

pgAdmin Overview

The screenshot displays the pgAdmin 4 web interface. The top navigation bar includes 'Admin', 'File', 'Object', 'Tools', and 'Help'. The left sidebar shows a tree view of database objects, with 'postgres' expanded to show 'Schemas (1)' and 'public' expanded to show various database features like 'Collations', 'Domains', 'FTS Configurations', etc. The main panel is titled 'Query Editor' and shows a SQL query: `SELECT * FROM film;`. Below the query editor, the 'Data Output' tab is active, displaying a table with 10 columns: `film_id` (integer), `title` (character varying (255)), `description` (text), `release_year` (integer), `language_id` (smallint), `rental_duration` (smallint), `rental_rate` (numeric (4,2)), and `length` (smallint). The table contains 6 rows of data.

	<code>film_id</code> [PK] integer	<code>title</code> character varying (255)	<code>description</code> text	<code>release_year</code> integer	<code>language_id</code> smallint	<code>rental_duration</code> smallint	<code>rental_rate</code> numeric (4,2)	<code>length</code> smallint
1	133	Chamber Italian	A Fateful Reflec...	2006	1	7	4.99	
2	384	Grosse Wonderful	A Epic Drama of...	2006	1	5	4.99	
3	8	Airport Pollock	A Epic Tale of a ...	2006	1	6	4.99	
4	98	Bright Encounters	A Fateful Yarn o...	2006	1	4	4.99	
5	1	Academy Dinosaur	A Epic Drama of...	2006	1	6	0.99	
6	2	Ace Goldfinger	A Astounding E...	2006	1	3	4.99	

PREFERENCES

The screenshot shows the 'Admin 4' application window with the 'Preferences' dialog box open. The 'CSV/TXT Output' option is selected in the left sidebar. The main area displays settings for CSV field separator (comma), CSV quote character (double quote), CSV quoting (Strings), and Replace null values with (NULL). A descriptive text explains that the NULL value string can be any arbitrary string, with quotes if desired. At the bottom right of the dialog are 'Cancel' and 'Save' buttons. In the background, a table with columns 'rental_rate' and 'length' is visible.

Admin 4

File ▾ Object ▾ Tools ▾ Help ▾

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

Preferences

- Keyboard shortcuts
- Miscellaneous
 - Themes
 - User language
- Paths
 - Binary paths
 - Help
- Query Tool
 - Auto completion
 - CSV/TXT Output**
 - Display
 - Editor
 - Explain
 - Keyboard shortcuts
 - Options
 - Results grid
 - SQL formatting
- Schema Diff
 - Display
- Storage
 - Options

CSV field separator: ,

CSV quote character: "

CSV quoting: Strings

Replace null values with: NULL

Specifies the string that represents a null value while downloading query results as CSV. You can specify any arbitrary string to represent a null value, with quotes if desired.

Cancel Save

	rental_rate numeric (4,2)	length smallint
7	4.99	
5	4.99	
6	4.99	
4	4.99	
6	0.99	
3	4.99	

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...

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.

Example syntax for **SELECT** statement:

SELECT column_name FROM table_name

SELECT C1, C3 FROM Table 1;

Database

Table 1

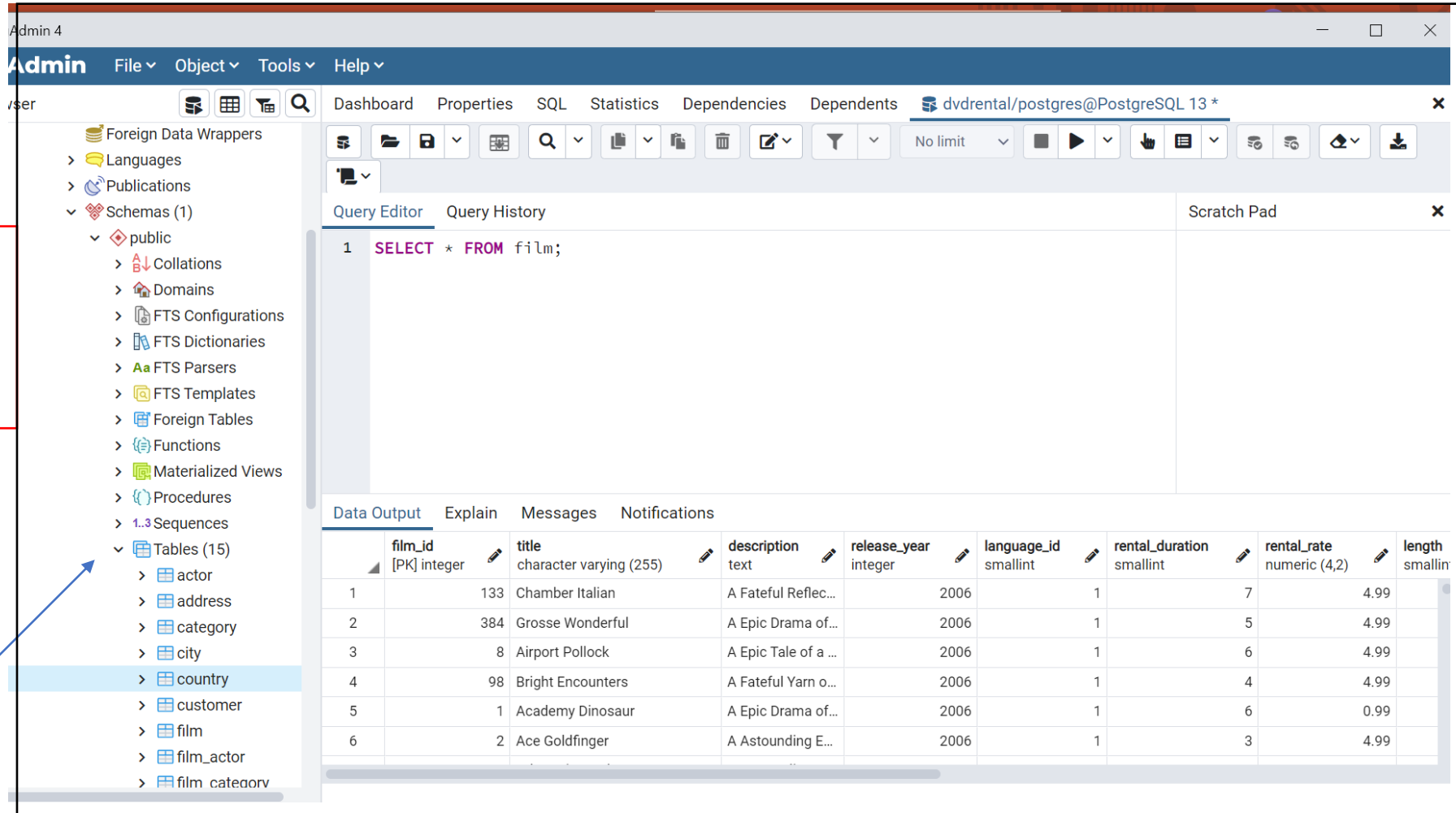
C1	C2	C3
X	23	a
Y	18	b
z	46	c

Table 2

Table 3

SELECT CLAUSE

Look at the tables
which are
inside our database



Admin 4

File Object Tools Help

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

Query Editor Query History Scratch Pad

```
1 SELECT * FROM film;
```

Data Output Explain Messages Notifications

	film_id [PK] integer	title character varying (255)	description text	release_year integer	language_id smallint	rental_duration smallint	rental_rate numeric (4,2)	length smallint
1	133	Chamber Italian	A Fateful Reflec...	2006	1	7	4.99	
2	384	Grosse Wonderful	A Epic Drama of...	2006	1	5	4.99	
3	8	Airport Pollock	A Epic Tale of a ...	2006	1	6	4.99	
4	98	Bright Encounters	A Fateful Yarn o...	2006	1	4	4.99	
5	1	Academy Dinosaur	A Epic Drama of...	2006	1	6	0.99	
6	2	Ace Goldfinger	A Astounding E...	2006	1	3	4.99	

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.

Basic syntax example:

```
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  
LIMIT 5;
```

We can use LIMIT 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](#)

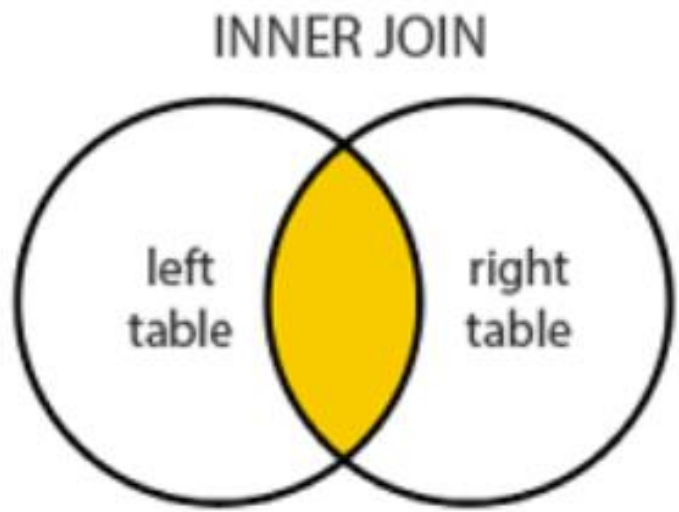
[SQL JOIN Examples](#)

[Wikipedia Page on SQL JOINS](#)

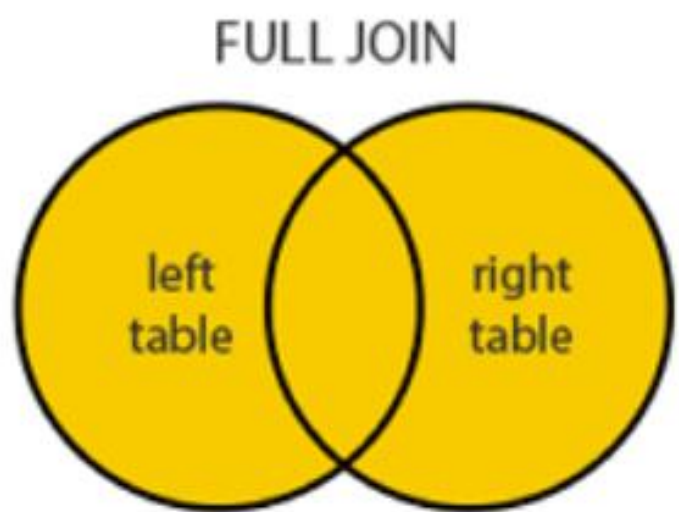
Four different types of JOINS

- 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.

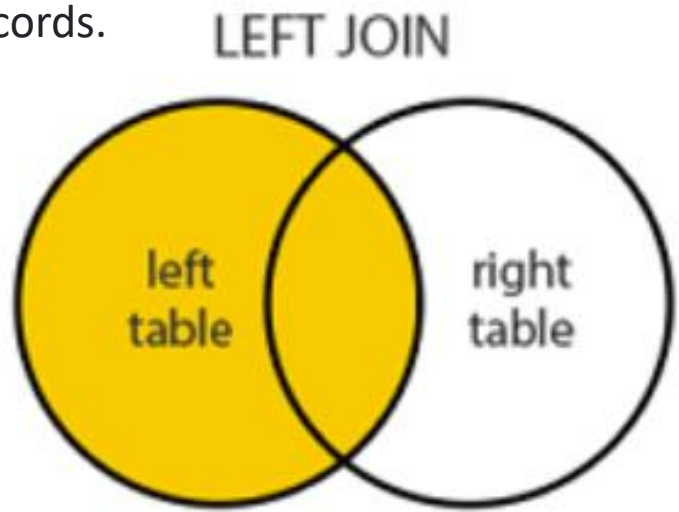
Select records that have matching values in both tables



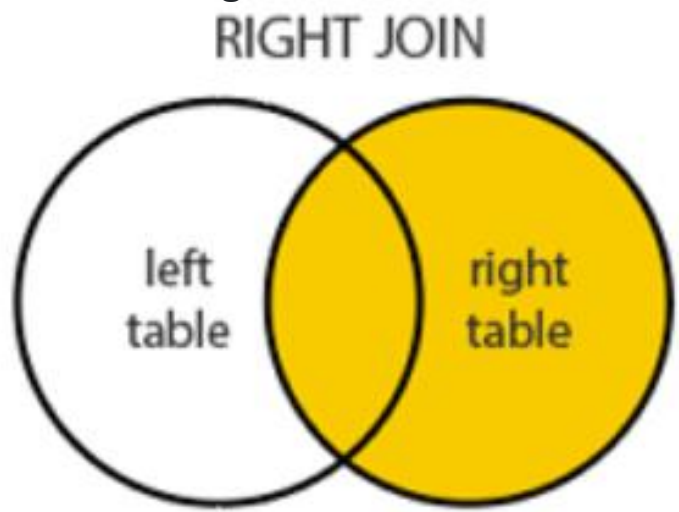
Selects all records that match either left or right table records



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



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



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
```

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
```

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
```

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

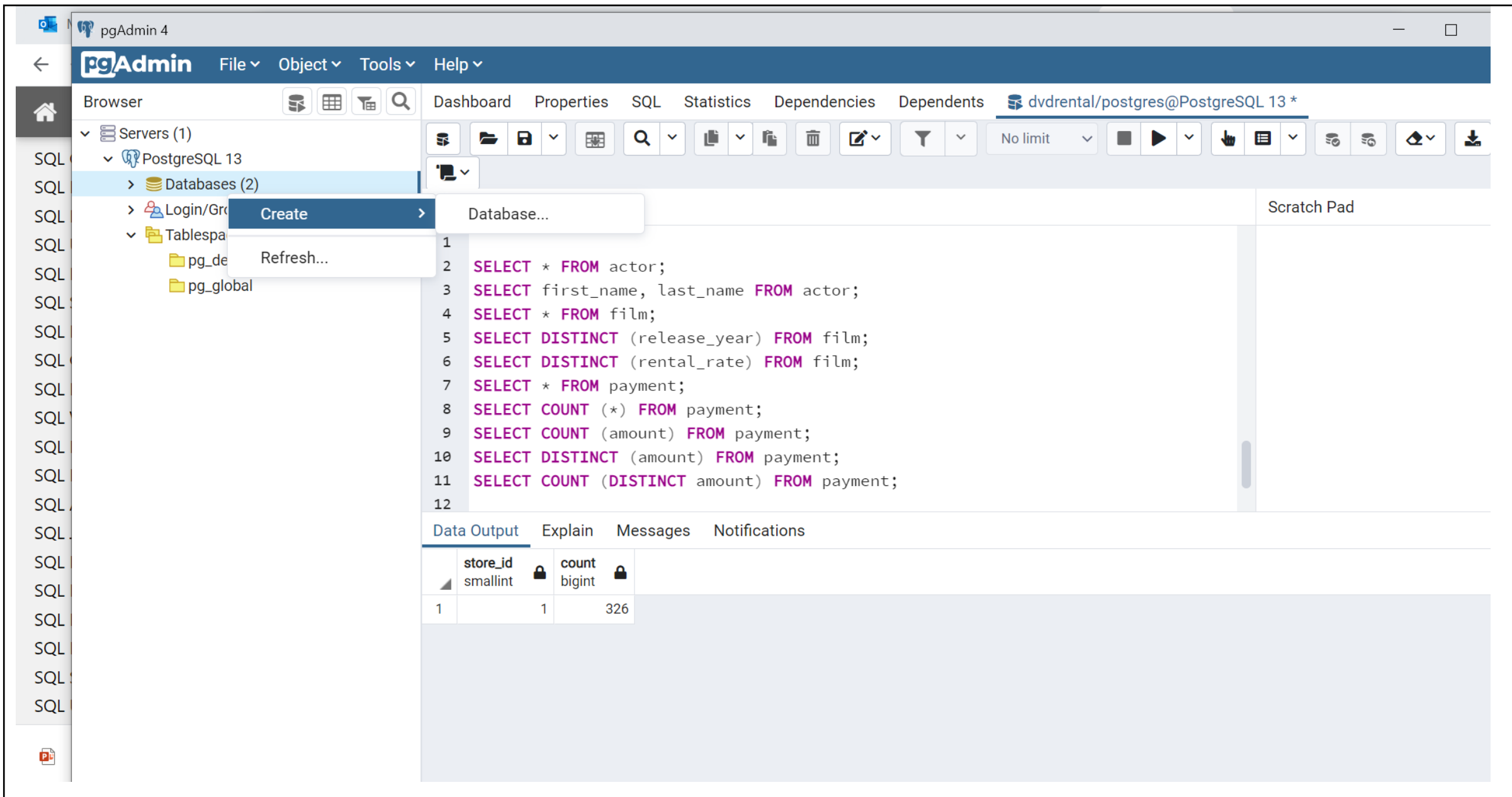
- Boolean
 - True or False
- Character
 - Char, varchar, and text
- Numeric
 - Integer and floating-point number
- Temporal
 - Date, time, timestamp, and interval

- 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.

- Full General Syntax:

- 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

File Object Tools Help

Browser

Servers (1)

- PostgreSQL 13
 - Databases (2)
 - Login/Groups
 - Tablespaces
 - pg_default
 - pg_global

Create Database...

Refresh...

SQL

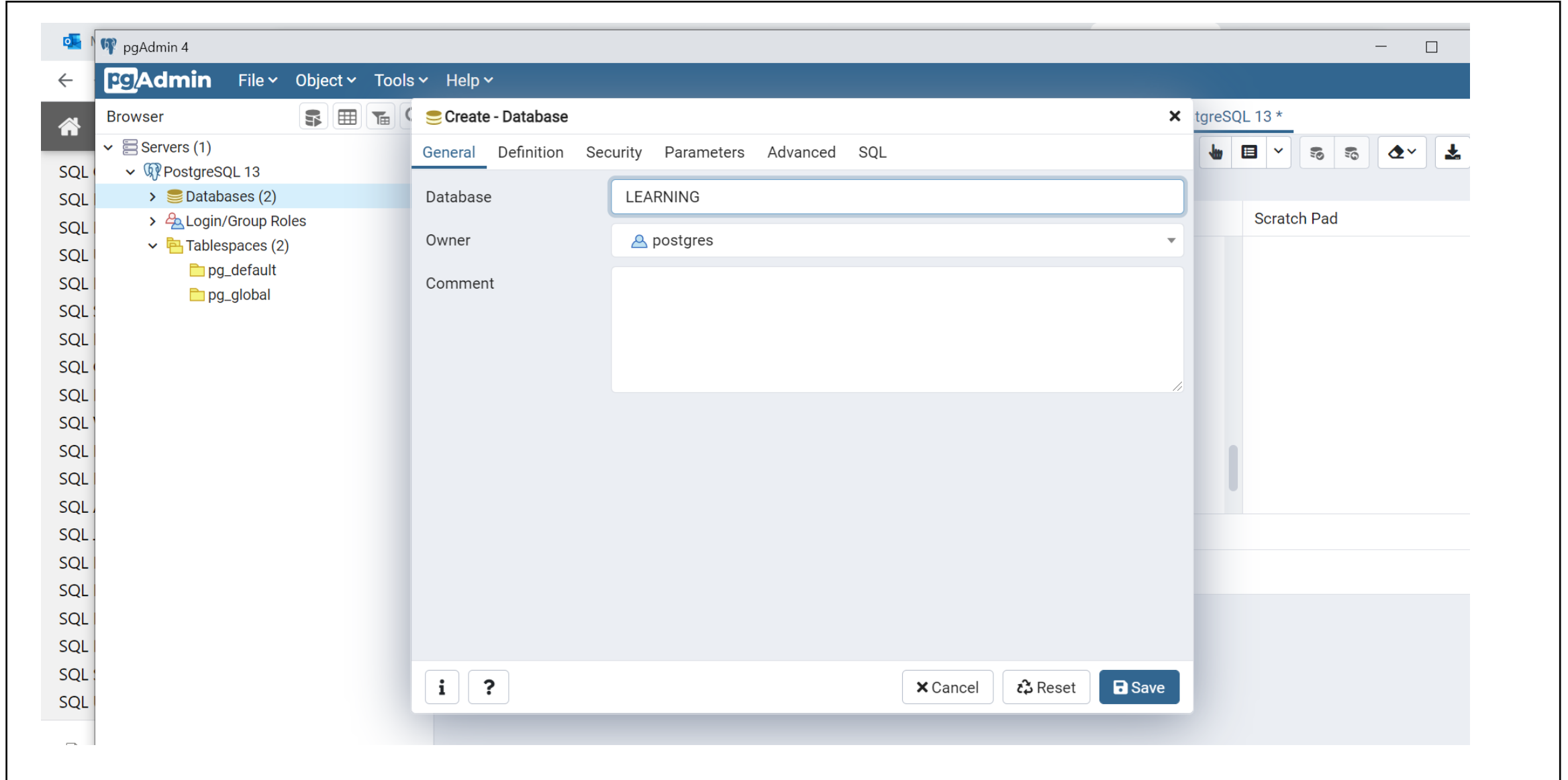
```
1
2 SELECT * FROM actor;
3 SELECT first_name, last_name FROM actor;
4 SELECT * FROM film;
5 SELECT DISTINCT (release_year) FROM film;
6 SELECT DISTINCT (rental_rate) FROM film;
7 SELECT * FROM payment;
8 SELECT COUNT (*) FROM payment;
9 SELECT COUNT (amount) FROM payment;
10 SELECT DISTINCT (amount)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

Scratch Pad

Data Output Explain Messages Notifications

	store_id smallint	count bigint
1	1	326

NAME THE DATABASE



REFRESH THE DATABASE

The screenshot shows the pgAdmin 4 web interface. The left sidebar displays a tree view of the database structure: Servers (1) > PostgreSQL 13 > Databases (3) > LEARNING. A right-click context menu is open over the 'LEARNING' database, with the 'Refresh...' option highlighted. The main panel shows the 'Query Editor' tab with a SQL query. Below the query editor, the 'Data Output' tab is active, displaying a table with two columns: 'store_id' and 'count'. The table contains one row with the values 'smallint' and '326'.

pgAdmin 4

File Object Tools Help

Browser

Servers (1)

PostgreSQL 13

Databases (3)

LEARNING

Create

Refresh...

Delete/Drop

CREATE Script

Disconnect Database...

Generate ERD (Beta)

Maintenance...

Backup...

Restore...

Grant Wizard...

Search Objects...

Query Tool

Properties...

Languages

Publications

Schemas (1)

Subscriptions

postgres

Casts

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;
```

Data Output Explain Messages Notifications

store_id	count
smallint	bigint
1	326

Query Tool

The screenshot displays the pgAdmin 4 web interface. The left sidebar shows a tree view of the database structure, including Servers (1), PostgreSQL 13, Databases (3), and the selected database LEARNING. A context menu is open over the LEARNING database, listing various actions such as Create, Refresh..., Delete/Drop, CREATE Script, Disconnect Database..., Generate ERD (Beta), Maintenance..., Backup..., Restore..., Grant Wizard..., Search Objects..., and the highlighted Query Tool. The main panel is divided into two tabs: Query Editor and Query History. The Query Editor tab is active, showing a SQL query with multiple SELECT statements. Below the query editor, there are tabs for Query Output, Explain, Messages, and Notifications. The Query Output tab is selected, displaying a table with two columns: store_id and count. The table contains one row with the value 1 for store_id and 326 for count.

pgAdmin 4

File Object Tools Help

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

Browser

Servers (1)

PostgreSQL 13

Databases (3)

LEARNING

Create

Refresh...

Delete/Drop

CREATE Script

Disconnect Database...

Generate ERD (Beta)

Maintenance...

Backup...

Restore...

Grant Wizard...

Search Objects...

Query Tool

Properties...

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;
```

Query Output

store_id	count
1	326

pgAdmin 4

pgAdmin

FileObjectToolsHelp

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

DashboardPropertiesSQLStatisticsDependenciesDependents

dvdrental/post...LEARNING/postgres@PostgreSQL 13 *

Query EditorQuery History

```
1 CREATE TABLE learning_table(  
2     bait varchar(50),  
3     av_spec varchar(50)  
4 );
```

Scratch Pad

Data OutputExplainMessagesNotifications

CREATE TABLE

Query returned successfully in 147 msec.

✓ Query returned successfully in 147 msec

CREATE TABLE

The screenshot displays the pgAdmin 4 web interface. On the left, the 'Browser' pane shows a tree view of database objects. Under 'Schemas (1)', the 'public' schema is expanded, and 'Tables (1)' is selected, showing the 'learning_table'. The main area is divided into two tabs: 'Query Editor' and 'Query History'. The 'Query Editor' tab is active, showing a SQL query to create a table named 'learning_table' with two columns: 'bait' of type 'varchar(50)' and 'av_spec' of type 'varchar(50)'. The query is as follows:

```
1 CREATE TABLE learning_table(  
2     bait varchar(50),  
3     av_spec varchar(50)  
4 );
```

Below the query editor, there are tabs for 'Data Output', 'Explain', 'Messages', and 'Notifications'. The 'Messages' tab is selected, showing the output of the query: 'CREATE TABLE' and 'Query returned successfully in 147 msec.'

IMPORT TABLE

The screenshot displays the pgAdmin 4 web interface. On the left, the 'Browser' pane shows a tree structure of database objects. Under 'Schemas (1)', the 'public' schema is expanded, and under 'Tables (1)', the 'learning...' table is selected. A context menu is open over this table, with 'Import/Export...' highlighted. The main pane shows the 'Query History' tab with a SQL query for creating a table: `CREATE TABLE learning_table (varchar(50), spec varchar(50))`. The 'Messages' tab at the bottom shows a success message: 'Query returned successfully in 147 msec.'

pgAdmin 4

File Object

Browser

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

Create

- Refresh...
- Count Rows
- Delete/Drop
- Drop Cascade
- Reset Statistics
- Import/Export...
- Maintenance...
- Scripts
- Truncate
- Backup...
- Restore...
- View/Edit Data
- Search Objects...
- Query Tool
- Properties...

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

Query History

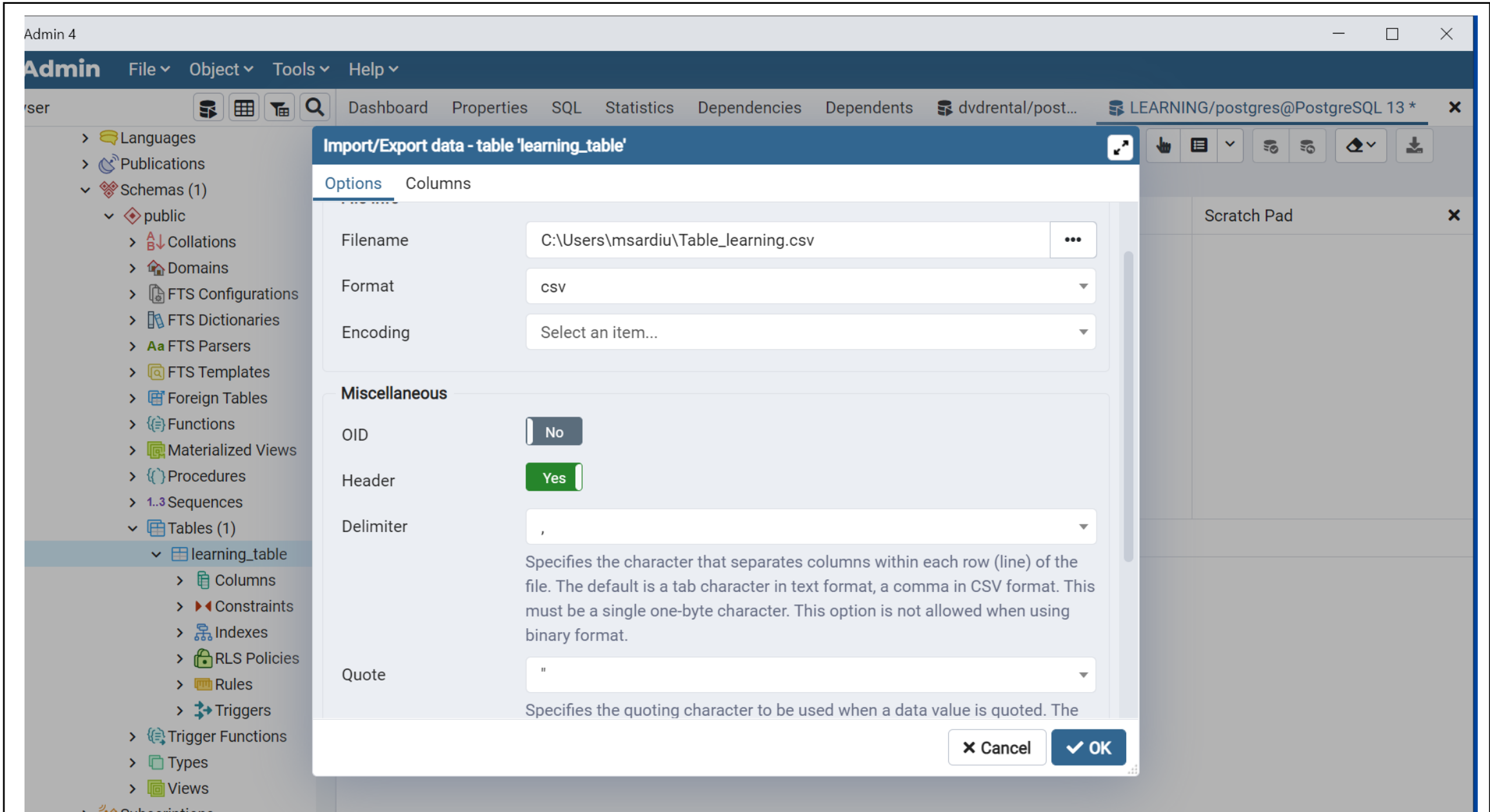
```
CREATE TABLE learning_table (
  varchar(50),
  spec varchar(50))
```

Scratch Pad

Messages

Query returned successfully in 147 msec.

IMPORT .CSV FILE



IMPORT .CSV FILE SUCCESSFULLY

Admin 4

Admin File Object Tools Help

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

Scratch Pad

Select file

C:\Users\msardiu\Table_learning.csv

Name	Size	Modified
Documents	4.0 kB	Wed Mar 17 08:47:31 202
Downloads	4.0 kB	Thu Mar 18 10:04:15 2021
Favorites	4.0 kB	Tue Mar 16 11:37:20 2021
Links	0.0 B	Tue Mar 16 10:07:51 2021
mcafee dlp quarantined files	0.0 B	Tue Mar 16 10:07:52 2021
Music	0.0 B	Tue Mar 16 10:07:51 2021
OneDrive	0.0 B	Tue Mar 16 10:08:52 2021
Pictures	0.0 B	Tue Mar 16 10:08:05 2021
Saved Games	0.0 B	Tue Mar 16 10:07:51 2021
Searches	4.0 kB	Tue Mar 16 10:08:04 2021
Table_learning.csv	443.3 kB	
Videos	0.0 B	

Show hidden files and folders? ☐

Import - Copying table data

Copying table data 'public.learning_table' on database 'LEARNING' and server (localhost:5432)

Thu Mar 18 2021 13:12:17 GMT-0500 (Central Daylight Time)

0.49 seconds

More details... Stop Process

Successfully completed.

Admin 4

Admin

File

Object

Tools

Help

Dashboard

Properties

SQL

Statistics

Dependencies

Dependents

dvdrental/post...

LEARNING/postgres@PostgreSQL 13 *

Scratch Pad

Import - Copying table data

Copying table data 'public.learning_table' on database 'LEARNING' and server (localhost:5432)

Thu Mar 18 2021 13:12:17 GMT-0500 (Central Daylight Time)

0.49 seconds

More details...

Stop Process

Successfully completed.

Admin 4

Admin

File

Object

Tools

Help

Dashboard

Properties

SQL

Statistics

Dependencies

Dependents

dvdrental/post...

LEARNING/postgres@PostgreSQL 13 *

Scratch Pad

Import - Copying table data

Copying table data 'public.learning_table' on database 'LEARNING' and server (localhost:5432)

Thu Mar 18 2021 13:12:17 GMT-0500 (Central Daylight Time)

0.49 seconds

More details...

Stop Process

Successfully completed.

Working with the import table

The screenshot displays the pgAdmin 4 web interface. On the left, the 'Schemas (1)' tree is expanded to show the 'public' schema, where the 'learning_table' is selected. The main panel is divided into a 'Query Editor' and a 'Data Output' section. The 'Query Editor' contains the SQL query: `SELECT * FROM learning_table`. The 'Data Output' section shows the results of the query in a table format.

Query Editor

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

Data Output

	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

The screenshot shows the pgAdmin 4 web interface. On the left is a 'Browser' pane with a tree view of the database structure. The 'public' schema is expanded, and 'Tables (1)' is selected, showing the 'learning_table'. The main area is divided into a 'Query Editor' and a 'Data Output' pane. The Query Editor contains a SQL query that counts the occurrences of different 'bait' values in the 'learning_table'. The Data Output pane shows the results of this query as a table with two columns: 'bait' and 'count'.

Query Editor

```
1 SELECT bait, COUNT(av_spec) FROM learning_table
2 GROUP BY bait
3 ORDER BY COUNT(av_spec);
4
5
6
```

Data Output

	bait character varying (50)	count bigint
1	SARS-CoV2 nsp5_C145A	365
2	SARS-CoV2 nsp10	443
3	SARS-CoV2 nsp14	475
4	SARS-CoV2 nsp5	540
5	SARS-CoV2 nsp15	559
6	SARS-CoV2 N	584

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 to remove rows from a table.

For example:

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

Between two tables

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

We can delete all rows

```
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 constraints
- 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 constraints involving the column.

General syntax

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