SUMMARY: DATA CHANGING OPERATIONS

SESSION OVERVIEW:

By the end of this session, the students will be able to:

- Understand the concepts of inserting data in the existing table.
- Understand all the operations related to the ALTER statement.
- Understand the concepts of the SHOW statement.

KEY TOPICS AND EXAMPLES:

Understand the concepts of inserting data in the existing table:

Uses of inserting data:

- **Add New Records:** Inserting data allows you to add new rows (records) to a table, which is essential for storing information about new entities in your database.
- **Update Existing Data:** You can also use insertion to update existing records by adding new rows with modified data. This is useful for keeping your database up-to-date with changing information.
- **Maintain Data Integrity:** By inserting data, you can ensure that your database maintains data integrity and consistency, which is crucial for accurate data analysis and reporting.

Basic Syntax:

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

Note:

- ***table_name: The name of the table into which you want to insert data.***
- ***(column1, column2, ...): Optional. If specified, it indicates the columns into which you want to insert data. If omitted, it means you're inserting data into all columns, and the values must be in the same order as the columns in the table.***
- ***VALUES (value1, value2, ...): The values to insert into the specified columns. The number of values must match the number of columns or the number of columns specified in the column list.***

1. **Inserting Data into All Columns:**

   If you want to insert data into all columns of a table, you can omit the column list as follows:

Syntax:

```
INSERT INTO table_name
VALUES (value1, value2, ...);
```

Example 1:

```
INSERT INTO  customers
VALUES (13099, 'MR.', 'Louis', 'Young', '17-09-1965', '', 'M', 'M',
'Louis59@learnsector.com', '$70,000', 2, 'Graduate Degree',
'Management', 'Y', '');
```

The above query will help you to insert a row whose customer key is 13099 and the other values must be in the same order as the columns in the table.

To observe if the above entry has been successfully inserted or not. We will be using the SELECT statement.

```
SELECT *
FROM Customers
WHERE customerKey= 13099;
```

The output of the above query will help return all the entries associated with all the columns corresponding to the customerKey as 13099.

2. **Inserting Data into Specific Columns:**

   If you want to insert data into specific columns, you need to specify the column list as follows:

Syntax:

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

Example 1:

```
Insert into customers (CustomerKey, Prefix, FirstName, LastName,
BirthDate)
values (13100, 'MRS.', 'Marry', 'Huang', '15-10-1956');
```

The above query will help you to insert a row whose customer key is 13100 and the other values must be in the same order as the columns specified in the table. We will only enter the values of the specified columns and it will only affect those rows.

To observe if the above entry has been successfully inserted or not. We will be using the SELECT statement.

```
SELECT *
FROM Customers
WHERE customerKey= 13100;
```

Note: The output of the above query will help return all the entries associated with all the columns corresponding to the CustomerKey as 13100. Once we run the query, we will observe the data has been inserted for those columns that have been mentioned, and NULL for those columns which haven't been specified.

3. **Inserting Multiple Rows:**

   You can use a single INSERT INTO statement to insert multiple rows of data by specifying multiple sets of values separated by commas as follows:

Syntax:

```
INSERT INTO table_name (column1, column2, ...)
VALUES (value1a, value2a, ...),
       (value1b, value2b, ...),
       (value1c, value2c, ...);
```

Example 1:

```
INSERT INTO customers (CustomerKey, Prefix, FirstName, LastName,
BirthDate)
VALUES (13101, 'MS.', 'Lucy', 'Walker', '01-09-1989'),
       (13102, 'MR.', 'Danny', 'Walter', '30-03-1978');
```

The above query will help you to insert multiple rows We will only enter the values of the specified columns and it will only affect those columns which are being specified in the query.

Note: The above query doesn't indicate that we can only enter two rows. We can put a comma and add as many rows as we want. If the columns are not mentioned that means we do not have the records for those columns for that particular customer and the record will show as NULL.

To observe if the above entry has been successfully inserted or not. We will be using the SELECT statement.

```
SELECT *
FROM Customers
WHERE customerKey IN (13101, 13102);
```

Note: The output of the above query will help return all the entries associated with all the columns corresponding to the CustomerKey as 13101 and 13102. Once we run the query, we will observe the data has been inserted for those columns that have been mentioned, and NULL for those columns that haven't been specified as the default constraint for other columns is set as NULL. (We will discuss this in the constraints section in the upcoming sessions)

4. **Inserting Data from Another Table:**

   Inserting data from one table into another table in SQL involves using the INSERT INTO statement with a SELECT statement to specify the source table.

Steps in detail:

- **Identify the Source and Destination Tables:** Determine the names of the source table (the table from which you want to insert data) and the destination table (the table into which you want to insert data).
- **Choose the Columns**: Identify the columns from the source table that you want to insert into

the destination table. You can select all columns or specify individual columns.

- **Write the INSERT INTO Statement:** Use the INSERT INTO statement with a SELECT statement to specify the source table.

Syntax:

```
INSERT INTO destination_table (column1, column2, ...)
SELECT column1, column2, ...
FROM source_table
WHERE condition;
```

NOTE: Replace destination_table with the name of the destination table, source_table with the name of the source table, and (column1, column2, ...) with the columns you want to insert into in the destination table.

- **Verify the Insertion**: After executing the INSERT INTO statement, you can verify that the data was inserted into the destination table by querying the destination table:

```
SELECT * FROM destination_table;
```

This command will show you the inserted data in the destination table.

5. **Inserting NULL Values:**

   Inserting NULL values into a table in SQL is useful when you want to represent missing or unknown data in a column.
   This is another way of entering NULL values in the table for the integrity of data. This method is effective only when we do not set the column constraint as NULL.

Syntax:

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

Example 1:

```
INSERT INTO customers (CustomerKey, Prefix, FirstName, LastName)
VALUES (13103, 'MS.', 'Gracy', null );
```

Note:

- ***If the tables do not have constraints set for the columns of the table, then we have to manually enter the NULL values while inserting the data in the table.***
- ***Another way of entering the NULL values in the table is using the UPDATE statement which we have learnt in the previous session.***

To observe if the above entry has been successfully inserted or not. We will be using the SELECT statement.

```
SELECT *
FROM Customers
```

```
WHERE customerKey= 13103;
```

<u>Understanding all the operations related to the ALTER statement:</u>

The ALTER statement in SQL is a powerful tool that allows you to make various changes to the structure of your database tables after they have been created. It is an essential part of database management, as it enables you to modify tables without having to drop and recreate them, preserving the existing data.

The ALTER statement can be used to perform a wide range of operations, including adding, modifying, or dropping columns, as well as adding or dropping constraints and indexes. This flexibility makes it a versatile tool for database administrators and developers.

1. **Adding a Column:**
   Adding a column to an existing table in SQL using the ALTER TABLE statement involves specifying the table name, the column name, and the data type of the new column. Additionally, you can specify other column attributes such as NOT NULL, default values, and constraints if needed.

Syntax:

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

Note: Replace table_name with the name of your table, column_name with the name of the new column, and datatype with the desired data type.

Example:
Suppose we have a Customers table and the table consists of all the required demographic details. After a while, we figured out that the table lacked the details of the customer's whereabouts. Thus, we decided to add a separate column in the table named “Country”. Here is how we can add a separate column in the table whenever we feel the requirement.

```
ALTER TABLE Customers
ADD Country Varchar(50);
```

To observe whether the above changes have been successfully made, we will use the SELECT statement.

```
SELECT *
FROM Customers
```

NOTE: The above query will return all the columns and rows available in the table. Once the whole table is printed, we will notice that the column that we have created will have all the rows of that particular column as NULL values. This is because whenever we import any CSV file into the MySQL Workbench, the software will automatically set all the column constraints as NULL.

After executing the ALTER TABLE statement, you can verify that the column was added by querying the table's structure:

```
DESC table_name;
```

If you want to populate the new column with values, you can use an UPDATE statement to do so:

```
UPDATE table_name
SET column_name
WHERE condition;
```

Example 1:

```
UPDATE Customers
SET Country= 'China'
WHERE CustomerKey= 13103;
```

2. **Adding a column after a specific location:**

   When you add a column to an existing table, it is added as the last column, i.e., the last ordinal position. But there will be moments when we wouldn't want to add the column as the last column. Thus, we will have to specify the column name as where we want to place the column using AFTER.

Syntax:

```
ALTER TABLE table_name
ADD new_column_name varchar(255)
AFTER Desired_column;
```

Note: Replace table_name with the name of the table, new_column_name with the name of the new column that we have added to the table, and desired_column with the column after which you want to insert the new column.

Example:

```
ALTER TABLE Customers
ADD Regions varchar(255) AFTER EmailAddress;
```

Output:
The above query will add a separate column named "Regions" and will be added after the EmailAddress column as specified.

3. **Changing a data type:**

   Changing the data type of a column in an existing table using the ALTER TABLE statement in SQL involves specifying the table name, the column name, and the new data type.

Syntax:

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

Note: Replace table_name with the name of your table, column_name with the name of the column you want to modify, and new_datatype with the new data type you want to assign to the column.

Example 1:

```
ALTER TABLE customers
MODIFY COLUMN EmailAddress varchar(50);
```

Output:
The above query will change the data type of the EmailAddress column from TEXT to VARCHAR(50).

4. **Renaming a column:**

   Renaming a column in a table using the ALTER TABLE statement in SQL involves specifying the current column name and the new column name.

Syntax:

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

Example 1:

```
ALTER TABLE Customers
RENAME COLUMN BirthDate TO DateOfBirth;
```

Output:
The above output of the query will change the name of the BirthDate Column to the DateOfBirth column. There are moments when we have to name the columns of the table to increase the integrity of the data.

5. **Deleting/dropping a column:**

   Deleting a column from a table in SQL using the ALTER TABLE statement involves specifying the column name that you want to delete.

Syntax:

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

Example:

```
ALTER TABLE customers
DROP COLUMN MyUnknownColumn;
```

Output:
If you go through the customers table, then you will notice that there is a column that is not required, and thus, you would like to delete that column for proper data integrity. The above query will drop that particular column named MyUnknownColumn from the customers table. This is the initial step of data cleaning.

Understanding the concepts of SHOW statement:

In SQL, the SHOW statement is not a standard SQL command. It is specific to certain database management systems (DBMS) and is used to display information about databases, tables, or other database objects. The exact syntax and functionality of the SHOW statement can vary between different DBMSs.

Uses of SHOW statements:

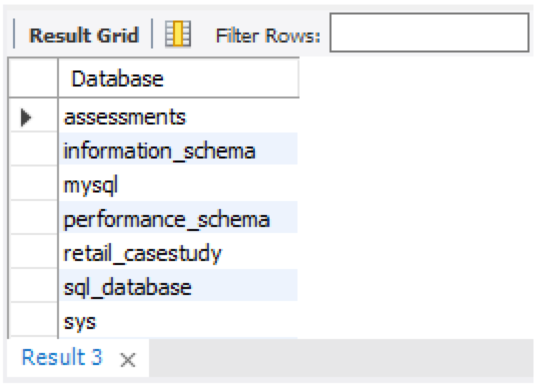
- **Display Database Objects:** The SHOW statement is used to display information about databases, tables, columns, indexes, and other database objects.
- **Schema Exploration:** It helps in exploring the database schema by listing available databases, tables, and their structures.
- **Learning and Documentation:** For beginners, the SHOW statement can be a valuable tool for learning about database structures and configurations. It also helps in documenting database schemas.

Syntax 1:

```
SHOW DATABASES;
```

Lists all databases on the MySQL server.

Output:



```
Database
assessments
information_schema
mysql
performance_schema
retail_casestudy
sql_database
sys
```

NOTE: These are the databases that are present in my system and will vary from person to person. The databases will vary as you name them and the databases you are working with or have imported into your system.

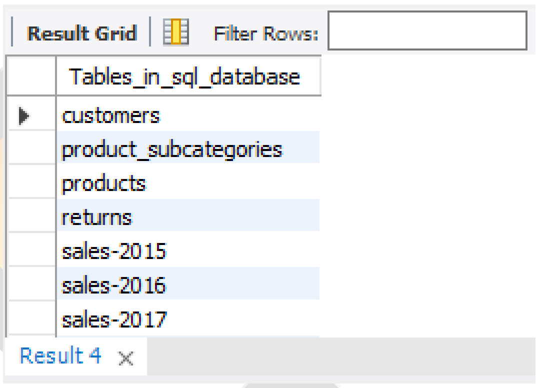
Syntax 2:

```
SHOW TABLES;
```

Lists all tables in the current database.

Output:

NOTE: The above query only fetches the names of the tables that are available in the default schema (The schema that you have set as default and wants to run all the queries on that particular schema.)



	Tables_in_sql_database
▶	customers
	product_subcategories
	products
	returns
	sales-2015
	sales-2016
	sales-2017

Syntax 3:

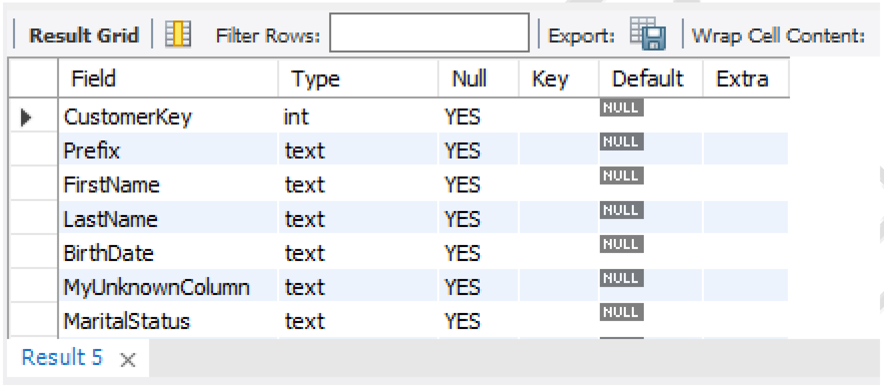
```
SHOW COLUMNS FROM table_name;
```

Lists the columns in a specific table.

Example 1:

```
SHOW COLUMNS FROM Customers;
```

Output:

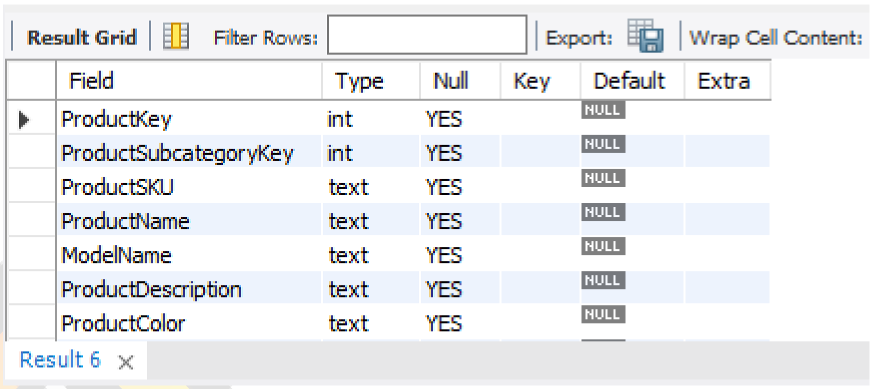


	Field	Type	Null	Key	Default	Extra
▶	CustomerKey	int	YES		NULL	
	Prefix	text	YES		NULL	
	FirstName	text	YES		NULL	
	LastName	text	YES		NULL	
	BirthDate	text	YES		NULL	
	MyUnknownColumn	text	YES		NULL	
	MaritalStatus	text	YES		NULL	

Example 2:

```
SHOW COLUMNS FROM Products;
```

Output:



Field	Type	Null	Key	Default	Extra
ProductKey	int	YES		NULL	
ProductSubcategoryKey	int	YES		NULL	
ProductSKU	text	YES		NULL	
ProductName	text	YES		NULL	
ModelName	text	YES		NULL	
ProductDescription	text	YES		NULL	
ProductColor	text	YES		NULL	

NOTE: Once you run the above query, the output will be similar to the DESCRIBE function that we have discussed in our previous sessions. So, the use case of SHOW COLUMN and DESCRIBE is the same.

Understanding of the concept of USE statement:

In SQL, the USE statement is used to switch the current database context to a different database. This statement is specific to certain database management systems (DBMS) and is not part of the SQL standard.

```
USE database_name;
```

Example:
Assume there are two databases, sales_db and hr_db. To switch to the sales_db database, you would use the following command:

```
USE sales_db;
```

After executing this statement, any subsequent SQL queries will be executed against the sales_db database unless specified otherwise.

Example to insert data in a particular database:
When writing SQL scripts that are intended to be run on different databases, using the USE statement at the beginning of the script ensures that the script operates on the correct database.

```
USE sales_db;

INSERT INTO customers (customer_id, customer_name) VALUES (1, 'John
Doe');
UPDATE orders SET order_status = 'Shipped' WHERE order_id = 101;
```

This is how we can utilize the USE statement followed by inserting data in the mentioned database.

Conclusion:
This session covered essential SQL operations critical for managing and manipulating database

structures and data effectively. The session began with the process of inserting data into tables, ensuring a comprehensive understanding of how to accurately populate tables with records. This was followed by an exploration of altering tables, which allows for modifying the structure of existing tables to adapt to evolving data requirements, such as adding, deleting, or modifying columns. These topics help to strengthen the fundamentals of SQL.