CASE:


- There are two main ways to use a CASE statement, either a general CASE or a CASE expression.
- Both methods can lead to the same results.
- Let's first show the syntax for a "general" CASE.


- General Syntax
  - CASE

```
WHEN condition1 THEN result1
WHEN condition2 THEN result2
ELSE some_other_result
END
```

SQL

- Simple Example
  - SELECT a,
    CASE WHEN a = 1 THEN 'one'
    WHEN a = 2 THEN 'two'
    ELSE 'other'
    END
    FROM test;

| a | Case |
|---|---|
| 1 | one |
| 2 | two |



SQL

- *CASE Expression Syntax*
  - CASE expression

    WHEN value1 THEN result1

    WHEN value2 THEN result2

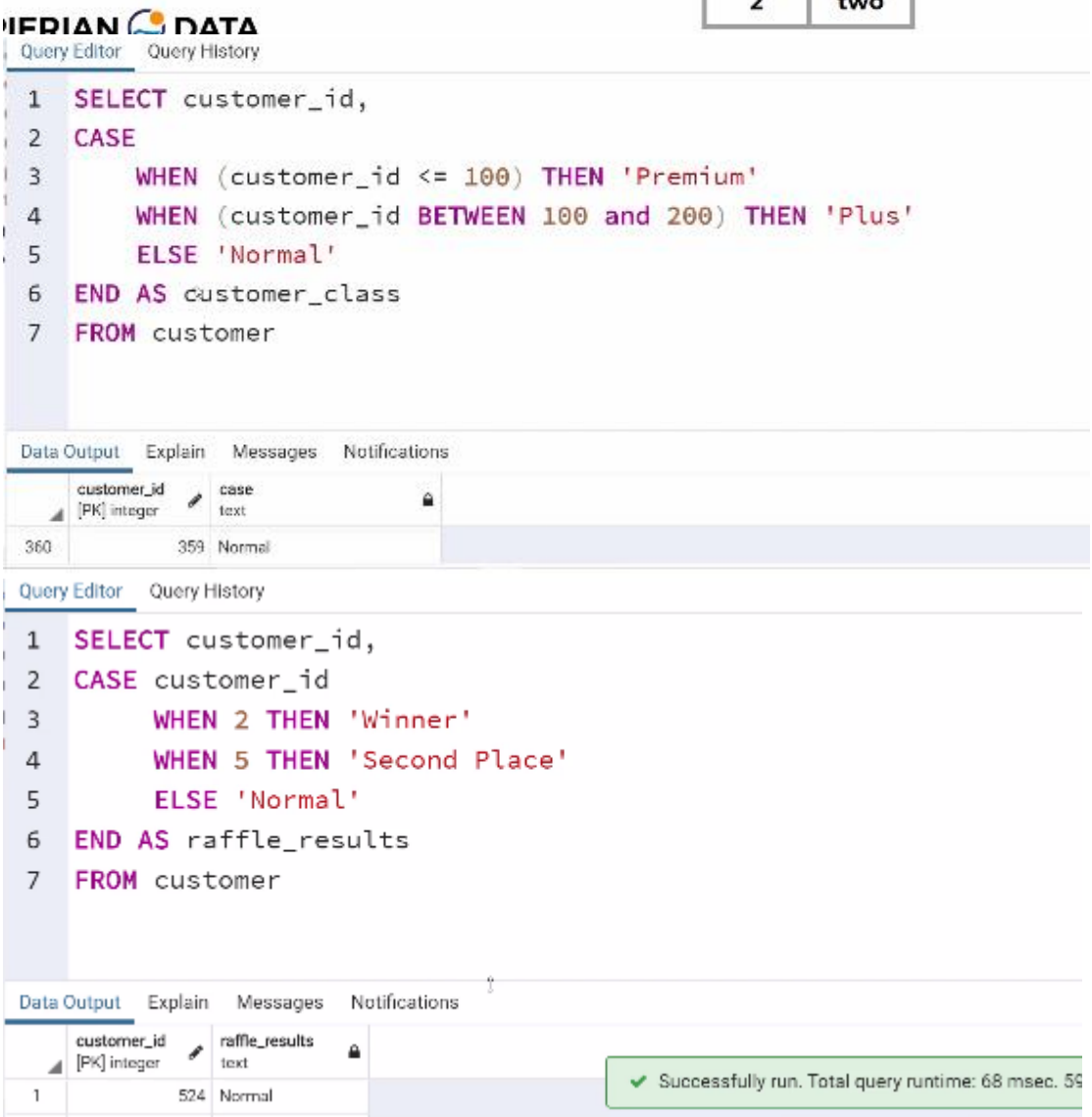
    ELSE some_other_result

    END

SQL

- Rewriting our previous example:
  - SELECT a,
    CASE a WHEN 1 THEN 'one'
    WHEN 2 THEN 'two'
    ELSE 'other'
    END
    FROM test;

| a | label |
|---|---|
| 1 | one |
| 2 | two |



dvdrental/postgres@PostgreSQL 12

Query Editor | Query History

```
1  SELECT
2  SUM(CASE rental_rate
3       WHEN 0.99 THEN 1
4       ELSE 0
5  END) AS number_of_bargains
6  FROM film
```

Data Output | Explain | Messages | Notifications

| | number_of_bargains bigint |
|---|---|
| 1 | 341 |

```
SELECT
SUM(CASE rental_rate
     WHEN 0.99 THEN 1
     ELSE 0
END) AS bargains,
SUM(CASE rental_rate
   WHEN 2.99 THEN 1
   ELSE 0
END) AS regular
```

Query Editor | Query History

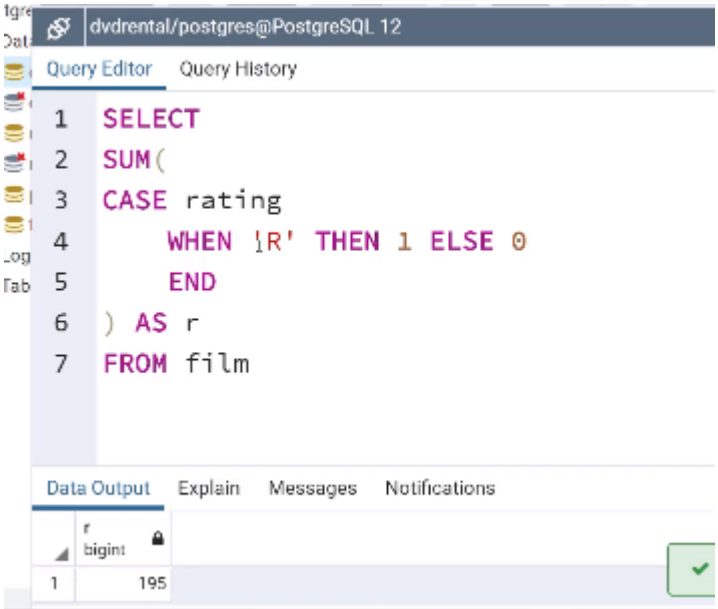
```
5  END) AS bargains,
6  SUM(CASE rental_rate
7     WHEN 2.99 THEN 1
8     ELSE 0
9  END) AS regular,
10 SUM(CASE rental_rate
11    WHEN 4.99 THEN 1
12    ELSE 0
13 END) AS premium
```

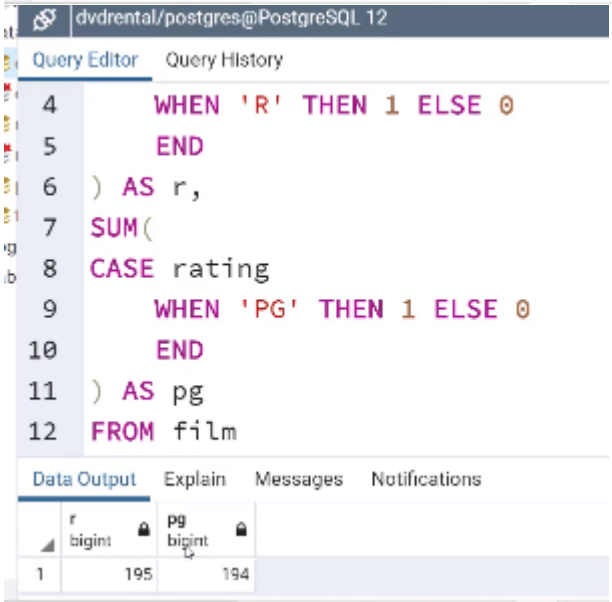
Q & A:

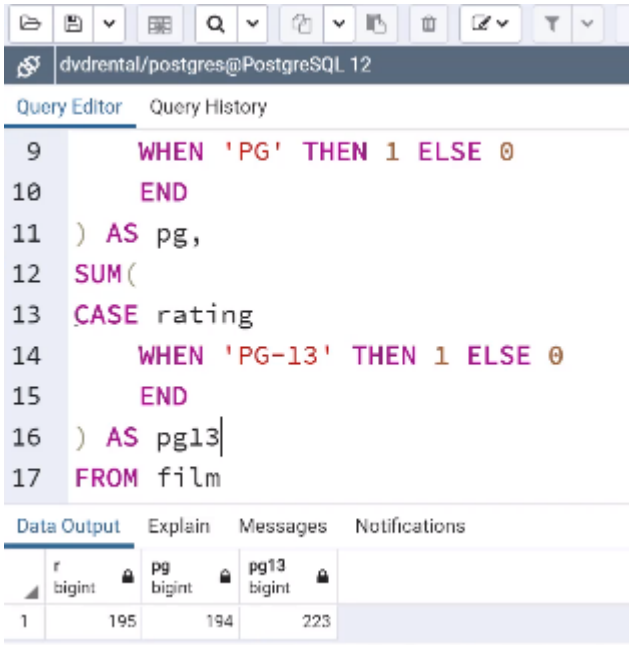
SQL

- We want to know and compare the various amounts of films we have per movie rating.
- Use CASE and the dvdrental database to re-create this table:

| r bigint | pg bigint | pg13 bigint |
|---|---|---|
| 195 | 194 | 223 |






SQL

- The COALESCE function accepts an unlimited number of arguments. It returns the first argument that is not null. If all arguments are null, the COALESCE function will return null.
  - COALESCE (arg_1,arg_2,...,arg_n)

SQL

- Example
  - SELECT COALESCE (1, 2)
    - 1
  - SELECT COALESCE(NULL, 2, 3)
    - 2


- Table of Products
  - Price and Discount in Dollars

| Item | Price | Discount |
|---|---|---|
| A | 100 | 20 |
| B | 300 | null |
| C | 200 | 10 |


- SELECT item,(price - discount) AS final FROM table

| Item | final |
|---|---|
| A | 80 |
| B | null |
| C | 190 |




SELECT item,(price - COALESCE(discount,0))

AS final FROM table

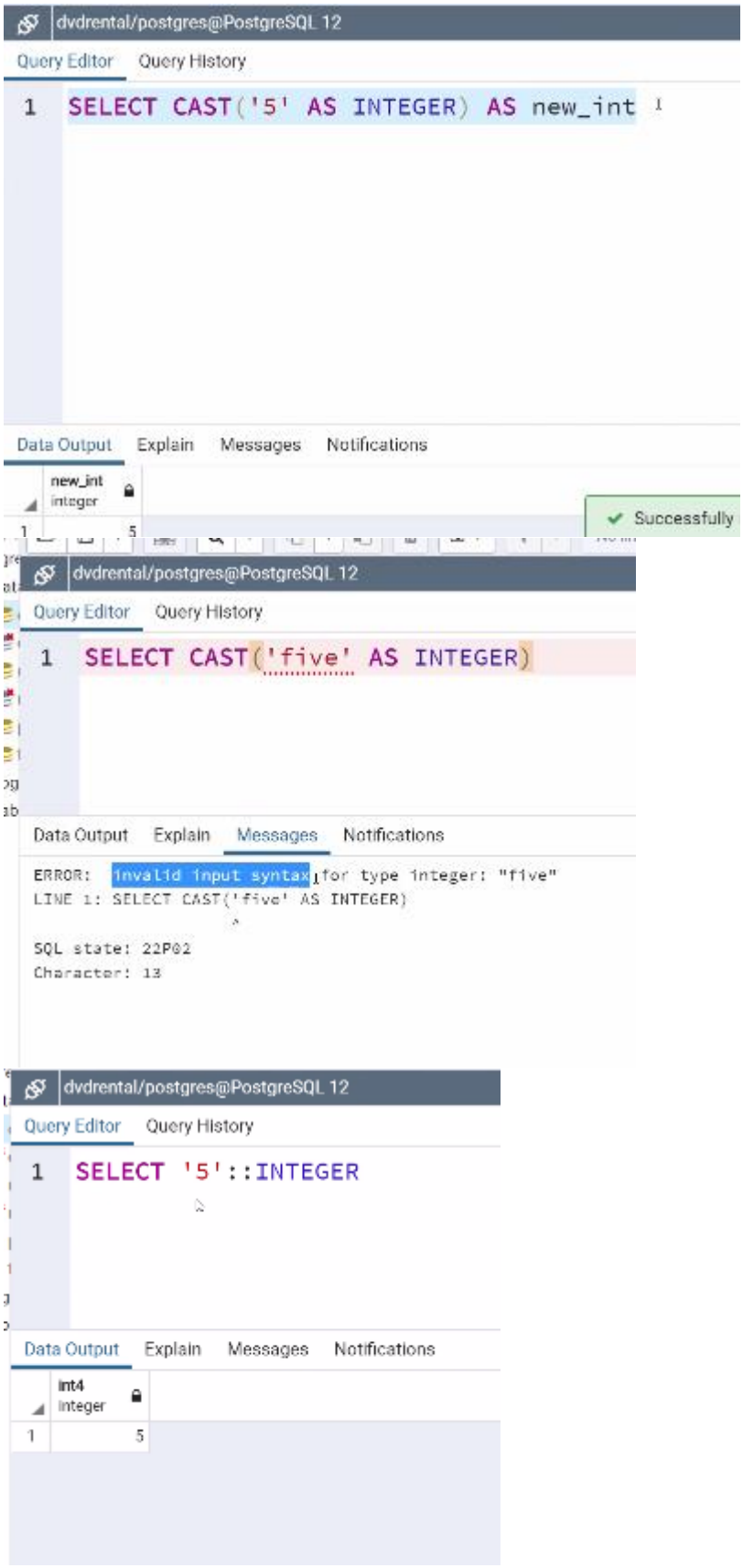
| Item | final |
|---|---|
| A | 80 |
| B | 300 |
| C | 190 |

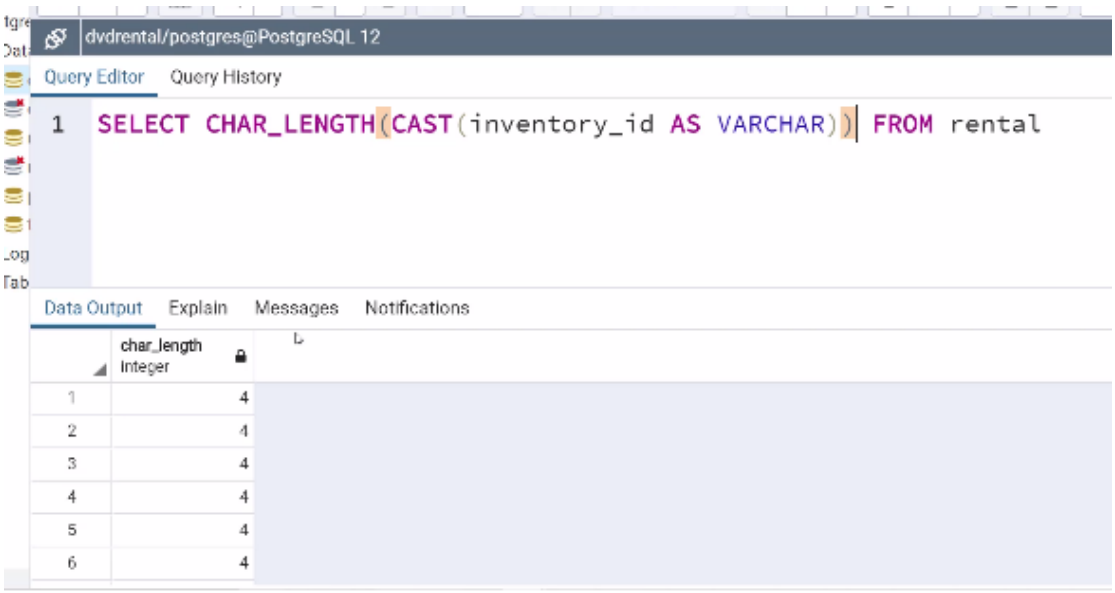

- The CAST operator let's you convert from one data type into another.
- Keep in mind not every instance of a data type can be CAST to another data type, it must be reasonable to convert the data, for example '5' to an integer will work, 'five' to an integer will not.


- Syntax for CAST function
  - SELECT CAST('5' AS INTEGER)

- PostgreSQL CAST operator
  - SELECT '5'::INTEGER


- Keep in mind you can then use this in a SELECT query with a column name instead of a single instance.
  - **SELECT CAST(date AS TIMESTAMP) FROM table**


dvdrental/postgres@PostgreSQL 12
Query Editor
Query History
1 SELECT CAST('5' AS INTEGER) AS new_int
Data Output
Explain
Messages
Notifications
new_int
integer
1 5
Successfully
dvdrental/postgres@PostgreSQL 12
Query Editor
Query History
1 SELECT CAST('five' AS INTEGER)
Data Output
Explain
Messages
Notifications
ERROR: invalid input syntax for type integer: "five"
LINE 1: SELECT CAST('five' AS INTEGER)
SQL state: 22P02
Character: 13
dvdrental/postgres@PostgreSQL 12
Query Editor
Query History
1 SELECT '5'::INTEGER
Data Output
Explain
Messages
Notifications
int4
integer
1 5





- The NULLIF function takes in 2 inputs and returns NULL if both are equal, otherwise it returns the first argument passed.
  - NULLIF(arg1,arg2)
- Example
  - NULLIF(10,10)
    - Returns NULL


- Let's jump to pgAdmin, quickly create this table and walk through solving the RATIO and why we may need NULLIF

| Name | Department |
|---|---|
| Lauren | A |
| Vinton | A |
| Claire | B |

testme/postgres@PostgreSQL 12

Query Editor | Query History

```
1  INSERT INTO depts(
2  first_name VARCHAR(50),
3  department VARCHAR(50)
4  )
```

Data Output | Explain | Messages | Notifications

```
CREATE TABLE

Query returned successfully in 62 msec.
```

testme/postgres@PostgreSQL 12

Query Editor | Query History

```
4  )
5  VALUES
6  ('Vinton','A'),
7  ('Lauren','A'),
8  ('Claire','B')
```

Data Output | Explain | Messages | Notifications

```
CREATE TABLE

Query returned successfully in 62 msec.
```

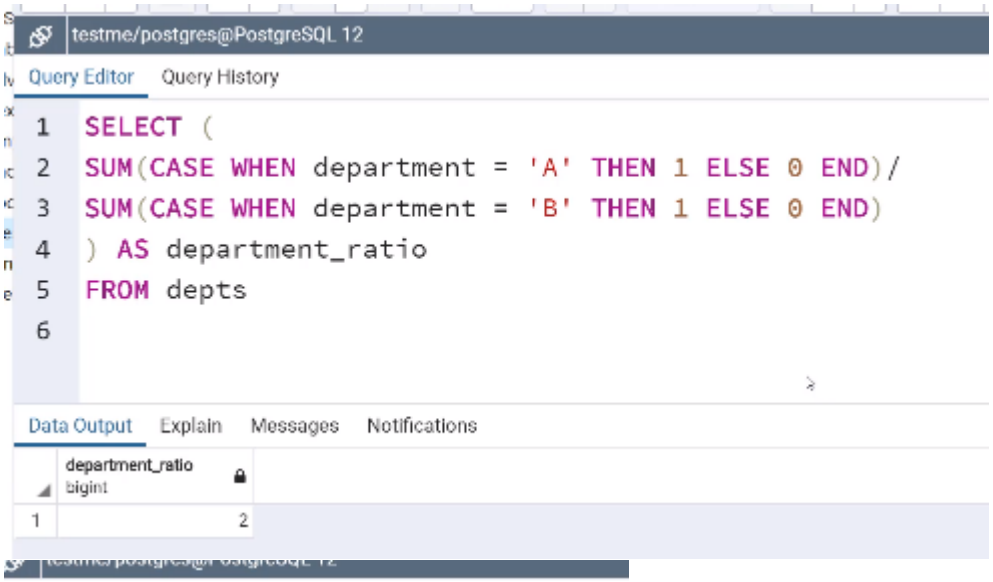
testme/postgres@PostgreSQL 12

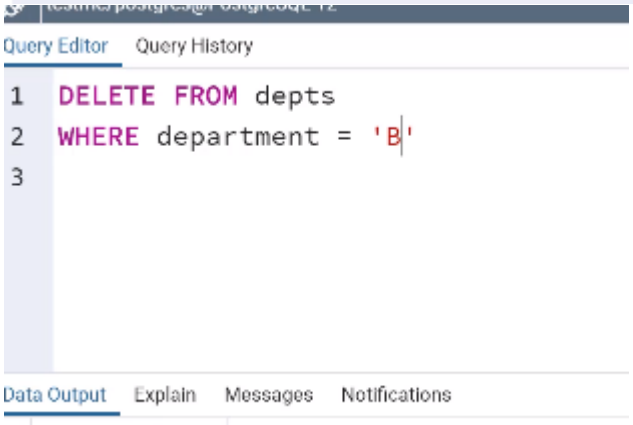
Query Editor | Query History

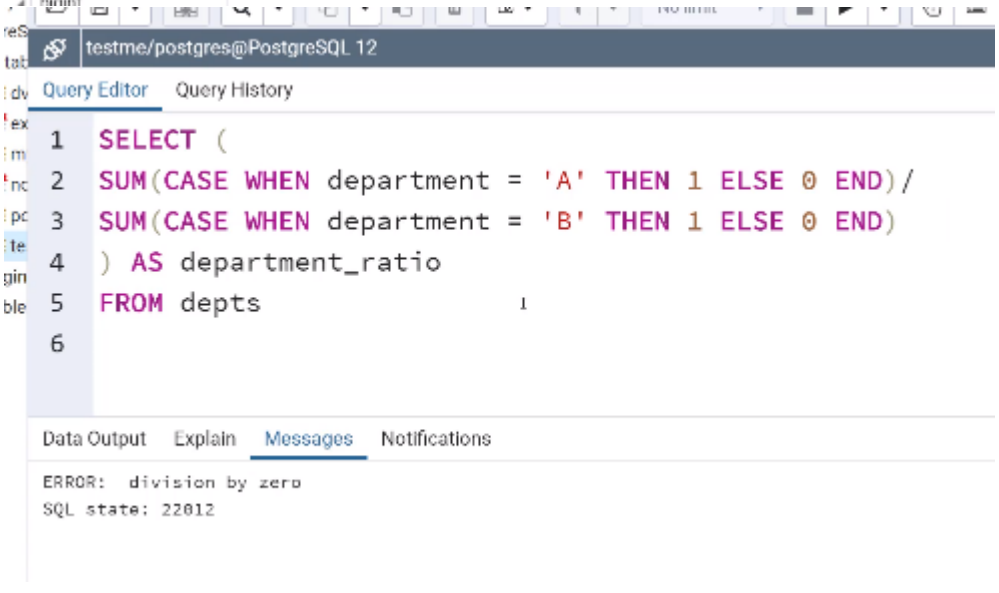
```
1  SELECT * FROM depts
2
```

Data Output | Explain | Messages | Notifications

| | first_name character varying (50) | department character varying (50) |
|---|---|---|
| 1 | Vinton | A |
| 2 | Lauren | A |
| 3 | Claire | B |


testme/postgres@PostgreSQL 12
Query Editor
Query History
1 SELECT (
2 SUM(CASE WHEN department = 'A' THEN 1 ELSE 0 END)/
3 SUM(CASE WHEN department = 'B' THEN 1 ELSE 0 END)
4 ) AS department_ratio
5 FROM depts
6
Data Output
Explain
Messages
Notifications
department_ratio
bigint
1
2


Query Editor
Query History
1 DELETE FROM depts
2 WHERE department = 'B'
3
Data Output
Explain
Messages
Notifications
department_ratio
bigint


testme/postgres@PostgreSQL 12
Query Editor
Query History
1 SELECT (
2 SUM(CASE WHEN department = 'A' THEN 1 ELSE 0 END)/
3 SUM(CASE WHEN department = 'B' THEN 1 ELSE 0 END)
4 ) AS department_ratio
5 FROM depts
6
Data Output
Explain
Messages
Notifications
ERROR: division by zero
SQL state: 22012

testme/postgres@PostgreSQL 12

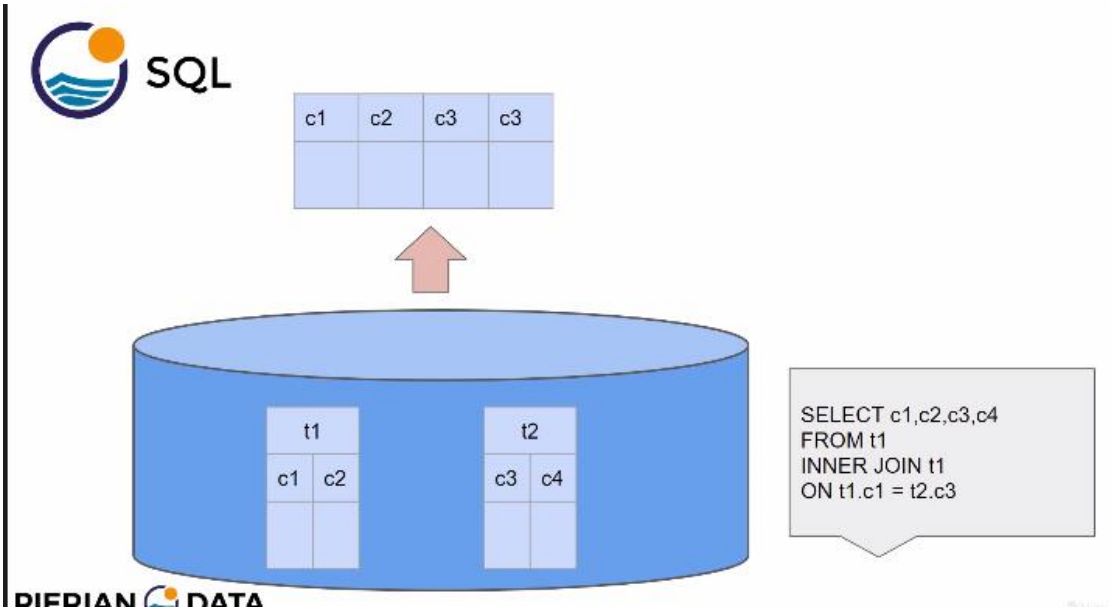
Query Editor | Query History

```
1  SELECT (
2  SUM(CASE WHEN department = 'A' THEN 1 ELSE 0 END)/
3  NULLIF(SUM(CASE WHEN department = 'B' THEN 1 ELSE 0 END),0)
4
5  ) AS department_ratio
6  FROM depts
7
```

Data Output | Explain | Messages | Notifications

| | department_ratio bigint |
|---|---|
| 1 | [null] |

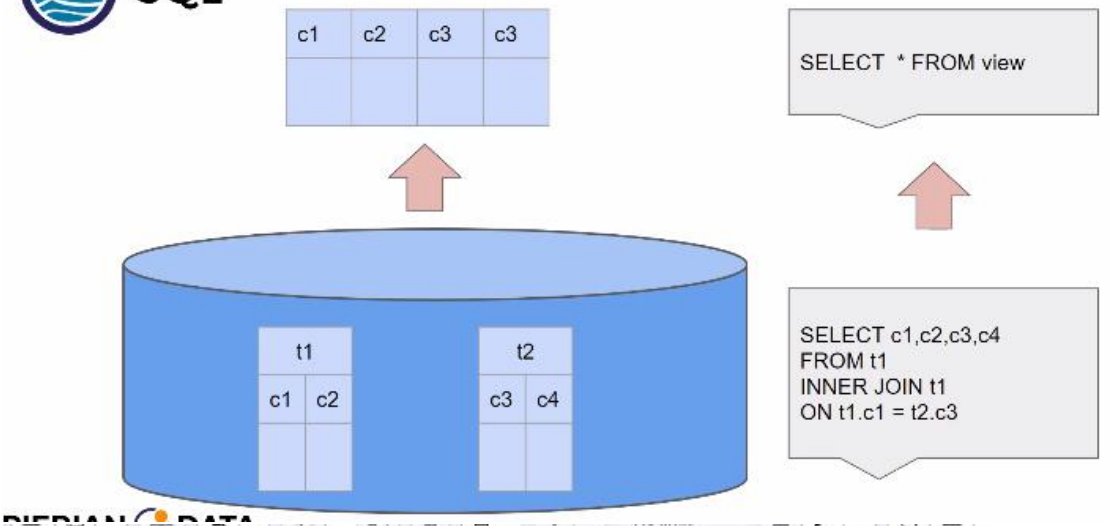

- Often there are specific combinations of tables and conditions that you find yourself using quite often for a project.
- Instead of having to perform the same query over and over again as a starting point, you can create a VIEW to quickly see this query with a simple call.



SQL

- A view is a database object that is of a stored query.
- A view can be accessed as a virtual table in PostgreSQL.
- Notice that a view does not store data physically, it simply stores the query.

SQL




dvdrental/postgres@PostgreSQL 12

Query Editor | Query History

```
1  SELECT first_name,last_name,address FROM customer
2  INNER JOIN address
3  ON customer.address_id = address.address_id
```

Data Output | Explain | Messages | Notifications

| | first_name character varying (45) | last_name character varying (45) | address character varying (50) |
|---|---|---|---|
| 1 | Jared | Ely | 1003 Qinhuangdao Street |
| 2 | Mary | Smith | 1913 Hanoi Way |
| 3 | Patricia | Johnson | 1121 Loja Avenue |
| 4 | Linda | Williams | 692 Joliet Street |
| 5 | Barbara | Jones | 1566 Inegl Manor |

dvdrental/postgres@PostgreSQL 12

Query Editor | Query History

```
1  CREATE VIEW customer_info AS
2  SELECT first_name,last_name,address FROM customer
3  INNER JOIN address
4  ON customer.address_id = address.address_id
```

Data Output | Explain | Messages | Notifications

```
CREATE VIEW

Query returned successfully in 48 msec.
```

dvdrental/postgres@PostgreSQL 12

Query Editor | Query History

```
1  SELECT * FROM customer_info
```

Data Output | Explain | Messages | Notifications

| | first_name character varying (45) | last_name character varying (45) | address character varying (50) |
|---|---|---|---|
| 1 | Jared | Ely | 1003 Qinhuangdao Street |
| 2 | Mary | Smith | 1913 Hanoi Way |
| 3 | Patricia | Johnson | 1121 Loja Avenue |
| 4 | Linda | Williams | 692 |
| 5 | Barbara | Jones | 150 |

Successfully run. Total quer

dvdrental/postgres@PostgreSQL 12

Query Editor | Query History

```
1  CREATE OR REPLACE VIEW customer_info AS
2  SELECT first_name,last_name,address,district FROM customer
3  INNER JOIN address
4  ON customer.address_id = address.address_id
```

Data Output | Explain | Messages | Notifications

```
CREATE VIEW

Query returned successfully in 59 msec.
```

dvdrental/postgres@PostgreSQL 12

Query Editor | Query History

```
1  DROP VIEW IF EXISTS customer_info
```

Data Output | Explain | Messages | Notifications

| | first_name character varying (45) | last_name character varying (45) | address character varying (50) | district character varying (20) |
|---|---|---|---|---|
| 1 | Jared | Ely | 1003 Qinhuangdao Street | West Java |
| 2 | Mary | Smith | 1913 Hanoi Way | Nagasaki |
| 3 | Patricia | Johnson | 1121 Loja Avenue | California |
| 4 | Linda | Williams | 692 Joliet Street | Attika |
| 5 | Barbara | Jones | 1566 Inegl Manor | Mandalay |

dvdrental/postgres@PostgreSQL 12

Query Editor | Query History

```
1  ALTER VIEW customer_info RENAME to c_info
```

Data Output | Explain | Messages | Notifications

```
ALTER VIEW

Query returned successfully in 48 msec.
```

Dashboard | Properties | SQL | Statistics | Dependencies | Dependents

dvdrental/postgres@PostgreSQL 12

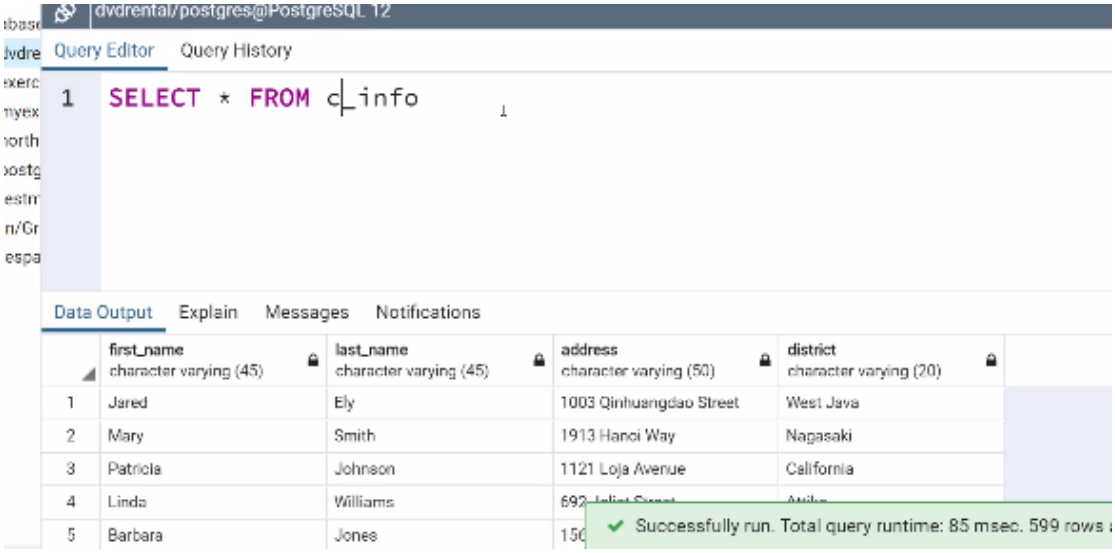
Query Editor | Query History

```
1  SELECT * FROM customer_info
```

Data Output | Explain | Messages | Notifications

```
ERROR:  relation "customer_info" does not exist
LINE 1: SELECT * FROM customer_info
                      ^

SQL state: 42P01
Character: 15
```


dvdrental/postgres@PostgreSQL 12
Query Editor
Query History
1 SELECT * FROM c_info
Data Output
Explain
Messages
Notifications
first_name character varying (45)
last_name character varying (45)
address character varying (50)
district character varying (20)
1 Jared Ely 1003 Qinhuangdao Street West Java
2 Mary Smith 1913 Hanoi Way Nagasaki
3 Patricia Johnson 1121 Loja Avenue California
4 Linda Williams
5 Barbara Jones
Successfully run. Total query runtime: 85 msec. 599 rows