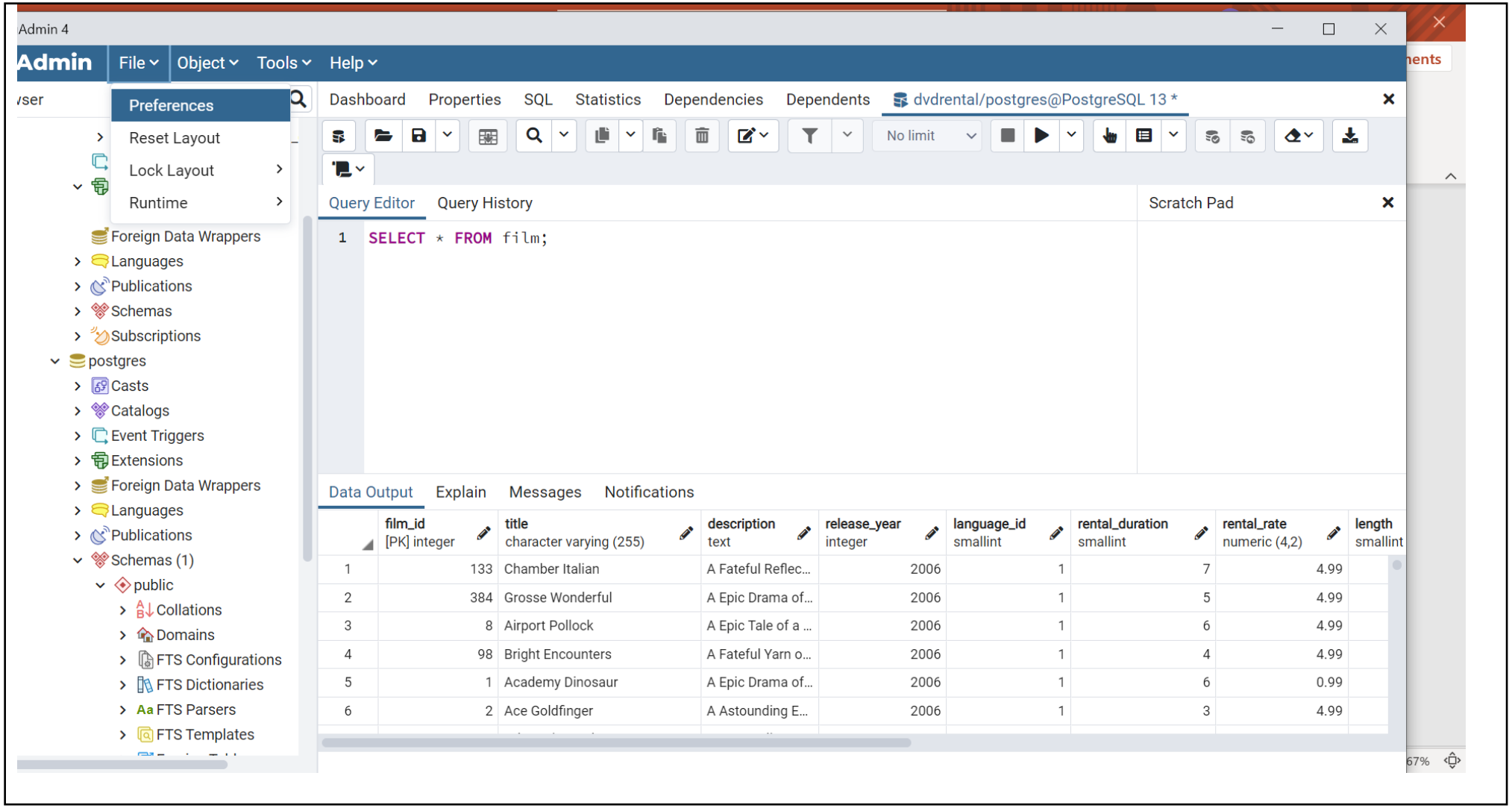
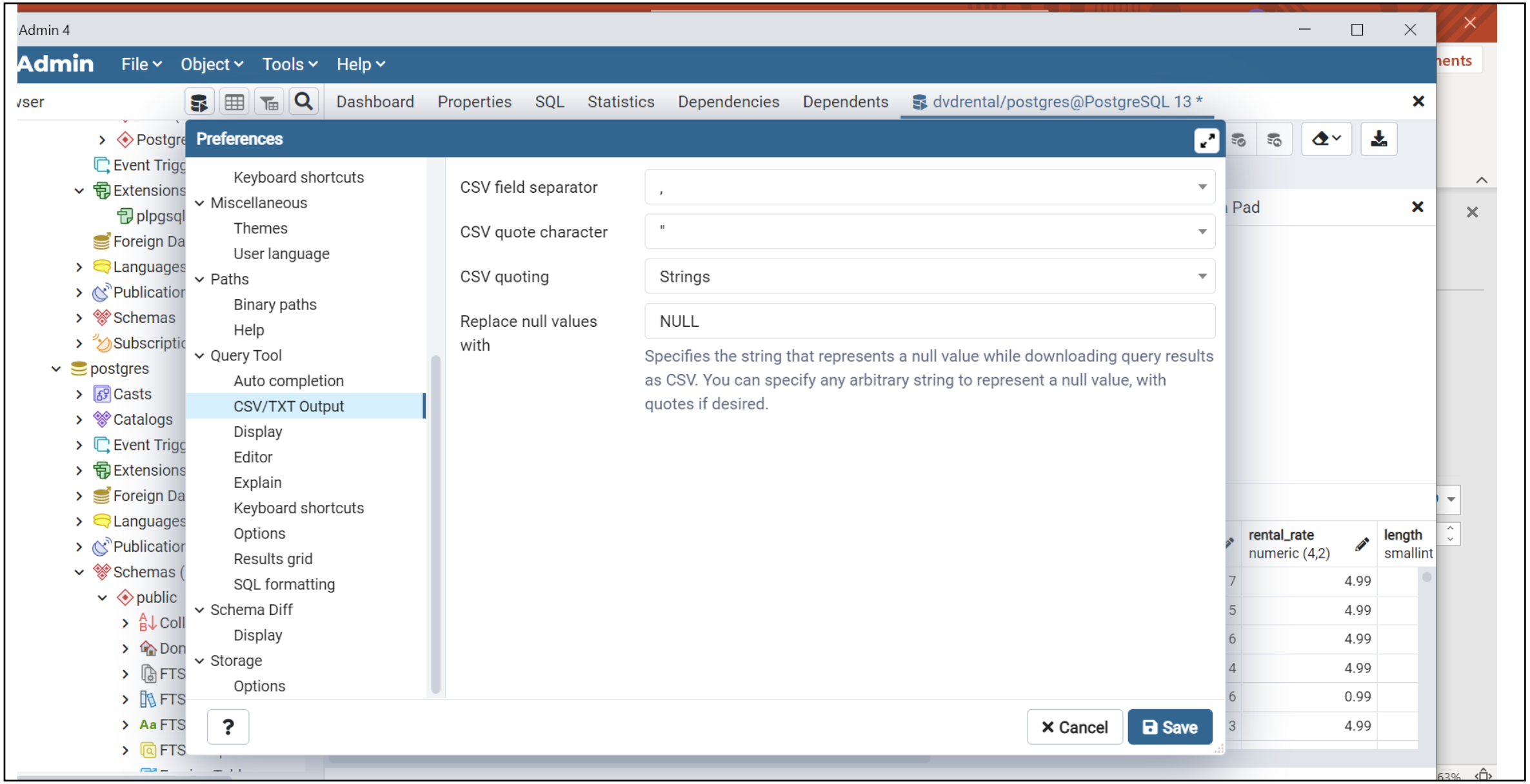
pgAdmin Overview



PREFERENCES



SQL Statement Fundamentals

- This section focuses on SQL syntax
- The syntax you learn here can be applied to any major type of SQL Database (MySQL, Oracle, etc…

https://www.tutorialspoint.com/sql/sql-drop-table.htm

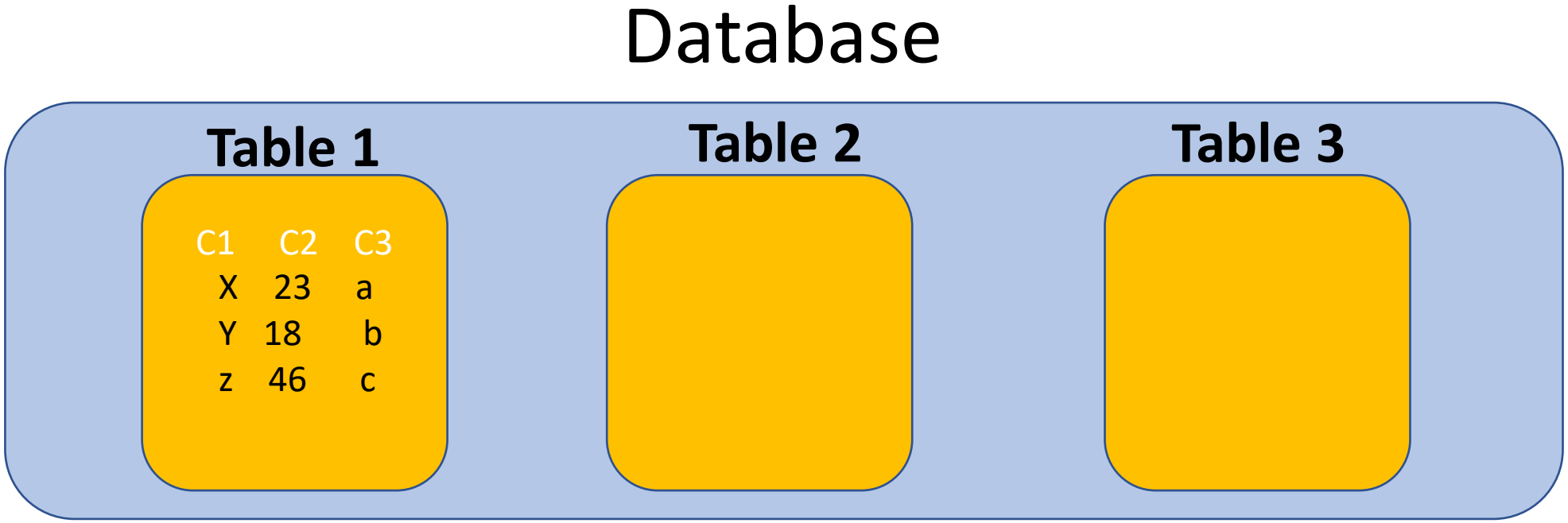
SELECT is the most common statement used, and it allows us to retrieve information from a table.

Later on we will learn how to combine **SELECT** with other statements to perform more complex queries.

<u>Example syntax for **SELECT** statement</u>:

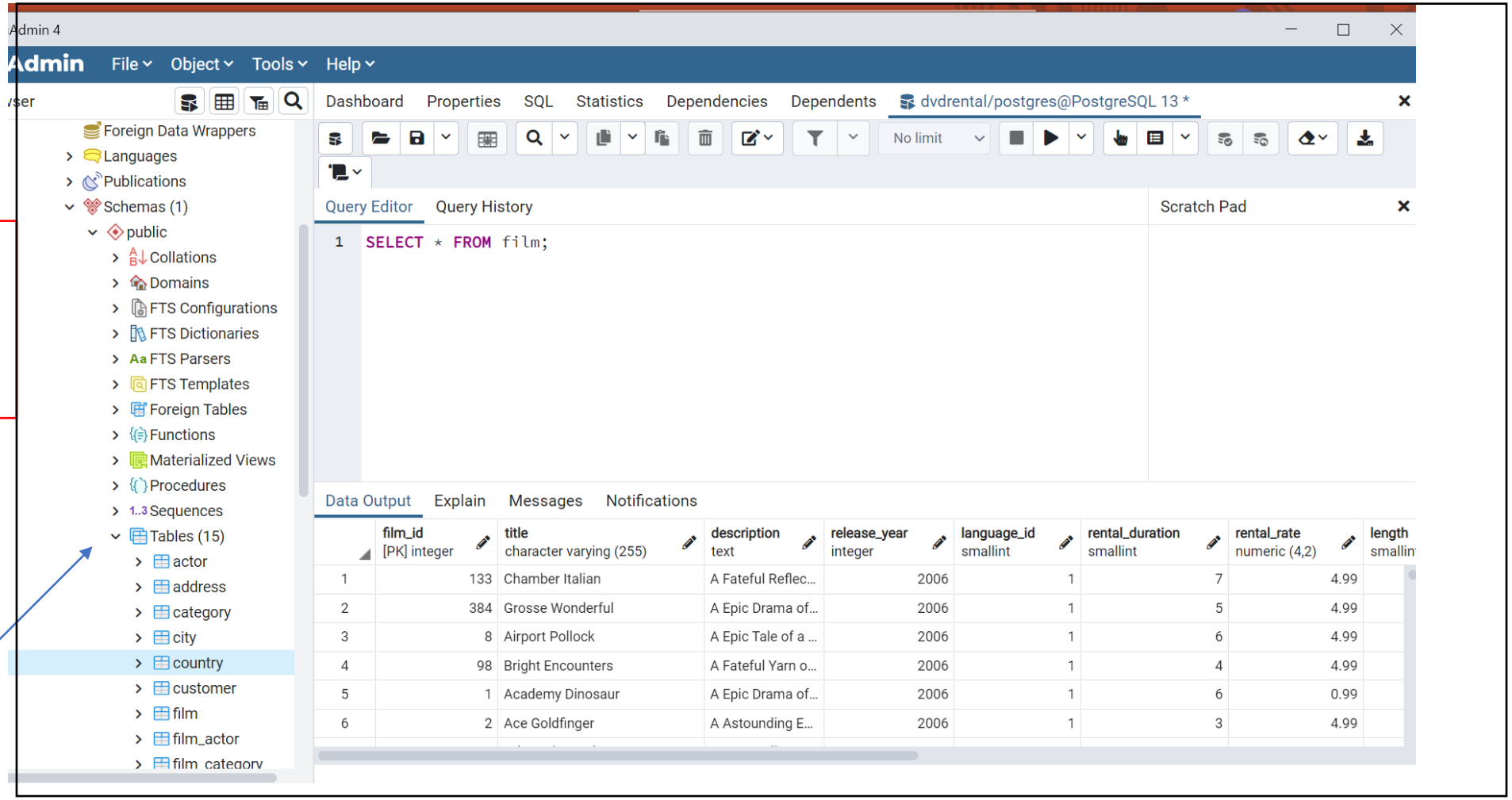
SELECT column_name FROM table_name

SELECT C1, C3 FROM Table 1;



SELECT CLAUSE

Look at the tables which are inside our database



The **DISTINCT** keyword operates on a column. The syntax looks like this:

SELECT DISTINCT column **FROM** table

Name	Choice
Zach	Green
David	Green
Claire	Yellow
David	Red

SELECT DISTINCT NAME **FROM** Table

Name
Zach
David
Claire

The `SELECT DISTINCT` statement is used to return only distinct (different) values

Some syntaxes to try

```
SELECT * FROM actor;
SELECT first_name, last_name FROM actor;

SELECT * FROM film;
SELECT DISTINCT (release_year) FROM film;
SELECT DISTINCT (rental_rate) FROM film;
```

- The COUNT function return the number of input rows that match a specific condition of a query.
- We can apply COUNT on a specific column or just pass COUNT (*), we will soon see this return the same result.
- COUNT is much more useful when is combined with other commands, such as DISTINCT

```
SELECT COUNT (DISTINCT name) FROM table;

SELECT COUNT (*) FROM payment;
SELECT COUNT (amount) FROM payment;
SELECT DISTINCT (amount) FROM payment;
SELECT COUNT (DISTINCT amount) FROM payment;
```

SELECT WHERE

- SELECT and WHERE are the most fundamental SQL statements and you will find yourself using them often!
- The WHERE statement allows us to specify conditions on columns for the rows to be returned.

<u>Basic syntax example:</u>

```
SELECT column1, column2
FROM TABLE
WHERE conditions;
```

- PostgreSQL provides a variety of standard operators to construct the conditions.

Comparison Operators

- Compare a column value to something

❑Is the price greater than $3.00?

❑Is the pet's name equal to "Sam"?

SQL Comparison Operators

Operator	Description
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
<>	Not equal to

Logical Operators:

Logical operators in SQL will return either true or false value.

Operators	Function	Example
AND	Check two conditions are true	Select * from emp where basic >= 10000 AND basic <= 20000;
OR	Check any of the two conditions are true	Select * from emp where basic >= 10000 OR dept = 'Sales';
NOT	Reversed the result of logical expression	Select * from emp where NOT(basic >= 10000 OR dept = 'Sales');

```
SELECT * FROM customer;

SELECT * FROM customer
WHERE first_name ='Jared’;

SELECT * FROM film
WHERE rental_rate > 4;

SELECT * FROM film
WHERE rental_rate > 4 AND replacement_cost >=19.99;


SELECT * FROM film
WHERE rental_rate > 4 AND replacement_cost >=19.99 AND rating ='R’;

SELECT COUNT(*)  FROM film
WHERE rental_rate > 4 AND replacement_cost >=19.99 AND rating =‘R’;
```

Result:34

```
SELECT COUNT(*) FROM film
WHERE rental_rate > 4 OR rating ='PG-13’;
```

Result: 482

ORDER BY

You can use ORDER BY to sort rows based on a column value in either ascending or descending order.

- Basic syntax for ORDER BY

```
SELECT column_1, column_2
  FROM table
 ORDER BY column_1 ASC/DESC
```

If you leave it blank , ORDER BY uses ASC by default

```
SELECT store_id,first_name,last_name
FROM customer
ORDER BY store_id DESC, first_name ASC
LIMT 5;
```

We can use LIMT as the number of rows we wish to print;

GROUP BY

- Group BY will allow us to aggregate data and apply functions to better understand how data is distributed per category
- SQL provides a variety of aggregate function

- Most Common Aggregate Functions:
  - AVG()- returns average value
  - COUNT()- returns number of values
  - MAX()= returns maximum value
  - MIN()-returns minimum value
  - SUM()-returns the sum of all values

- MIN and MAX functions

```
SELECT MIN(replacement_cost) FROM film;
SELECT MAX(replacement_cost) FROM film

SELECT MIN(replacement_cost),
MAX(replacement_cost) FROM film;
```

- COUNT function

```
SELECT COUNT(film_id) FROM film;
```

- AVG function

```
SELECT ROUND(AVG(replacement_cost),2) FROM film;
```

- SUM function

```
SELECT SUM(replacement_cost) FROM film;
```

```
SELECT customer_id, COUNT(amount) FROM payment
GROUP BY customer_ID
ORDER BY COUNT(amount) DESC
```

```
SELECT staff_id,customer_id, SUM(amount) FROM payment
GROUP BY staff_id,customer_ID
ORDER BY staff_id,customer_id
```

```
SELECT DATE(payment_date), SUM(amount) FROM payment
GROUP BY DATE(payment_date)
ORDER BY SUM(amount)
```

- HAVING

```
SELECT store_id, COUNT(*) FROM customer
GROUP BY store_id
HAVING COUNT(*) >300
```

```
SELECT store_id, SUM(*) FROM customer
GROUP BY store_id
HAVING SUM(*) >300
```

In this section we will discuss how to combine tables in our database with the use of JOIN statements! Sometime JOINS can be tricky for beginners, I mention throughout the next lectures Venn Diagrams for JOINs, so if you want more resources after reviewing the lectures, check out the following helpful links:

SQL JOINS Explained with Venn Diagrams

SQL JOIN Examples

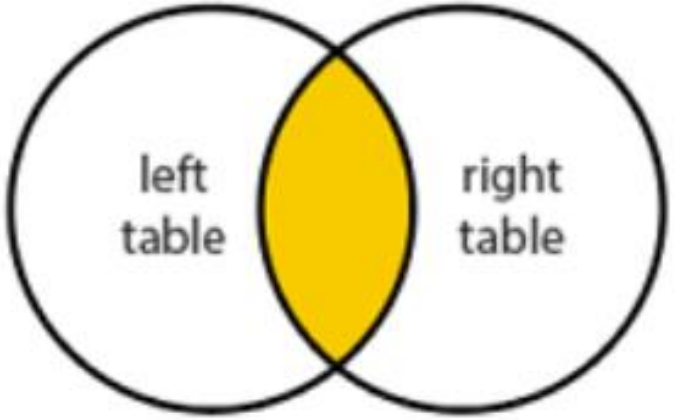
Wikipedia Page on SQL JOINS

Four different types of JOINs

1.(INNER) JOIN: Select records that have matching values in both tables.
2.FULL (OUTER) JOIN: Selects all records that match either left or right table records.
3.LEFT (OUTER) JOIN: Select records from the first (left-most) table with matching right table records.
4.RIGHT (OUTER) JOIN: Select records from the second (right-most) table with matching left table records.

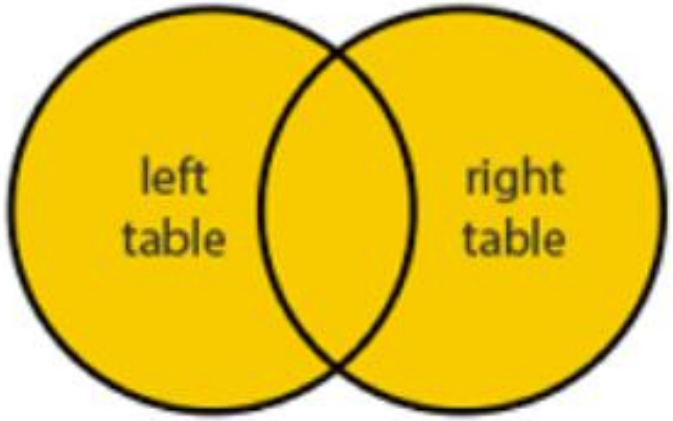
Select records that have matching values in both tables

INNER JOIN



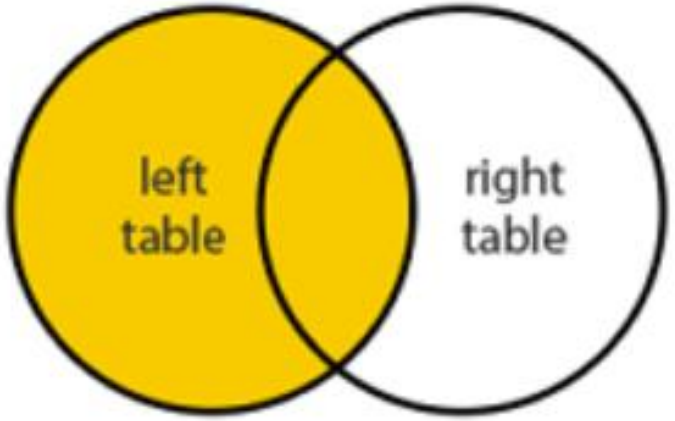
Selects all records that match either left or right table records

FULL JOIN



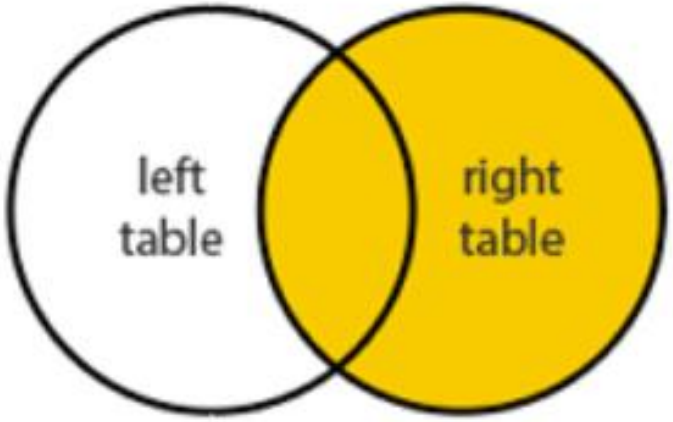
Select records from the first (left-most) table with matching right table records.

LEFT JOIN



Select records from the second (right-most) table with matching left table records.

RIGHT JOIN



https://www.dofactory.com/sql/join

INNER JOIN

```
SELECT *  FROM TABLE A
INNER JOIN TABLE B
ON TABLEA.col_match=TABLEB.col_match
```

```
SELECT * FROM payment
INNER JOIN customer
ON payment.customer_id=customer.customer_id
```

LEFT JOIN

```
SELECT *  FROM TABLE A
LEFT OUTER JOIN TABLE B
ON TABLEA.col_match=TABLEB.col_match
```

```
SELECT film.film_id,title, inventory_id,store_id
FROM film
LEFT JOIN inventory
ON inventory.film_id=film.film_id
```

FULL JOIN

```
SELECT * FROM TABLEA
FULL OUTER JOIN TABLEB
ON TABLEA.col_match=TABLEB.col_match
```

```
SELECT * FROM payment
INNER JOIN customer
ON payment.customer_id=customer.customer_id
```

RIGHT JOIN

```
SELECT *  FROM TABLE A
RIGHT OUTER JOIN TABLE B
ON TABLEA.col_match=TABLEB.col_match
```

```
SELECT *  FROM TABLE A
LEFT JOIN TABLE B
ON TABLEA.col_match=TABLEB.col_match
```

CREATING DATABASES AND TABLES

Section Overview

- Data Types
- Primary and Foreign Keys
- Constrains
- CREATE
- INSERT
- DELETE, ALTER, DROP

- Boolean
  - True or False
- Character
  - Char, varchar, and text
- Numeric
  - Integer and floating-point number
- Temporal
  - Date, time, timestamp, and interval

- A primary key is a column, or a group of columns used to identify a row uniquely in a table
- For example, in our dvdrental database we saw customers had a unique, non-null customer_id column as their primary key.

- A foreign key is a field or group of fields in a table that uniquely identifies a row in another table
- A foreign key is defined in a table that references to the primary key of the another table.

- <u>Full General Synatx</u>:

  - CREATE TABLE table_name (
    column_name TYPE column_constraint,
    column_name TYPE column_constraint,
    Table_constraint table_constraint
) INHERITS existing_table_name;

CREATE DATABASE

pgAdmin 4

pgAdmin File Object Tools Help

Browser

Servers (1)
PostgreSQL 13
Databases (2)
Login/Gro
Tablespa
pg_de
pg_global

Create
Refresh...
Database...

Dashboard Properties SQL Statistics Dependencies Dependents dvdrental/postgres@PostgreSQL 13 *

No limit

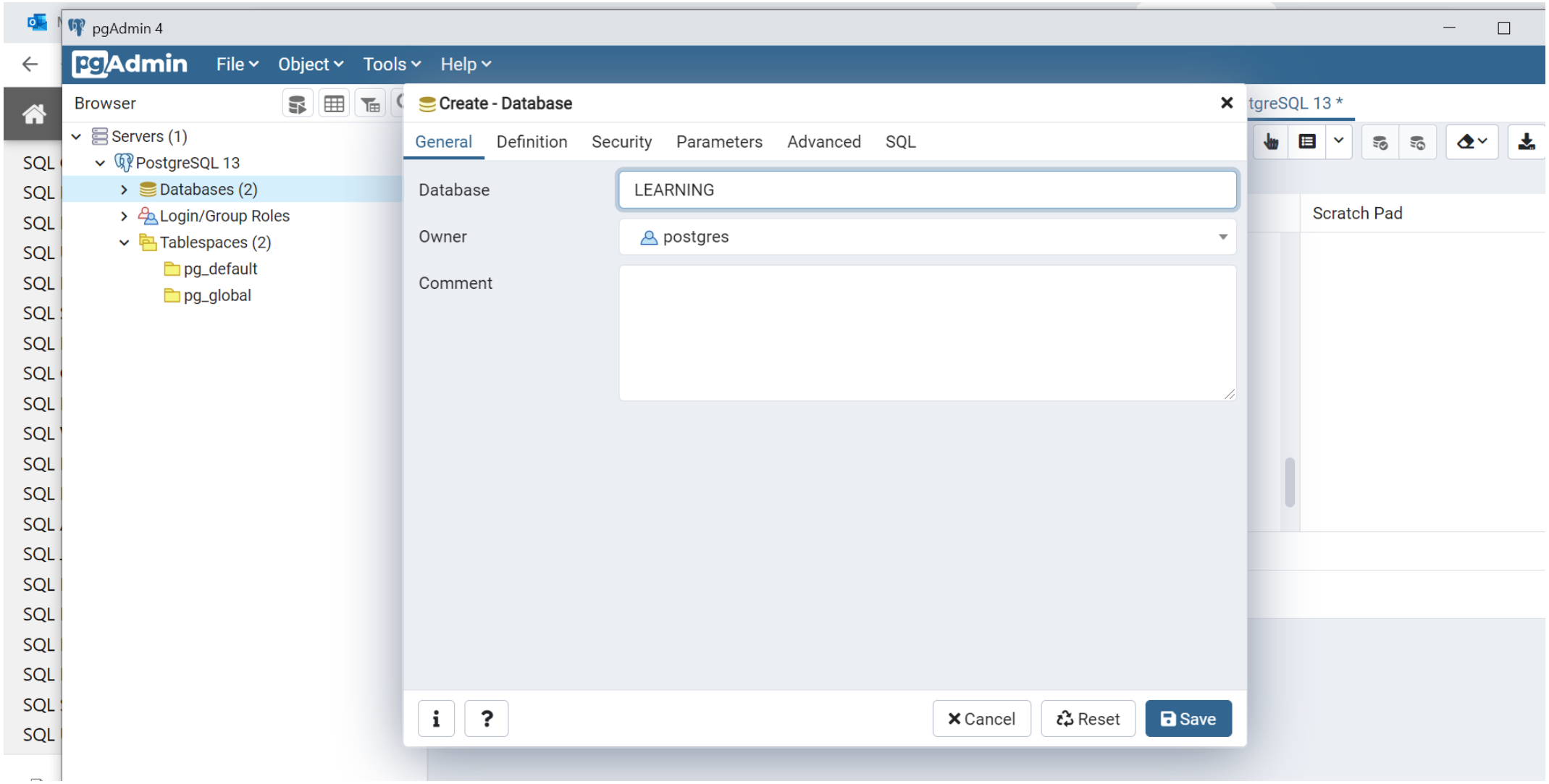
Scratch Pad

```
1
2  SELECT * FROM actor;
3  SELECT first_name, last_name FROM actor;
4  SELECT * FROM film;
5  SELECT DISTINCT (release_year) FROM film;
6  SELECT DISTINCT (rental_rate) FROM film;
7  SELECT * FROM payment;
8  SELECT COUNT (*) FROM payment;
9  SELECT COUNT (amount) FROM payment;
10 SELECT DISTINCT (amount) FROM payment;
11 SELECT COUNT (DISTINCT amount) FROM payment;
12
```

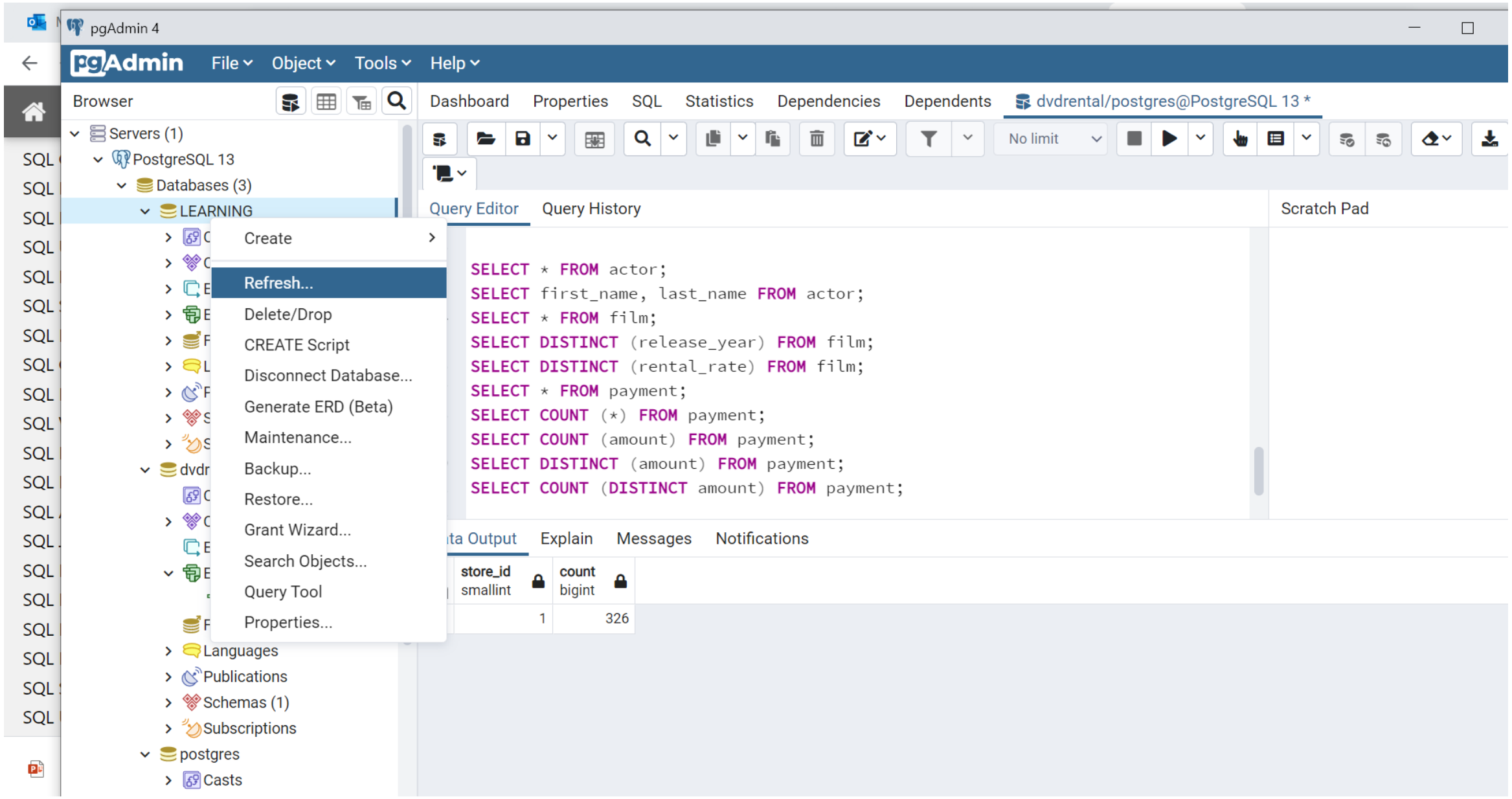
Data Output Explain Messages Notifications

	store_id smallint	count bigint
1	1	326

NAME THE DATABASE



REFRESH THE DATABASE



Query Tool

pgAdmin 4

File Object Tools Help

Browser

Servers (1)
PostgreSQL 13
Databases (3)
LEARNING
dvdre
postgres
Languages
Publications
Schemas (1)
Subscriptions
Casts

Create
Refresh...
Delete/Drop
CREATE Script
Disconnect Database...
Generate ERD (Beta)
Maintenance...
Backup...
Restore...
Grant Wizard...
Search Objects...
Query Tool
Properties...

Dashboard Properties SQL Statistics Dependencies Dependents dvdrental/postgres@PostgreSQL 13 *

No limit

Query Editor Query History Scratch Pad

```
SELECT * FROM actor;
SELECT first_name, last_name FROM actor;
SELECT * FROM film;
SELECT DISTINCT (release_year) FROM film;
SELECT DISTINCT (rental_rate) FROM film;
SELECT * FROM payment;
SELECT COUNT (*) FROM payment;
SELECT COUNT (amount) FROM payment;
SELECT DISTINCT (amount) FROM payment;
SELECT COUNT (DISTINCT amount) FROM payment;
```

a Output Explain Messages Notifications

store_id smallint	count bigint
1	326

pgAdmin 4

pgAdmin File Object Tools Help

Browser

- Servers (1)
  - PostgreSQL 13
    - Databases (3)
      - LEARNING
        - Casts
        - Catalogs
        - Event Triggers
        - Extensions
        - Foreign Data Wrappers
        - Languages
        - Publications
        - Schemas
        - Subscriptions
      - dvdrental
        - Casts
        - Catalogs (2)
        - Event Triggers
        - Extensions (1)
          - plpgsql
        - Foreign Data Wrappers
        - Languages
        - Publications
        - Schemas (1)
        - Subscriptions
      - postgres
        - Casts

Dashboard Properties SQL Statistics Dependencies Dependents dvdrental/post... LEARNING/postgres@PostgreSQL 13 *

No limit

Query Editor Query History Scratch Pad

```
CREATE TABLE learning_table(
    bait varchar(50),
    av_spec varchar(50)
);
```

Data Output Explain Messages Notifications

```
CREATE TABLE

Query returned successfully in 147 msec.
```

Query returned successfully in 147 msec

CREATE TABLE

pgAdmin 4

File Object Tools Help

Browser

- Languages
- Publications
- Schemas (1)
  - public
    - Collations
    - Domains
    - FTS Configurations
    - FTS Dictionaries
    - FTS Parsers
    - FTS Templates
    - Foreign Tables
    - Functions
    - Materialized Views
    - Procedures
    - Sequences
    - Tables (1)
      - learning_table
        - Columns
        - Constraints
        - Indexes
        - RLS Policies
        - Rules
        - Triggers
    - Trigger Functions
    - Types
    - Views

Dashboard Properties SQL Statistics Dependencies Dependents dvdrental/post... LEARNING/postgres@PostgreSQL 13 *

No limit

Query Editor Query History Scratch Pad

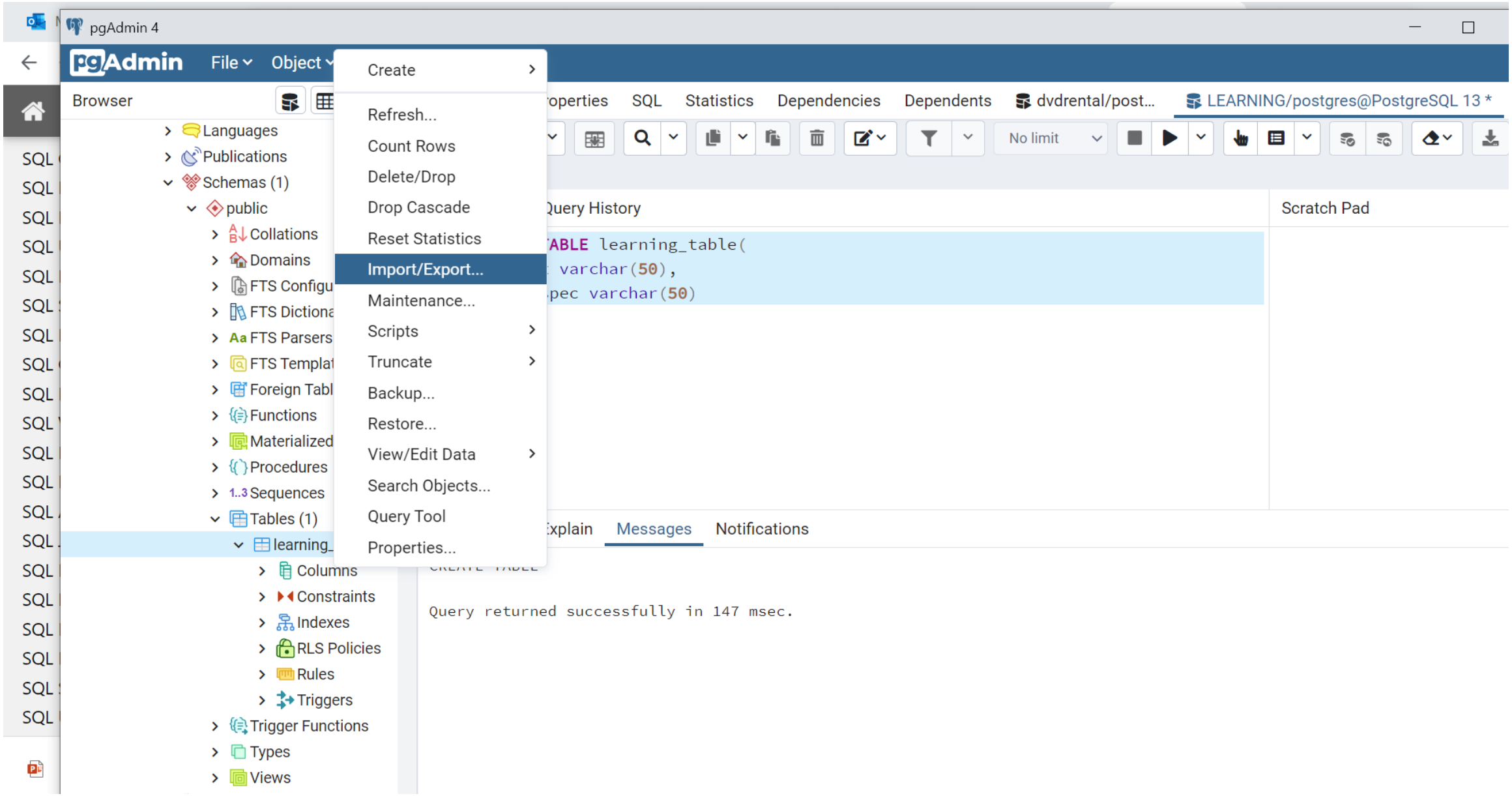
```
CREATE TABLE learning_table(
    bait varchar(50),
    av_spec varchar(50)
);
```

Data Output Explain Messages Notifications

```
CREATE TABLE

Query returned successfully in 147 msec.
```

IMPORT TABLE



IMPORT .CSV FILE

Admin 4

Admin File Object Tools Help

Dashboard Properties SQL Statistics Dependencies Dependents dvdrental/post... LEARNING/postgres@PostgreSQL 13 *

- Languages
- Publications
- Schemas (1)
  - public
    - Collations
    - Domains
    - FTS Configurations
    - FTS Dictionaries
    - FTS Parsers
    - FTS Templates
    - Foreign Tables
    - Functions
    - Materialized Views
    - Procedures
    - Sequences
    - Tables (1)
      - learning_table
        - Columns
        - Constraints
        - Indexes
        - RLS Policies
        - Rules
        - Triggers
    - Trigger Functions
    - Types
    - Views

Import/Export data - table 'learning_table'

Options Columns

Filename: C:\Users\msardiu\Table_learning.csv

Format: csv

Encoding: Select an item...

Miscellaneous

OID: No

Header: Yes

Delimiter: ,

Specifies the character that separates columns within each row (line) of the file. The default is a tab character in text format, a comma in CSV format. This must be a single one-byte character. This option is not allowed when using binary format.

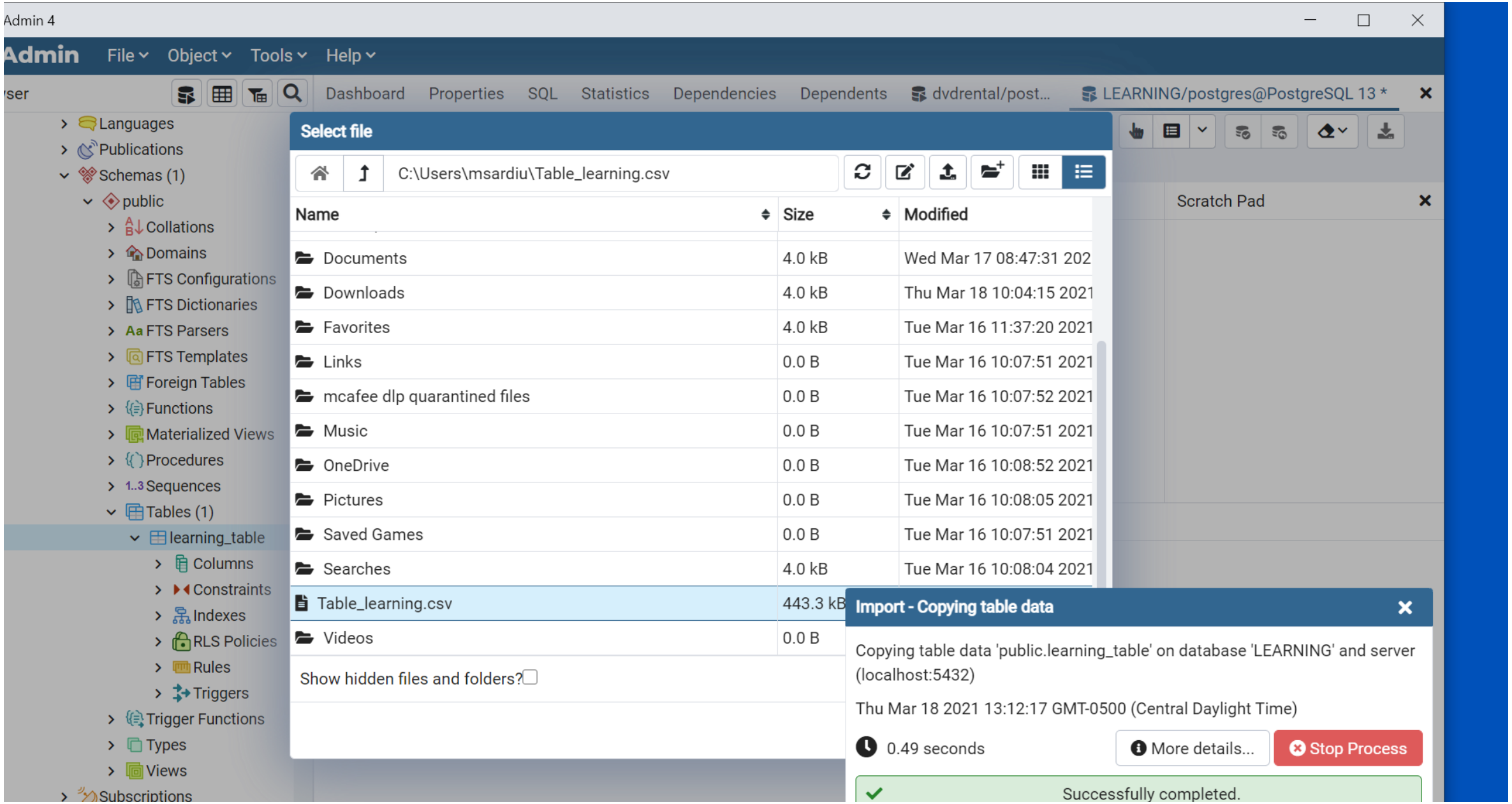
Quote: "

Specifies the quoting character to be used when a data value is quoted. The

Cancel OK

Scratch Pad

IMPORT .CSV FILE SUCCESFULLY



Working with the import table

Admin 4

Admin File Object Tools Help

Dashboard Properties SQL Statistics Dependencies Dependents dvdrental/post... LEARNING/postgres@PostgreSQL 13 *

No limit

- Languages
- Publications
- Schemas (1)
  - public
    - Collations
    - Domains
    - FTS Configurations
    - FTS Dictionaries
    - FTS Parsers
    - FTS Templates
    - Foreign Tables
    - Functions
    - Materialized Views
    - Procedures
    - Sequences
    - Tables (1)
      - learning_table
        - Columns
        - Constraints
        - Indexes
        - RLS Policies
        - Rules
        - Triggers
    - Trigger Functions
    - Types
    - Views
- Subscriptions

Query Editor Query History

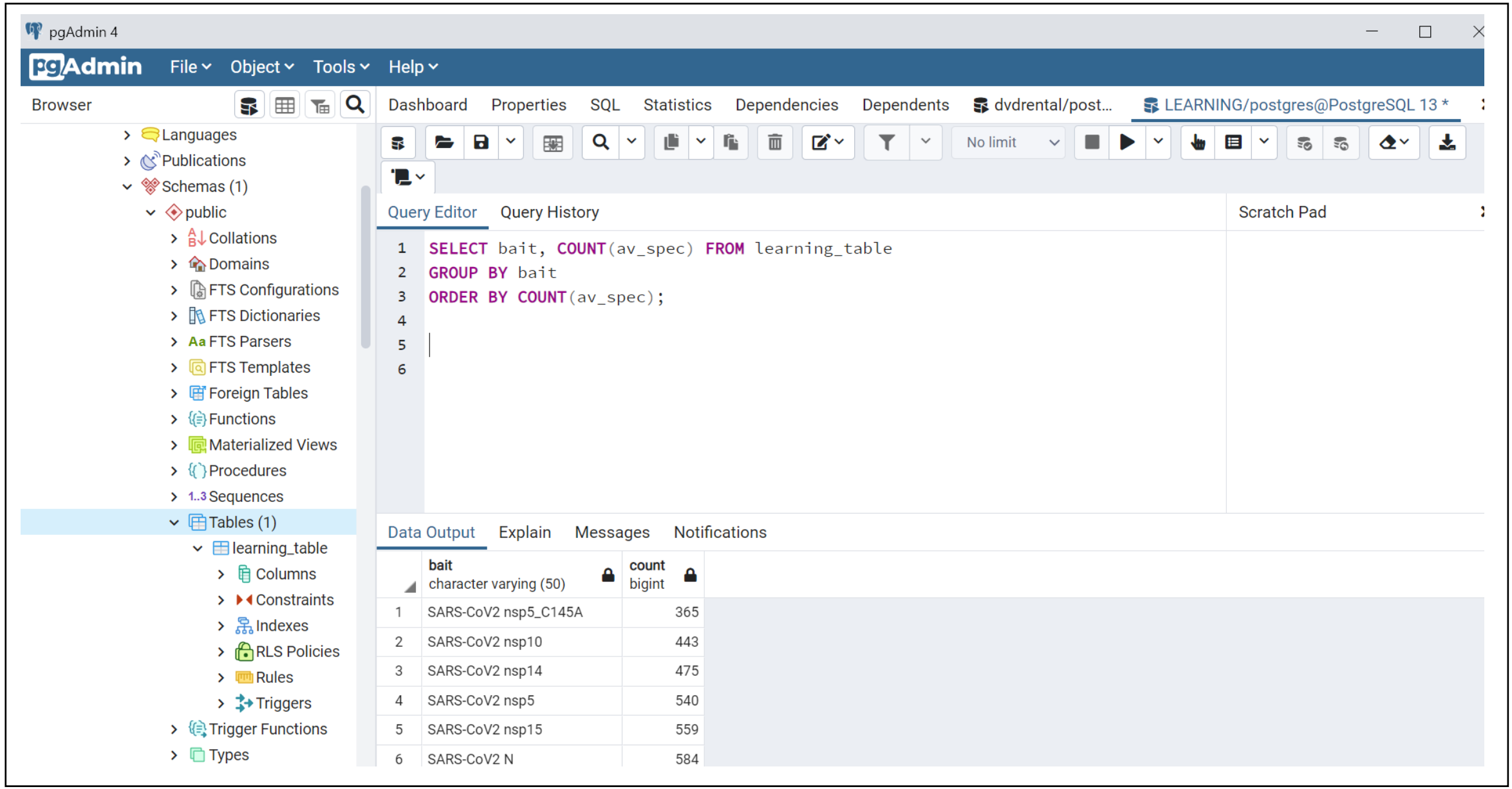
Scratch Pad

```
1 SELECT * FROM learning_table
2
```

Data Output Explain Messages Notifications

	bait character varying (50)	av_spec character varying (50)
1	SARS-CoV2 E	1.67
2	SARS-CoV2 E	0.33
3	SARS-CoV2 E	3.33
4	SARS-CoV2 E	0.33
5	SARS-CoV2 E	1.33
6	SARS-CoV2 E	0.67
7	SARS-CoV2 E	0.67

Working with the import table



INSERT values manually

```
CREATE TABLE account (
        user_id SERIAL PRIMARY KEY,
        username varchar(50) UNIQUE NOT NULL,
        password varchar(50) NOT NULL,
        email varchar(25)
        )
```

```
INSERT INTO account(username, password, email)
VALUES
('Mihaela','password','msardiu')
```

DELETE CLAUSE

WE can use the DELETE clause <u>to remove rows from a table</u>.
For example:

```
DELETE FROM table
WHERE row_id=1
```

<u>Between two tables</u>

```
DELETE FROM table A
USING table B
WHERE tableA.id=TABLEB.id
```

<u>We can delete all rows</u>

```
DELETE * FROM table
```

ALTER CLAUSE

The ALTER clause allows for changes to an existing table structure, such as:

- Adding, dropping or renaming columns
- Changing a column data type
- Set DEFAULT values for a column
- Add CHECK constrains
- Rename table

Syntax to try

```
ALTER TABLE information
RENAME TO new_info

ALTER TABLE new_info
RENAME COLUMN person to people
```

DROP CLAUSE

- DROP allows for the complete removal of a column in a table.
- In PostgreSQL this will also automatically remove all of its indexes and constrains involving the column.

<u>General syntax</u>

```
ALTER TABLE table_name
DROP COLUMN col_name
```