DSA 610 Redesign, Lecture 6 Outline

Lecture Outline: SQL Queries with Examples
Duration: 50 minutes

---

1. Introduction to SQL (5 minutes)

- **Objective:** Provide a brief overview of SQL, its purpose, and common uses.
- **Content:**
    - **Definition:**
        - **SQL (Structured Query Language):** A domain-specific language used to manage and manipulate relational databases.
    - **Key Uses:**
        - **Data Retrieval:** Querying databases to extract data.
        - **Data Manipulation:** Inserting, updating, and deleting records.
        - **Data Definition:** Creating and modifying database schemas.
        - **Data Control:** Managing access and permissions to data.
    - **Example Databases:**
        - **MySQL, PostgreSQL, SQLite, SQL Server, Oracle**

---

2. Basic SQL Query Structure (5 minutes)

- **Objective:** Understand the fundamental components of an SQL query.
- **Content:**
    - **SQL Syntax:**
        - **SELECT:** Specifies the columns to retrieve.
        - **FROM:** Indicates the table(s) from which to retrieve data.
        - **WHERE:** Filters rows based on specified conditions.
        - **ORDER BY:** Sorts the result set by one or more columns.
        - **LIMIT/OFFSET:** Limits the number of rows returned and skips a specified number of rows.
    - **Example:**

```
SELECT column1, column2
FROM table_name
WHERE condition
ORDER BY column1 DESC
LIMIT 10 OFFSET 5;
```

Basic Data Retrieval Queries (10 minutes)

- **Objective:** Learn how to retrieve data from a single table using simple SQL queries.
- **Content:**
    - **SELECT Statement:**
        - **Selecting Specific Columns:**
            - Retrieve specific columns from a table.
            - **Example:**

```
SELECT first_name, last_name
FROM employees;
```

Selecting All Columns:

- Retrieve all columns from a table using *.

- **Example:**

```
SELECT *
FROM employees;
```

Filtering Results with WHERE:

- **Basic Conditions:** Using equality, inequality, and comparison operators.
- **Example:**

```
SELECT first_name, last_name
FROM employees
WHERE department = 'Sales' AND salary > 50000;
```

Using IN, BETWEEN, and LIKE:

- **IN:** Filtering based on a list of values.
- **BETWEEN:** Filtering within a range.
- **LIKE:** Pattern matching with wildcards (% and _).
- **Examples:**

```
SELECT first_name, last_name
FROM employees
WHERE department IN ('Sales', 'Marketing');

SELECT first_name, last_name
FROM employees
WHERE salary BETWEEN 40000 AND 60000;

SELECT first_name, last_name
FROM employees
WHERE first_name LIKE 'J%';
```

Aggregation and Grouping (10 minutes)

- **Objective:** Perform aggregation operations and group data for analysis.
- **Content:**
    - **Aggregate Functions:**
        - **COUNT, SUM, AVG, MIN, MAX:** Functions to summarize data.
        - **Examples:**

```
SELECT COUNT(*) AS total_employees
FROM employees;

SELECT AVG(salary) AS average_salary
FROM employees;

SELECT MIN(salary) AS lowest_salary, MAX(salary) AS highest_salary
FROM employees;
```

GROUP BY Clause:

- **Grouping Data:** Group data by one or more columns and apply aggregate functions.
- **HAVING Clause:** Filtering groups based on aggregate conditions.
- **Examples:**

```
SELECT department, COUNT(*) AS num_employees
FROM employees
GROUP BY department;

SELECT department, AVG(salary) AS average_salary
FROM employees
GROUP BY department
HAVING AVG(salary) > 50000;
```

Joining Tables (10 minutes)

- **Objective:** Learn how to combine data from multiple tables using various types of joins.
- **Content:**
    - **Types of Joins:**
        - **INNER JOIN:** Returns rows that have matching values in both tables.
        - **LEFT (OUTER) JOIN:** Returns all rows from the left table and the matched rows from the right table.
        - **RIGHT (OUTER) JOIN:** Returns all rows from the right table and the matched rows from the left table.
        - **FULL (OUTER) JOIN:** Returns all rows when there is a match in either table.
    - **Examples:**

```
-- Inner Join Example
SELECT employees.first_name, employees.last_name, departments.department_name
FROM employees
INNER JOIN departments ON employees.department_id = departments.department_id;

-- Left Join Example
SELECT employees.first_name, employees.last_name, departments.department_name
FROM employees
LEFT JOIN departments ON employees.department_id = departments.department_id;

-- Full Join Example
SELECT employees.first_name, employees.last_name, departments.department_name
FROM employees
FULL JOIN departments ON employees.department_id = departments.department_id;
```

Cross Join:

- **Definition:** Returns the Cartesian product of two tables.
- **Example:**

```
SELECT a.product_name, b.store_name
FROM products a
CROSS JOIN stores b;
```

Subqueries and Nested Queries (5 minutes)

- **Objective:** Learn how to use subqueries to make complex queries more efficient.
- **Content:**
    - **Subqueries in SELECT:**
        - Using a subquery to calculate a value within a larger query.
        - **Example:**

```
SELECT first_name, last_name, (SELECT department_name
                               FROM departments
                               WHERE employees.department_id = departments.department_id) AS department
FROM employees;
```

Subqueries in WHERE:

- Filtering results based on a subquery.
- **Example:**

```
SELECT first_name, last_name
FROM employees
WHERE salary > (SELECT AVG(salary) FROM employees);
```

Practical Examples and Q&A (5 minutes)

- **Objective:** Reinforce learning with practical examples and address any questions.
- **Content:**
    - **Examples:** Go through practical examples with the class, answering any queries that arise.
        - **Complex Query Example:** Combining JOINs, GROUP BY, and subqueries.
        - **Hands-On Exercise:** Challenge students with a query to write on their own.
    - **Q&A Session:** Encourage students to ask questions about specific SQL concepts or queries they struggle with.

Key Takeaways

- **Basic Queries:** Understanding how to retrieve and filter data using SQL.
- **Aggregation:** Ability to summarize and group data for analysis.
- **Joins:** Combining data from multiple tables using different types of joins.
- **Subqueries:** Using nested queries to solve complex problems.

Resources:
SQL Tutorials: https://www.w3schools.com/sql/, https://www.sqltutorial.org/, https://www.sqltutorial.org/, https://www.tutorialspoint.com/sql/index.htm

Lecture Outline: Joining Tables and Aggregation in Data Analysis
Duration: 50 minutes

---

1. Introduction to Joining Tables in Data Analysis (5 minutes)

- **Objective:** Provide context on why joining tables is essential in data analysis.
- **Content:**
    - **Definition:**

        - **Joining Tables:** Combining data from two or more tables based on a related column.
    - **Purpose in Data Analysis:**
        - **Data Integration:** Merging different data sources to create a unified dataset.
        - **Enrichment:** Adding additional information to a dataset by pulling in related data (e.g., customer demographics, product details).
        - **Handling Normalized Data:** Working with relational databases where data is divided into multiple tables to reduce redundancy.
    - **Examples of Use Cases:**
        - **Customer Transactions:** Joining sales records with customer information to analyze purchasing behavior.
        - **Employee Performance:** Merging employee details with performance metrics for a comprehensive view.

---

2. Types of Joins and Their Applications (15 minutes)

- **Objective:** Understand different types of SQL joins and how they are applied in data analysis.
- **Content:**
    - **INNER JOIN:**
        - **Definition:** Returns rows with matching values in both tables.
        - **Use Case:** Analyzing records where there is complete information in both tables (e.g., orders with customer information).
        - **Example:**

```
SELECT orders.order_id, customers.customer_name
FROM orders
INNER JOIN customers ON orders.customer_id = customers.customer_id;
```

LEFT (OUTER) JOIN:

- **Definition:** Returns all rows from the left table and matched rows from the right table.
- **Use Case:** Retaining all records from the primary table, even if there's no corresponding data in the joined table (e.g., all customers, including those with no orders).
- **Example:**

```
SELECT customers.customer_name, orders.order_id
FROM customers
LEFT JOIN orders ON customers.customer_id = orders.customer_id;
```

RIGHT (OUTER) JOIN:

- **Definition:** Returns all rows from the right table and matched rows from the left table.
- **Use Case:** Less common, but useful in specific situations where the right table is primary.
- **Example:**

```
SELECT orders.order_id, customers.customer_name
FROM orders
RIGHT JOIN customers ON orders.customer_id = customers.customer_id;
```

FULL (OUTER) JOIN:

- **Definition:** Returns all rows when there is a match in either table.
- **Use Case:** Analyzing datasets where you want to retain all information from both tables, identifying where data may be missing on either side.

- **Example:**

```
SELECT customers.customer_name, orders.order_id
FROM customers
FULL OUTER JOIN orders ON customers.customer_id = orders.customer_id;
```

-
    - **Practical Examples:**
        - **Merging Sales Data with Product Information:** To analyze total sales per product category.
        - **Combining User Data with Website Activity Logs:** To assess user engagement.

---

3. Introduction to Aggregation in Data Analysis (5 minutes)

- **Objective:** Explain the purpose and importance of aggregation in data analysis.
- **Content:**
    - **Definition:**
        - **Aggregation:** Summarizing data by grouping and applying aggregate functions (e.g., SUM, AVG, COUNT).
    - **Purpose:**
        - **Data Summarization:** Reducing the dataset to focus on key metrics (e.g., total sales, average salary).
        - **Identifying Trends:** Aggregating data over time or categories to uncover patterns.
        - **Data Reduction:** Simplifying large datasets for easier analysis and visualization.
    - **Use Cases in Data Analysis:**
        - **Monthly Revenue Reports:** Summarizing daily sales data into monthly totals.
        - **Customer Segmentation:** Calculating average spend per customer segment.
        - **Operational Efficiency:** Summarizing production data to evaluate efficiency.

---

4. Aggregate Functions and Grouping Data (15 minutes)

- **Objective:** Learn how to apply aggregate functions and group data for analysis.
- **Content:**
    - **Common Aggregate Functions:**
        - **COUNT:** Counting the number of rows or occurrences.
        - **SUM:** Adding up numerical values.
        - **AVG:** Calculating the average of a set of values.
        - **MIN/MAX:** Finding the smallest or largest values.
        - **Example:**

```
SELECT department, COUNT(*) AS employee_count, AVG(salary) AS average_salary
FROM employees
GROUP BY department;
```

GROUP BY Clause:

- **Purpose:** Grouping data by one or more columns to apply aggregate functions.
- **Using HAVING:** Filtering groups based on aggregate conditions.
- **Example:**

```
SELECT product_category, SUM(sales) AS total_sales
FROM sales_data
GROUP BY product_category
HAVING SUM(sales) > 10000;
```

-
    - **Practical Example:**
        - **Sales Data Analysis:** Grouping sales by region and product category to identify top-performing regions and products.

---

5. Working with Aggregated Data in Analysis (10 minutes)

- **Objective:** Understand how to interpret and use aggregated data for further analysis and decision-making.
- **Content:**
    - **Interpretation of Aggregated Data:**
        - **Trends and Patterns:** Recognizing key trends in the summarized data.
        - **Comparisons:** Comparing different groups or categories based on the aggregated metrics.
        - **Example:** Using average sales per region to decide where to allocate marketing resources.
    - **Challenges and Considerations:**
        - **Loss of Detail:** Aggregation can obscure granular insights; it's important to balance summary with detail.
        - **Outliers and Anomalies:** Aggregated data might mask outliers, so it's essential to analyze these separately if necessary.
    - **Visualization of Aggregated Data:**
        - **Bar Charts, Pie Charts, Line Graphs:** Common ways to visualize aggregated data.
        - **Example:** Visualizing total sales per month with a line graph to observe trends over time.
    - **Case Study:** Analyzing aggregated data in a business context to make strategic decisions (e.g., identifying top-performing products).

---

6. Practical Implementation & Q&A (5 minutes)

- **Objective:** Apply the concepts through a hands-on example and address any questions.
- **Content:**
    - **Example:** A brief exercise where students write queries to join tables and aggregate data.
    - **Q&A Session:** Open the floor for questions, discussing any challenges students face when working with joins and aggregations.

Key Takeaways

- **Joining Tables:** Understanding the importance of combining data from multiple sources for comprehensive analysis.
- **Aggregation:** Learning how to summarize and interpret data to identify trends and make informed decisions.
- **SQL Techniques:** Gaining practical skills in writing SQL queries for joining tables and aggregating data.

Resources:
Merge, Join, Concatenate and Compare in Pandas:
https://pandas.pydata.org/docs/user_guide/merging.html
SQL Using Python: https://www.geeksforgeeks.org/sql-using-python/
Python SQLite3: https://www.geeksforgeeks.org/python-sqlite/
Working with SQLite3: https://www.freecodecamp.org/news/work-with-sqlite-in-python-handbook/

Lecture Outline: Data Transformation, Feature Engineering, Dummy Variables, and Working with SQL in Python
Duration: 50 minutes

---

1. Introduction to Data Transformation in Python (10 minutes)

- **Objective:** Understand the basics of data transformation and its importance in data analysis.
- **Content:**
  - **Definition:**
    - **Data Transformation:** The process of converting data from one format or structure into another.
  - **Importance in Data Analysis:**
    - **Preparing Data for Modeling:** Ensuring that data is in the right format for machine learning models.
    - **Enhancing Data Quality:** Cleaning and structuring data to improve the accuracy of analysis.
  - **Common Data Transformation Techniques:**
    - **Scaling and Normalization:** Adjusting the scale of data.
    - **Handling Missing Data:** Filling in or removing missing values.
    - **Data Type Conversion:** Changing data types for compatibility (e.g., converting strings to integers).
  - **Example:** Importing a dataset and applying basic transformations using pandas.

```
import pandas as pd

# Load dataset
df = pd.read_csv('data.csv')

# Scaling and normalization
df['scaled_column'] = (df['original_column'] - df['original_column'].min()) / (df['original_column'].max() - df['original_column'].min())

# Handling missing data
df.fillna(0, inplace=True)

# Data type conversion
df['date_column'] = pd.to_datetime(df['date_column'])
```

Feature Engineering (10 minutes)

- **Objective:** Learn how to create new features from existing data to improve model performance.
- **Content:**
  - **Definition:**

        - **Feature Engineering:** The process of using domain knowledge to create new input features that enhance the performance of a machine learning model.
    - **Why Feature Engineering Matters:**
        - **Improving Model Accuracy:** Better features can lead to more accurate predictions.
        - **Capturing Complex Patterns:** Creating features that help the model understand the data better.
    - **Common Feature Engineering Techniques:**
        - **Creating Interaction Terms:** Combining features to capture relationships.
        - **Polynomial Features:** Raising features to a power to capture non-linear relationships.
        - **Extracting Date Components:** Breaking down dates into day, month, year, etc.
        - **Example:**

```
# Interaction term
df['interaction_feature'] = df['feature1'] * df['feature2']

# Polynomial feature
df['poly_feature'] = df['original_feature'] ** 2

# Extracting date components
df['year'] = df['date_column'].dt.year
df['month'] = df['date_column'].dt.month
df['day'] = df['date_column'].dt.day
```

Dummy Variables (One-Hot Encoding) (10 minutes)

- **Objective:** Understand how to convert categorical variables into a format that can be provided to machine learning algorithms.
- **Content:**
    - **Definition:**
        - **Dummy Variables:** Binary variables created from categorical variables to represent the presence or absence of a category.
    - **Why Use Dummy Variables:**
        - **Compatibility with Models:** Many machine learning models require numerical input, so categorical data must be transformed.
    - **How to Create Dummy Variables:**
        - **One-Hot Encoding:** Creating a new binary column for each category.
        - **Example:**

```
# Categorical variable
df['category'] = ['A', 'B', 'C', 'A', 'B']

# One-Hot Encoding using pandas
df_dummies = pd.get_dummies(df['category'], prefix='category')

# Combine with original DataFrame
df = pd.concat([df, df_dummies], axis=1)
```

```
# Drop the original categorical column
df.drop('category', axis=1, inplace=True)
```

Removing Columns and Data Cleanup (5 minutes)

- **Objective:** Learn how to remove unnecessary columns and clean up data in Python.
- **Content:**
    - **Why Remove Columns:**
        - **Reduce Noise:** Unnecessary or redundant columns can confuse models and reduce accuracy.
        - **Simplify the Dataset:** Focus on the most relevant features for analysis.
    - **How to Remove Columns:**
        - **Dropping Columns:** Using pandas to remove specific columns.
        - **Example:**

```
# Drop multiple columns
df.drop(['column1', 'column2'], axis=1, inplace=True)
```

Further Data Cleanup:

- **Removing Duplicates:** Ensure there are no repeated rows.
- **Renaming Columns:** For better readability.
- **Example:**

```
# Remove duplicates
df.drop_duplicates(inplace=True)

# Rename columns
df.rename(columns={'old_name': 'new_name'}, inplace=True)
```

Working with SQL in Python (10 minutes)

- **Objective:** Learn how to interact with SQL databases directly from Python for data extraction and analysis.
- **Content:**
    - **Why Integrate SQL with Python:**
        - **Seamless Data Retrieval:** Query data from databases directly into Python for analysis.
        - **Complex Queries:** Perform SQL operations and then continue analysis in Python.
    - **Connecting to a SQL Database:**
        - **Using sqlite3 or SQLAlchemy:** Libraries to connect Python to SQL databases.
        - **Example with SQLite:**

```
import sqlite3
import pandas as pd

# Connect to database
conn = sqlite3.connect('example.db')

# Execute SQL query and load data into a DataFrame
df_sql = pd.read_sql_query("SELECT * FROM table_name", conn)

# Close the connection
```

conn.close()

Example with SQLAlchemy (for more complex databases):
from sqlalchemy import create_engine
import pandas as pd

# Create engine to connect to PostgreSQL database
engine = create_engine('postgresql://user:password@localhost/dbname')

# Query data
df_sqlalchemy = pd.read_sql("SELECT * FROM table_name", engine)

-
    - **Common Use Cases:**
        - **Querying large datasets stored in databases.**
        - **Combining SQL querying with Python-based data analysis and visualization.**

---

6. Practical Example & Q&A (5 minutes)

- **Objective:** Apply the learned concepts through a practical example and address any student questions.
- **Content:**
    - **Hands-On Exercise:**
        - **Example:** Import a dataset, perform feature engineering, create dummy variables, and remove unnecessary columns. Connect to a SQL database, retrieve data, and combine it with the existing DataFrame for further analysis.
    - **Q&A Session:** Open discussion for students to ask questions about the topics covered in the lecture.

Key Takeaways

- **Data Transformation:** Learn the basics of preparing and transforming data in Python.
- **Feature Engineering:** Understand how to create new features to enhance model performance.
- **Dummy Variables:** Convert categorical data into a format usable by machine learning models.
- **SQL Integration:** Gain practical skills in querying SQL databases directly from Python.

Resources:
SQLite in Python: https://www.tutorialspoint.com/sqlite/sqlite_python.htm
Python SQL Libraries: https://realpython.com/python-sql-libraries/
Best Python Libraries for SQL: https://dev.to/jconn4177/guide-to-the-best-python-libraries-and-modules-for-sql-21p0
Python Libraries for Database Management: https://www.apriorit.com/dev-blog/web-python-libraries-for-database-management
Python Databases 101: https://builtin.com/data-science/python-database