Introduction to SQL

Data Boot Camp

Lesson 9.1

Congratulations!!

You have made it this far!

Tap yourself on your shoulder for such an accomplishment.

Be very proud of yourself!

Learning Outcomes

By the end of this unit, you will be able to:

01

Create a data model to represent the objects and relationships in a dataset.

02

Create schemas, tables, and databases for relational data.

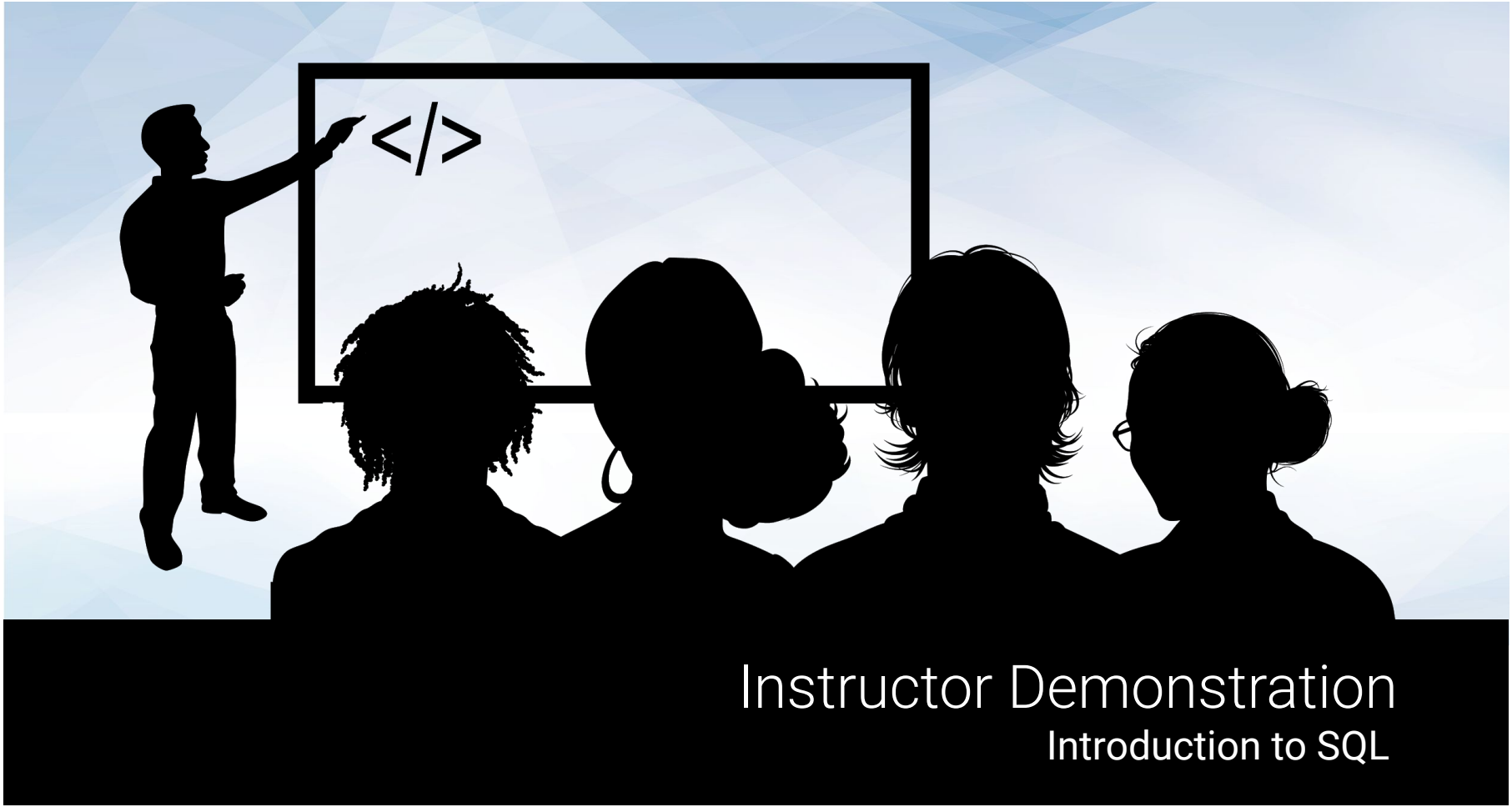
03

Retrieve data using advanced database queries.

Class Objectives

By the end of today's class, you will:

- Install PostgreSQL and pgAdmin on your computer.
- Create databases and tables using pgAdmin.
- Define SQL data types, primary keys, and unique values.
- Load CSV files into a database.
- Query data from a database.
- Articulate the four basic functions of CRUD and apply them to a database.
- Combine data from multiple tables using JOINs.

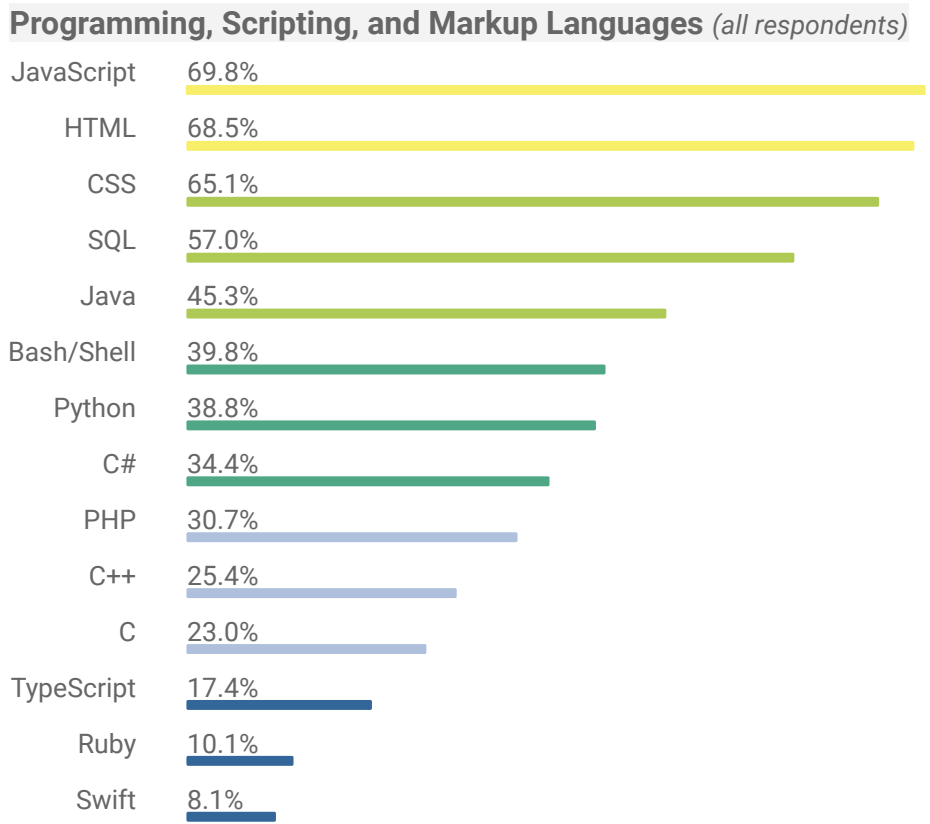


Instructor Demonstration

Introduction to SQL

Introduction to SQL

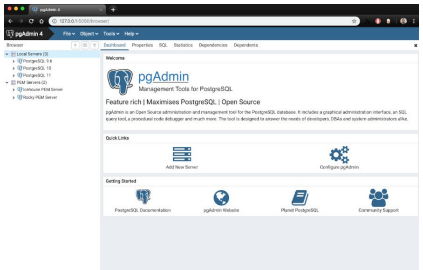
- **S**tructured **Q**uery **L**anguage (SQL) is one of the main query languages used to access data within relational databases.
- **SQL** is designed to efficiently handle large amounts of data, which is a highly valued capability for organizations.
- Experienced **SQL** programmers are in high demand.

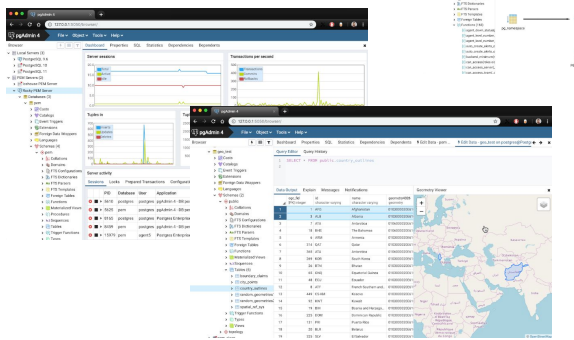


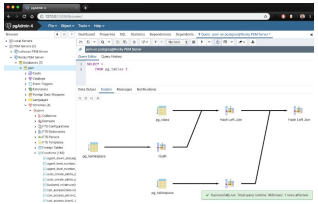
Introduction to SQL

- PostgreSQL (typically referred to as "Postgres") is an object-relational database system that uses the SQL language.
  - Database Engine
  - Open Source
  - Great Functionality

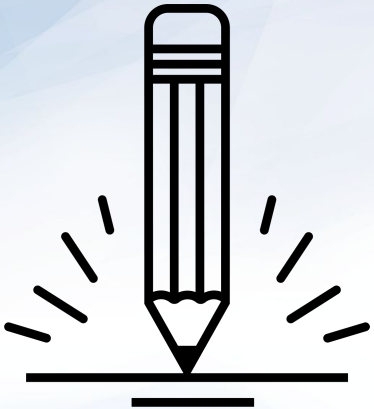
- **pgAdmin**
  - Is a management tool used for working with Postgres. It simplifies the creation, maintenance, and use of database objects.







Check your SLACK!!!



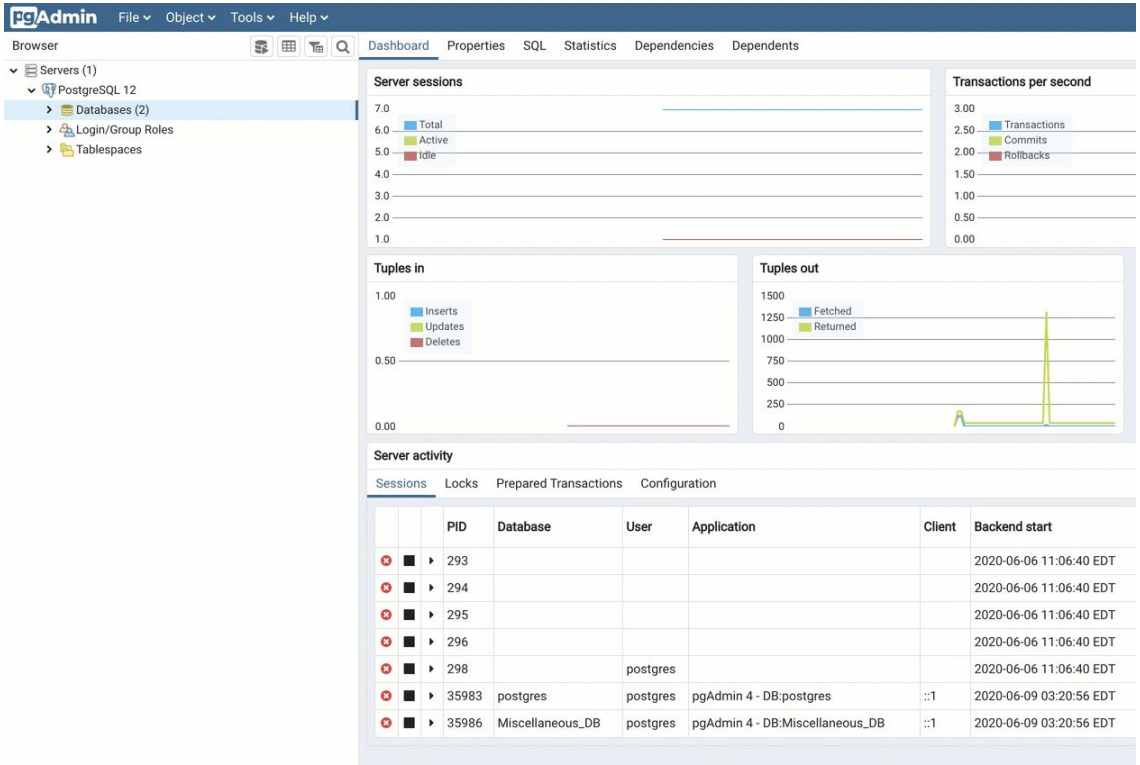
Activity: Create a Database

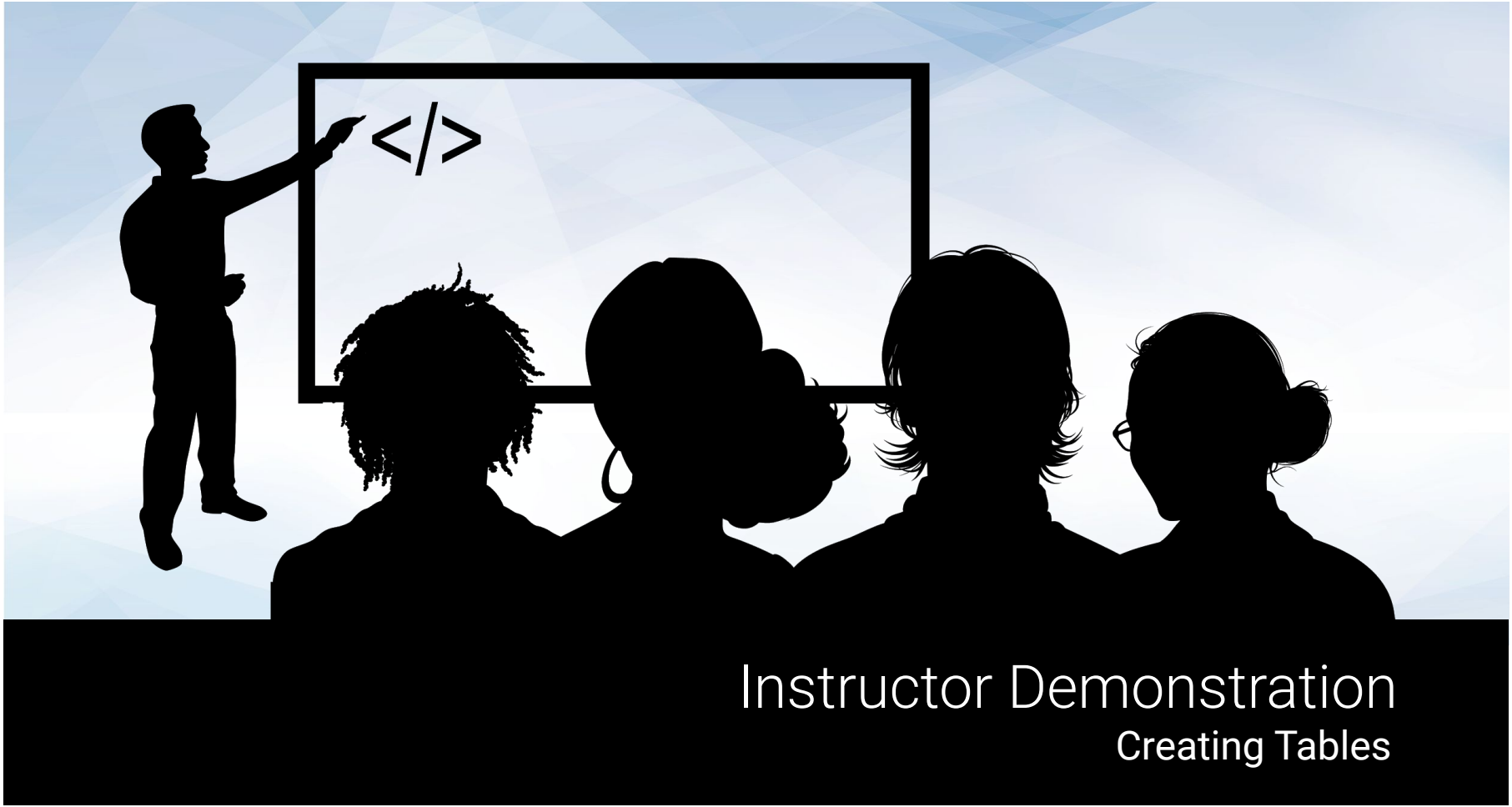
In this activity, everyone will create a database in pgAdmin.

Suggested Time:
5 Minutes

Create a Database

- In the pgAdmin editor, right-click the newly established server to create a new database.
- From the menu, select **Create**, and then select **Database** to create a new database
- Enter **animals_db** as the database name. Make sure the owner is set as the default postgres, and then click **Save**.





Instructor Demonstration

Creating Tables

Creating Tables

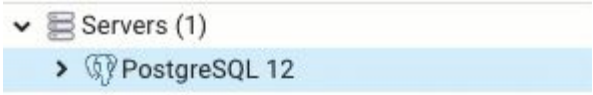
- In order to create tables you need couple of things:
  - Code editor (Query Tool).
  - Queries.

- How to open the code editor.

  - Locate the left-hand menu in **pgAdmin**.
  - Click on **PostgreSQL 12** to expand the menu.

    - Click on **Databases** to expand the menu.

      - Right-click postgres (or whatever database you have created) and select **Query Tool**.



***Note**: You can also select Query Tool from the Tools drop-down menu at the top of the screen.*

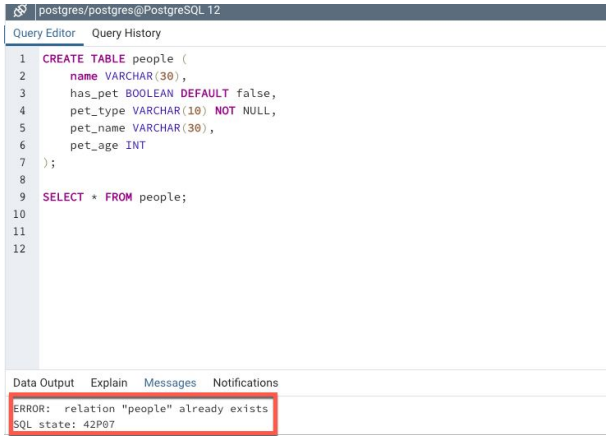
Creating Tables

- **Queries:**
  - `CREATE TABLE people (<COLUMNS>);` creates a table called people with the columns listed within the parentheses.
  - `name VARCHAR(30) NOT NULL` creates a `name` column, which can hold character strings of up to 30 characters and will not allow null fields.
  - The `NOT NULL` constraint requires the name field to have a value specified.
  - `pet_type VARCHAR(10) NOT NULL`, creates a `pet_type` in the same manner as the name column is created. The only difference is the number of characters allowed in the column.
  - `has_pet BOOLEAN DEFAULT false` creates a `has_pet` column that holds either true or false values, though the default value is now set as false.
  - `pet_name VARCHAR(30)` creates a pet_name column, which can hold character strings of up to 30 characters and will allow null fields.
  - `pet_age INT` creates a `pet_age` column, which can hold whole numbers.

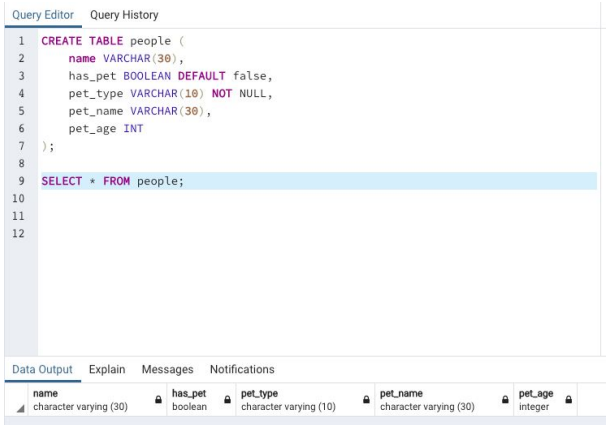
***Note**: The semicolon at the end of the statement tells pgAdmin that this line of code has concluded.*

Creating Tables

- **Queries:**
  - `SELECT * FROM <table name>;` visualize the structure of the table.



- Make sure you highlight the query otherwise you will encounter an error.



Creating Tables

- The query below operates as it reads.
  - Data is inserted to the `people` table in the following columns.
  - Data is the following, in the same order columns were stipulated.

```
INSERT INTO people (name, has_pet, pet_type, pet_name, pet_age)
VALUES ('Jacob', true, 'dog', 'Misty', 10),
  ('Ahmed', true, 'rock', 'Rockington', 100),
  ('Peter', true, 'cat', 'Franklin', 2),
  ('Dave', true, 'dog', 'Queso', 1);
```

- The query below only extracts the data from the column `pet_name` in the `people` table.

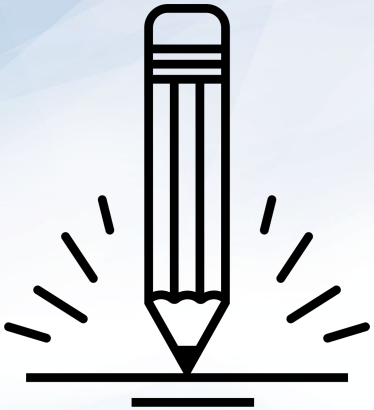
```
SELECT pet_name
FROM people;
```

Note: *Single quotations must be used for insert strings; otherwise, an error will result.*

Creating Tables

- The query below will return only dog(s), with their names that are younger than 5 years old.

```
SELECT pet_type, pet_name
FROM people
WHERE pet_type = 'dog'
AND pet_age < 5;
```



Activity: Creating Tables

In this activity, you will recreate and a table from an image provided.

Suggested Time:
15 Minutes

Activity: Creating Tables

- Create a new database in pgAdmin named `city_info`.
- Using the query tool, create an empty table named `cities`. Be sure to match the data types!
- Insert data into the new table. The result should match table **A**.
- Query the table to recreate table **B**.

A

	city character varying (30)	state character varying (30)	population integer
1	Alameda	California	79177
2	Mesa	Arizona	496401
3	Boerne	Texas	16056
4	Anaheim	California	352497
5	Tucson	Arizona	535677
6	Garland	Texas	238002

B

	city character varying (30)
1	Alameda
2	Mesa
3	Boerne
4	Anaheim
5	Tucson
6	Garland

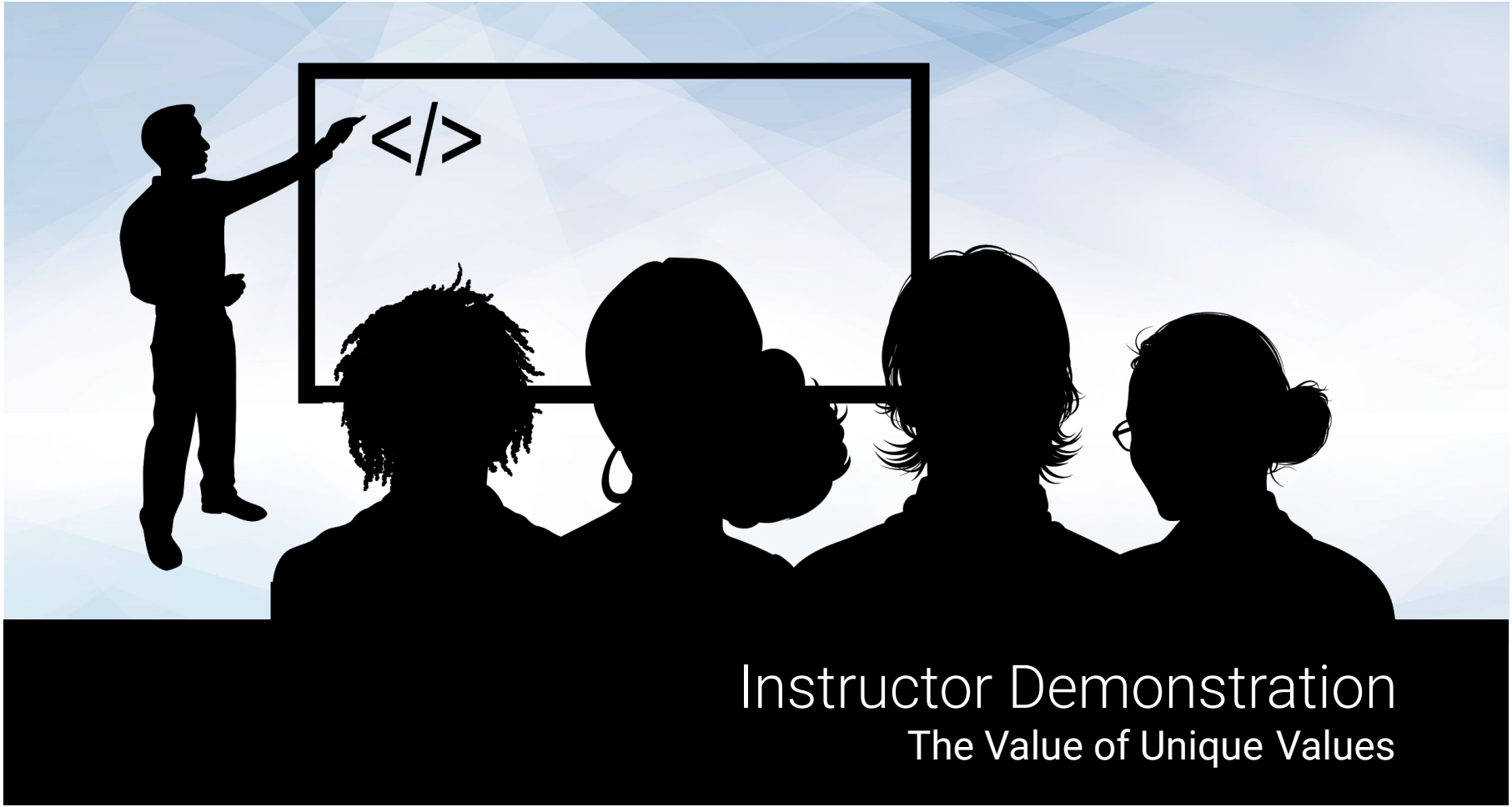
Activity: Creating Tables

- **Bonus:**
  - Filter the table to view only cities in Arizona.
  - Filter the table to view only cities with a population of less than 100,000.
  - Filter the table to view California cities with a population of less than 100,000.

- **Hints:**

  - For the second bonus question, you will need to use a WHERE clause to filter the original query.
  - For the third bonus question, an AND clause will also be necessary.

Time’s Up! Let’s Review.



Instructor Demonstration

The Value of Unique Values

The Value of Unique Values

- The query below will insert a duplicate data into our `people` table.

```
INSERT INTO people (name, has_pet, pet_type, pet_name, pet_age)
VALUES ('Ahmed', true, 'rock', 'Rockington', 100);
```

- Writing the query below will delete the duplicate entry.

```
DELETE FROM people
WHERE name = 'Ahmed';
```

  - In fact the duplicated data was deleted, so was the original.
  - The reason both was deleted is because both contains the name ‘Ahmed’.
  - SQL automatic assumes that the query is telling to delete every column containing the name ‘Ahmed’.

The Value of Unique Values

- `SERIAL` generates a new value for each inserted record in the table.
- By default, the starting value is 1, and it will increase by 1.
- The data type for the id column is

```sql
-- Re-create the table "people" within animals_db
CREATE TABLE people (
  id SERIAL PRIMARY KEY,
  name VARCHAR(30) NOT NULL,
  has_pet BOOLEAN DEFAULT false,
  pet_type VARCHAR(10) NOT NULL,
  pet_name VARCHAR(30),
  pet_age INT
);

-- Insert data into the table
INSERT INTO people (name, has_pet, pet_type, pet_name, pet_age)
VALUES ('Jacob', true, 'dog', 'Misty', 10),
  ('Ahmed', true, 'rock', 'Rockington', 100),
  ('Ahmed', true, 'rock', 'Rockington', 100),
  ('Peter', true, 'cat', 'Franklin', 2),
  ('Dave', true, 'dog', 'Queso', 1),
  ('Dave', true, 'dog', 'Pringles', 7);

-- Query all fields from the table
SELECT *
FROM people;
```

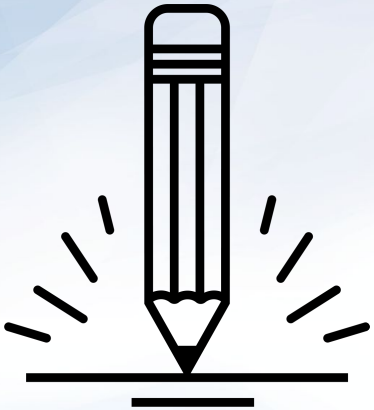
The Value of Unique Values

- This query delete duplicate data while preserving original data.

```sql
SELECT id, name, pet_name, pet_age
FROM people
WHERE name = 'Dave';

UPDATE people
SET has_pet = true, pet_name = 'Rocket', pet_age = 8
WHERE id = 6;

DELETE FROM people
WHERE id = 3;
```



Activity: Making and Using an ID

In this activity, you will recreate a table and then query, insert and update data.

Suggested Time:
10 Minutes

Activity: Making and Using an ID

- Create a new database named `programming_db`.
- Recreate the `programming_languages` table using the table below:

	id integer	language character varying (20)	rating integer
1	1	HTML	95
2	2	JS	99
3	3	JQuery	98
4	4	MySQL	70
5	5	MySQL	70

- Query the table to return the rows containing MySQL, and then delete one of the duplicates.
- Change the rating for HTML to 90.

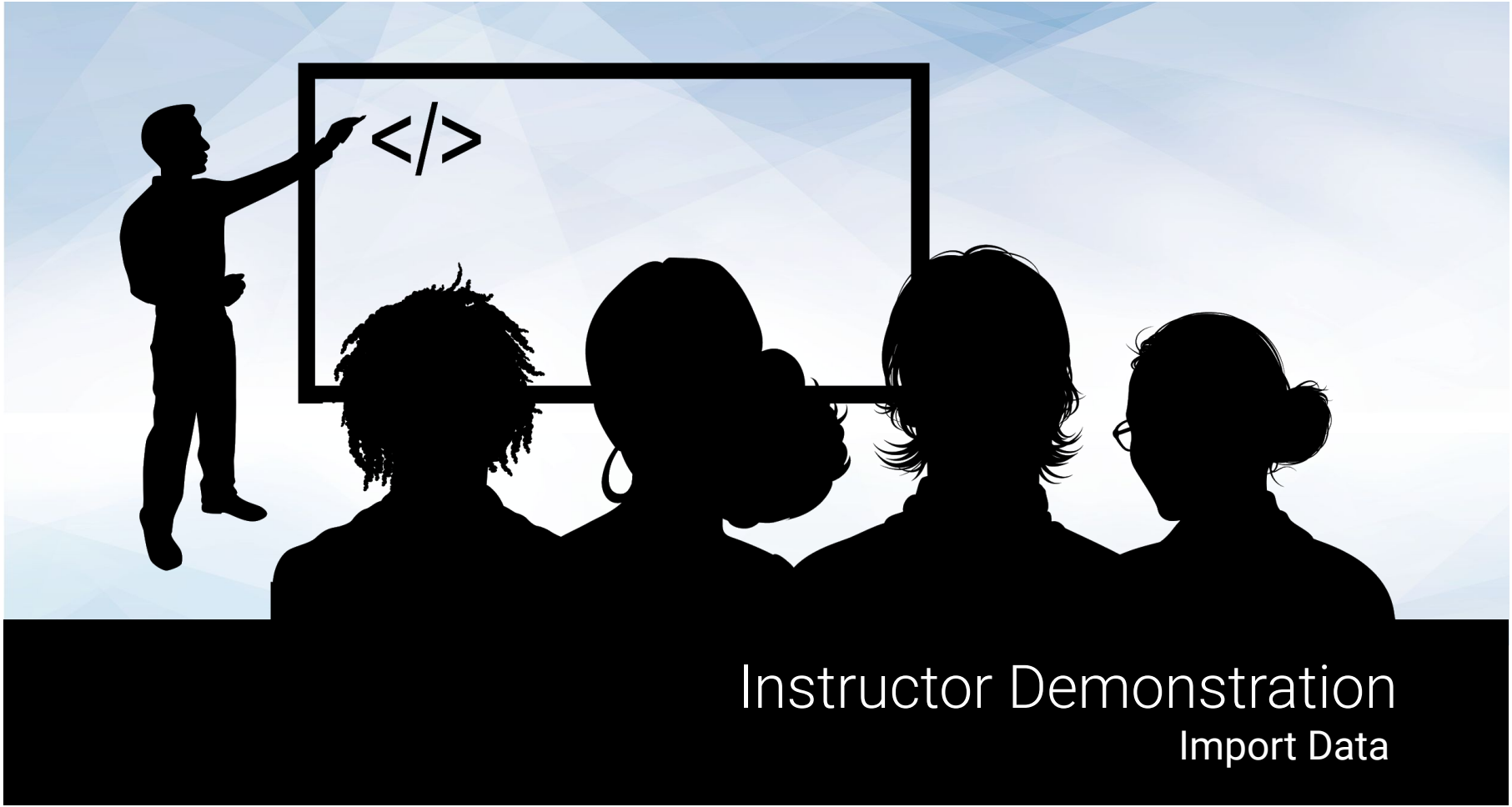
- Insert a few more rows of data for additional programming languages by adding the `language` and `rating` of your choice to the `programming_languages` table.
- Change the name of the JS language to JavaScript.
- **Bonus:**
    - Research how to add columns to a table. Then create a Boolean column named mastered that has a default value of true.
    - Start looking into the concept of joins in SQL. (This concept will be covered later in the lesson.)

Time’s Up! Let’s Review.

Countdown timer

15:00

(with alarm)



Instructor Demonstration

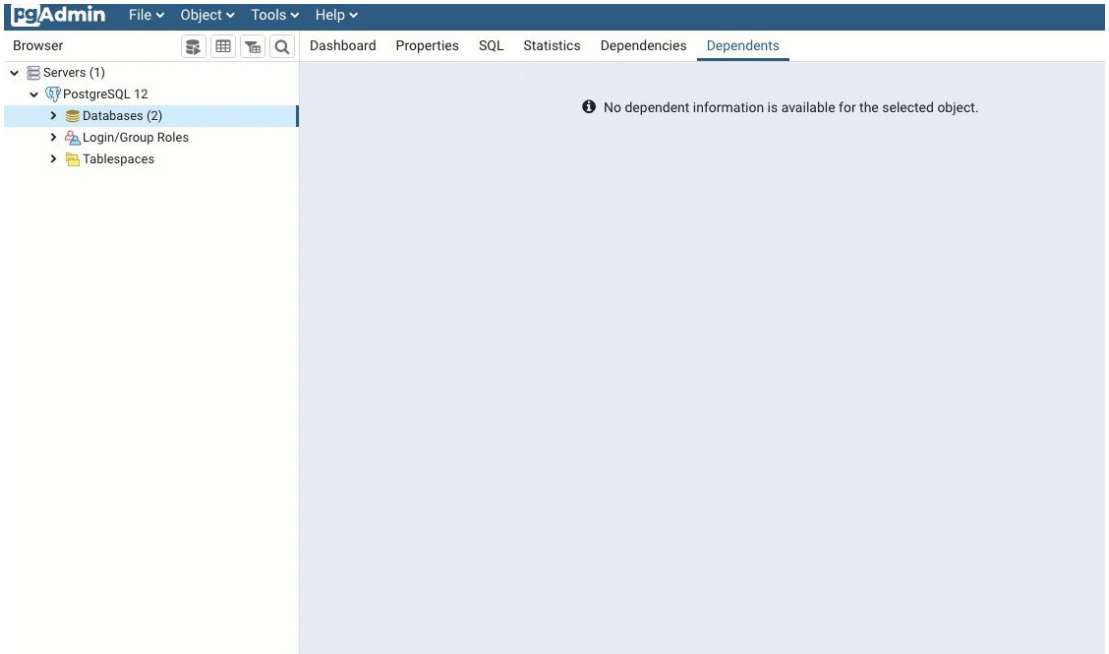
Import Data

Import Data

1. Create a new database.

   - Right-click **Databases.**

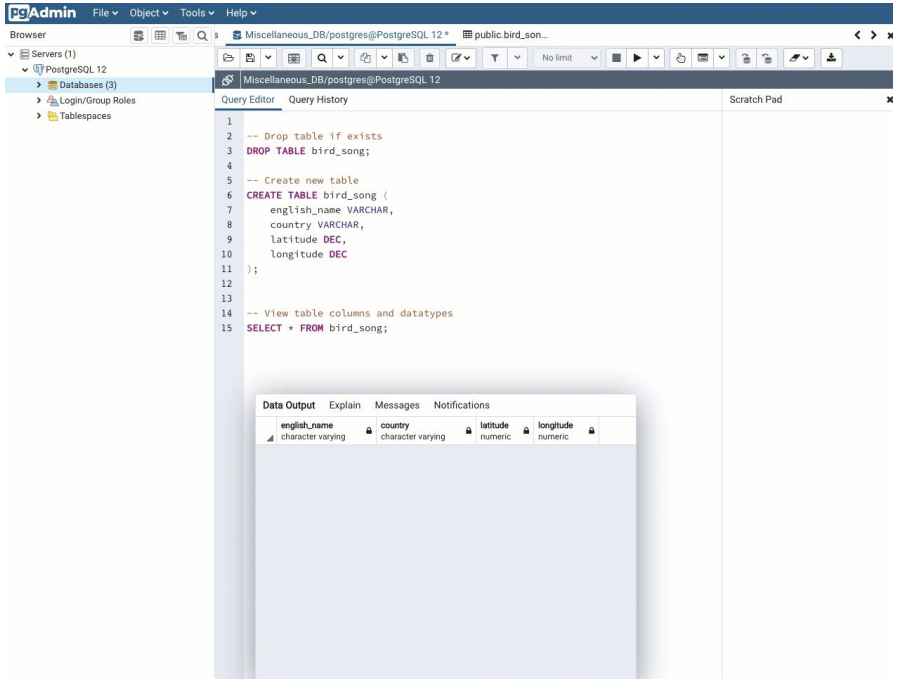
     - Create.
     - Insert database name.
     - Hit “**Save**”.



Import Data

2. Create a new table.

3. Import data.
   - Right-click **Miscellaneous_DB** from the left-hand menu and select Refresh.
   - Scroll down to **Schemas** and expand that menu, and then expand the Tables menu.
   - Right-click the new table and select **Import/Export** from the menu.
   - In the Options tab, complete the following steps:
     - Slide the **Import/Export** tab to Import
     - Click on the dot menu to navigate to the `birdsong.csv` file on your computer
     - Slide the Header tab to Yes.
     - Select the **comma** from the drop-down menu to set it as the Delimiter.
     - Leave the other fields as they are, and then click OK.



Import Data

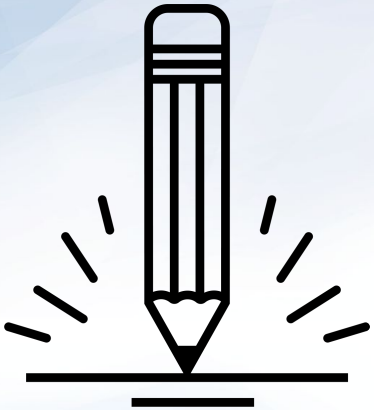
4. Verify data has been imported.

```
-- View table columns and datatypes
SELECT * FROM bird_song;
```

Data Output | Explain | **Messages** | Notifications

```
INSERT 0 100

Query returned successfully in 69 msec.
```



Activity: Hide and Seek

In this activity, you will create a new table and import data from a CSV file.

Suggested Time:
10 Minutes

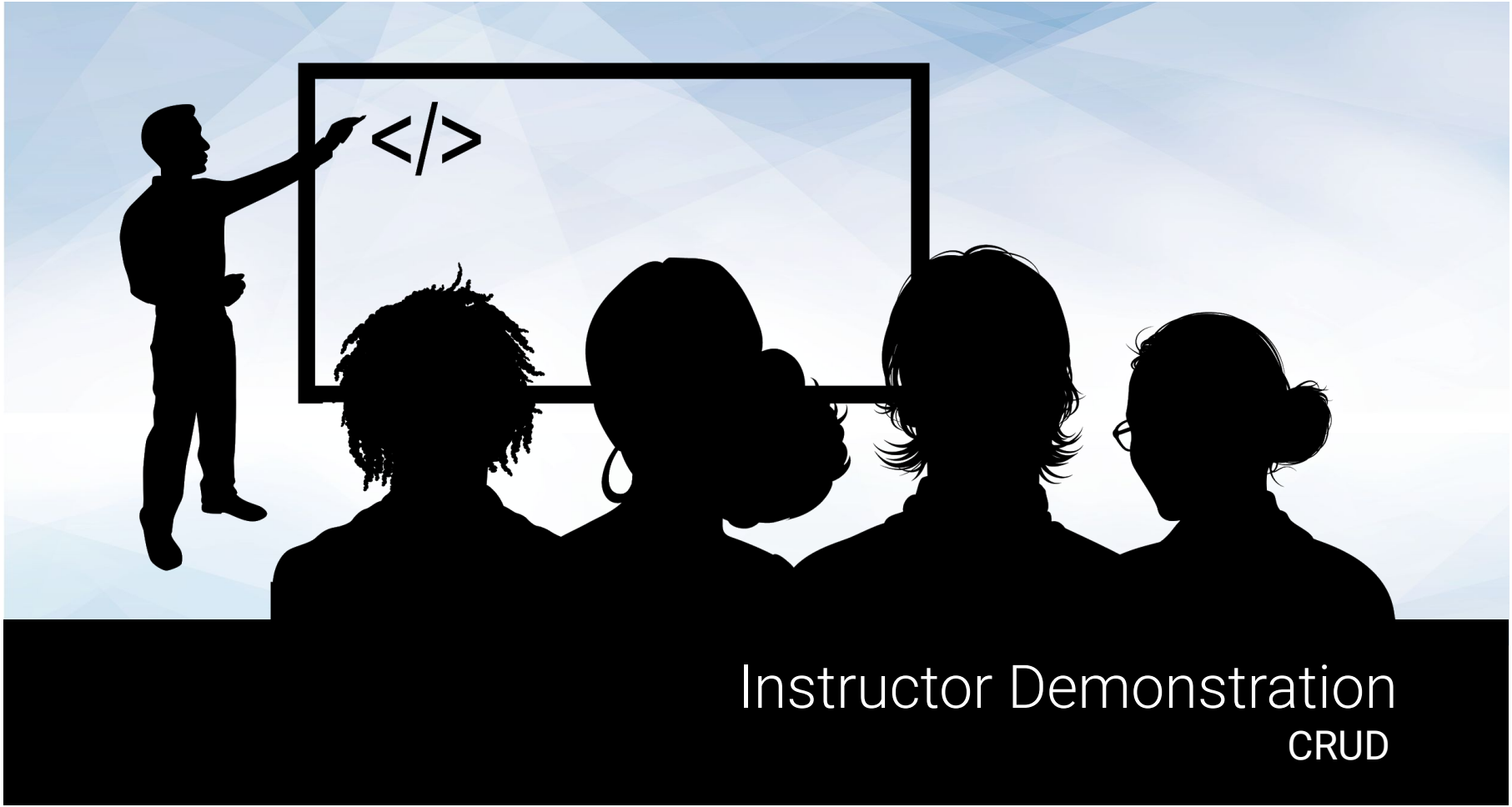
Activity: Hide and Seek

- Create a new table in the `Miscellaneous_DB` database called `wordassociation`.
- Import the data from the `wordassociation_AC.csv` file in the Resources folder.
- Create a query in which the data in the `word1` column is `stone`.
- Create a query that collects all rows in which the author is within the range 0–10.
- Create a query that searches for any rows that have `pie` in their `word1` or `word2` columns.

- **Bonus**:
  - Import to the `wordassociation_AC.csv` to the `wordassociation` table explore filtering on the `source` column.
  - Create a query that will collect all rows with a `source` of BC.
  - Create a query that will collect all rows with a `source` of BC and an author range between 333 and 335.

Note: Data provided by Kaggle.

Time’s Up! Let’s Review.



Instructor Demonstration

CRUD

CRUD
Create Read Update Delete

- While an unusual acronym, is a set of tools that are persistently used throughout programming. CRUD stands for Create, Read, Update, and Delete.

Create	INSERT table info (column1, column2, column3)
Read	SELECT * FROM table
Update	UPDATE table SET column1 = VALUE WHERE id = 1
Delete	DELETE FROM table WHERE id = 5

- These tools are fundamental to all programming languages—not just SQL.

Wildcard: % and _

- Wildcards are used to substitute zero, one, or multiple characters in a string. The keyword **LIKE** indicates the use of a wildcard.

```
SELECT *
FROM actor
WHERE last_name LIKE ‘Will%’;
```

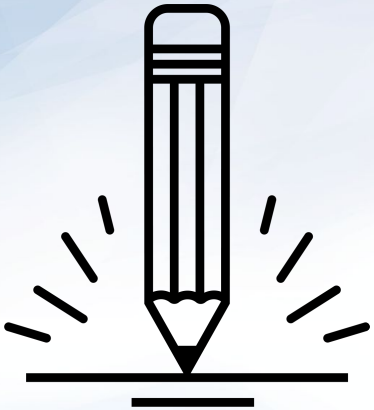
The **%** will substitute **zero**, **one**, or **multiple** characters in a query.

For example, all of the following will match: **Will**, **Willa**, and **Willows**.

```
SELECT *
FROM actor
WHERE first_name LIKE ‘_AN’;
```

The **_** will substitute one, and only one, character in a query.

_AN returns all actors whose first name contains three letters, the second and third of which are **AN**.



Activity: Using CRUD

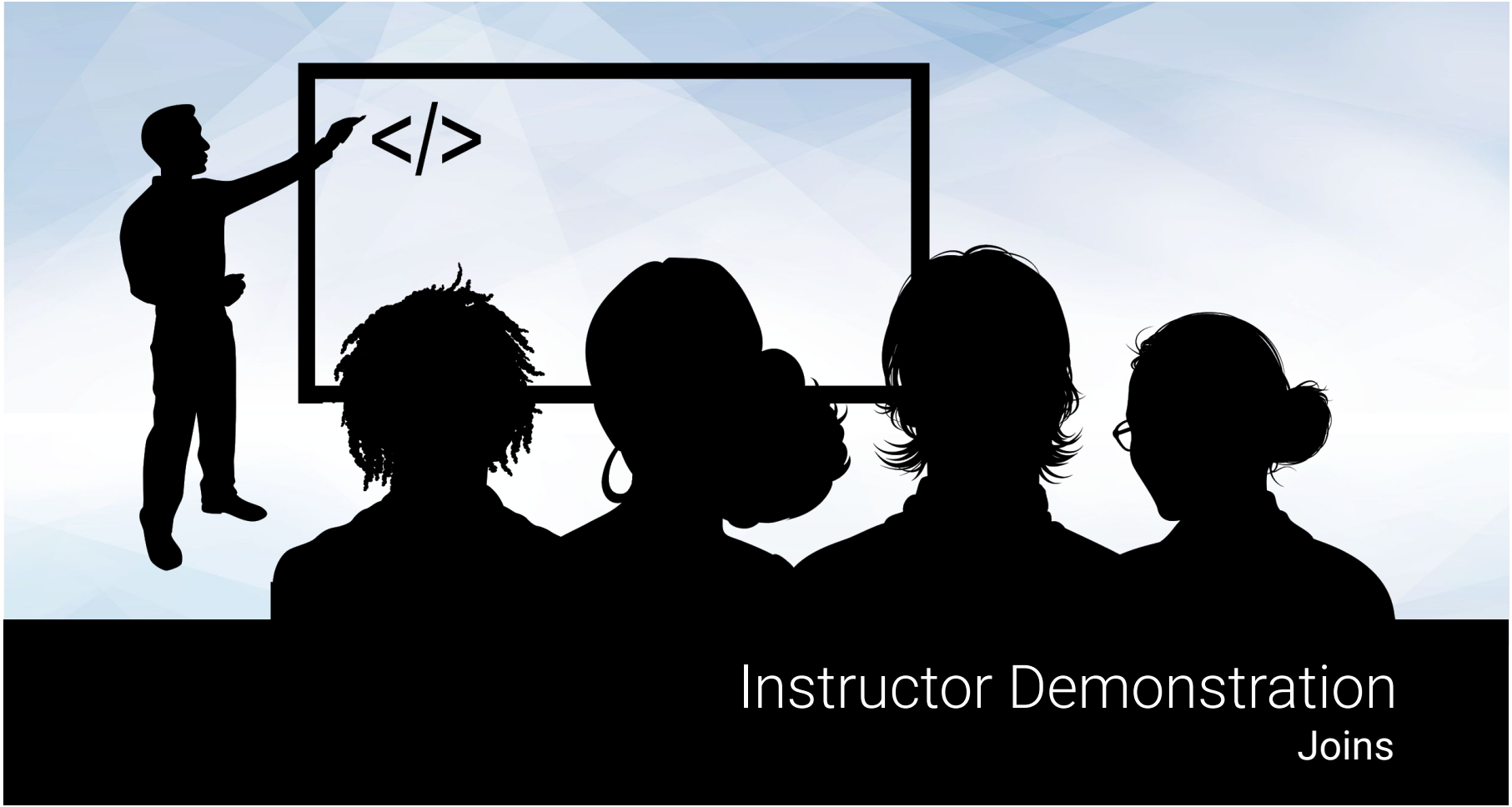
In this activity, you will utilize CRUD operations (Create, Read, Update, Destroy) on the provided data.

Suggested Time:
20 Minutes

Activity: Hide and Seek

- Create a new database named `GlobalFirePower` in pgAdmin.
- Create a table called `firepower` in the `GlobalFirePower` database by copying the code provided in `schema.sql` into a new query window in pgAdmin. Import the data from `GlobalFirePower.csv` using the Import/Export tool.
- Find the rows that have a `ReservePersonnel` of 0 and remove these rows from the dataset.
- Let's find which country only has one `FighterAircraft`, and take note of it.
- Every country in the world at least deserves one `FighterAircraft` — it only seems fair. Let's add one to each nation that has none.
- Oh no! By updating this column, the values within `TotalAircraftStrength` column are now off for those nations! We need to add 1 to the original number but not for the country that already had 1 `FighterAircraft`.
- Find the Averages for `TotalMilitaryPersonnel`, `TotalAircraftStrength`, `TotalHelicopterStrength`, and `TotalPopulation`, and rename the columns with their designated average.
- **Bonus:**
    - After creating your new nation and some parts of your military strategy, add the average values you calculated to the appropriate columns in the newly created rows. Update their values in any way you wish!

Time’s Up! Let’s Review.



Instructor Demonstration

Joins

Joins

- In our given scenario `player_id` column of the `players` table and the `loser_id/winner_id` columns of the `matches` table have matching values.
    - In that case we can join these tables together utilizing the **INNER JOIN**:

```
SELECT players.first_name, players.last_name, players.hand, matches.loser_rank
FROM matches
INNER JOIN players ON
players.player_id=matches.loser_id;
```

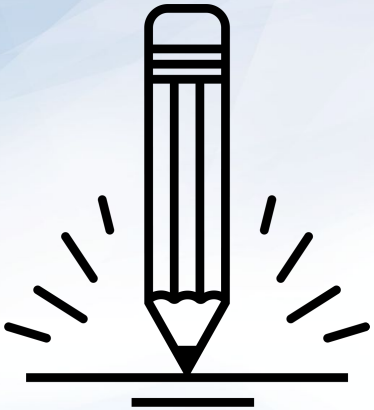
- A more advanced **INNER JOIN** solution.

```
-- Advanced INNER JOIN solution
SELECT p.first_name, p.last_name, p.hand, m.loser_rank
FROM matches AS m
INNER JOIN players AS p ON
p.player_id=m.loser_id;
```

Joins

The Primary Types of Joins used with PostgreSQL:

- `INNER JOIN` returns records that have matching values in both tables.
- `LEFT JOIN` returns all records from the left table and the matched records from the right table.
- `RIGHT JOIN` returns all records from the right table and the matched records from the left table.
- `CROSS JOIN` returns records that match every row of the left table with every row of the right table. This type of join has the potential to make very large tables.
- `FULL OUTER JOIN` places null values within the columns that do not match between the two tables, after an inner join is performed.



Activity: Joining the NBA

In this activity, you will be using joints to query NBA player seasonal statistics.

Suggested Time:
20 Minutes

Activity: Joining the NBA

- Create a new database named `NBA_DB` and create two new tables with pgAdmin named `players` and `seasons_stats`.
- Copy the code from `schema.sql` to create the tables, and then import the corresponding data from `Players.csv` and `Seasons_Stats.csv`.

Note: Remember to refresh the database; newly created tables will not immediately appear.

Activity: Joining the NBA

- Perform joins that will generate the following outputs.
  - Basic Information Table:

id integer	player character varying	height integer	weight integer	college character varying	born integer	position character varying	tm character varying
0	Cliff Barker	188	83	University of Kentucky	1921	SG	INO
0	Cliff Barker	188	83	University of Kentucky	1921	SG	INO
0	Cliff Barker	188	83	University of Kentucky	1921	SG	INO
1	Ralph Beard	178	79	University of Kentucky	1927	G	INO
1	Ralph Beard	178	79	University of Kentucky	1927	G	INO
2	Charlie Black	196	90	University of Kansas	1921	F-C	TOT

  - Percents Stats:

player_id integer	college character varying	year numeric	position character varying	two_point_percentage numeric	fg_percentage numeric	ft_percentage numeric	ts_percentage numeric
0	University of Kentucky	1950	SG	0.372	0.372	0.708	0.435
0	University of Kentucky	1951	SG	0.252	0.252	0.649	0.322
0	University of Kentucky	1952	SG	0.298	0.298	0.588	0.343
1	University of Kentucky	1950	G	0.363	0.363	0.762	0.422
1	University of Kentucky	1951	G	0.368	0.368	0.775	0.435
2	University of Kansas	1950	F-C	0.278	0.278	0.651	0.346

Time’s Up! Let’s Review.


The End