Working with Tables in SQL

1. Creating a Table

Example Scenario: Student Management System

Imagine you are building a student management system for a school. You need a table to store student information, such as their name, age, grade, and contact details. Here's how you can create a "Students" table:

```
CREATE TABLE Students (
    StudentID INT PRIMARY KEY AUTO_INCREMENT,
    FirstName VARCHAR(50) NOT NULL,
    LastName VARCHAR(50) NOT NULL,
    Age INT,
    Grade VARCHAR(10),
    ContactNumber VARCHAR(15),
    Email VARCHAR(100)
);
```

Explanation

- `StudentID`: An auto-incrementing primary key to uniquely identify each student.
- `FirstName` and `LastName`: Strings with a maximum length of 50 characters, and they are required (`NOT NULL`).
- `Age`: Stores the student's age as an integer.
- `Grade`: Stores the student's grade (e.g., "10th", "12th").
- `ContactNumber` and `Email`: Additional contact information.

2. Modifying a Table

Example Scenario: Adding an Address Column

After some time, the school wants to add the students' addresses to the table. You can modify the table using the `ALTER TABLE` command:

```
ALTER TABLE Students

ADD Address VARCHAR(255);
```

Explanation

- `ALTER TABLE Students`: Specifies which table to modify.
- `ADD Address VARCHAR(255)`: Adds a new column called `Address` to store up to 255 characters.

3. Dropping a Table

Example Scenario: Removing the Students Table

If the student management system is no longer needed, you might need to drop the "Students" table to free up the database.

```
DROP TABLE Students;
```

Explanation

- `DROP TABLE Students`: Permanently deletes the "Students" table and all its data.

Best Practices

- Always back up data before dropping or altering tables.
- Use constraints like `NOT NULL`, `PRIMARY KEY`, and `AUTO_INCREMENT` to maintain data integrity.
- Avoid using `DROP TABLE` lightly in a production environment.