Summary: Generative AI for SQL Coding

Session Overview:

In this session, the students will be able to:

- Introduction to Gen AI as a Helper in SQL Coding
- Crafting Effective Prompts to Generate Accurate SQL Code
- Debugging AI-Generated SQL Code
- Practical Tips for Optimizing SQL Code with Generative AI

Dataset Overview:

sales_data dataset

- **Sale_ID** - Unique identifier for each sale transaction.
- **Customer_ID** - Identifier for the customer making the purchase.
- **Product_ID** - Identifier for the product being sold.
- **Store_ID** - Identifier for the store where the sale occurred.
- **Sale_Date** - Date when the sale transaction took place.
- **Quantity** - Number of units sold in the transaction.
- **Total_Amount** - Total revenue generated from the sale.

products_data dataset

- **Product_ID** - Unique identifier for each product.
- **Product_Name** - Name of the product.
- **Category** - Category or type of product (e.g., Electronics, Home Supplies).
- **Price** - Price of a single unit of the product.

stores_data dataset

- **Store_ID** - Unique identifier for each store.
- **Store_Name** - Name of the store.
- **Location** - City where the store is located.

Instructions for Instructor/Learners: Create a new column, **Discount**, in the **sales_data** table. The Discount should be calculated as a percentage based on **(Quantity * Price)** before proceeding with any tasks.

1. Query Generation

AI can generate SQL queries based on natural language instructions, helping with:

- Basic SELECT queries.
- Complex JOIN operations.
- Aggregations and filtering.

Prompt Example:
"Write a query to find the total sales for each store."

Generated SQL:

```
SELECT Store_ID, SUM(Total_Amount) AS Total_Sales
FROM Sales
GROUP BY Store_ID;
```

2. Query Optimization

AI helps rewrite SQL queries for better performance, especially for large datasets.

Prompt Example:
"Optimize this query to reduce execution time on a large dataset."

Original Query:

```
SELECT *
FROM Sales
WHERE Sale_Date BETWEEN '2023-01-01' AND '2023-01-31';
```

Optimized Query:

```
SELECT Sale_ID, Store_ID, Total_Amount
FROM Sales
WHERE Sale_Date >= '2023-01-01' AND Sale_Date <= '2023-01-31';
```

3. Code Explanation

AI can explain SQL queries in plain language, making it easier to understand complex code.

Prompt Example:
"Explain the following query:

SELECT Product_Name, SUM(Quantity) AS Total_Quantity
FROM Sales
JOIN Products ON Sales.Product_ID = Products.Product_ID
GROUP BY Product_Name;"

Explanation:
This query calculates the total quantity of each product sold by joining the Sales table with the Products table on Product_ID.

Step-by-Step Process: Using Gen AI for SQL Coding

Step 1: Define the Problem Clearly

- Provide clear, natural language descriptions of the task.
- Include key details such as column names and table relationships.

  Example:
  "Find the total quantity of products sold for each category in the Products table."

Step 2: Craft a Prompt

Structure your prompt to include:

1. **The task:** *What do you want?*
2. **The tables:** *What data is available?*
3. **Expected output:** *What should the result look like?*

Prompt Example:
"Write a SQL query to calculate the total quantity of products sold for each category in the Products table. Use the Sales table to get the quantity sold and join it with the Products table."

Step 3: Refine the Query

AI may generate a query that needs refinement based on:

- Business requirements.
- Table relationships.
- Dataset constraints (e.g., missing values).

Generated Query:

```
SELECT Category, SUM(Quantity) AS Total_Quantity
FROM Sales
```

```
JOIN Products ON Sales.Product_ID = Products.Product_ID
GROUP BY Category;
```

Refined Query (if needed):

```
SELECT p.Category, SUM(s.Quantity) AS Total_Quantity
FROM Sales s
JOIN Products p ON s.Product_ID = p.Product_ID
WHERE s.Quantity > 0
GROUP BY p.Category
```

Step 4: Test and Validate

- Execute the generated query on the database.
- Compare the output with expected results.
- Check for edge cases (e.g., missing Product_ID).

Practical Examples

Example 1: Total Sales by Store

Task: Find the total sales amount for each store.
Generated Query:

```
SELECT Store_ID, SUM(Total_Amount) AS Total_Sales
FROM Sales
GROUP BY Store_ID;
```

Example 2: Best-Selling Product

Task: Identify the product with the highest quantity sold.
Generated Query:

```
SELECT Product_ID, SUM(Quantity) AS Total_Quantity
FROM Sales
GROUP BY Product_ID
ORDER BY Total_Quantity DESC
LIMIT 1;
```

Example 3: Sales by Location

Task: Find the total sales for each store location.
Generated Query:

```
SELECT s.Location, SUM(sa.Total_Amount) AS Total_Sales
FROM Sales sa
JOIN Stores s ON sa.Store_ID = s.Store_ID
GROUP BY s.Location;
```

Common Pitfalls and How to Avoid Them

1. **Ambiguity in Prompts**:
   Issue: *"Find total sales."*
   Fix: *"Find total sales amount for each store in the Sales table."*
2. **Missing Relationships**:
   Ensure AI is aware of the joins between tables.
3. **Ignoring Edge Cases**:
   Check for nulls, missing relationships, or unexpected values.

Crafting Effective Prompts to Generate Accurate SQL Code

Key Principles for Crafting Prompts:

1. Clearly Define the Task

- Be specific about what you need.
- Mention tables, relationships, and expected output.

Bad Prompt:
"Find the total sales."

Good Prompt:
"Write a SQL query to find the total sales amount (Total_Amount) for each store (Store_ID) in the Sales table."

2. Include Context About the Data

- Describe column names, data types, and table relationships.
- Specify joins if needed.

Bad Prompt:
"Find the best-selling product."

Good Prompt:
"Write a query to find the product with the highest total quantity sold. Use the Sales table for quantities and join it with the Products table on Product_ID to get product details."

3. Specify the Output Format

- Define how the result should look.
- Mention columns, aggregation, and sorting.

Bad Prompt:
"Show sales by category."

Good Prompt:
"Write a query to calculate the total sales amount (Total_Amount) for each product category (Category) in the Products table. Sort the results in descending order of total sales."

4. Address Edge Cases

- Mention constraints, such as null values or specific date ranges.

Bad Prompt:
"Get sales in 2023."

Good Prompt:
"Write a query to calculate total sales (Total_Amount) for each store in 2023. Include only transactions with non-null Total_Amount and filter Sale_Date to the year 2023."

5. Refine Iteratively

- Start simple and add details step by step.
- Test initial outputs before adding complexity.

Initial Prompt:
"Get total sales by store."
Refinement Prompt:
"Add a column to calculate the total quantity sold (Quantity) for each store along with the total sales amount (Total_Amount)."

Practical Examples

Example 1: Sales by Store

Task: Find total sales for each store.

Prompt:
"Write a SQL query to calculate the total sales amount (Total_Amount) for each store (Store_ID) in the Sales table."

Generated Query:

```
SELECT Store_ID, SUM(Total_Amount) AS Total_Sales
FROM Sales
GROUP BY Store_ID;
```

Example 2: Revenue by Product Category

Task: Calculate total sales for each product category.

Prompt:
"Write a SQL query to calculate total sales (Total_Amount) for each product category (Category) in the Products table. Use the Sales table to get the sales data and join it with the Products table on Product_ID."

Generated Query:

```
SELECT p.Category, SUM(s.Total_Amount) AS Total_Sales
FROM Sales s
JOIN Products p ON s.Product_ID = p.Product_ID
GROUP BY p.Category;
```

Example 3: Sales in 2023

Task: Calculate total sales in 2023.

Prompt:
"Write a SQL query to calculate total sales amount (Total_Amount) for each store in 2023. Filter Sale_Date to include only transactions in the year 2023."

Generated Query:

```
SELECT Store_ID, SUM(Total_Amount) AS Total_Sales
FROM Sales
WHERE YEAR(Sale_Date) = 2023
GROUP BY Store_ID;
```

Example 4:

Task: Find the product with the highest total quantity sold.

Prompt:
"Write a SQL query to identify the product (Product_ID) with the highest total quantity sold (Quantity). Use the Sales table for quantities and join it with the Products table to include product details."

Generated Query:

```
SELECT p.Product_Name, SUM(s.Quantity) AS Total_Quantity
FROM Sales s
JOIN Products p ON s.Product_ID = p.Product_ID
GROUP BY p.Product_Name
ORDER BY Total_Quantity DESC
LIMIT 1;
```

Common Mistakes in Prompt Crafting

1. Being Too Vague

- **Issue:** *"Get total sales."*
- **Fix:** Be explicit: *"Calculate total sales (Total_Amount) for each store (Store_ID)."*

2. Missing Relationships

- **Issue:** Omitting joins when needed.
  Example: *"Find sales by category."*
- **Fix:** Include joins: *"Join the Sales table with the Products table on Product_ID to get categories."*

3. Not Defining Constraints

- **Issue:** Ignoring edge cases.
  Example: *"Show total sales for 2023."*
- **Fix:** Add constraints: *"Include only non-null values for Total_Amount."*

Steps to Craft an Effective Prompt

1. **Describe the Goal:** Clearly define what you want.
2. **Mention Tables and Columns:** Include table names and relationships.
3. **Specify Aggregations:** Indicate whether you need sums, counts, or averages.
4. **Address Filters and Constraints:** Include any conditions like date ranges or null handling.
5. **Refine Iteratively:** Test the output and refine for missing details.

Debugging AI-Generated SQL Code

1. Common Issues in AI-Generated SQL Code:

1. **Incorrect Joins**
    - AI might not generate proper joins due to ambiguous table relationships.
    - **Example Issue:** Missing join condition between Sales and Products.
2. **Column Ambiguity**
    - AI might not alias columns properly, leading to ambiguity when columns with the same name exist in multiple tables.
    - **Example Issue:** *"Column 'Product_ID' is ambiguous."*
3. **Syntax Errors**
    - Missing commas, mismatched parentheses, or incorrect use of SQL keywords.
    - **Example Issue:** *"Syntax error near 'GROUP'."*
4. **Performance Bottlenecks**
    - AI might generate inefficient queries for large datasets.
    - **Example Issue:** Full table scans instead of indexed filtering.
5. **Incorrect Aggregations**
    - AI may not group data correctly or might use the wrong aggregate function.
    - **Example Issue:** Using COUNT instead of SUM.
6. **Incomplete Filters**
    - Filters might miss critical conditions.
    - **Example Issue:** Missing date range in a sales query.

2. Step-by-Step Debugging Process:

Step 1: Understand the Query

1. Read the AI-generated SQL query carefully.
2. Identify table names, joins, filters, and aggregations.
3. Verify if the query matches the task requirements.

Example:
Task: Find the total revenue by store location.
Generated Query:

```
SELECT Store_ID, SUM(Total_Amount) AS Total_Revenue
FROM Sales
GROUP BY Store_ID;
```

Issue: Missing join with the Stores table to include Location.

Correct Query:

```
SELECT st.Location, SUM(sa.Total_Amount) AS Total_Revenue
FROM Sales sa
JOIN Stores st ON sa.Store_ID = st.Store_ID
GROUP BY st.Location;
```

Step 2: Validate the Query Syntax

1. Run the query on your database to check for syntax errors.
2. Look for missing keywords, incorrect aliases, or unmatched parentheses.

Example Syntax Issue:

```
SELECT Product_ID, SUM(Quantity) Total_Quantity
FROM Sales
GROUP BY Product_ID;
```

Error Message: *"Syntax error near 'Total_Quantity'."*

Fix: Add AS for the alias:

```
SELECT Product_ID, SUM(Quantity) AS Total_Quantity
FROM Sales
GROUP BY Product_ID;
```

Step 3: Check Joins

1. Ensure all necessary joins are included.
2. Verify join conditions to avoid cartesian products.

Example Join Issue:

```
SELECT Product_Name, SUM(Total_Amount) AS Total_Revenue
FROM Sales
JOIN Products;
```

Error Message: *"Missing join condition."*

Fix: Add the join condition:

```
SELECT p.Product_Name, SUM(s.Total_Amount) AS Total_Revenue
FROM Sales s
JOIN Products p ON s.Product_ID = p.Product_ID
GROUP BY p.Product_Name;
```

Step 4: Verify Filters

1. Ensure filters are applied correctly.
2. Check for edge cases like null values or incorrect date formats.

Example Filter Issue:

```
SELECT Store_ID, SUM(Total_Amount) AS Total_Revenue
FROM Sales
WHERE Sale_Date = '2023-01';
```

Error Message: *"Invalid date format."*

Fix: Use the LIKE operator for partial matching or filter a date range:

```
SELECT Store_ID, SUM(Total_Amount) AS Total_Revenue
FROM Sales
WHERE Sale_Date LIKE '2023-01%';
```

OR

```
SELECT Store_ID, SUM(Total_Amount) AS Total_Revenue
FROM Sales
WHERE Sale_Date BETWEEN '2023-01-01' AND '2023-01-31';
```

Step 5: Test for Accuracy

1. Run the query and inspect the output.
2. Compare the results with expected values.

Debugging Examples

Example 1: Mismatched Column Name

Task: Find the total quantity sold by product.

Generated Query:

```
SELECT Product_ID, SUM(Quantities) AS Total_Quantity
FROM Sales
GROUP BY Product_ID;
```

Error Message: *"Unknown column 'Quantities'."*

Fix: Correct the column name to Quantity:

```
SELECT Product_ID, SUM(Quantity) AS Total_Quantity
FROM Sales
GROUP BY Product_ID;
```

Example 2: Cartesian Product

Task: Find total sales by product category.

Generated Query:

```
SELECT Category, SUM(Total_Amount) AS Total_Sales
FROM Sales, Products
GROUP BY Category;
```

Error Message: *"Ambiguous column 'Category'."*

Fix: Use an explicit JOIN with a proper condition:

```
SELECT p.Category, SUM(s.Total_Amount) AS Total_Sales
```

```
FROM Sales s
JOIN Products p ON s.Product_ID = p.Product_ID
GROUP BY p.Category;
```

Example 3: Missing Filter

Task: Find total sales in January 2023.

Generated Query:

```
SELECT SUM(Total_Amount) AS Total_Sales
FROM Sales;
```

Issue: No date filter applied.

Fix: Add a WHERE clause:

```
SELECT SUM(Total_Amount) AS Total_Sales
FROM Sales
WHERE Sale_Date BETWEEN '2023-01-01' AND '2023-01-31';
```

Example 4: Incorrect Aggregation

Task: Find the average revenue per store.

Generated Query:

```
SELECT Store_ID, SUM(Total_Amount) AS Avg_Revenue
FROM Sales
GROUP BY Store_ID;
```

Issue: Incorrect alias Avg_Revenue for SUM.

Fix: Use AVG for correct aggregation:

```
SELECT Store_ID, AVG(Total_Amount) AS Avg_Revenue
FROM Sales
GROUP BY Store_ID;
```

Tips for Effective Debugging

1. **Run in Stages:** Break complex queries into smaller parts.
   - Test joins, filters, and aggregations separately.
2. **Inspect Intermediate Outputs:** Use LIMIT or select fewer columns to check interim results.
3. **Use Aliases:** Alias tables and columns for clarity and to avoid ambiguity.
4. **Read Error Messages:** SQL error messages often point directly to the problem.
5. **Ask Gen AI for Debugging:**
   Prompt Example:
   "This query is throwing a syntax error near 'GROUP'. Can you help me debug it? Here's the query: [query]."

Practical Tips for Optimizing SQL code

1. Start with a Clear Prompt

- Clearly define the problem, dataset, and expected output.
- Include table names, column names, and relationships.
- Example Prompt:
  "Optimize a query to calculate total sales (Total_Amount) by product category (Category) in the Products table, using a join with the Sales table."

2. Leverage AI for Query Simplification

- Use AI to rewrite complex queries into simpler, more efficient ones.
- Example: Replace nested subqueries with a JOIN operation to reduce redundancy.

3. Optimize Aggregations

- Use targeted aggregations (SUM, AVG, etc.) instead of fetching unnecessary data.
- Ensure proper grouping in GROUP BY clauses to avoid performance bottlenecks.

4. Avoid Full Table Scans

- Add filters in WHERE clauses to limit the scope of data.
- Example: *"Filter rows for Sale_Date in 2023 to avoid scanning the entire table."*

5. Use Proper Joins

- Ensure correct join types (INNER JOIN, LEFT JOIN, etc.) to minimize unnecessary rows.
- Use indexed columns in join conditions to improve performance.

6. Test AI-Generated Queries

- Run queries with real data to verify output and execution time.
- Use database-specific tools like EXPLAIN to analyze query performance.

7. Refactor Iteratively

- Use AI to refine queries step-by-step:
    1. Start with a basic query.
    2. Add optimizations (e.g., filters, indexes).
    3. Validate results at each stage.