Group: 4

Aquila Mucubaquire

Avinash Bunga

Ebenezer Amo

Selorm Kwaku Soga

Information Systems and Business Analytics, Park University

CIS622DLAF2P2023 Data Architecture for Business Analytics

Professor: Gulnoza Khakimova

Nov 19, 2023

Unit 5: Show-and-Tell

Detailed Summary for Unit 5 Show-and-Tell

It is my pleasure to submit our detailed summary, reflecting the diligent work we have done with SQL using the PgAdmin tool and the insights we have gathered on the utility of Python for database-related tasks.

Introduction to Database Creation with PgAdmin:

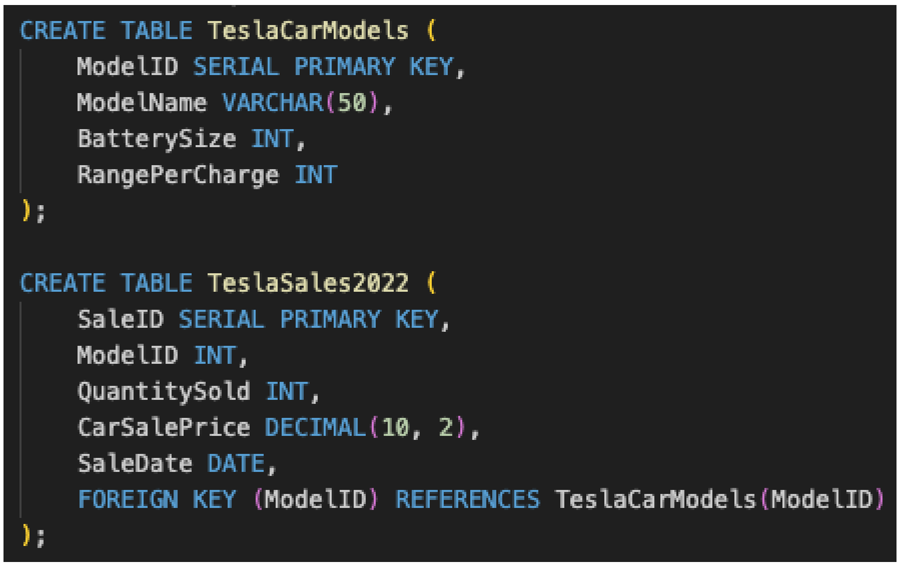
Our journey began with creating a database named “**Tesla**” in PgAdmin Using its GUI. This was a crucial first step to organize our data infrastructure, which would store and relate information about Tesla car models and their corresponding sales data for 2022, adhering to the following steps:

1. Initiated pgAdmin and connected to the PostgreSQL server.
2. Selected the 'Databases' node, right-clicked‘ and chose 'Create > Database'.
3. Provided the database name "**Tesla**" and confirmed the creation parameters.

Creating Relational Tables in SQL:

Our SQL journey began with the creation of two interconnected tables. The first, “**TeslaCarModels”**, was devised to hold data on various Tesla car models, while the second, “**TeslaSales2022”**, recorded sales data. The relationship was established through a primary and foreign key to maintain referential integrity. Here are the queries that laid the foundation for our relational database schema, adhering to the following steps:

The 'Query Tool' was used to execute the SQL script, thereby creating the schema's tables and relationships.



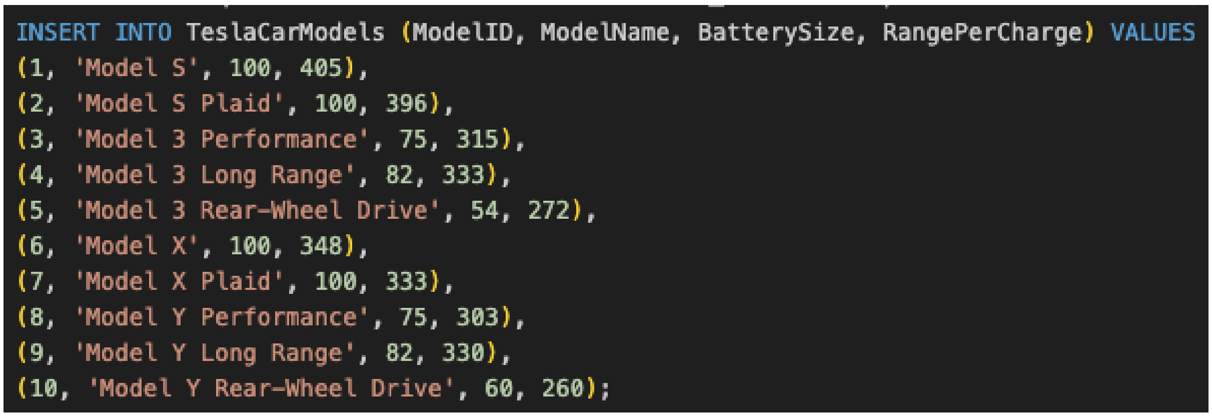
```sql
CREATE TABLE TeslaCarModels (
    ModelID SERIAL PRIMARY KEY,
    ModelName VARCHAR(50),
    BatterySize INT,
    RangePerCharge INT
);

CREATE TABLE TeslaSales2022 (
    SaleID SERIAL PRIMARY KEY,
    ModelID INT,
    QuantitySold INT,
    CarSalePrice DECIMAL(10, 2),
    SaleDate DATE,
    FOREIGN KEY (ModelID) REFERENCES TeslaCarModels(ModelID)
);
```

Data Insertion into Tables:

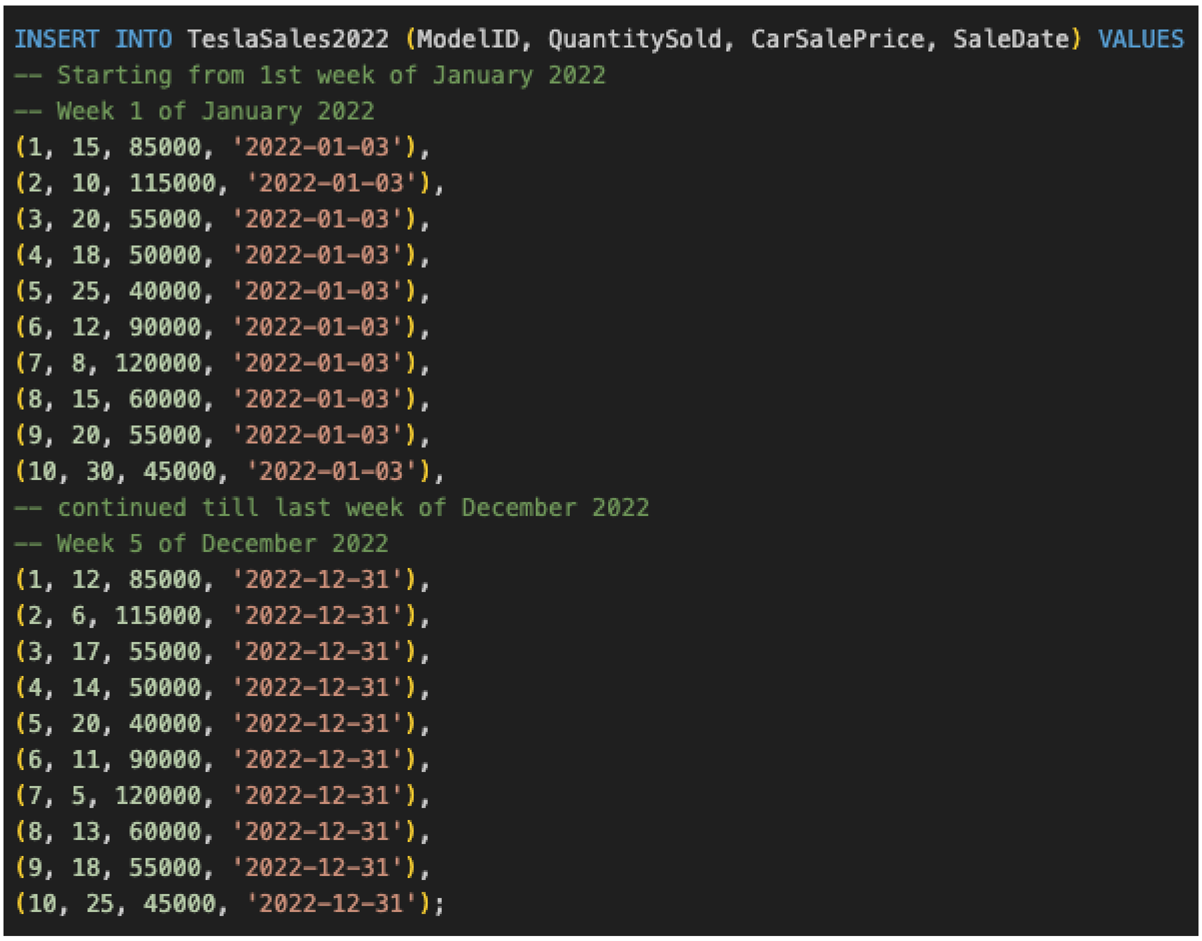
Once our tables were structured, we populated “**TeslaCarModels”** with a rich dataset detailing the specifications of various Tesla models. Simultaneously, “**TeslaSales2022”** was filled with granular weekly sales data, painting a complete picture of the year's sales trends.

- SQL query for data insertion into “**TeslaCarModels**”



```sql
INSERT INTO TeslaCarModels (ModelID, ModelName, BatterySize, RangePerCharge) VALUES
(1, 'Model S', 100, 405),
(2, 'Model S Plaid', 100, 396),
(3, 'Model 3 Performance', 75, 315),
(4, 'Model 3 Long Range', 82, 333),
(5, 'Model 3 Rear-Wheel Drive', 54, 272),
(6, 'Model X', 100, 348),
(7, 'Model X Plaid', 100, 333),
(8, 'Model Y Performance', 75, 303),
(9, 'Model Y Long Range', 82, 330),
(10, 'Model Y Rear-Wheel Drive', 60, 260);
```

- Sample SQL query for data insertion into “**TeslaSales2022**”

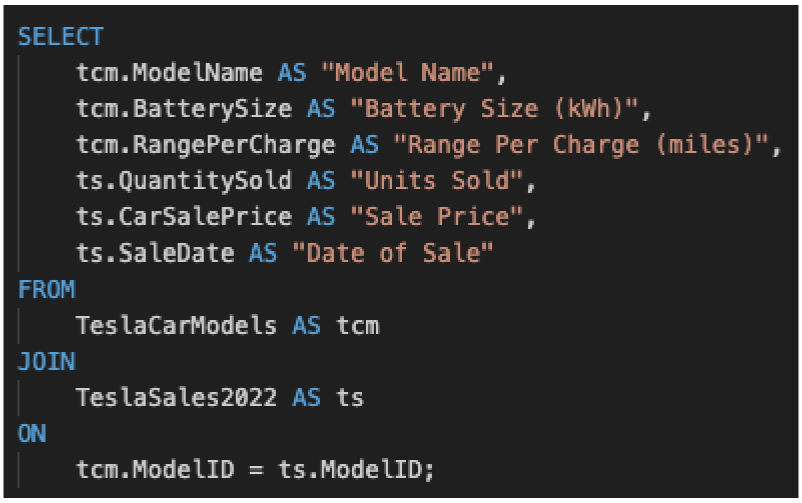


```sql
INSERT INTO TeslaSales2022 (ModelID, QuantitySold, CarSalePrice, SaleDate) VALUES
-- Starting from 1st week of January 2022
-- Week 1 of January 2022
(1, 15, 85000, '2022-01-03'),
(2, 10, 115000, '2022-01-03'),
(3, 20, 55000, '2022-01-03'),
(4, 18, 50000, '2022-01-03'),
(5, 25, 40000, '2022-01-03'),
(6, 12, 90000, '2022-01-03'),
(7, 8, 120000, '2022-01-03'),
(8, 15, 60000, '2022-01-03'),
(9, 20, 55000, '2022-01-03'),
(10, 30, 45000, '2022-01-03'),
-- continued till last week of December 2022
-- Week 5 of December 2022
(1, 12, 85000, '2022-12-31'),
(2, 6, 115000, '2022-12-31'),
(3, 17, 55000, '2022-12-31'),
(4, 14, 50000, '2022-12-31'),
(5, 20, 40000, '2022-12-31'),
(6, 11, 90000, '2022-12-31'),
(7, 5, 120000, '2022-12-31'),
(8, 13, 60000, '2022-12-31'),
(9, 18, 55000, '2022-12-31'),
(10, 25, 45000, '2022-12-31');
```

DriveLink for the full query.

Joining Tables for Consolidated Insights:

We crafted a SQL join statement to merge the model details with sales data. This allowed us to view and analyze the data cohesively, linking each sale to its specific car model.



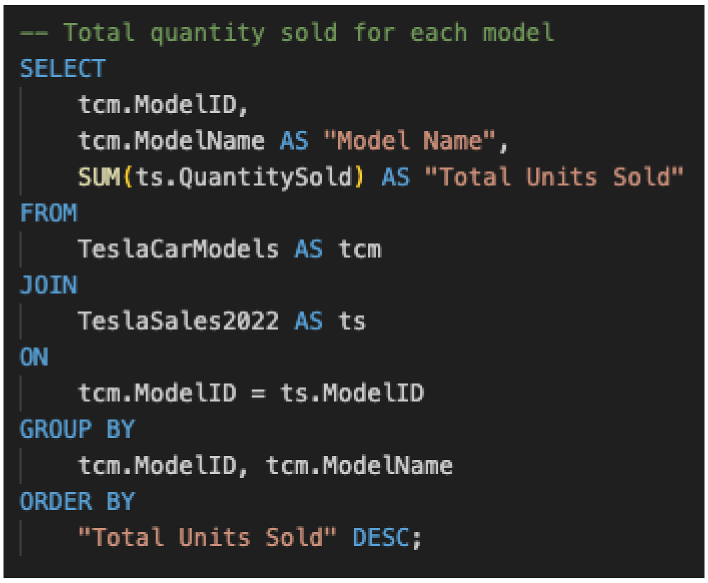
```sql
SELECT
    tcm.ModelName AS "Model Name",
    tcm.BatterySize AS "Battery Size (kWh)",
    tcm.RangePerCharge AS "Range Per Charge (miles)",
    ts.QuantitySold AS "Units Sold",
    ts.CarSalePrice AS "Sale Price",
    ts.SaleDate AS "Date of Sale"
FROM
    TeslaCarModels AS tcm
JOIN
    TeslaSales2022 AS ts
ON
    tcm.ModelID = ts.ModelID;
```

DriveLink for the full query.

Enhancing Our Database with Additional SQL Operations:

We did not stop at simple joins; we pushed our SQL skills further by conducting additional operations:

1. Aggregating the total quantity of each model sold.

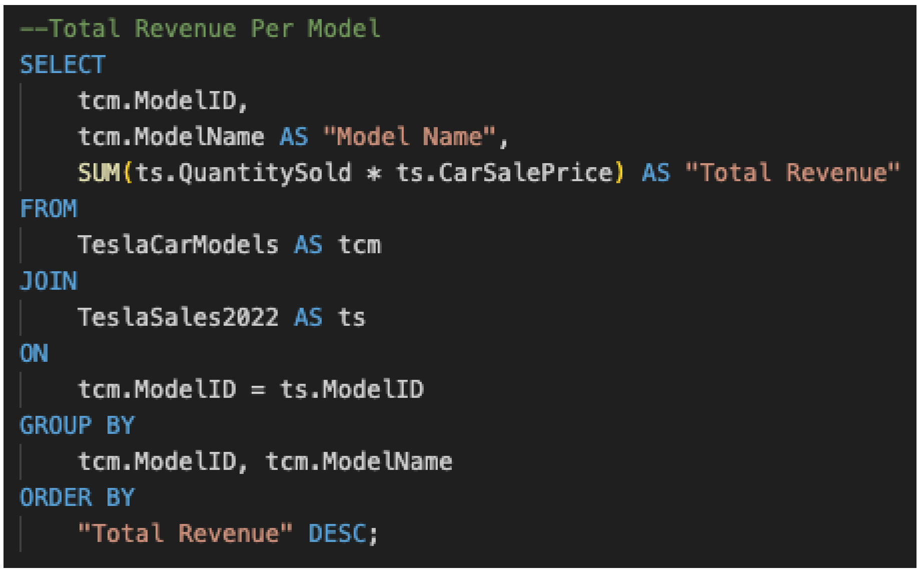


```sql
-- Total quantity sold for each model
SELECT
    tcm.ModelID,
    tcm.ModelName AS "Model Name",
    SUM(ts.QuantitySold) AS "Total Units Sold"
FROM
    TeslaCarModels AS tcm
JOIN
    TeslaSales2022 AS ts
ON
    tcm.ModelID = ts.ModelID
GROUP BY
    tcm.ModelID, tcm.ModelName
ORDER BY
    "Total Units Sold" DESC;
```

DriveLink for the full query.

2. Calculating the total revenue generated per model.

These additional SQL operations provided us with deeper business intelligence and showcased the real-world utility of our database.



```sql
--Total Revenue Per Model
SELECT
    tcm.ModelID,
    tcm.ModelName AS "Model Name",
    SUM(ts.QuantitySold * ts.CarSalePrice) AS "Total Revenue"
FROM
    TeslaCarModels AS tcm
JOIN
    TeslaSales2022 AS ts
ON
    tcm.ModelID = ts.ModelID
GROUP BY
    tcm.ModelID, tcm.ModelName
ORDER BY
    "Total Revenue" DESC;
```

DriveLink for the full query.

Python’s Edge Over SQL:

While SQL is potent for database manipulation, Python shines with its ability to handle data beyond the confines of databases. We found Python, with its libraries like Pandas, to be exceptionally beneficial for tasks such as data cleaning, transformation, advanced analysis, and automation—capabilities that SQL is not designed to handle natively.

Conclusion:

Our SQL-driven project adeptly demonstrates database management and complex querying, while Python's prowess in broader data processing underscores its complementary role. Together, they form a powerful duo for data analytics. For a complete overview of our SQL queries and methodical process, please refer to the complete SQL Query File.

I am immensely proud of our team's commitment and collaborative achievements.

Best Regards,

Avinash Bunga

Unit 5 - CIS622

Reference:

Samarth, V. (2023, October 25). *SQL vs Python: Finding the Best Language for Your Need and Career*. Emeritus. https://emeritus.org/in/learn/data-science-sql-vs-python/