Data Manipulation in SQL

In this session, we'll delve deeper into the techniques used to update the structure of the data within SQL databases effectively.

Objective:

• Modifying and Updating New Tables:

- Modify the structure of existing tables using the `ALTER TABLE` statement
- Update data within these tables using the `UPDATE` statement. This includes adding new columns, changing data types, and updating existing records.

• Understand and Implement Primary Keys:

- Define and create primary keys to uniquely identify records in a table.
- Add and remove primary keys from existing tables.

• Utilize Auto-Increment Fields:

- Configure auto-increment fields to automatically generate unique identifiers for new records.
- Reset auto-increment values as needed.

1. Modifying and Updating New Tables

You can modify the structure of an existing table using the `ALTER TABLE` statement and update the data using the `UPDATE` statement.

Altering Table Structure

Syntax

```
ALTER TABLE table_name
ADD column_name datatype;
```

Example

Add a new column `record_date` to the `users` table.

```
ALTER TABLE users
ADD record_date Date;
```

Modify Table

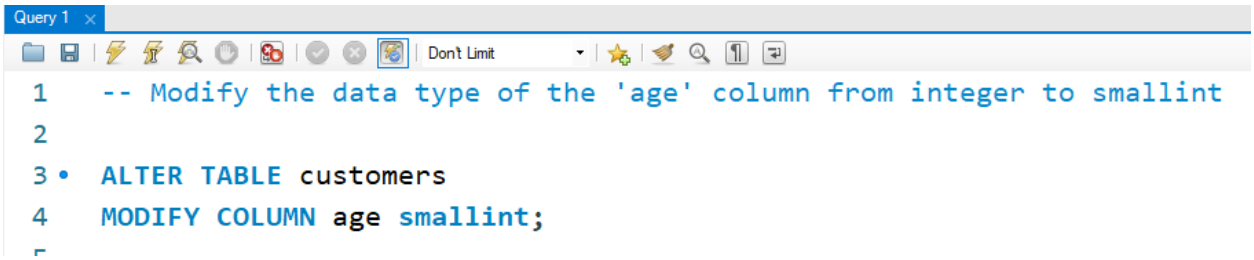
Syntax to Modify a column data type

```
ALTER TABLE table_name
MODIFY COLUMN column_name NEW_DATA_TYPE;
```

Example

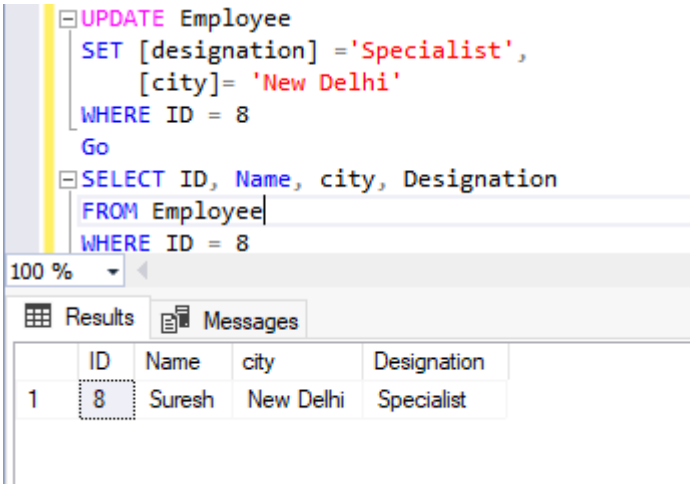
Modify the data type of the 'age' column from integer to smallint

```
ALTER TABLE customers MODIFY COLUMN age smallint;
```



Updating Data

General Example of updating data in table.



Syntax

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

Example

Update the salary of the employee with `employee_id` 1.

```
UPDATE employees
SET salary = 80000.00
WHERE employee_id = 1;
```

5. Working with Primary Keys

Introduction

A primary key is a field in a table which uniquely identifies each row/record in that table. Primary keys must contain unique values and cannot contain NULL values.

Creating a Primary Key

To create a primary key when creating a table, use the following syntax:

```
CREATE TABLE table_name (
    column1 datatype PRIMARY KEY,
    column2 datatype,
    ...
);
```

Example

```
CREATE TABLE Students (
    StudentID int NOT NULL PRIMARY KEY,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int
);
```

Adding a Primary Key to an Existing Table

If a table already exists and you want to add a primary key:

```
ALTER TABLE table_name
ADD PRIMARY KEY (column_name);
```

Example

```
ALTER TABLE Students
ADD PRIMARY KEY (StudentID);
```

Removing a Primary Key

To remove a primary key constraint:

```
ALTER TABLE table_name
DROP PRIMARY KEY;
```

Example

```
ALTER TABLE Students
DROP PRIMARY KEY;
```

6. Auto-Increments

Introduction

Auto-increment allows a unique number to be generated automatically when a new record is inserted into a table.

General Example of auto increment.

```
CREATE TABLE Company
    (
    CompanyId INTEGER Primary Key AUTOINCREMENT,
    CompanyName VARCHAR(200),
    CompanyNumber VARCHAR(50),
    AddressLine1 VARCHAR(200),
    AddressLine2 VARCHAR(100),
    City VARCHAR(100),
    State VARCHAR(50),
    PostalCode VARCHAR(50),
    Country VARCHAR(100),
    IsFortune500 CHAR(1)
    );
```

Creating an Auto-Increment Field

To create an auto-increment field, you can use the `AUTO_INCREMENT` attribute in MySQL or the `SERIAL` data type in PostgreSQL.

MySQL Example

```
CREATE TABLE Students (
    StudentID int NOT NULL AUTO_INCREMENT,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    PRIMARY KEY (StudentID)
);
```

Resetting Auto-Increment Value

To reset the auto-increment value in MySQL:

```
ALTER TABLE table_name AUTO_INCREMENT = value;
```

Example

```
ALTER TABLE Students AUTO_INCREMENT = 1000;
```

7. Transaction in SQL:

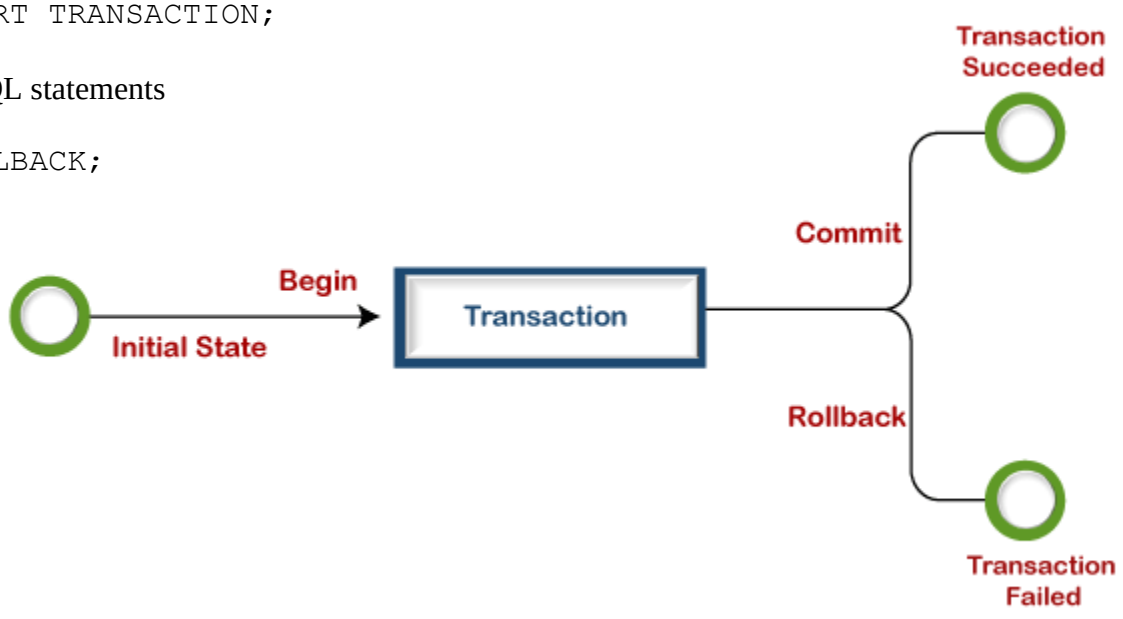
A transaction in SQL is a sequence of SQL statements that are executed as a single unit. Either all of the statements in a transaction are executed successfully, or none of them are. Transactions are used to ensure that data is consistent and accurate in a database.

Syntax:

```
START TRANSACTION;
-- SQL statements
COMMIT;
```

or

```
START TRANSACTION;
-- SQL statements
ROLLBACK;
```



Example 1:

-- Start a transaction

```
Start TRANSACTION;
```

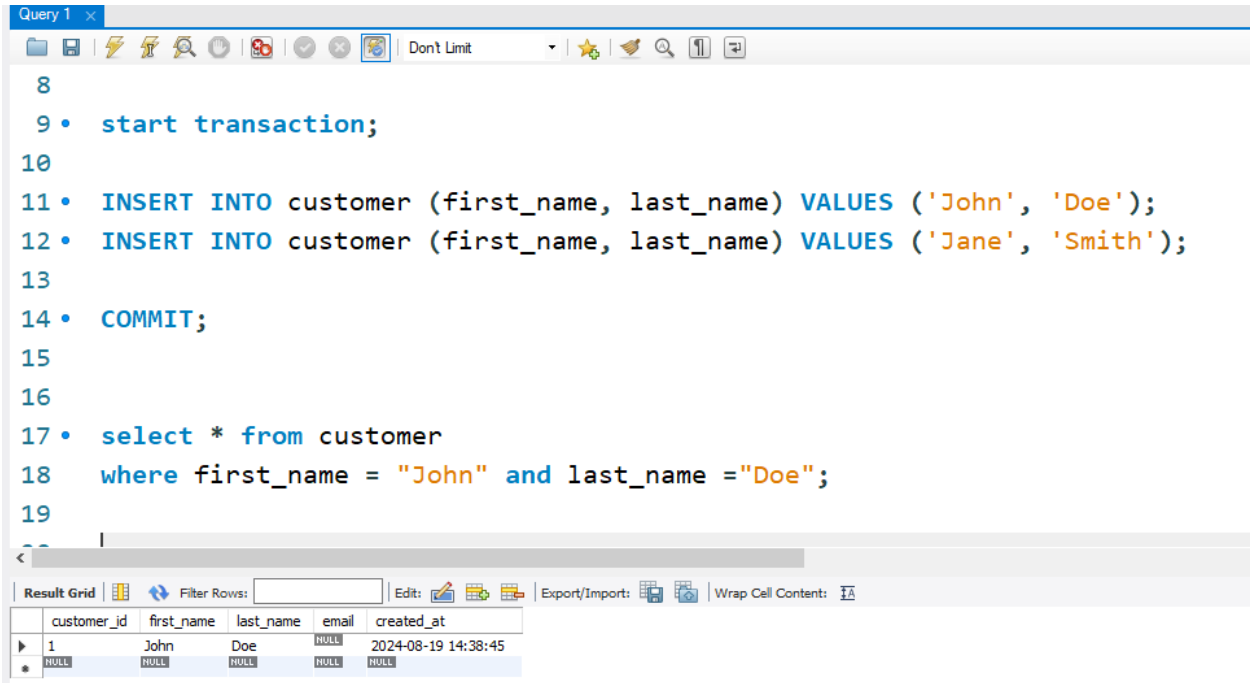
-- Insert data

```
INSERT INTO customer (first_name, last_name) VALUES ('John', 'Doe);

INSERT INTO customer (first_name, last_name) VALUES ('Jane', 'Smith');
```

-- Commit the transaction

```
COMMIT;
```

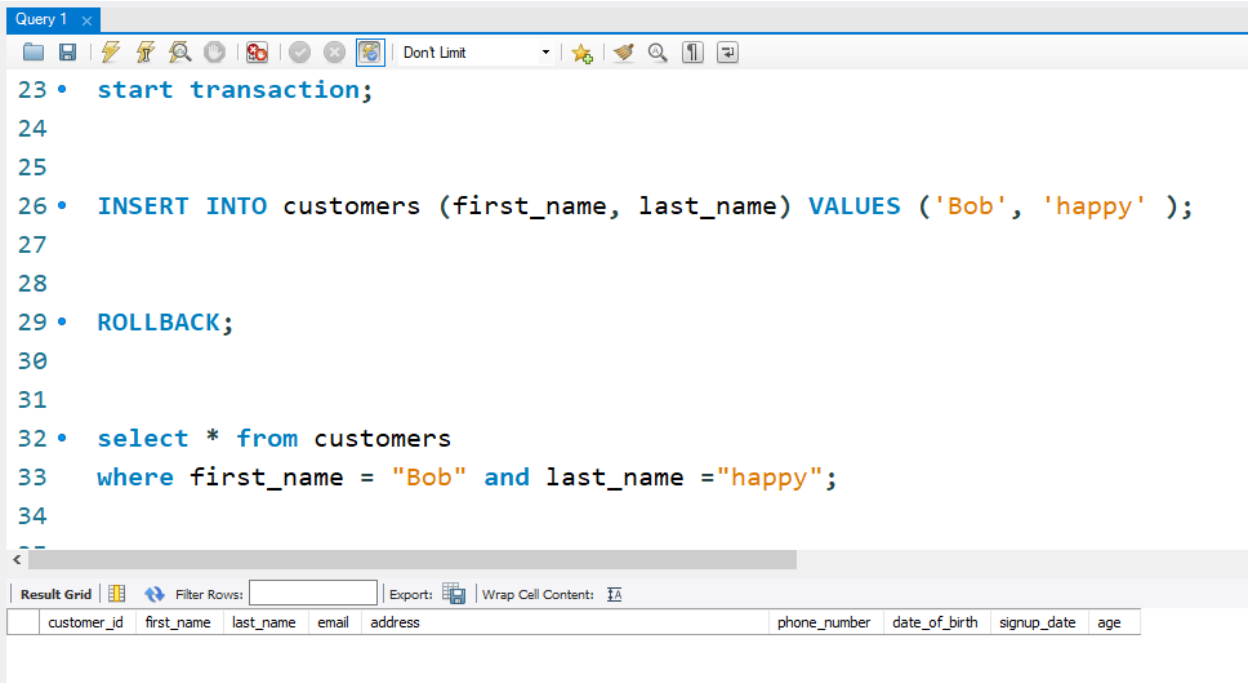


Example 2:

```
start transaction;

INSERT INTO customers (first_name, last_name) VALUES ('Bob', 'happy' );

ROLLBACK;
```



Best Practices

- Always use the `WHERE` clause to specify which records to update.
- Test your `UPDATE` statements with a `SELECT` query first.

Remember to always end your SQL statements with a semicolon (;) and to use appropriate whitespace and indentation for readability. Utilize these techniques to efficiently filter, sort, and update data in your SQL databases. Happy querying!