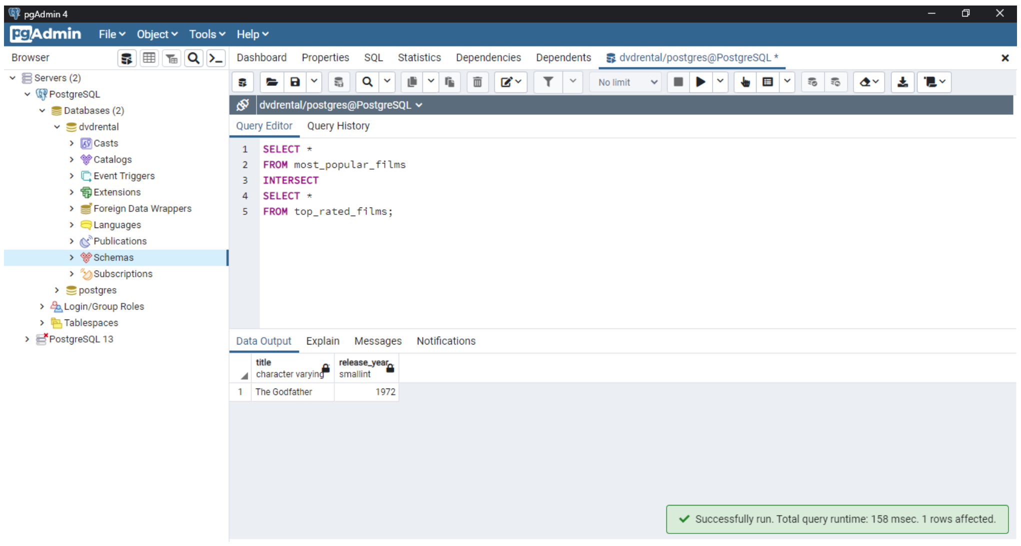
	title character varying	release_year smallint
1	The Shawshank Redemption	1994
2	The Godfather	1972
3	12 Angry Men	1957

The `most_popular_films` table:



	title character varying	release_year smallint
1	An American Pickle	2020
2	The Godfather	1972
3	Greyhound	2020

To get popular films which are also top rated films, you use the `INTERSECT` operator as follows:



```
SELECT *
FROM most_popular_films
INTERSECT
SELECT *
FROM top_rated_films;
```

```
  title               release_year
  character varying   smallint
1 The Godfather       1972
```

Successfully run. Total query runtime: 158 msec. 1 rows affected.

The result set returns one film that appears on both tables.

Introduction to the PostgreSQL EXCEPT operator

Like the UNION and INTERSECT operators, the EXCEPT operator returns rows by comparing the result sets of two or more queries.

The EXCEPT operator returns distinct rows from the first (left) query that are not in the output of the second (right) query.

The following illustrates the syntax of the EXCEPT operator.

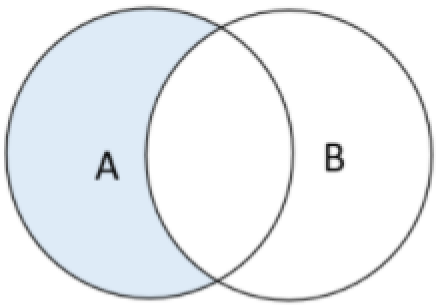
```
SELECT select_list
FROM A
EXCEPT
SELECT select_list
FROM B;
```

The queries that involve in the EXCEPT need to follow these rules:

The number of columns and their orders must be the same in the two queries.

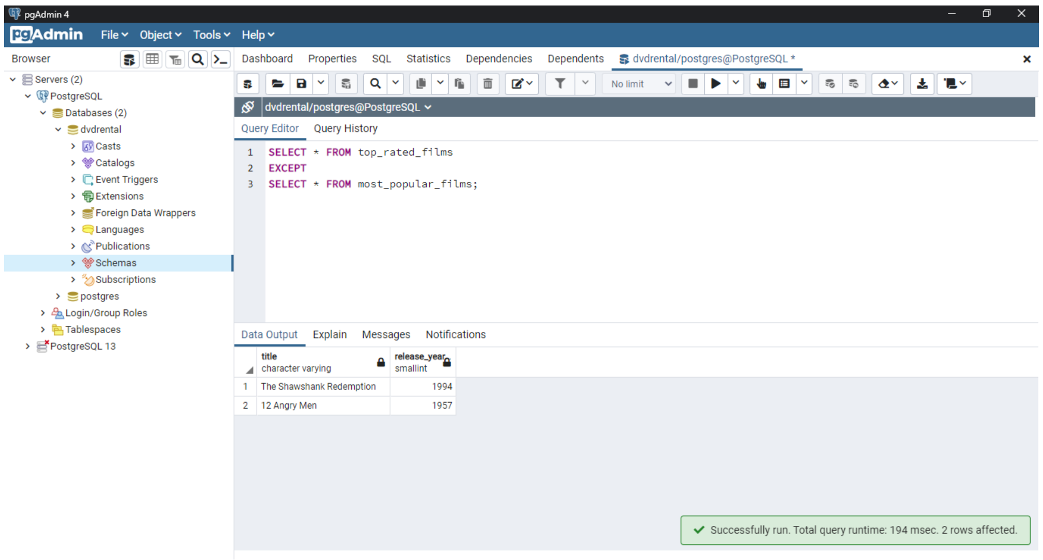
The data types of the respective columns must be compatible.

The following Venn diagram illustrates the EXCEPT operator:



PostgreSQL EXCEPT operator examples

The following statement uses the EXCEPT operator to find the top-rated films that are not popular:

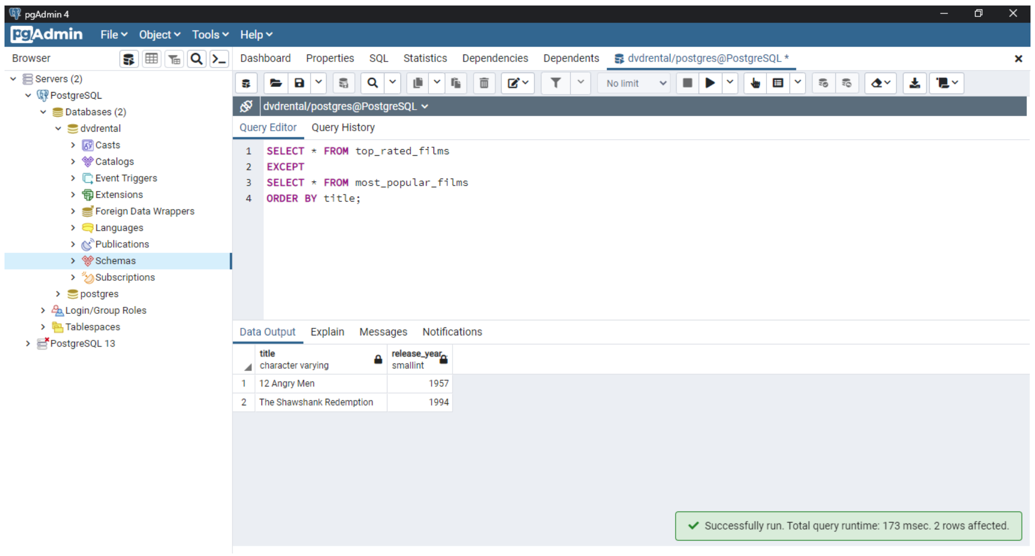


```
SELECT * FROM top_rated_films
EXCEPT
SELECT * FROM most_popular_films;
```

```
   title                      release_year
   character varying          smallint
1  The Shawshank Redemption   1994
2  12 Angry Men               1957
```

Successfully run. Total query runtime: 194 msec. 2 rows affected.

The following statement uses the ORDER BY clause in the query to sort result sets returned by the EXCEPT operator:



```
SELECT * FROM top_rated_films
EXCEPT
SELECT * FROM most_popular_films
ORDER BY title;
```

```
   title                      release_year
   character varying          smallint
1  12 Angry Men               1957
2  The Shawshank Redemption   1994
```

Successfully run. Total query runtime: 173 msec. 2 rows affected.

Notice that we placed the ORDER BY clause at the end of the statement to sort films by title.