MIT 213 – DATA MANAGEMENT

TUTORIAL 3: Data Manipulation with SQL

1 Data Manipulation with SQL

Data manipulation is a fundamental aspect of working with databases. SQL (Structured Query Language) provides a powerful set of tools for manipulating and transforming data within a database. With SQL, you can perform a wide range of operations, including querying, updating, inserting, and deleting data, allowing you to retrieve and modify information to meet your specific needs.

1.1 SELECT Statement

The SELECT statement is used to retrieve data from one or more database tables. It allows you to specify the columns you want to retrieve, apply filtering conditions, and sort the results.

Example:

```
SELECT column1, column2
FROM table_name
WHERE condition
ORDER BY column1;
```

1.2 UPDATE Statement

The UPDATE statement is used to modify existing records in a table. It allows you to update one or more columns with new values based on specified conditions.

Example:

```
UPDATE table_name
SET column1 = value1, column2 = value2
WHERE condition;
```

1.3 INSERT Statement

The INSERT statement is used to insert new records into a table. It allows you to specify the columns and their corresponding values for the new records.

Example:

```
INSERT INTO table_name (column1, column2)
VALUES (value1, value2);
```

1.4 DELETE Statement

The DELETE statement is used to remove one or more records from a table based on specified conditions.

Example:

```
DELETE FROM table_name
WHERE condition;
```

1.5 JOINs

JOINs are used to combine data from multiple tables based on related columns. Different types of JOINs, such as INNER JOIN, LEFT JOIN, RIGHT JOIN, and FULL JOIN, allow you to retrieve data based on different matching conditions.

Example:

```sql
SELECT column1, column2
FROM table1
INNER JOIN table2
ON table1.column_name = table2.column_name;
```

1.6 Aggregation Functions

SQL provides various aggregation functions, such as SUM, AVG, COUNT, MAX, and MIN, to perform calculations on groups of rows. These functions allow you to summarize and analyze data.

Example:

```sql
SELECT COUNT(*)
FROM table_name;
```

1.7 Subqueries

Subqueries are queries nested within another query. They allow you to perform complex operations by using the results of one query as input for another query.

Example:

```sql
SELECT column1
FROM table_name
WHERE column2 IN (SELECT column2 FROM another_table);
```

SQL provides a rich set of data manipulation capabilities that enable you to retrieve, modify, and analyze data stored in databases. By understanding the syntax and functionality of SQL statements, you can efficiently manipulate data to extract meaningful insights and support decision-making processes.

Note: The examples provided above are simplified for illustration purposes. In practice, you may need to consider additional factors such as table relationships, data types, and security measures when working with databases.

2 INSERT, UPDATE, and DELETE Statements for Modifying Data

In SQL, the INSERT, UPDATE, and DELETE statements are essential for modifying data within database tables. These statements allow you to add new records, update existing records, and delete unwanted records, providing flexibility in managing and maintaining data integrity.

2.1 INSERT Statement

The INSERT statement is used to add new records to a table. It allows you to specify the columns and their corresponding values for the new records.

Example: Let's consider a table called "employees" with columns for "employee_id," "first_name," and "last_name." We will insert a new employee into the table.

```sql
-- Create the "employees" table
CREATE TABLE employees (
  employee_id INT PRIMARY KEY,
  first_name VARCHAR(50),
  last_name VARCHAR(50)
);

-- Insert a new employee
INSERT INTO employees (employee_id, first_name, last_name)
```

```
VALUES (1, 'John', 'Doe');
```
In the above example, we create the "employees" table using the CREATE TABLE statement, specifying the columns and their data types. Then, we use the INSERT INTO statement to add a new employee with an employee ID of 1, first name 'John', and last name 'Doe' into the table.

2.2 UPDATE Statement

The UPDATE statement is used to modify existing records in a table. It allows you to update one or more columns with new values based on specified conditions.

Example: Let's update the last name of an employee in the "employees" table.

```
-- Update the last name of an employee
UPDATE employees
SET last_name = 'Smith'
WHERE employee_id = 1;
```
In the above example, we use the UPDATE statement to change the last name of the employee with an employee ID of 1 to 'Smith'. The WHERE clause specifies the condition for which employee to update.

2.3 DELETE Statement

The DELETE statement is used to remove one or more records from a table based on specified conditions.

Example: Let's delete an employee from the "employees" table.

```
-- Delete an employee
DELETE FROM employees
WHERE employee_id = 1;
```
In the above example, we use the DELETE statement to remove the employee with an employee ID of 1 from the "employees" table. The WHERE clause specifies the condition for which employee to delete.

The INSERT, UPDATE, and DELETE statements provide powerful tools for modifying data in SQL. Whether you need to add new records, update existing records, or delete unwanted records, these statements allow you to maintain the integrity and accuracy of your data.

Note: The specific table schema and data used in the code examples may vary depending on the context of your application. It is important to adapt the code to match your table structure and data requirements.

By leveraging the INSERT, UPDATE, and DELETE statements, you can effectively manipulate the data stored in your database tables, ensuring that the information remains up to date and relevant.

3 Managing Transactions and Ensuring Data Integrity

In SQL, transactions play a crucial role in maintaining data integrity and ensuring that database operations are performed reliably and consistently. Transactions allow you to group multiple SQL statements into a single logical unit, ensuring that all changes are either committed together or rolled back in case of failures or errors.

3.1 ACID Properties

Transactions adhere to the ACID properties, which stand for Atomicity, Consistency, Isolation, and Durability. These properties guarantee the reliability and integrity of database operations.

- **Atomicity**: Transactions are indivisible units of work. They are either executed in their entirety or not at all. If any part of a transaction fails, the entire transaction is rolled back, and the database is left unchanged.
- **Consistency**: Transactions bring the database from one consistent state to another. Data modifications within a transaction must adhere to predefined validation rules, preserving the integrity and validity of the data.
- **Isolation**: Transactions are executed in isolation from each other. Each transaction operates as if it is the only transaction being executed, preventing interference and maintaining data integrity.
- **Durability**: Once a transaction is committed, its changes are permanent and survive any subsequent failures. The database ensures that committed changes are durable and can be recovered in case of system crashes or errors.

3.2 Transaction Control Statements

SQL provides transaction control statements to manage transactions explicitly:

- BEGIN TRANSACTION: Begins a new transaction.
- COMMIT: Commits the changes made within a transaction, making them permanent.
- ROLLBACK: Undoes the changes made within a transaction, reverting the database to its state before the transaction started.
- SAVEPOINT: Sets a savepoint within a transaction, allowing partial rollback to a specific point.

Example: Let's consider a scenario where we need to transfer funds between two bank accounts and ensure the integrity of the transaction.

```
-- Create the "accounts" table
CREATE TABLE accounts (
  account_number INT PRIMARY KEY,
  balance DECIMAL(10,2)
);

-- Begin a new transaction
BEGIN TRANSACTION;

-- Update the balance of Account A
UPDATE accounts
SET balance = balance - 500
WHERE account_number = 123;

-- Update the balance of Account B
UPDATE accounts
SET balance = balance + 500
WHERE account_number = 456;

-- Commit the transaction
COMMIT;
```

In the above example, we create the "accounts" table using the CREATE TABLE statement, defining the columns for the account number and balance. We then begin a new transaction

using the BEGIN TRANSACTION statement. Within the transaction, we update the balances of two bank accounts, subtracting 500 units from Account A and adding 500 units to Account B. Finally, we commit the transaction using the COMMIT statement, ensuring that the changes are made permanently.

If an error or failure occurs during the transaction, we can use the ROLLBACK statement to undo the changes made within the transaction and revert the database to its previous state.

By employing transactions and adhering to the ACID properties, you can ensure the reliability, consistency, isolation, and durability of your database operations, maintaining data integrity and providing a solid foundation for managing complex data manipulations.

Note: The specific table schema and data used in the code examples may vary depending on the context of your application. It is important to adapt the code to match your table structure and data requirements.