Table of Contents

Practice

1. SQL LIKE Operator

1A. Purpose

The SQL LIKE operator is used in a WHERE clause to search for a specified pattern in a column. It allows you to perform partial matches on string data, making it especially useful when you don't know the exact value you're searching for but know part of it. It is case-insensitive in some databases (e.g., MySQL), while in others (e.g., PostgreSQL), it can be case-sensitive unless modified.

1B. Syntax

The basic syntax for using the LIKE operator is:

```
SELECT column1, column2, ...
FROM table_name
WHERE column_name LIKE  pattern;
```

Wildcard Characters Used with LIKE

- % (percent sign): Represents zero, one, or multiple characters.
- _ (underscore): Represents a single character.

1C. Examples of SQL LIKE Operator

a. Using % Wildcard for Pattern Matching

A table called **Employees**:

EmployeeID	FirstName	LastName	Department
E01	John	Doe	Sales
E02	Jane	Doe	Marketing
E03	Alice	Johnson	IT
E04	Bob	Anderson	Sales
E05	Charlie	Lee	HR

Example 1: Find Employees Whose Last Names Start with 'D'

- The % after 'D' means it will match any characters that follow the letter "D". So, any last names starting with "D" will be retrieved.

```
SELECT FirstName, LastName
FROM Employees
WHERE LastName LIKE 'D%';
```

Result:

FirstName	LastName
John	Doe
Jane	Doe

Example 2: Find Employees Whose Last Names End with 'son'

- The % before 'son' means it will match any characters that come before "son", so any last name ending with "son" will be returned.

```
SELECT FirstName, LastName
FROM Employees
WHERE LastName LIKE '%son';
```

Result:

FirstName	LastName
Alice	Johnson

b. Using _ Wildcard for Single Character Match

Example 3: Find Employees Whose First Names Are Four Characters Long and Start with 'J'

- The three underscores ___ represent exactly three characters, so this query finds first names that are exactly four characters long and starts with "J".

```
SELECT FirstName, LastName
FROM Employees
 WHERE FirstName LIKE 'J___';
```

Result:

FirstName	LastName
John	Doe
Jane	Doe

c. Combining LIKE with Other Conditions

Example 4: Find Employees from the Sales Department Whose Last Names Start with 'A'

- This query uses both the LIKE operator and an exact match condition (Department = 'Sales') to filter employees whose last names start with "A" and who work in the Sales department.

- **Summary of SQL LIKE Operator Usage:**
  - **Purpose**: Perform pattern matching in string data.
  - **Syntax**: WHERE column_name LIKE pattern.
  - **Wildcards**:
    - % matches zero or more characters.
    - _ matches exactly one character.
  - **Applications**: Can be combined with other conditions (AND, OR) and used for complex database string searches.

```
SELECT FirstName, LastName, Department
FROM Employees
WHERE Department = 'Sales' AND LastName LIKE 'A%';
```

Result:

FirstName	LastName	Department
Bob	Anderson	Sales

2. WILDCARD Characters

2A. Purpose

Wildcard characters in SQL are used to substitute for one or more characters in a string when performing searches. They are most commonly used with the LIKE operator to allow for pattern matching in string data. The purpose is to find records where the data meets a partial pattern rather than an exact match.

2B. Wildcard Characters

- **%**: Represents zero, one, or multiple characters.
- _: Represents a single character.
- **[ ]**: Allows you to specify a range or a set of characters.
- **[^] or !**: Represents any character not in the set specified.

2C. Syntax

Wildcards are generally used in combination with the LIKE operator:

```
SELECT column1, column2, ...
FROM table_name
WHERE column_name LIKE pattern;
```

2D. Examples: % Wildcard: Match Zero, One, Or Multiple Characters

The % wildcard allows for flexible pattern matching. It can match any sequence of characters (including no characters at all).

Find Employees Whose First Name Starts with 'J' Suppose we have a table:

Employees:

EmployeeID	FirstName	LastName	Department
E01	John	Doe	Sales
E02	Jane	Doe	Marketing
E03	Alice	Johnson	IT
E04	Bob	Anderson	Sales
E05	Charlie	Lee	HR

Explanation: The % after 'J' means that any number of characters can follow 'J', so all first names starting with 'J' will be returned.

```
SELECT FirstName, LastName
FROM Employees
WHERE FirstName  LIKE 'J%';
```

Result:

FirstName	LastName
John	Doe
Jane	Doe

2E. _ Wildcard: Match Exactly One Character

The _ wildcard represents exactly one character. This is useful when you want to match strings with a specific length or format.

Explanation: The three underscores ___ represent exactly three additional characters after 'J', so only four-letter first names starting with 'J' are returned.

Find Employees Whose First Name is Four Characters Long and Starts with 'J'

```
SELECT FirstName, LastName
FROM Employees
WHERE FirstName LIKE 'J___';
```

Result:

FirstName	LastName
John	Doe
Jane	Doe

2F. [ ] Wildcard: Match Any Single Character Within A Range Or Set

The [ ] wildcard allows you to specify a range or a set of characters. It will match any single character within the brackets.

Explanation: This pattern will match any first name that starts with either 'J', 'A', or 'M'. The % allows for any number of characters to follow.

Find Employees Whose First Name Starts with 'J', 'A', or 'C'

```
SELECT FirstName, LastName
FROM Employees
WHERE FirstName LIKE '[JAC]%';
```

Result:

FirstName	LastName
John	Doe
Jane	Doe
Alice	Johnson
Charlie	Lee

2G. [^] or ! Wildcard: Match Any Character *Not* In A Set Or Range

The [^] or ! wildcard negates (invalidates) the set of characters, meaning that it will match any character *not* in the specified range.

Explanation: This query returns employees whose first names do not start with 'J'. The [^J] pattern ensures that the first character is any letter except 'J'.

Find Employees Whose First Name Does Not Start with 'J'

```
SELECT FirstName, LastName
FROM Employees
WHERE FirstName LIKE '[^J]%';
```

Result:

FirstName	LastName
Bob	Anderson
Charlie	Lee

Summary of Wildcard Characters:

- %: Matches zero or more characters (e.g., LIKE 'a%' matches "apple", "axe", "anything").
- _: Matches exactly one character (e.g., LIKE 'a_' matches "an", "as", "at").
- [ ]: Matches any single character within the specified range or set (e.g., LIKE '[a-c]%' matches "apple", "banana", but not "dog").
- [^]: Matches any character *not* in the specified set (e.g., LIKE '[^a-c]%' matches "dog", "elephant", but not "apple").

Wildcards are powerful tools for performing flexible searches and filtering data based on patterns rather than exact values.

3. IN Operator

3A. Purpose

The SQL IN operator is used to filter records based on whether a column's value matches any value in a specified list of values. It simplifies complex OR conditions by allowing you to specify multiple values in a single query, making it more concise and readable.

3B. KEY POINTS:

- Used in the WHERE clause to check if a value matches any of the values in a list.
- Reduces the need for multiple OR conditions.
- Can handle both numeric and string values

3C. Syntax

The basic syntax of the IN operator is:

```
SELECT column1, column2, ...
FROM table_name
WHERE column_name IN (value1, value2, ...);
```

Alternatively, it can be used in a NOT IN format to exclude records with specific values:

```
SELECT column1, column2, ...
FROM table_name
WHERE column_name NOT IN (value1, value2, ...);
```

3D. Example 1: Using IN to Filter Specific Departments

Suppose we have a table named **Employees**:

EmployeeID	FirstName	LastName	Department
E01	John	Doe	Sales
E02	Jane	Smith	Marketing
E03	Alice	Johnson	IT
E04	Bob	Anderson	Sales
E05	Charlie	Lee	HR
E06	Emily	Davis	IT

Explanation: The IN operator checks whether the value in the Department column is either "Sales" or "IT", and returns those rows.

You want to retrieve all employees who work in the "Sales" or "IT" departments.

```
SELECT FirstName, LastName, Department
FROM Employees
WHERE Department IN ('Sales', 'IT');
```

Result:

FirstName	LastName	Department
John	Doe	Sales
Bob	Anderson	Sales
Alice	Johnson	IT
Emily	Davis	IT

3E.Example 2: Using NOT IN to Exclude Specific Department

Explanation: The NOT IN operator excludes rows where the Department is either "Sales" or "IT", so only employees from "Marketing" and "HR" are returned.

You want to retrieve employees who do not work in the "Sales" or "IT" departments.

```
SELECT FirstName, LastName, Department
FROM Employees
WHERE Department NOT IN ('Sales', 'IT');
```

Result:

FirstName	LastName	Department
Jane	Smith	Marketing
Charlie	Lee	HR

3F. Example 3: Using IN with Numeric Values

Explanation: The IN operator checks if the EmployeeID is either 1, 3, or 5, and returns those rows.

You want to find employees whose EmployeeID is either 1, 3, or 5.

```
SELECT FirstName, LastName, EmployeeID
FROM Employees
WHERE EmployeeID IN (1,3,5);
```

Result:

FirstName	LastName	EmployeeID
John	Doe	1
Alice	Johnson	2
Charlie	Lee	3

3G. Example 4: Using IN with Subqueries

The **Departments** table:

DepartmentID	Department	Manager
D01	Sales	Manager 1
D02	Marketing	Manager 2
D03	IT	Manager 1
D04	HR	Manager 3

Explanation: The subquery retrieves departments managed by either "Manager1" or "Manager2" (Sales, IT, and Marketing), and then the outer query returns employees in those departments.

You want to retrieve all employees who are in departments managed by certain managers. The Departments table contains manager details, and you can use a subquery to fetch departments managed by "Manager1" or "Manager2".

Subquery with **IN**:

```
SELECT FirstName, LastName, Department
FROM Employees
WHERE Department IN (SELECT Department FROM Departments WHERE Manager IN ('Manager1',
'Manager2'));
```

Result:

FirstName	LastName	Department
John	Doe	Sales
Bob	Anderson	Sales

3H. Example 5: Using IN with Multiple Columns (MySQL)

In some SQL databases like MySQL, you can use the IN operator with multiple columns by comparing sets of values:

Explanation: The IN operator checks for the exact combination of FirstName and Department to match both conditions together.

You want to retrieve employees who work in specific departments and have specific first names

```
SELECT FirstName, LastName, Department
FROM Employees
WHERE (FirstName, Department) IN (('John', 'Sales'), ('Alice', 'IT'));
```

Result:

FirstName	LastName	Department
John	Doe	Sales
Alice	Johnson	IT

Summary of SQL IN Operator:

- **Purpose**: Filter records where a column's value matches any value in a list or set of values.
- **Syntax**:
  - For matching values: WHERE column_name IN (value1, value2, ...).
  - For excluding values: WHERE column_name NOT IN (value1, value2, ...).
- **Examples**:
  - Simple list matching (IN and NOT IN).
  - Numeric value matching.
  - Subqueries.
  - Multiple-column matching.

The IN operator simplifies queries and reduces the need for repetitive OR conditions.

4. UPDATE Statement:

4A. Purpose

The SQL UPDATE statement is used to modify existing records in a table. You can update one or more columns for one or more records in a table, depending on the conditions you specify. It is essential to use the WHERE clause when updating specific records; otherwise, all records in the table will be updated.

4B. Syntax

The basic syntax for the UPDATE statement is:

- table_name: The table where the update will take place.
- column1, column2: The columns that you want to update.
- value1, value2: The new values you want to assign to the columns.
- condition: The condition to specify which records to update. If no condition is provided, all rows will be updated.

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

Important Notes:

- If you omit(exclude) the WHERE clause, **every record in the table** will be updated.
- You can update multiple columns in one statement.

4C. Example 1: Update a Single Column

Suppose we have a table called Employees:

EmployeeID	FirstName	LastName	Department	Salary
E01	John	Doe	Sales	50000
E02	Jane	Smith	Marketing	55000
E03	Alice	Johnson	IT	60000
E04	Bob	Anderson	Sales	50000
E05	Charlie	Lee	HR	45000

Explanation: The UPDATE statement modifies the Salary column where the FirstName is "John" and the LastName is "Doe". Only the matching row is updated.

You want to increase the salary of "John Doe" to $55,000.

```
UPDATE Employees
SET Salary = 55000
WHERE FirstName = 'John' AND LastName = 'Doe';
```

Result:

EmployeeID	FirstName	LastName	Department	Salary
E01	John	Doe	Sales	55000
E02	Jane	Smith	Marketing	55000
E03	Alice	Johnson	IT	60000
E04	Bob	Anderson	Sales	50000
E05	Charlie	Lee	HR	45000

4D. Example 2: Update Multiple Columns

Explanation: The Department and Salary columns are both updated for "Jane Smith". The condition ensures that only her record is affected.

You want to update "Jane Smith's" department to "Sales" and increase her salary to $60,000.

```
UPDATE Employees
SET Department = 'Sales', Salary = 60000
WHERE FirstName = 'Jane' AND LastName = 'Smith';
```

Result:

EmployeeID	FirstName	LastName	Department	Salary
E01	John	Doe	Sales	55000
E02	Jane	Smith	Sales	60000
E03	Alice	Johnson	IT	60000
E04	Bob	Anderson	Sales	50000
E05	Charlie	Lee	HR	45000

4E. Example 3: Update Multiple Rows

Explanation: The WHERE Department = 'Sales' condition filters the rows where the department is "Sales", and the Salary column for those employees is increased by 5000. All employees in the Sales department are affected.

You want to increase the salary of all employees in the "Sales" department by $5,000.

```
UPDATE Employees
SET Salary = Salary + 5000
WHERE Department = 'Sales';
```

Result:

EmployeeID	FirstName	LastName	Department	Salary
E01	John	Doe	Sales	60000
E02	Jane	Smith	Sales	60000
E04	Bob	Anderson	Sales	55000

4F. Example 4: Update All Records (No WHERE Clause)

Explanation: Since no WHERE clause is provided, the Salary column is updated for all rows, increasing each employee's salary by 10%.

You want to give a flat 10% bonus to all employees by increasing their salary by 10%.

```
UPDATE Employees
SET Salary = Salary * 1.10;
```

Result:

EmployeeID	FirstName	LastName	Department	Salary
E01	John	Doe	Sales	66000
E02	Jane	Smith	Sales	66000
E03	Alice	Johnson	IT	66000
E04	Bob	Anderson	Sales	60500
E05	Charlie	Lee	HR	49500

4G. Example 5: Using a Subquery with UPDATE

The SalaryRules table:

Department	SalaryIncrease
Sales	5000
IT	3000
HR	4000

Explanation: The UPDATE statement modifies the Salary of each employee by adding the value from the SalaryRules table that matches the employee's department. This results in a customized salary increase based on department.

You want to increase the salary of employees based on a set of rules defined in another table (SalaryRules). The SalaryRules table specifies salary increases for each department.

Update with Subquery:

```
UPDATE Employees
SET Salary = Salary + (SELECT SalaryIncrease FROM SalaryRules WHERE Employees.Department =
SalaryRules.Department);
```

Result:

EmployeeID	FirstName	LastName	Department	Salary
E01	John	Doe	Sales	71000
E02	Jane	Smith	Sales	71000
E03	Alice	Johnson	IT	69000
E04	Bob	Anderson	Sales	65500
E05	Charlie	Lee	HR	53500

Summary of SQL UPDATE Statement:

- **Purpose**: Modify existing data in a table.

- **Syntax**:

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

- **Key Points:**
  - Always use a WHERE clause to specify which rows to update; otherwise, all rows in the table will be affected.
  - You can update multiple columns in a single UPDATE statement.
  - You can use subqueries to perform more dynamic updates based on data from other tables.

Examples show how the UPDATE statement can be used to modify single or multiple rows and columns, with or without conditions.

5. ALTER Statement:

5A. Purpose

The SQL ALTER statement is used to modify the structure of an existing database table. It allows you to add, delete, or modify columns, as well as other changes to the table, such as renaming columns or changing their data types. The ALTER statement provides flexibility in modifying tables without having to recreate them.

5B. Syntax

The syntax for the ALTER statement varies based on what you are trying to do. Below are some common use cases:

1. Add a new column:

```
ALTER TABLE table_name
ADD column_name data_type;
```

2. Drop (delete) a column:

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

3. Rename a column (MySQL):

```
ALTER TABLE table_name
RENAME COLUMN old_column_name TO new_column_name;
```

4. Rename the table:

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

5C. Example 1: Adding a New Column

Suppose you have a table called **Employees**:

EmployeeID	FirstName	LastName	Department	Salary
E01	John	Doe	Sales	50000
E02	Jane	Smith	Marketing	55000
E03	Alice	Johnson	IT	60000
E04	Bob	Anderson	Sales	50000
E05	Charlie	Lee	HR	45000

Explanation: The ALTER TABLE ... ADD command added a new column called HireDate with the DATE data type to store employee hire dates.

You want to add a HireDate column to store the date each employee was hired.

```
ALTER TABLE Employees
ADD HireDate DATE ;
```

Result:

Now, the table structure will have an additional column:

EmployeeID	FirstName	LastName	Department	Salary	HireDate
E01	John	Doe	Sales	50000	NULL
E02	Jane	Smith	Marketing	55000	NULL
E03	Alice	Johnson	IT	60000	NULL
E04	Bob	Anderson	Sales	50000	NULL
E05	Charlie	Lee	HR	45000	NULL

5D. Example 2: Dropping a Column

Explanation: The ALTER TABLE ... DROP COLUMN command removed the Department column from the Employees table.

You no longer need the Department column, so you decide to remove it.

```
ALTER TABLE Employees
DROP COLUMN Department;
```

Result:

The Department column is removed from the table:

EmployeeID	FirstName	LastName	Salary	HireDate
E01	John	Doe	50000	NULL
E02	Jane	Smith	55000	NULL
E03	Alice	Johnson	60000	NULL
E04	Bob	Anderson	50000	NULL
E05	Charlie	Lee	45000	NULL

5E. Example 3: Modifying a Column

Explanation: The ALTER TABLE ... MODIFY command changed the Salary column to DECIMAL(10, 2), which allows it to store numbers with up to 10 digits, 2 of which are after the decimal point.

You want to change the data type of the Salary column from INTEGER to DECIMAL so it can store more precise values.

```
ALTER TABLE Employees
MODIFY Salary DECIMAL(10, 2);
```

Result:

Now, the Salary column can store values like 50000.00 with two decimal places.

EmployeeID	FirstName	LastName	Salary	HireDate
E01	John	Doe	50000.00	NULL
E02	Jane	Smith	55000.00	NULL
E03	Alice	Johnson	60000.00	NULL
E04	Bob	Anderson	50000.00	NULL
E05	Charlie	Lee	45000.00	NULL

5F. Example 4: Renaming a Column

Explanation: The ALTER TABLE ... RENAME COLUMN command changed the name of the column from LastName to Surname.

You want to rename the LastName column to Surname.

```
ALTER TABLE Employees
RENAME COLUMN LastName TO Surname;
```

Result:

The column LastName is now renamed to Surname:

EmployeeID	FirstName	Surname	Salary	HireDate
E01	John	Doe	50000.00	NULL
E02	Jane	Smith	55000.00	NULL
E03	Alice	Johnson	60000.00	NULL
E04	Bob	Anderson	50000.00	NULL
E05	Charlie	Lee	45000.00	NULL

5G. Example 6: Adding a Constraint

Explanation: The ALTER TABLE ... ADD CONSTRAINT command adds a unique constraint to the Email column, ensuring all values in this column are unique.

You want to add a unique constraint on the Email column to ensure that no two employees have the same email address.

```
ALTER TABLE Employees
ADD CONSTRAINT unique_email UNIQUE (Email);
```

Result: Now, the Email column will have a unique constraint, preventing duplicate values.

Summary of SQL ALTER Statement:

- **Purpose**: Modify the structure of an existing table (e.g., add, delete, or modify columns).

- **Syntax**:
  - **Add a column**: ALTER TABLE table_name ADD column_name data_type;
  - **Drop a column**: ALTER TABLE table_name DROP COLUMN column_name;
  - **Modify a column**: ALTER TABLE table_name MODIFY column_name new_data_type;
  - **Rename a column**: ALTER TABLE table_name RENAME COLUMN old_column_name TO new_column_name;
  - **Rename the table**: ALTER TABLE old_table_name RENAME TO new_table_name;
  - **Add constraints**: ALTER TABLE table_name ADD CONSTRAINT ...;

Examples demonstrate how the ALTER statement can be used to make structural changes to tables, such as adding and removing columns, renaming columns and tables, modifying column data types, and adding constraints.

6. EXEC with sp_rename:

6A. Purpose

The SQL EXEC operator is used to execute a stored procedure or a dynamic SQL statement. When used with the system-stored procedure sp_rename, it allows you to rename database objects such as tables, columns, indexes, or user-defined data types.

The sp_rename system stored procedure is a Microsoft SQL Server feature, specifically used to rename objects in the database.

Purpose of sp_rename:

- **Rename database objects** such as:
  - Tables
  - Columns
  - Indexes
  - Data types

  Important: sp_rename is specific to Microsoft SQL Server and cannot be used in other database systems like MySQL or PostgreSQL.

6B. Syntax

- old_object_name: The current name of the object you want to rename.
- new_object_name: The new name you want to assign to the object.
- object_type: Optional. Specifies the type of object you are renaming. Some common object types are:
  - COLUMN: Renaming a column in a table.
  - INDEX: Renaming an index.
  - No object type is needed when renaming a table.

The basic syntax for using sp_rename is:

```
EXEC sp_rename 'old_object_name', 'new_object_name', 'object_type';
```

Key Points:

- sp_rename changes only the **name** of the object; it does not alter the object's definition or data.
- You need to be cautious while renaming columns, as this may affect queries, views, and stored procedures that reference the old column name.

6C. Example 1: Renaming a Table

Suppose you have a table called **Employees**:

EmployeeID	FirstName	LastName	Department	Salary
E01	John	Doe	Sales	50000
E02	Jane	Smith	Marketing	55000
E03	Alice	Johnson	IT	60000
E04	Bob	Anderson	Sales	50000
E05	Charlie	Lee	HR	45000

Explanation: The EXEC sp_rename command renames the table from Employees to Staff. All the data remains intact, and only the table name is changed.

You want to rename the Employees table to Staff.

```
EXEC sp_rename 'Employees', 'Staff';
```

Result:

The table Employees is now renamed to Staff.

Before renaming:

```
SELECT * FROM Employees;
```

After renaming:

```
SELECT * FROM Staff;
```

EmployeeID	FirstName	LastName	Department	Salary
E01	John	Doe	Sales	50000
E02	Jane	Smith	Marketing	55000
E03	Alice	Johnson	IT	60000
E04	Bob	Anderson	Sales	50000
E05	Charlie	Lee	HR	45000

6D. Example 2: Renaming a Column

Explanation: The EXEC sp_rename command renames the column LastName to Surname in the Staff table. After the renaming, you can refer to the column using the new name (Surname).

You want to rename the LastName column to Surname in the Staff table.

```
EXEC sp_rename 'Staff.LastName', 'Surname', 'COLUMN';
```

Result:

The column LastName has now been renamed to Surname.

Before renaming:

```
SELECT LastName FROM Staff;
```

After renaming:

```
SELECT Surname FROM Staff;
```

EmployeeID	FirstName	Surname	Department	Salary
E01	John	Doe	Sales	50000
E02	Jane	Smith	Marketing	55000
E03	Alice	Johnson	IT	60000
E04	Bob	Anderson	Sales	50000
E05	Charlie	Lee	HR	45000

6E. Example 3: Renaming an Index

Explanation: The EXEC sp_rename command changes the name of the index on the FirstName column from IDX_FirstName to IDX_EmployeeFirstName. The functionality of the index remains the same, only the name is changed.

Suppose you have an index on the FirstName column called IDX_FirstName, and you want to rename it to IDX_EmployeeFirstName.

```
EXEC sp_rename 'Staff.IDX_FirstName', 'IDX_EmployeeFirstName', 'INDEX';
```

Result: The index IDX_FirstName is now renamed to IDX_EmployeeFirstName.

6F. Example 4: Renaming a User-Defined Data Type

Explanation: The EXEC sp_rename command changes the name of the user-defined data type from employee_type to staff_type. Any columns or variables using this data type will now refer to it as staff_type.

You have a user-defined data type called employee_type, and you want to rename it to staff_type.

```
EXEC sp_rename 'employee_type', 'staff_type', 'USERDATATYPE';
```

Result: The user-defined data type employee_type is renamed to staff_type.

Summary of EXEC sp_rename:

- **Purpose**: Renames database objects such as tables, columns, indexes, or user-defined data types.

- Syntax:

```
EXEC sp_rename 'old_object_name', 'new_object_name', 'object_type';
```

- **Key Points**:
  - It can rename tables, columns, indexes, or data types.
  - Use COLUMN when renaming a column and INDEX when renaming an index.
  - The object type is optional when renaming tables.

Examples show how to rename various database objects using sp_rename. It is an essential command for managing and modifying the structure of database objects without impacting the

Exercise

Question 1:

- **Table Name**: Publisher
- **Primary Key**: PubliserID

PubliserID	Name	Address
P01	Pearson	Bukit Jalil
P02	Deitel	Puchong
P03	Rainbow	Subang
P04	MacHill	Kuala Lumpur

Attributes	Data Type
PublisherID	nvarchar(50)
Name	nvarchar(50)
Address	nvarchar(50)

- **Table Name**: Book
- **Primary Key**: BookID
- **Foreign Key:** PublisherID

BookID	Name	Author	Price	PublishedDate	PublisherID
B01	Maths	J.Wenton	50.60	10 Jan 2016	P01
B02	Science	S.Hanson	100.00	12 Feb 2016	P01
B03	English	K.Vince	89.30	9 March 2016	P02
B03	Biology	K.Vince	150.80	24 April 2016	P03
B05	Computing	J.Denzin	NULL	NULL	NULL

Attributes	Data Type
BookID	nvarchar(50)
Name	nvarchar(50)
Author	nvarchar(50)
Price	decimal(10,2)
PublisherDate	date
PurblisherID	nvarchar(50)

a. Using MS SQL Server, create a new database Lab5

b. Write a query to create the tables given above

c. Write a query to add each row of data to the tables

Answer a:

CREATE DATABASE Lab5

Answer b:

CREATE TABLE Publisher(PublisherID nvarchar(50) PRIMARY KEY, Name nvarchar(50), Address nvarchar(50))	CREATE TABLE Book(BookID nvarchar(50) PRIMARY KEY, Name nvarchar(50), Author nvarchar(50), Price decimal(10,2), PublishedDate date, PublisherID nvarchar(50) FOREIGN KEY REFERENCES Publisher(PublisherID))

Answer c:

INSERT INTO Publisher (PublisherID, Name, Address) VALUES ('P01','Pearson','Bukit Jalil'), ('P02','Deitel','Puchong'), ('P03','Rainbow','Subang'), ('P04','Machill','Kuala Lumpur')	INSERT INTO Book (BookID, Name, Author,Price,PublishedDate,PublisherID) Values ('B01', 'Math', 'J.Wenton', 50.60,'10 Jan 2016','P01'), ('B02', 'Science', 'S.Hanson', 100.00,'12 Feb 2016','P01'), ('B03', 'English', 'K.Vince', 89.30,'9 March 2016','P02'), ('B04', 'Biology', 'K.Vince', 150.80,'24 April 2016','P03'), ('B05', 'Computing', 'J.Denzin', NULL,NULL,NULL)

Question 2: Using Where Clause and SQL 'Like' operator

a. Display a list of Publishers where publisher's name starts with the alphabet 'r'

- **SELECT * FROM Publisher WHERE Name LIKE 'r%';**
-

	PublisherID	Name	Address
1	P03	Rainbow	Subang
2	P03	Rainbow	Subang

b. Display a list of Publishers where publisher's name ends with the alphabet 'n'

- **SELECT * FROM Publisher WHERE Name LIKE '%n';**
-

	PublisherID	Name	Address
1	P01	Pearson	Serdang

c. Display a list of Books where book name contains the alphabet 'a' in the second position

- **SELECT * FROM Book WHERE Name LIKE '_a%';**
-

	BookID	Name	Author	Price	PublishedDate	PublisherID
1	B01	Math	J.Wenton	50.60	2016-01-10	P01

d. Display a list of Books where the book name begins with the alphabet 'b' and is at least 2 characters in length

- **SELECT * FROM Book WHERE Name LIKE 'b_%';**
-

	BookID	Name	Author	Price	PublishedDate	PublisherID
1	B04	Biology	K.Vince	150.80	2016-04-24	P03

e. Display a list of Books where book name contains the alphabet 'i' in any position

- **SELECT * FROM Book Where Name LIKE'%i%';**
-

	BookID	Name	Author	Price	PublishedDate	PublisherID
1	B02	Science	S.Hanson	100.00	2017-09-03	P01
2	B03	English	K.Vince	98.00	2019-04-29	P02
3	B05	Computing	J.Denzin	NULL	NULL	NULL

f. Display a list of Books where the book name begins with the alphabet 'e' and ends with 'h'

- **SELECT * FROM Book Where Name LIKE'e%h';**
-

	BookID	Name	Author	Price	PublishedDate	PublisherID
1	B03	English	K.Vince	98.00	2019-04-29	P02

Question 3: Using Where Clause and multiple operators: LIKE, Order By

a. Display a list of Books where the author's name contains the alphabet 'n' in any position, and order the result in ascending order by author name and descending order by price.

- **SELECT* FROM Book WHERE Author LIKE '%n%' ORDER BY Author asc, Price desc;**

	BookID	Name	Author	Price	PublishedDate	PublisherID
1	B05	Computing	J.Denzin	NULL	NULL	NULL
2	B01	Math	J.Wenton	50.60	2016-01-10	P01
3	B04	Biology	K.Vince	150.80	2016-04-24	P03
4	B03	English	K.Vince	98.00	2019-04-29	P02
5	B02	Science	S.Hanson	100.00	2017-09-03	P01

b. Display a list of Books where author's name ends with the alphabet 'e' and is at least 3 characters in length, order the result in descending order by book name.

- **SELECT*FROM Book WHERE Author LIKE '%__e'Order by Name DESC;**

	BookID	Name	Author	Price	PublishedDate	PublisherID
1	B03	English	K.Vince	98.00	2019-04-29	P02
2	B04	Biology	K.Vince	150.80	2016-04-24	P03

Question 4: Using Where Clause and SQL 'In' operator

a. Display a list of Publishers whose address is Puchong or Subang

- **SELECT * FROM Publisher Where Address IN ('Puchong', 'Subang');**
-

	PublisherID	Name	Address
1	P02	Deitel	Puchong
2	P03	Rainbow	Subang

b. Display a list of Books whose price is 50 or 100

- **SELECT*FROM Book WHERE Price IN (50, 100);**
-

	BookID	Name	Author	Price	PublishedDate	PublisherID
1	B02	Science	S.Hanson	100.00	2017-09-03	P01

c. Display a list of Books whose name is Maths or Science or English, and order the results in ascending order by price

- **SELECT *FROM Book WHERE NAME IN('Math' , 'Scince' ,'English')Order by Price ASC;**
-

	BookID	Name	Author	Price	PublishedDate	PublisherID
1	B01	Math	J.Wenton	50.60	2016-01-10	P01
2	B03	English	K.Vince	98.00	2019-04-29	P02

Question 5: Using Where Clause and SQL 'Update' Statement

a. Publisher 'Pearson' had changed its address from Bukit Jalil to Serdang, update this

- **UPDATE Publisher**

  SET Address = 'Serdang' --SET is where we change the data

  WHERE PublisherID = 'P01';

-

	PublisherID	Name	Address
1	P01	Pearson	Serdang
2	P02	Deitel	Puchong
3	P03	Rainbow	Subang
4	P04	Machill	Kuala Lumpur

b. The price of the English Book written by K.Vince has changed from 89.30 to 99.30, update this.

- **UPDATE Book**

  SET Price = 99.30

  WHERE BookID = 'B03' AND Price = '89.30';

-

	BookID	Name	Author	Price	PublishedDate	PublisherID
1	B01	Math	J.Wenton	50.60	2016-01-10	P01
2	B02	Science	S.Hanson	100.00	2017-09-03	P01
3	B03	English	K.Vince	99.30	2019-04-29	P02
4	B04	Biology	K.Vince	150.80	2016-04-24	P03
5	B05	Computing	J.Denzin	NULL	NULL	NULL

c. Change the PublishedDate of Science book written by 'S.Hanson' to '3 September 2017'

- **UPDATE Book**

  SET PublishedDate = '3 September 2017'

  WHERE Name = 'Science';

-

	BookID	Name	Author	Price	PublishedDate	PublisherID
1	B01	Math	J.Wenton	50.60	2016-01-10	P01
2	B02	Science	S.Hanson	100.00	2017-09-03	P01
3	B03	English	K.Vince	99.30	2019-04-29	P02
4	B04	Biology	K.Vince	150.80	2016-04-24	P03
5	B05	Computing	J.Denzin	NULL	NULL	NULL

e. Update the price and the published date of the English book to ‘98’ and ’29 April 2019’

- **UPDATE Book**

  SET PRICE = 98 ,PublishedDate = '29 April 2019'

  WHERE Name = 'English';

	BookID	Name	Author	Price	PublishedDate	PublisherID
1	B01	Math	J.Wenton	50.60	2016-01-10	P01
2	B02	Science	S.Hanson	100.00	2017-09-03	P01
3	B03	English	K.Vince	98.00	2019-04-29	P02
4	B04	Biology	K.Vince	150.80	2016-04-24	P03
5	B05	Computing	J.Denzin	NULL	NULL	NULL

Question 6: Using SQL 'Alter Table' Statement

a. Change the data type of the Book name column to varchar(50)

- **ALTER TABLE Book**

  ALTER COLUMN Name varchar(50);

- Columns
  - BookID (PK, nvarchar(50), not null)
  - Name (varchar(50), null)
  - Author (nvarchar(50), null)
  - Price (decimal(10,2), null)
  - PublishedDate (date, null)
  - PublisherID (FK, nvarchar(50), null)

b. Add another column to the Publisher table, name it 'Telephone', and use integer as the data type

- **ALTER TABLE Publisher**

  ADD Telephone integer;

- Columns
  - PublisherID (PK, nvarchar(50), r
  - Name (nvarchar(50), null)
  - Address (nvarchar(50), null)
  - Telephone (int, null)

c. Change the newly added column in the Publisher table, change it from 'Telephone' to 'ContactNumber', and use an integer as the data type. (Hint: use sp_rename)

- **EXEC sp_rename 'Publisher.Telephone', 'ContactNumber';**

	PublisherID	Name	Address	ContactNumber
1	P01	Pearson	Serdang	NULL
2	P02	Deitel	Puchong	NULL
3	P03	Rainbow	Subang	NULL
4	P04	Machill	Kuala Lumpur	NULL

d. Delete the newly added column 'ContactNumber'

- **ALTER TABLE Publisher**

  DROP COLUMN ContactNumber;

-

	PublisherID	Name	Address
1	P01	Pearson	Serdang
2	P02	Deitel	Puchong
3	P03	Rainbow	Subang
4	P04	Machill	Kuala Lumpur

Question 7: Using SQL 'Delete' Statement

a. Delete a record, for the biology book written by 'K.Vince'.

- **DELETE FROM Book Where Name = 'Biology' AND Author = 'K.Vince'**
-

	BookID	Name	Author	Price	PublishedDate	PublisherID
1	B01	Math	J.Wenton	50.60	2016-01-10	P01
2	B02	Science	S.Hanson	100.00	2017-09-03	P01
3	B03	English	K.Vince	98.00	2019-04-29	P02
4	B05	Computing	J.Denzin	NULL	NULL	NULL

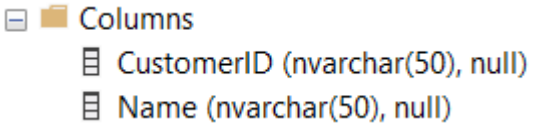
b. Remove the Maths book written by 'J.Wenton'

- **DELETE FROM Book WHERE Name = 'Math' and Author = 'J.Wenton'**
-

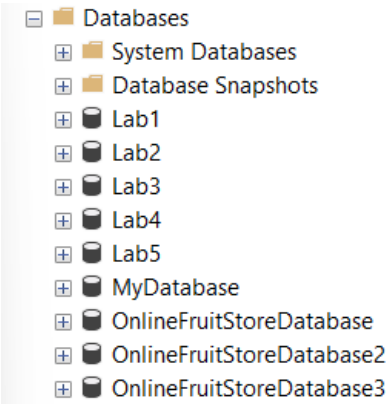
	BookID	Name	Author	Price	PublishedDate	PublisherID
1	B02	Science	S.Hanson	100.00	2017-09-03	P01
2	B03	English	K.Vince	98.00	2019-04-29	P02
3	B05	Computing	J.Denzin	NULL	NULL	NULL

Question 8: Using SQL 'Drop' Statement

a. Create a new table 'Customer', put ID and Name as attributes (use suitable data type), then use the 'Drop' statement to delete the whole table.

- **CREATE TABLE Customer(**
  CustomerID nvarchar(50),
  Name nvarchar(50)
- 

b. Create a new database 'MyDatabase', then use the 'Drop' statement to delete the whole Database.

- **CREATE DATABASE MyDatabase**
- 

- **DROP DATABASE MyDatabase**
-