Fundamentals of SQL

DATA DEFINITION LANGUAGE

CRUD OPERATIONS

- Creating a Table
- Reading a Table
- Updating a Table
- Deleting a Table

Creating a Table

1. **CREATE DATABASE**: As implied by its name, this keyword creates a new database.
2. **USE DATABASE**
3. **CREATE TABLE**
4. **INSERT VALUES INTO TABLE**

```
CREATE DATABASE database_name;
USE database_name;


CREATE TABLE table_name (
    column1 datatype constraint,
    column2 datatype constraint,
    column3 datatype constraint,
    ...
);

// Example


CREATE DATABASE College;
USE College;

CREATE TABLE Student(
    id int primary key,
    name varchar(50),
    email varchar(50)
);
```

Inserting Values into the Table

```
INSERT INTO table_name (column1, column2, column3, ...)
VALUES (value1, value2, value3, ...);

//Example

INSERT INTO Student VALUES
    (1, 'Raj', 'raj@gmail.com'),
    (2, 'Susmitha', 'susmitha@gmail.com'),
    (3, 'Manish', 'manish@gmail.com');
```

- The values can also be inserted from FILES

Updating the Values

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

// Example

UPDATE Student
SET email = "rajendran@gmail.com"
WHERE id = 1;

SELECT * FROM Student;

// Update without a condition

UPDATE Student
SET email = 'default@example.com';


// Multiple Condition

UPDATE Student
SET email = CASE
    WHEN name = "Raj" THEN "raj@gmail.com"
    WHEN name = "Mani" THEN "manish@gmail.com"
ELSE email
END;
```

Retrieving Data From Table

- **SELECT:** helps to retrieve the data from the database

    - While using `SELECT` command, `*` indicates that all records should be retrieved from the `Employee` table.

- **FROM**: Used in conjunction with `SELECT` to specify the table from which to fetch data.

```
SELECT column_names FROM table_name;

// Example

SELECT employee_id FROM Employee;
SELECT * FROM Employee; // retrieves all the columns from the
```

- **WHERE**: Used to filter records.

```
SELECT name FROM College WHERE department = "AI & DS";
```

Deleting a Record, Attribute, Table, Database

Here are the SQL commands for deleting records, attributes (columns), tables, and databases.

1. Deleting a Record

To delete specific records from a table, use the `DELETE` statement. The `WHERE` clause is essential to specify which records to delete.

Syntax:

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

Example:

```
DELETE FROM Student
WHERE id = 1;
```

This command deletes the record from the `Student` table where the `id` is 1.

2. Deleting an Attribute (Column)

To remove a column from a table, use the `ALTER TABLE` statement with the `DROP COLUMN` clause.

Syntax:

```
ALTER TABLE table_name
DROP COLUMN column_name;
```

Example:

```
ALTER TABLE Student
DROP COLUMN email;
```

This command removes the `email` column from the `Student` table.

3. Deleting a Table

To delete an entire table and all of its records, use the `DROP TABLE` statement.

Syntax:

```
DROP TABLE table_name;
```

Example:

```
DROP TABLE Student;
```

This command permanently deletes the `Student` table.

4. Deleting a Database

To delete an entire database along with all its tables and data, use the `DROP DATABASE` statement.

Syntax:

```
DROP DATABASE database_name;
```

Example:

```
DROP DATABASE College;
```

This command permanently deletes the `College` database and all its tables.

TRUNCATE TABLE

- deletes all data from the table without deleting the structure of the table

ALTER Operations

1. Add
2. Drop Column
3. Modify
4. Change
5. Rename to

The `ALTER TABLE` statement in SQL is used to modify an existing table structure. You can use it to add new columns, modify existing columns, or rename columns and tables. Below are examples for each of these operations:

1. Adding a Column

To add a new column to an existing table, use the `ADD` clause.

Syntax:

```
ALTER TABLE table_name
ADD column_name data_type [constraint];
```

Example:

```
ALTER TABLE Student
ADD age INT;
```

This command adds a new column named `age` of type `INT` to the `Student` table.

```
ALTER TABLE table_name
DROP COLUMN column_name;
```

Example:

```
ALTER TABLE Student
DROP COLUMN age;
```

2. Modifying a Column

To modify the definition of an existing column (like changing its data type or constraints), use the `MODIFY` or `ALTER` clause, depending on the SQL database you are using.

Syntax:

```
ALTER TABLE table_name
MODIFY column_name new_data_type [constraint];
```

Example:

```
ALTER TABLE Student
MODIFY age TINYINT;  -- Change age data type to TINYINT
```

3. Renaming a Column

To rename an existing column, you can use the `CHANGE` clause in MySQL or the `RENAME COLUMN` clause in other databases like PostgreSQL or SQL Server.

MySQL Example:

```
ALTER TABLE Student
CHANGE student_age age TINYINT;  -- Rename 'student_age' ba
ck to 'age'
```

PostgreSQL/SQL Server Example:

```
ALTER TABLE Student
RENAME COLUMN age TO student_age;  -- Rename 'age' to 'stud
ent_age'
```

4. Renaming a Table

To rename an entire table, use the `RENAME TO` clause.

Syntax:

```
ALTER TABLE old_table_name
RENAME TO new_table_name;
```

Example:

```
ALTER TABLE Student
RENAME TO Students;  -- Rename 'Student' table to 'Student
s'
```

NOTES

To know the details about a table use `DESCRIBE` command

Example:

`DESCRIBE Student`

- If you ever want to remove duplicate elements from a table, use `SELECT DISTINCT`
- Use `ORDER BY` to sort the result