Welcome!

WWCode San Francisco - Backend Study Group

June 21, 2023

- We'll start in a moment :)
- We are **RECORDING** tonight's event
- We may plan to take screenshots for social media
- If you are comfortable, turn the video ON. If you want to be anonymous, then turn the video off
- We'll make some time for Q&A at the end of the presentation
- Feel free to take notes
- Online event best practices:
    - Don't multitask. Distractions reduce your ability to remember concepts
    - Mute yourself when you aren't talking
    - We want the session to be interactive
    - Use the 'Raise Hand' feature to ask questions
- **By attending our events, you agree to comply with our Code of Conduct**

WOMEN WHO CODE /san-francisco

Introduction & Agenda

- Welcome from WWCode!
- Our mission: Empower women to excel in technology careers
- Backend Study Group: Learn and discuss backend engineering concepts

Prachi Shah
Instructor
Sr. Software Engineer, Unity
Director, WWCode SF

Anjali Bajaj
Host & Lead, WWCode SF

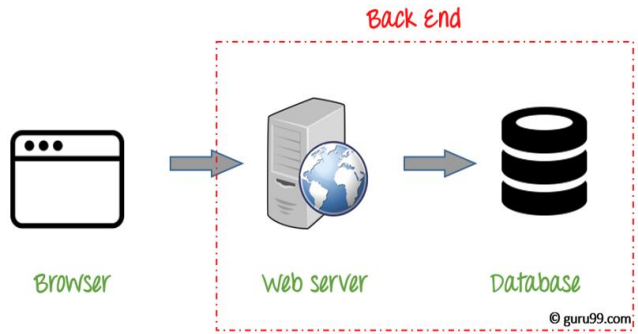
- Fundamentals of SQL
- Query Execution
- Data Types
- Clauses
- Functions
- Operators
- Relationships
- Joins
- Nested Queries
- Q & A

WOMEN WHO CODE /san-francisco

Backend Engineering

- Design, build and maintain server-side web applications

- Concepts: Client-server architecture, networking, APIs, web fundamentals, microservices, databases, security, operating systems, etc.



- Tech Stack: Java, PHP, .NET, C#, Ruby, Python, REST, AWS, Node, SQL, NoSQL, etc.

SQL

- SQL stands for Structured Query Language.
- It is a standard language for managing and manipulating relational databases.
- SQL allows you to define, manipulate, and query the data stored in a database.
- SQL was first developed at IBM by Donald D. Chamberlin and Raymond F. Boyce in the 1970s.
- It was initially called SEQUEL (Structured English Query Language).
- SQL became an ANSI standard in 1986 and an ISO standard in 1987.
- Over the years, different variations of SQL have been developed, including MySQL, Oracle, SQL Server, and PostgreSQL.

Query Execution

- SQL queries consist of one or more statements that instruct the database what to do.
- The most common SQL statements are:
  - SELECT: retrieves data from one or more tables.
  - INSERT: inserts new records into a table.
  - UPDATE: modifies existing records in a table.
  - DELETE: deletes records from a table.
- Data Manipulation Language (DML):
  - DML is used for retrieving, inserting, updating, and deleting data in a database.
  - Common DML statements include SELECT, INSERT, UPDATE, and DELETE.
- Data Definition Language (DDL):
  - DDL is used for creating, altering, and deleting database objects.
  - Common DDL statements include CREATE, ALTER, and DROP.

Query Execution

SELECT Statement:

- The SELECT statement is used to query data from one or more tables.
- Syntax: SELECT column1, column2, ... FROM table_name WHERE condition;
- Example: SELECT * FROM employees WHERE salary > 50000;
- DML statement

INSERT Statement:

- The INSERT statement is used to insert new records into a table.
- Syntax: INSERT INTO table_name (column1, column2, ...) VALUES (value1, value2, ...);
- Example: INSERT INTO customers (name, email) VALUES ('John Doe', 'johndoe@sql.com');
- DML statement

Query Execution

UPDATE Statement:

- The UPDATE statement is used to modify existing records in a table.
- Syntax: UPDATE table_name SET column1 = value1, column2 = value2 WHERE condition;
- Example: UPDATE products SET price = 10.99 WHERE id = 1;
- DML statement

DELETE Statement:

- The DELETE statement is used to delete records from a table.
- Syntax: DELETE FROM table_name WHERE condition;
- Example: DELETE FROM orders WHERE order_date < '2023-01-01';
- DML statement

Query Execution

CREATE Statement:

- The CREATE statement is used to create a new table in the database.
- Syntax: CREATE TABLE table_name (column1 data_type, column2 data_type, ...);
- Example: CREATE TABLE employees (id INT, name VARCHAR(50), salary DECIMAL(10,2));
- DDL statement

ALTER Statement:

- The ALTER statement is used to modify the structure of an existing table.
- Syntax: ALTER TABLE table_name ADD column_name data_type;
- Example: ALTER TABLE employees ADD email VARCHAR(100);
- DDL statement

Query Execution

DROP Statement

- The DROP statement is used to delete a table or other database objects.
- Syntax: DROP TABLE table_name;
- Example: DROP TABLE employees;
- DDL statement

Data Types

- Data types define the type of data that can be stored in a column of a table.
- Common data types include:
  - INT: Integer values.
  - VARCHAR(n): Variable-length character strings with a maximum length of 'n'.
  - CHAR(n): Fixed-length character strings with a length of 'n'.
  - FLOAT: Floating-point numbers.
  - DATE: Date values.
  - BOOLEAN: Boolean (true/false) values.
- INTEGER (INT): Used to store whole numbers. Example: age INT, quantity INT
- DECIMAL (DEC, NUMERIC): Used to store fixed-point decimal numbers with precision and scale. Example: price DECIMAL(10, 2), tax DECIMAL(5, 2)
- VARCHAR: Used to store variable-length character strings. Example: name VARCHAR(50), address VARCHAR(100)

Data Types

- CHAR: Used to store fixed-length character strings. Example: status CHAR(1), country CHAR(3)
- DATE: Used to store dates. Example: birth_date DATE, order_date DATE
- TIMESTAMP: Used to store date and time values. Example: created_at TIMESTAMP, updated_at TIMESTAMP
- BOOLEAN: Used to store true/false or logical values. Example: is_active BOOLEAN, has_discount BOOLEAN
- FLOAT: Used to store floating-point numbers. Example: weight FLOAT, temperature FLOAT
- BLOB (Binary Large Object): Used to store binary data, such as images or files. Example: image BLOB, document BLOB

Clauses

- SQL offers many advanced features and functions for complex data retrieval and manipulation.
- JOIN: combines rows from two or more tables based on a related column.
- GROUP BY: groups rows based on a column and performs aggregate functions.
- HAVING: filters the result set based on aggregate function conditions.
- WHERE: Filters rows based on a specified condition.
- ORDER BY: Sorts the result set based on one or more columns.
- GROUP BY: Groups rows based on a column or columns.
- Example: SELECT name, SUM(salary) as total_salary FROM employees **WHERE** age > 30 **GROUP BY** name **HAVING** total_salary > 50000 **ORDER BY** name;
- Here, we select the name and calculate the sum of salary for employees above 30 years old, group the result by name, filter out groups with a total salary less than 50000, and finally, order the result by name.

Functions

- SQL functions are built-in functions provided by the SQL language that perform various operations on data or return computed values.
- There are four types of functions:
  - Aggregate
  - String
  - Date
  - Mathematical

Functions

- Aggregate Functions:
  - AVG(): Calculates the average value of a column.
  - SUM(): Calculates the sum of values in a column.
  - COUNT(): Returns the number of rows or non-null values in a column.
  - MIN(): Returns the minimum value in a column.
  - MAX(): Returns the maximum value in a column.
- String Functions:
  - CONCAT(): Concatenates two or more strings.
  - SUBSTRING(): Extracts a portion of a string.
  - UPPER(): Converts a string to uppercase.
  - LOWER(): Converts a string to lowercase.
  - LENGTH(): Returns the length of a string.
- Example: SELECT AVG(salary) FROM employees;

Functions

- Date Functions:
  - NOW(): Returns the current date and time.
  - DATE(): Extracts the date portion from a datetime value.
  - YEAR(): Extracts the year from a date or datetime value.
  - MONTH(): Extracts the month from a date or datetime value.
  - DAY(): Extracts the day from a date or datetime value.
- Mathematical Functions:
  - ABS(): Returns the absolute value of a number.
  - ROUND(): Rounds a number to a specified number of decimal places.
  - CEILING(): Rounds a number up to the nearest integer.
  - FLOOR(): Rounds a number down to the nearest integer.
  - POWER(): Raises a number to a specified power.
- Example: SELECT POWER(2, 3); // Raises 2 to the power of 3

Operators

- SQL operators are symbols or keywords used to perform operations on data or compare values in SQL statements. Some commonly used SQL operators include:
- Arithmetic Operators:
  - + (addition)
  - - (subtraction)
  - * (multiplication)
  - / (division)
  - % (modulus)
- Comparison Operators:
  - = (equal to)
  - <> or != (not equal to)
  - < (less than) and > (greater than)
  - <= (less than or equal to) and >= (greater than or equal to)

Operators

- Logical Operators:
  - AND (logical AND)
  - OR (logical OR)
  - NOT (logical NOT)
- String Operators:
  - || (concatenation)
- Example:
  - Arithmetic: SELECT 5 + 3; // Addition: Result = 8
  - Comparison: SELECT * FROM employees WHERE salary <= 50000;
  - Logical: SELECT * FROM employees WHERE age > 30 AND department = 'IT';
  - String: SELECT * FROM products WHERE description LIKE '%apple%';

Relationships

- Relationships between tables define how tables are related or connected to each other based on their keys.
- One-to-One (1:1) Relationship:
  - Each record in Table A is associated with exactly one record in Table B, and vice versa.
  - Example: A "Person" table and an "Address" table, where each person has only one address.
- One-to-Many (1:N) Relationship:
  - Each record in Table A can be associated with multiple records in Table B, but each record in Table B is associated with only one record in Table A.
  - Example: A "Department" table and an "Employee" table, where one department can have multiple employees, but each employee belongs to only one department.

Relationships

- Many-to-Many (N:N) Relationship:
  - Each record in Table A can be associated with multiple records in Table B, and vice versa.
  - A separate "junction" or "linking" table is used to establish the relationship, typically containing the primary keys of both tables.
  - Example: A "Student" table and a "Course" table, where each student can enroll in multiple courses, and each course can have multiple students.
- Relationships are defined using primary keys and foreign keys to establish connections between related tables, enabling efficient data retrieval and maintaining data integrity within the database.

Relationships

Primary Keys:

- A primary key is a column or a set of columns that uniquely identifies each record in a table.
- It ensures the uniqueness and integrity of data within the table.
- Primary keys have the following characteristics:
  - Each table can have only one primary key.
  - Primary key values must be unique and cannot be NULL (empty).
  - Primary keys are often created using the INT or VARCHAR data types.

Relationships

Foreign Keys:

- A foreign key is a column or a set of columns in a table that references the primary key of another table.
- It establishes a relationship between two tables based on their common key values.
- Foreign keys have the following characteristics:
    - They enforce referential integrity, ensuring that data in the referencing table (foreign key table) corresponds to existing data in the referenced table (primary key table).
    - Foreign key values can have NULL values if the relationship allows it.
    - Foreign keys can be used to establish different types of relationships between tables, such as one-to-one, one-to-many, or many-to-many.

Joins

- Inner Join: Returns only the matched rows from both tables. This query joins the "employees" and "departments" tables based on the "department_id" column, and retrieves the employee names and their corresponding department names.
- Example: SELECT employees.employee_name, departments.department_name FROM employees INNER JOIN departments ON employees.department_id = departments.department_id;

Table: employees

employee_id	employee_name	department_id
1	John Smith	1
2	Emma Johnson	2
3	Michael Brown	1

Table: departments

department_id	department_name
1	Sales
2	Marketing
3	Finance

Joins

- Left Join: Returns all rows from the left table and the matched rows from the right table. This query performs a left join between the "employees" and "departments" tables, and retrieves all employee names along with their corresponding department names. If a department doesn't have any employees, it will still be included in the result with a NULL value in the department name.
- Example: SELECT employees.employee_name, departments.department_name
  FROM employees LEFT JOIN departments
  ON employees.department_id = departments.department_id;

Joins

- Right Join: Returns all rows from the right table and the matched rows from the left table. This query performs a right join between the "employees" and "departments" tables, and retrieves all department names along with the corresponding employee names. If an employee doesn't belong to any department, the employee name will be NULL.
- Example: SELECT employees.employee_name, departments.department_name
  FROM employees RIGHT JOIN departments
  ON employees.department_id = departments.department_id;

Nested Queries

- Nested queries, also known as subqueries, are queries within another query. They are used to retrieve data from one table based on the result of another query.

Table: employees

employee_id	employee_name	department_id
1	John Smith	1
2	Emma Johnson	2
3	Michael Brown	1

Table: departments

department_id	department_name
1	Sales
2	Marketing
3	Finance

Nested Queries

- Nested queries, also known as subqueries, are queries within another query. They are used to retrieve data from one table based on the result of another query.
- Example: Retrieve employees from the Sales department:
  ```
  SELECT employee_name FROM employees
  WHERE department_id = (SELECT department_id FROM departments WHERE
  department_name = 'Sales');
  ```
- Here, the nested query (SELECT department_id FROM departments WHERE department_name = 'Sales') retrieves the department ID for the department with the name 'Sales'. This result is then used as a condition in the outer query WHERE clause to filter the employees based on their department.

Backend Study Group

Backend Study Group:

- Presentations on GitHub and session recordings available on WWCode YouTube channel
- June 29th, 2023: Big data pipelines 101
- July 6th, 2023: Ruby on Rails 101
- August 3rd, 2023: Domain Driver Design (DDD)
- September 21st, 2023: Web Development 101
- October 12th, 2023: DevOps 101
- November 2nd, 2023: How to be a good Backend Engineer?

Women Who Code:

- Technical Tracks and Digital Events for more events
- Join the Digital mailing list for updates about WWCode
- Contacts us at: contact@womenwhocode.com
- Join our Slack workspace and join *#backend-study-group*!

You can unmute and talk or use the chat

WOMEN WHO CODE® /san-francisco