

SQL Commands With Examples

Jagriti Goswami

Outline

- SQL Overview
- Installation And Setup
- Creating A Database
- Creating A Table
- Inserting Values
- Retrieving All Rows And Columns
- Retrieving Subset Of Rows
- Retrieving Top N number Rows
- Retrieving Top X percent Rows
- Retrieving Subset Of Columns
- Counting Rows
- Updating Data
- Deleting Data
- Deleting Table
- Inserting Rows
- Copy Rows From One Table Into another
- Using Aliases
- The NULL Value
- NOT NULL Constraint
- DEFAULT Constraint
- UNIQUE Constraint
- Primary Key Constraint
- Foreign Key Constraint
- Changing A Schema
- Import Files
- Saving Query Results
- Filtering Data: WHERE Clause
- Filtering Data: LIKE Operator
- Filtering Data: IN Operator
- Removing Duplicates
- Ordering Results
- Grouping Data
- Conditional Expressions
- Accessing Related Tables
- Join Multiple Tables
- String Functions
- Numeric Functions
- Date And Time Functions
- Aggregating Data
- Concatenating Column Data
- Searching For Patterns
- UNION Operator
- Transactions

SQL Overview

- Structured Query Language (**SQL**)
- Creates a database and defines its structure
- Performs querying database
- Controls database security
- SQL has two sublanguage: Data control Language (DCL) & Data Manipulating Language (DML)
- DCL deals with defining database (creating new table, fields) and with database security
- DML deals with queries and data manipulation
- Databases that uses SQL: **MS SQL Server, Oracle, MySQL, SQLite, IBM DB2, Postgre SQL, Big Data**
- Install Azure Data Studio
- Tools used for SQL:
 - **Azure Data Studio** (Windows, Mac OS, Linux)
 - **SQL Server Management Studio (SSMS)** (Windows)
 - **SQL Server Data Tools (SSDT)** (Windows)
 - **Visual Studio Code (T-SQL)** (Windows, Mac OS, Linux)
 - **SQL Workbench**

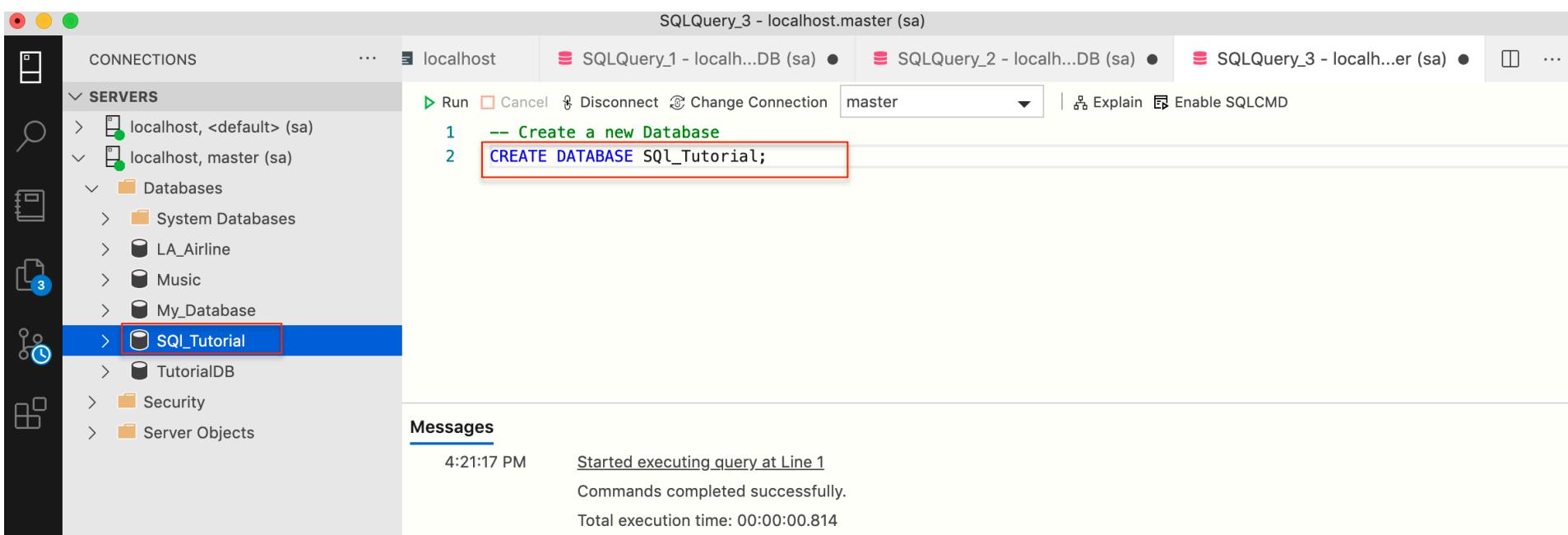
Installation and Setup

➤ Installation of Microsoft SQL Server 2017 on macOS Mojave version 10.14.4

1. Download and install **Docker** → Create Docker account ([Link](#))
 - Install Docker : Download Docker Community Edition for Mac. ([Link](#)).
 - This enables to run SQL Server from within a Docker container
 - To install Docker, download **.dmg** file → drag the **Docker.app** icon to **Application** folder
 - To launch Docker, open Docker → provide mac password .
 - Docker installs its networking components and links to the Docker apps
 - Increase the memory: from Docker icon select **Preferences** → change memory to 4GB (because SQL Server needs at least 3.25GB) → click **Apply & Restart**
2. Download and install **SQL Server**: On Terminal run → **\$ docker pull microsoft/mssql-server-linux**
3. Launch the **Docker Image**: open a Terminal window and run → **\$ docker run -d -name sql_server_demo -e 'ACCEPT_EULA=Y' -e 'SA_PASSWORD=reallyStrongPwd123' -p 1433:1433 microsoft/mssql-server-linux**
(name, password can be changed)
4. Check the Docker container: run → **\$ docker ps**
5. Install **sql-cli**: run **\$ npn install -g sql-cli** OR **\$ sudo npn install -g sql-cli**
6. Connect to SQL server: run → **\$ mssql -u sa -p reallyStrongPwd123**
7. Install **Azure Data Studio** ([Link](#)) & Connect to SQL server: open **Azure Data Studio** → **Welcome** page → under **Start** click **New Connection** → provide Server Name: **localhost**, Authentication Type: **SQL Login**, user name: **sa**, password: **reallyStrongPwd123**, database name: <default>, server group: <default>

Creating a Database

- Start **Azure Data Studio** (see instructions on installation and setup slide)
- Create a new database
 - Right-click on server, **localhost** → select **New Query** --> run following SQL command
CREATE DATABASE database_name;



The screenshot shows the Azure Data Studio interface. On the left is a sidebar with various icons and a list of databases under the 'Servers' section. The 'SQL_Tutorial' database is selected and highlighted with a red box. In the main pane, there is a query editor titled 'SQLQuery_3 - localhost.master (sa)'. The query window contains two lines of SQL code: '1 -- Create a new Database' and '2 CREATE DATABASE SQL_Tutorial;'. The second line is also highlighted with a red box. Below the query editor is a 'Messages' section showing the execution status: 'Started executing query at Line 1', 'Commands completed successfully.', and 'Total execution time: 00:00:00.814'.

```
-- Create a new Database
CREATE DATABASE SQL_Tutorial;
```

Creating a Table

- Create a new Table name **test** in the **SQL_Tutorial** Database
 - Change the connection context from **Master** to **SQL_Tutorial**
 - Run the following command as shown in below figure →

The screenshot shows the SSMS interface with the following details:

- Connections:** SQLQuery_2 - localhost.TutorialDB (sa)
- Servers:** localhost, <default> (sa), localhost, master (sa)
- Databases:** System Databases, LA_Airline, Music, My_Database, SQL_Tutorial
- Tables:** A table named **dbo.test** is selected and highlighted with a red box.
- Scripting:** The following T-SQL script is displayed in the query window:

```
1  -- Create a new table name test
2  CREATE TABLE test (
3      a INTEGER,
4      b TEXT
5 );
6
7  -- Selecting columns or show the results
8  SELECT * FROM test;
```
- Results:** The results pane shows a single row with columns **a** and **b**, both containing the value **a**.

Inserting Values

- Inserting values into table name **test** in the **SQL_Tutorial** Database
- Run the following command as shown in below figure →

The screenshot shows the SSMS interface with the following details:

- Object Explorer (Left):** Shows the database structure under the **localhost** server node. A red box highlights the **dbo.test** table entry.
- Query Window (Top Right):** Titled **SQLQuery_2 - localhost.SQL_Tutorial (sa)**. It contains the following SQL code:

```
-- Insert values in the table
SELECT * FROM test;

INSERT INTO test (a, b)
VALUES (1, 'a')

INSERT INTO test VALUES (2, 'b')
INSERT INTO test VALUES (3, 'c')

-- Select and run command "SELECT * FROM test;" to see the result
```
- Results Grid (Bottom Right):** Displays the data from the **test** table:

	a	b
1	1	a
2	2	b
3	3	c

A red box highlights the entire results grid.

Example Tables: Customer

- DB SQL_Tutorial, Table **Customer**:

```
1 CREATE TABLE Customer (
2     Customer_ID INT,
3     Name VARCHAR(20),
4     Address TEXT,
5     City VARCHAR(20),
6     State VARCHAR(20),
7     Zip INT,
8     Email TEXT
9 );
10
11    SELECT * FROM Customer;
```

Customer						
Customer_ID	Name	Address	City	State	Zip	Email
1	Sean Smith	110 Convoy St	San Diego	CA	92123	seansmith@wor...
2	Som Sharma	121 Pepper Tree Ln	Poway	CA	92064	somsharma@wor...
3	Jag Gos	235 Costa Verde	Irvine	CA	92112	jagos@work.com
4	Lili Golzs	211 Dorian St	San Francisco	CA	93678	liligolzs@wor...
5	Ryan Tomas	123 Indian St	Los Angeles	CA	91328	ryantomas@wor...

- Inserting values into table **Customer**:

```
1 -- Inserting values in the table Customer
2 SELECT * FROM Customer;
3
4 INSERT INTO Customer VALUES (1, 'Sean Smith', '110 Convoy St', 'San Diego', 'CA', 92123, 'seansmith@work.com')
5 INSERT INTO Customer VALUES (2, 'Som Sharma', '121 Pepper Tree Ln', 'Poway', 'CA', 92064, 'somsharma@work.com')
6 INSERT INTO Customer VALUES (3, 'Jag Gos', '235 Costa Verde', 'Irvine', 'CA', 92112, 'jagos@work.com')
7 INSERT INTO Customer VALUES (4, 'Lili Golzs', '211 Dorian St', 'San Francisco', 'CA', 93678, 'liligolzs@work.com')
8 INSERT INTO Customer VALUES (5, 'Ryan Tomas', '123 Indian St', 'Los Angeles', 'CA', 91328, 'ryantomas@work.com')
```

Example Tables: item, sale

➤ DB SQL_Tutorial, Table item:

```
1 CREATE TABLE item (
2     id INT,
3     name VARCHAR(20),
4     description TEXT
5 );
6
7 SELECT * FROM item;
```

➤ DB SQL_Tutorial, Table sale:

```
1 CREATE TABLE sale (
2     id INT,
3     item_id INT,
4     customer_id INT,
5     quantity INT,
6     price INT
7 );
8
9 SELECT * FROM sale;
```

item

id	name	description
1	T-shirt	Clothing
2	Pant	Clothing
3	Kitchen Towel	Household
4	Smart TV	Electronics
5	Perfume	Beauty

➤ Inserting values in table item:

```
1 -- Inserting data into table item
2 SELECT * FROM item;
3
4 INSERT INTO item VALUES (1, 'T-shirt', 'Clothing');
5 INSERT INTO item VALUES (2, 'Pant', 'Clothing');
6 INSERT INTO item VALUES (3, 'Kitchen Towel', 'Household');
7 INSERT INTO item VALUES (4, 'Smart TV', 'Electronics');
8 INSERT INTO item VALUES (5, 'Perfume', 'Beauty');
```

➤ Inserting values in table sale:

```
1 -- Inserting data into table sale
2 SELECT * FROM sale;
3
4 INSERT INTO sale VALUES (1, 2, 2, 3, 200);
5 INSERT INTO sale VALUES (2, 1, 4, 2, 150);
6 INSERT INTO sale VALUES (3, 4, 1, 1, 500);
7 INSERT INTO sale VALUES (4, 2, 5, 3, 200);
8 INSERT INTO sale VALUES (5, 5, 2, 3, 60);
```

sale

id	item_id	customer_id	quantity	price
1	2	2	3	200
2	1	4	2	150
3	4	1	1	500
4	2	5	3	200
5	5	2	3	60

Retrieving All Columns And Rows

- Selecting all columns and rows from table **Customer**
- Run the following command as shown in below figure →

The screenshot shows the SSMS interface with the following details:

- Left Sidebar:** Shows the "CONNECTIONS" section with a tree view of servers, databases, and objects.
- Top Bar:** Displays the connection information: "SQLQuery_3 - localhost.SQL_Tutorial (sa)".
- Toolbar:** Includes buttons for "Run", "Cancel", "Disconnect", "Change Connection", "SQL Tutorial" (selected), "Explain", and "Enable SQLCMD".
- Query Editor:** Contains the following T-SQL code:

```
1 -- Select all columns and rows
2 SELECT * FROM Customer;
3
4
```

The second line, `SELECT * FROM Customer;`, is highlighted with a red box.
- Results Grid:** A table titled "Results" showing the output of the query. The columns are: Customer_ID, Name, Address, City, State, Zip, and Email. The data is as follows:

	Customer_ID	Name	Address	City	State	Zip	Email
1	1	Sean Smith	110 Convoy St	San Diego	CA	92123	seansmith@wor...
2	2	Som Sharma	121 Pepper Tree Ln	Poway	CA	92064	somsharma@wor...
3	3	Jag Gos	235 Costa Verde	Irvine	CA	92112	jagos@work.com
4	4	Lili Golzs	211 Dorian St	San Francisco	CA	93678	liligolzs@wor...
5	5	Ryan Tomas	123 Indian St	Los Angeles	CA	91328	ryantomas@wor...

Retrieving Subset Of Rows

- Selecting specific rows from table **Customer**
- Run the following command as shown in below figure →

The screenshot shows the SSMS interface with the following details:

- Servers:** localhost, <default> (sa), localhost, master (sa), Databases (System Databases, LA_Airline, Music, My_Database), and SQL_Tutorial (Tables: dbo.Customer, dbo.item, dbo.sale, dbo.test, Views).
- Connections:** Welcome, localhost, SQLQuery_1 - localhost...DB (sa), SQLQuery_2 - localhost...al (sa).
- Toolbar:** Run, Cancel, Disconnect, Change Connection, SQL Tutorial, Explain, Enable SQLCMD.
- Query Window:** The query entered is:

```
1 -- Selecting specific rows from a table
2 SELECT * FROM Customer WHERE City = 'San Diego';
3
```
- Results Window:** The results show one row from the Customer table:

C...	Name	Address	City	State	Zip	Email
1	Sean Smith	110 Convoy St	San Diego	CA	92123	seansmith@work.com

Retrieving Top N number Rows

- Retrieving top N number rows from table **Disease** using **SELECT TOP** statement followed by number of rows
- Run the following command as shown in below figure →

The screenshot shows the SSMS interface with the following details:

- Left Sidebar (Object Explorer):** Shows the database structure under "localhost, master (sa) \ SQL_Tutorial". It includes nodes for Servers, Databases (System Databases, LA_Airline, Music, My_Database), Tables (Customer, Disease, item, Orders, sale, test_tb_1, test_tb_2, test_tb_3, Titanic), Views, Synonyms, Programmability, External Resources, Service Broker, Storage, and Security.
- Top Bar:** Displays the connection information: Welcome, localhost, SQLQuery_2 - localhost.SQL_Tutorial (sa), and SQLQuery_1 - localhost.SQL_Tutorial (sa). It also has buttons for Run, Cancel, Disconnect, Change Connection, Explain, and Enable SQLCMD.
- Query Editor:** Contains the T-SQL command: `1 SELECT TOP 10 * FROM Disease;`
- Results Grid:** A table titled "Results" showing the top 10 rows of the "Disease" table. The columns are: Disease, County, Year, Sex, Cases, Population, Lower_95_CI, Upper_95_CI, and Rate. The data is as follows:

	Disease	County	Year	Sex	Cases	Population	Lower_95_CI	Upper_95_CI	Rate
1	Amebiasis	ALAMEDA	2001	FEMALE	7	746596	0.377	1.932	0.938*
2	Amebiasis	ALAMEDA	2001	MALE	9	718968	0.572	2.376	1.252*
3	Amebiasis	ALAMEDA	2001	TOTAL	16	1465564	0.624	1.773	1.092*
4	Anaplasmosis and Ehrlichia...	ALAMEDA	2001	FEMALE	0	746596	0	0.494	-
5	Anaplasmosis and Ehrlichia...	ALAMEDA	2001	MALE	0	718968	0	0.513	-
6	Anaplasmosis and Ehrlichia...	ALAMEDA	2001	TOTAL	0	1465564	0	0.252	-
7	Anthrax	ALAMEDA	2001	FEMALE	0	746596	0	0.494	-
8	Anthrax	ALAMEDA	2001	MALE	0	718968	0	0.513	-
9	Anthrax	ALAMEDA	2001	TOTAL	0	1465564	0	0.252	-
10	Babesiosis	ALAMEDA	2001	FEMALE	0	746596	0	0.494	-

Retrieving Top X percent Rows

- Retrieving top x percent rows from table **Disease** using **SELECT TOP x PERCENT** statement
- Run the following command as shown in below figure →

The screenshot shows the SSMS interface. On the left is the Object Explorer with a tree view of servers, databases, tables, and other objects. In the center is a query window titled 'SQLQuery_2 - localhost.SQL_Tutorial (sa)' containing the following SQL code:

```
1 SELECT TOP 10 PERCENT * FROM Disease;
```

The results grid below shows 16 rows of data from the 'Disease' table. The first 12 rows are highlighted with a red border. The columns are: Disease, County, Year, Sex, Cases, Population, Lower_95_CI, Upper_95_CI, and Rate.

	Disease	County	Year	Sex	Cases	Population	Lower_95_CI	Upper_95_CI	Rate
1	Amebiasis	ALAMEDA	2001	FEMALE	7	746596	0.377	1.932	0.938*
2	Amebiasis	ALAMEDA	2001	MALE	9	718968	0.572	2.376	1.252*
3	Amebiasis	ALAMEDA	2001	TOTAL	16	1465564	0.624	1.773	1.092*
4	Anaplasmosis and Ehrlichia...	ALAMEDA	2001	FEMALE	0	746596	0	0.494	-
5	Anaplasmosis and Ehrlichia...	ALAMEDA	2001	MALE	0	718968	0	0.513	-
6	Anaplasmosis and Ehrlichia...	ALAMEDA	2001	TOTAL	0	1465564	0	0.252	-
7	Anthrax	ALAMEDA	2001	FEMALE	0	746596	0	0.494	-
8	Anthrax	ALAMEDA	2001	MALE	0	718968	0	0.513	-
9	Anthrax	ALAMEDA	2001	TOTAL	0	1465564	0	0.252	-
10	Babesiosis	ALAMEDA	2001	FEMALE	0	746596	0	0.494	-
11	Babesiosis	ALAMEDA	2001	MALE	0	718968	0	0.513	-
12	Babesiosis	ALAMEDA	2001	TOTAL	0	1465564	0	0.252	-
13	Botulism, Foodborne	ALAMEDA	2001	FEMALE	0	746596	0	0.494	-
14	Botulism, Foodborne	ALAMEDA	2001	MALE	0	718968	0	0.513	-
15	Botulism, Foodborne	ALAMEDA	2001	TOTAL	0	1465564	0	0.252	-
16	Botulism, Other	ALAMEDA	2001	FEMALE	0	746596	0	0.494	-

Retrieving Subset Of Columns

- Selecting specific columns and all rows from table **Customer**
- Run the following command as shown in below figure →

The screenshot shows the SSMS interface with the following details:

- Left Sidebar:** Displays the Object Explorer with connections to localhost, master, and a database named SQL_Tutorial. Under SQL_Tutorial, there are tables: dbo.Customer, dbo.item, dbo.sale, dbo.test, Views, Synonyms, Programmability, External Resources, and Service Broker.
- Top Bar:** Shows the connection to localhost, the current database SQL_Tutorial, and two other windows titled "SQLQuery_1 - localhost.SQI_Tutorial (sa)" and "SQLQuery_2 - localhost.SQI_Tutorial (sa)".
- Toolbar:** Includes buttons for Run, Cancel, Disconnect, Change Connection, Explain, and Enable SQLCMD.
- Query Window:** Contains the following T-SQL code:

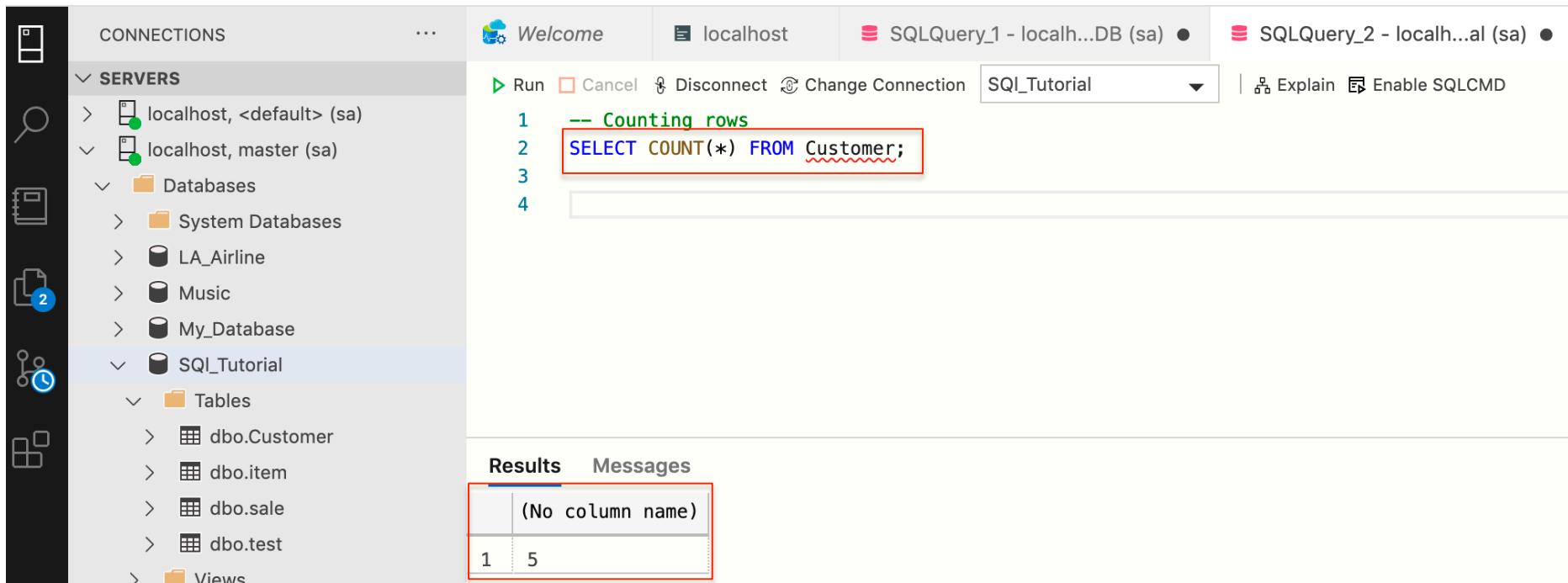
```
1 -- Selecting all rows and specific columns from table
2 SELECT Customer_ID , Name, Address, Email FROM Customer;
3
4
```

The second line of the query is highlighted with a red box.
- Results Grid:** Shows the output of the query with the following data:

	Customer_ID	Name	Address	Email
1	1	Sean Smith	110 Convoy St	seansmith@work.com
2	2	Som Sharma	121 Pepper Tree Ln	somsharma@work.com
3	3	Jag Gos	235 Costa Verde	jagos@work.com
4	4	Lili Golzs	211 Dorian St	liligolzs@work.com
5	5	Ryan Tomas	123 Indian St	ryantomas@work.com

Counting Rows

- Counting rows in table **Customer**
- Run the following command as shown in below figure →



The screenshot shows the SSMS interface with the following details:

- Left Sidebar:** Shows the "CONNECTIONS" section with two connections: "localhost, <default> (sa)" and "localhost, master (sa)". Under "localhost, master (sa)", there are nodes for "Databases", "System Databases", "LA_Airline", "Music", "My_Database", and "SQL_Tutorial". "SQL_Tutorial" is expanded, showing "Tables" with "dbo.Customer", "dbo.item", "dbo.sale", "dbo.test", and "Views".
- Top Bar:** Includes "Welcome", "localhost", "SQLQuery_1 - localh...DB (sa)", "SQLQuery_2 - localh...al (sa)", "Run", "Cancel", "Disconnect", "Change Connection", "SQL Tutorial" (selected), "Explain", and "Enable SQLCMD".
- Query Window:** Contains the following code:

```
1 -- Counting rows
2 SELECT COUNT(*) FROM Customer;
3
4
```

The second line, "SELECT COUNT(*) FROM Customer;", is highlighted with a red box.
- Results Window:** Labeled "Results" and "Messages". It displays a table with one row:

(No column name)
1 5

The entire table is also highlighted with a red box.

Updating Data

- Updating data in table **Customer**
- Run the following command as shown in below figure →

The screenshot shows the SSMS interface with the following details:

- Servers:** localhost, <default> (sa), localhost, master (sa) (selected).
 - Databases:** System Databases, LA_Airline, Music, My_Database, SQL_Tutorial (selected).
 - Tables:** dbo.Customer, dbo.item, dbo.sale, dbo.test, Views, Synonyms, Programmability, External Resources, Service Broker, Storage, Security, TutorialDB, Security, Server Objects.
- SQL Query Window:** Title: SQLQuery_2 - localhost.SQL_Tutorial (sa).
 - Toolbar: Run, Cancel, Disconnect, Change Connection, SQL_Tutorial (selected), Explain, Enable SQLCMD.
 - Text:

```
1 -- Updating data
2 -- Using UPDATE, SET statement, and WHERE Clause
3 -- Before:
4 SELECT * FROM Customer;
5
6 -- After:
7 UPDATE Customer SET Address = '203 Parkway Blvd', zip = '92022' WHERE Customer_ID = 3;
8
9 SELECT * FROM Customer;
```
- Results Grid:** Shows the state of the Customer table before and after the update.

	Customer_ID	Name	Address	City	State	Zip	Email
1	1	Sean S...	110 Convoy St	San Diego	CA	92123	seansmith@wor...
2	2	Som Sh...	121 Pepper T...	Poway	CA	92064	somsharma@wor...
3	3	Jag Gos	235 Costa Ve...	Irvine	CA	92112	jagos@work.com
4	4	Lili G...	211 Dorian St	San Francisco	CA	93678	liligolzs@wor...
5	5	Ryan T...	123 Indian St	Los Angeles	CA	91328	ryantomas@wor...

	Customer_ID	Name	Address	City	State	Zip	Email
1	1	Sean Smith	110 Convoy St	San Diego	CA	92123	seansmith@work.com
2	2	Som Sharma	121 Pepper Tree Ln	Poway	CA	92064	somsharma@work.com
3	3	Jag Gos	203 Parkway Blvd	Irvine	CA	92022	jagos@work.com
4	4	Lili Golzs	211 Dorian St	San Francisco	CA	93678	liligolzs@work.com
5	5	Ryan Tomas	123 Indian St	Los Angeles	CA	91328	ryantomas@work.com

Deleting Data

- Deleting data in table **Customer**
- Run the following command as shown in below figure →

The screenshot shows the SSMS interface with the following details:

- Left Sidebar:** Shows the Object Explorer with the tree structure of the database. The **SQL_Tutorial** database is selected.
- Top Bar:** Displays the connection status for **localhost, <default> (sa)**, **localhost, master (sa)**, and **SQLQuery_2 - localhost.SQL_Tutorial (sa)**. It also includes buttons for Run, Cancel, Disconnect, Change Connection, Explain, and Enable SQLCMD.
- Query Editor:** Contains the following T-SQL code:

```
1  -- Deleting data
2  SELECT * FROM Customer;
3
4  -- Delete data where Customer_id = 2
5  DELETE FROM Customer WHERE Customer_ID = 2;
6
7  -- Table after data deletion
8  SELECT * FROM Customer;
```

The line `DELETE FROM Customer WHERE Customer_ID = 2;` is highlighted with a red box.
- Results Grid:** Shows the initial data from the **Customer** table. The second row (Customer ID 2) is highlighted with a red box.

	Customer_ID	Name	Address	City	State	Zip	Email
1	1	Sean Smith	110 Convoy St	San Diego	CA	92123	seansmith@work.c...
2	2	Som Sharma	121 Pepper Tree Ln	Poway	CA	92064	somsharma@work.c...
3	3	Jag Gos	203 Parkway Blvd	Irvine	CA	92022	jagos@work.com
4	4	Lili Golzs	211 Dorian St	San Francisco	CA	93678	liligolzs@work.c...
5	5	Ryan Tomas	123 Indian St	Los Angeles	CA	91328	ryantomas@work.c...
- Results Grid (Second Query):** Shows the data after the delete operation. The second row (Customer ID 2) is missing.

	Customer_ID	Name	Address	City	State	Zip	Email
1	1	Sean Smith	110 Convoy St	San Diego	CA	92123	seansmith@work.com
2	3	Jag Gos	203 Parkway Blvd	Irvine	CA	92022	jagos@work.com
3	4	Lili Golzs	211 Dorian St	San Francisco	CA	93678	liligolzs@work.com
4	5	Ryan Tomas	123 Indian St	Los Angeles	CA	91328	ryantomas@work.com

Deleting Table

- Deleting existing table **test_tb** from **SQL_Tutorial** Database
- Run the following command as shown in below figure →

The screenshot shows the SSMS interface. On the left is the Object Explorer (File Explorer) with a tree view of servers, databases, tables, and views. The 'Tables' node under 'SQL_Tutorial' is selected. On the right is the 'SQLQuery_2 - localhost.SQL_Tutorial (sa)' window. It contains a query editor with the following code:

```
1  -- Delete a table
2  DROP TABLE test_tb;
3
4  -- Delete a table if exists
5  DROP TABLE IF EXISTS test_tb;
```

The second line ('DROP TABLE test_tb;') is highlighted with a red box. The fifth line ('DROP TABLE IF EXISTS test_tb;') is highlighted with a blue box. Below the query editor is the 'Messages' pane, which displays the execution log:

4:38:41 PM Started executing query at Line 5
Commands completed successfully.
Total execution time: 00:00:00.001

Inserting Rows

- Inserting rows into table
- Run the following command as shown in below figure →

The screenshot shows the SSMS interface. On the left is the Object Explorer tree, which includes connections, servers (localhost, master), databases (System Databases, LA_Airline, Music, My_Database), and the SQL_Tutorial database (Tables, Views, Synonyms, Programmability, External Resources, Service Broker, Storage, Security, TutorialDB, Security, Server Objects). The right side has tabs for Welcome, localhost, SQLQuery_1, and SQLQuery_2. The SQLQuery_2 tab is active, showing a query window with the following code:

```
SQLQuery_2 - localhost.SQL_Tutorial (sa)
Welcome    localhost    SQLQuery_1 - localh...DB (sa) •    SQLQuery_2 - localh...al (sa) •
Run Cancel Disconnect Change Connection SQL_Tutorial Explain Enable SQLCMD

1 -- Create a new table
2 CREATE TABLE test_tb_2 (c INT, d TEXT);
3
4 -- Inserting rows into a table
5 INSERT INTO test_tb_2 VALUES (1, 'a');
6 INSERT INTO test_tb_2 (c, d) VALUES (2, 'e');
7 INSERT INTO test_tb_2 DEFAULT VALUES;
8 INSERT INTO test_tb_2 (d) VALUES ('f');
9
10 -- Inserting rows from another table
11 INSERT INTO test_tb_2 (c, d) SELECT a, b FROM test;
12
13 -- Retrieve all the rows and columns
14 SELECT * FROM test_tb_2;
15
```

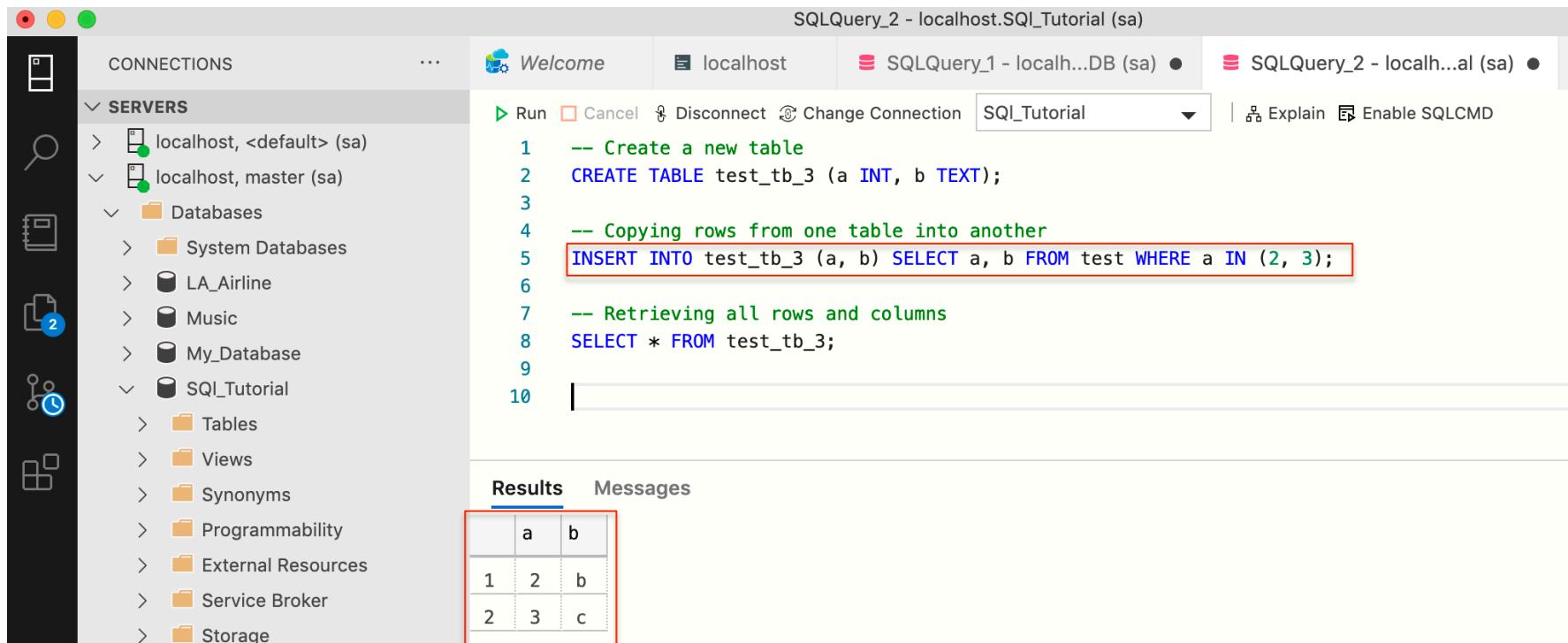
The code is divided into sections: creating the table, inserting rows directly, inserting rows from another table, and retrieving all rows. The section from line 5 to line 8 is highlighted with a red box. The section from line 11 to line 12 is also highlighted with a red box.

Below the query window is a Results grid showing the data inserted into the table:

	c	d
1	1	a
2	2	e
3	NULL	NULL
4	NULL	f
5	1	a
6	2	b
7	3	c
8	4	d

Copying Rows From One Table Into Another

- Copying rows from table **test** into table **test_tb_3** using WHERE clause-- WHERE a IN (2, 3)
- Run the following command as shown in below figure →



The screenshot shows the SSMS interface with the following details:

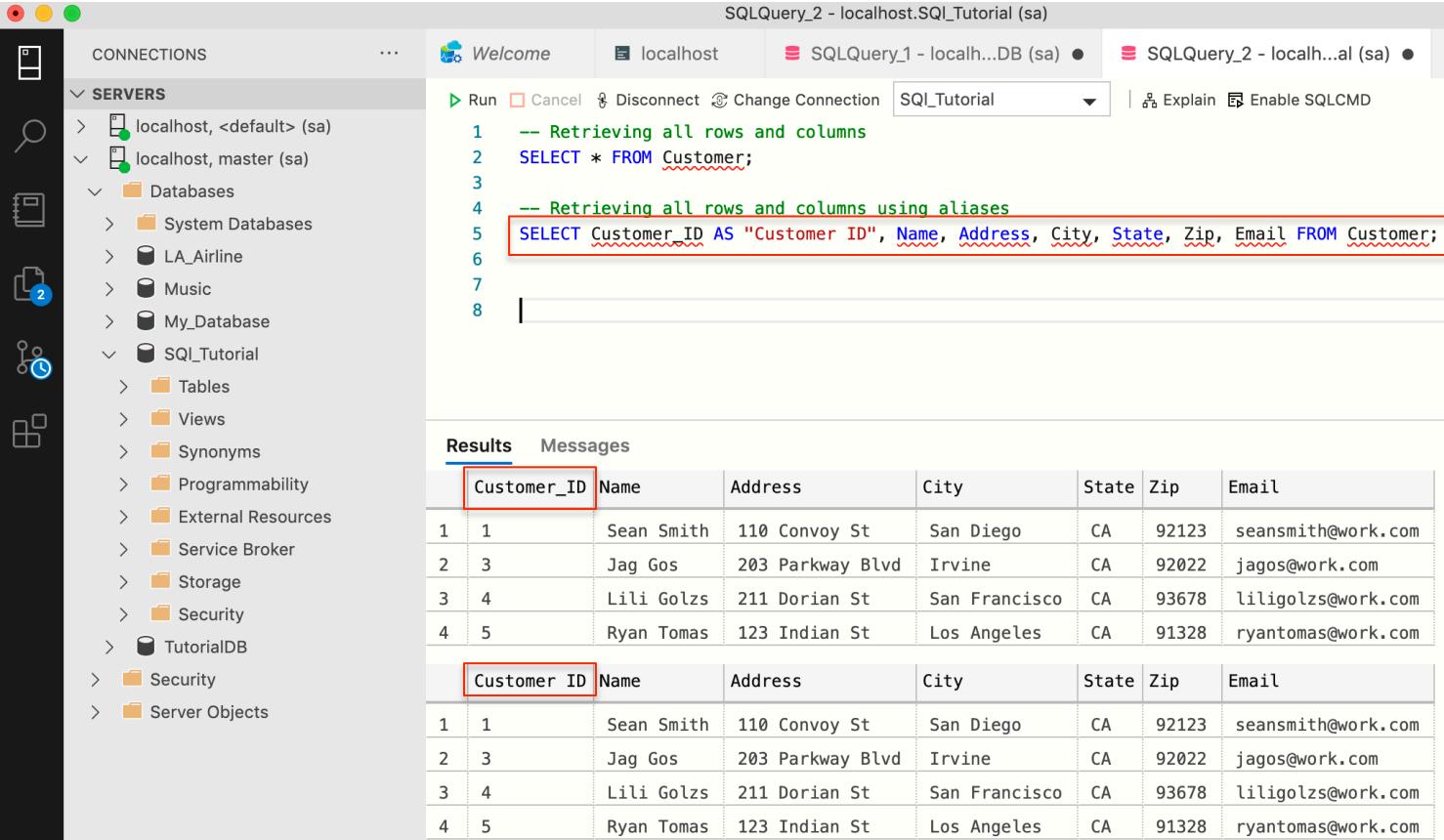
- Servers:** localhost, <default> (sa), localhost, master (sa), Databases (System Databases, LA_Airline, Music, My_Database), SQL_Tutorial (Tables, Views, Synonyms, Programmability, External Resources, Service Broker, Storage).
- Query Window:** SQLQuery_2 - localhost.SQL_Tutorial (sa). It contains the following T-SQL code:

```
1 -- Create a new table
2 CREATE TABLE test_tb_3 (a INT, b TEXT);
3
4 -- Copying rows from one table into another
5 INSERT INTO test_tb_3 (a, b) SELECT a, b FROM test WHERE a IN (2, 3);
6
7 -- Retrieving all rows and columns
8 SELECT * FROM test_tb_3;
9
10
```
- Results Grid:** A table showing the copied data with columns a and b. The data is:

	a	b
1	2	b
2	3	c

Using Aliases

- Retrieving rows and columns using aliases
- Run the following command as shown in below figure →



The screenshot shows the SSMS interface with the following details:

- Left Sidebar:** Shows the "CONNECTIONS" section with "localhost, <default> (sa)" selected. Below it, the "SERVERS" tree view includes "localhost, master (sa)", "Databases" (with "System Databases", "LA_Airline", "Music", "My_Database"), "SQI_Tutorial" (with "Tables", "Views", "Synonyms", "Programmability", "External Resources", "Service Broker", "Storage", "Security"), and "TutorialDB", "Security", "Server Objects".
- Top Bar:** Displays "SQLQuery_2 - localhost.SQI_Tutorial (sa)".
- Toolbar:** Includes "Run", "Cancel", "Disconnect", "Change Connection" (set to "SQI_Tutorial"), "Explain", and "Enable SQLCMD".
- Query Window:** Contains the following T-SQL code:

```
1 -- Retrieving all rows and columns
2 SELECT * FROM Customer;
3
4 -- Retrieving all rows and columns using aliases
5 SELECT Customer_ID AS "Customer ID", Name, Address, City, State, Zip, Email FROM Customer;
```

The second query is highlighted with a red border.
- Results Grid:** Two results grids are displayed. The first grid, under the "Results" tab, has columns: Customer_ID, Name, Address, City, State, Zip, Email. The second grid, also under "Results", has the same columns. Both grids show the same data for four rows:

	Customer_ID	Name	Address	City	State	Zip	Email
1	1	Sean Smith	110 Convoy St	San Diego	CA	92123	seansmith@work.com
2	3	Jag Gos	203 Parkway Blvd	Irvine	CA	92022	jagos@work.com
3	4	Lili Golzs	211 Dorian St	San Francisco	CA	93678	liligolzs@work.com
4	5	Ryan Tomas	123 Indian St	Los Angeles	CA	91328	ryantomas@work.com

	Customer ID	Name	Address	City	State	Zip	Email
1	1	Sean Smith	110 Convoy St	San Diego	CA	92123	seansmith@work.com
2	3	Jag Gos	203 Parkway Blvd	Irvine	CA	92022	jagos@work.com
3	4	Lili Golzs	211 Dorian St	San Francisco	CA	93678	liligolzs@work.com
4	5	Ryan Tomas	123 Indian St	Los Angeles	CA	91328	ryantomas@work.com

The NULL Value

- Retrieving rows with NULL value from table **test_tb_2**
- Retrieving rows which are NOT NULL from table **test_tb_2**
- Run the following command as shown in below figure →

The screenshot shows the SSMS interface with the following details:

- Connections:** Shows connections to localhost, master, and the current database (SQL_Tutorial).
- Query Window:** Title: SQLQuery_2 - localhost.SQL_Tutorial (sa).
Content:

```
1  -- Retrieve all rows and columns
2  SELECT * FROM test_tb_2;
3
4  -- Retrieve all the rows where c is NULL
5  SELECT * FROM test_tb_2 WHERE c IS NULL;
6
7  -- Retrieve all the rows where c is NOT NULL
8  SELECT * FROM test_tb_2 WHERE c IS NOT NULL;
```
- Results Grid:** Displays two tables of data. The first table has columns c and d, with rows 1, 2, 3, 4, 5, 6, and 7. The second table also has columns c and d, with rows 1, 2, and 3.
- Bottom Status Bar:** Shows AZURE and SQL SERVER BIG DATA CLUSTERS.

Constraining columns: NOT NULL

- Create a new table test_tb_1 with “NOT NULL” Constraint
- The insertion of a row which doesn’t have a value will fail
- Run the following command as shown in below figure →

The screenshot shows the SSMS interface with the following details:

- Left Sidebar:** Shows the "CONNECTIONS" section with "localhost, <default> (sa)" selected. Below it, the "SERVERS" tree shows "localhost, master (sa)", "Databases" (with "System Databases" expanded), and "SQL_Tutorial" (with "Tables" expanded to show "dbo.Customer", "dbo.item", "dbo.sale", "dbo.test", "dbo.test_tb_2", "dbo.test_tb_3"). Other nodes like "Views", "Synonyms", etc., are also listed.
- Top Bar:** Displays "SQLQuery_2 - localhost.SQL_Tutorial (sa)".
- Toolbar:** Includes "Run", "Cancel", "Disconnect", "Change Connection" (set to "SQL_Tutorial"), "Explain", and "Enable SQLCMD".
- Query Window:** Contains the following T-SQL script:

```
1 -- Create new table with NOT NULL constraint
2 CREATE TABLE test_tb_1 (
3     a INT NOT NULL,
4     b TEXT NOT NULL,
5     c TEXT,
6 );
7
8 -- Insert a row with 3 values (insertion successful)
9 INSERT INTO test_tb_1 (a, b, c) VALUES (1, 'My', 'SQL');
10
11 -- Insert a row without values (insertion failed)
12 -- Insert a row with 2 values in b and c
13 INSERT INTO test_tb_1 (b, c) VALUES ('My', 'SQL');
14
15 -- Retrieve all rows and columns
16 SELECT * FROM test_tb_1;
17
18
```
- Messages Window:** Shows the execution log:
 - 12:59:00 PM Started executing query at Line 13
 - Msg 515, Level 16, State 2, Line 13
 - Cannot insert the value NULL into column 'a', table 'SQL_Tutorial.dbo.test_tb_1'; column does not allow nulls. INSERT fails.

The last two items are highlighted with a red border.
- Status Bar:** Shows "The statement has been terminated." and "Total execution time: 00:00:00.002".

Constraining columns: DEFAULT

- Create a new table **test**
- Insert row values with DEFAULT constraint
- Run the following command as shown in below figure →

The screenshot shows the SSMS interface. On the left is the Object Explorer with a tree view of servers, databases, tables, and other objects. The central area is a query window titled "SQLQuery_2 - localhost.SQL_Tutorial (sa)". It contains the following SQL script:

```
1 -- Drop table if exists
2 DROP TABLE IF EXISTS test;
3
4 -- Create a table test
5 CREATE TABLE test (a INT DEFAULT 1, b TEXT, c TEXT);
6
7 -- Insert row with 3 values (column a, b, c)
8 INSERT INTO test (a, b, c) VALUES (22, 'John', 'CA');
9
10 -- Insert row with 2 values (column b, c)
11 INSERT INTO test (b, c) VALUES ('Shan', 'AZ');
12
13 SELECT * FROM test;
14
15
```

The lines of code for creating the table and inserting rows with the DEFAULT constraint are highlighted with a red border.

At the bottom, there are tabs for "Results" and "Messages". The "Results" tab displays the following table:

	a	b	c
1	22	John	CA
2	1	Shan	AZ

Constraining columns: UNIQUE

- Create a new table **test**
- Insert row values with UNIQUE constraint
- NULL value or NULL state is exempt from the UNIQUE constraint
- Run the following command as shown in below figure →

The screenshot shows the SSMS interface with the following details:

- Left Sidebar:** Shows the "CONNECTIONS" section with "localhost, <default> (sa)" and "localhost, master (sa)". It also displays a tree view of "Databases", "Tables", and other objects under "SQL_Tutorial".
- Top Bar:** Displays "SQLQuery_2 - localhost.SQL_Tutorial (sa)" and three tabs: "localhost", "SQLQuery_1 - localh...DB (sa)", and "SQLQuery_2 - localh...al (sa)".
- Query Window:** Contains the following T-SQL script:

```
1  -- Drop table test if exists
2  DROP TABLE IF EXISTS test;
3
4  -- Create table test
5  CREATE TABLE test (a INT UNIQUE, b TEXT, c TEXT);
6
7  -- Insert same value in column c
8  INSERT INTO test (a, b, c) VALUES (1, 'John', 'HONDA')
9  INSERT INTO test (a, b, c) VALUES (1, 'John', 'HONDA')
10 SELECT * FROM test;
11
```
- Results Tab:** Shows the output of the query:

```
Started executing query at Line 1
(1 row affected)
```
- Messages Tab:** Shows the error message:

```
Msg 2627, Level 14, State 1, Line 9
Violation of UNIQUE KEY constraint 'UQ__test__3BD0198FA10DBB5C'. Cannot insert duplicate key in object 'dbo.test'. The duplicate key value is (1).
```
- Bottom Status:** Shows "The statement has been terminated.", "(1 row affected)", and "Total execution time: 00:00:00.026".

Primary Key Constraint On Create Table

- An ID column is a column with unique, sequential values for each row in a table
- Create ID column with auto increment integer primary key
- Auto increment Primary key in SQL Server is defined using IDENTITY and PRIMARY KEY constraint
- Run the following command as shown in below figure →

The screenshot shows the SSMS interface with the following details:

- Servers:** localhost, master (sa) is selected.
- Tables:** The `test` table is being created with the following schema:
 - `id INT IDENTITY PRIMARY KEY`: The `id` column is defined as an integer with an identity constraint and set as the primary key.
 - `First_name TEXT`: The `First_name` column is defined as text.
 - `Last_name TEXT`: The `Last_name` column is defined as text.
 - `Age INT`: The `Age` column is defined as an integer.
- Results:** The data inserted into the `test` table is displayed in a grid:

	<code>id</code>	<code>First_name</code>	<code>Last_name</code>	<code>Age</code>
1	1	John	Smith	32
2	2	Shan	Sharm	38
3	3	Jim	Tang	42

Multi-Column Primary Key Constraint On Create Table

- Define PRIMARY KEY constraint on multiple columns when the table is created.
- Run the following command as shown in below figure →

The screenshot shows the SSMS interface with the following details:

- Servers View:** Shows connections to localhost, master, and a database named SQL_Tutorial. The SQL_Tutorial database is expanded to show its tables (Customer, item, sale, test), columns, and keys.
- SQL Query Window:** Titled "SQLQuery_2 - localhost.SQL_Tutorial (sa)". It contains the following SQL code:

```
2
3 -- Create a table by defining a PRIMARY KEY constraint on multiple columns/Composite
4 CREATE TABLE test (
5     id INT NOT NULL,
6     First_name VARCHAR(20),
7     Last_name VARCHAR(20) NOT NULL,
8     Age INT,
9     CONSTRAINT ID_Last_name PRIMARY KEY (id, Last_name)
10 );
11
12 INSERT INTO test (id, First_name, Last_name, Age) VALUES (1, 'John', 'Smith', 32);
13 INSERT INTO test (id, First_name, Last_name, Age) VALUES (2, 'Shan', 'Sharm', 38);
14 INSERT INTO test (id, First_name, Last_name, Age) VALUES (3, 'Jim', 'Tang', 42);
15 SELECT * FROM test;
16
```
- Results View:** Displays the data inserted into the table:

	id	First_name	Last_name	Age
1	1	John	Smith	32
2	2	Shan	Sharm	38
3	3	Jim	Tang	42

Multi-Column Primary Key Constraint On Existing Table

- Define PRIMARY KEY constraint on multiple columns on existing table.
- Run the following command as shown in below figure →

The screenshot shows the SSMS interface with the following details:

- Servers:** localhost, <default> (sa), localhost, master (sa), localhost, System Databases, LA_Airline, Music, My_Database, SQL_Tutorial.
- Databases:** SQL_Tutorial is selected.
- Tables:** dbo.Customer, dbo.item, dbo.sale, dbo.test.
- Keys:** A red box highlights the "Keys" node under the dbo.test table.
- SQL Query:** The query window contains the following T-SQL code:

```
1  DROP TABLE IF EXISTS test;
2
3  -- Create a table
4  CREATE TABLE test (
5      id INT NOT NULL,
6      First_name VARCHAR(20),
7      Last_name VARCHAR(20) NOT NULL,
8      Age INT,
9  );
10
11 INSERT INTO test (id, First_name, Last_name, Age) VALUES (1, 'John', 'Smith', 32);
12 INSERT INTO test (id, First_name, Last_name, Age) VALUES (2, 'Shan', 'Sharm', 38);
13 INSERT INTO test (id, First_name, Last_name, Age) VALUES (3, 'Jim', 'Tang', 42);
14 SELECT * FROM test;
15
16 -- Create Primary/Composite key while table is already created
17 ALTER TABLE test
18     ADD CONSTRAINT ID_Last_name PRIMARY KEY (id, Last_name);
19
```
- Results:** The results pane shows the data inserted into the test table:

	id	First_name	Last_name	Age
1	1	John	Smith	32
2	2	Shan	Sharm	38
3	3	Jim	Tang	42
- Messages:** No messages are present.

Disable, Enable and Drop A Primary Key Constraint

- Enable, Disable, and Drop a PRIMARY KEY constraint
- Run the following command as shown in below figure →

The screenshot shows the SSMS interface with the following details:

- Left pane (Object Explorer):** Shows the database structure. The "Keys" node under "dbo.test" is highlighted with a red box.
- Top bar:** CONNECTIONS, Welcome, localhost, SQLTutorial (selected), SQLQuery_1 - localhost...DB (sa), SQLQuery_2 - localhost...al (sa).
- Toolbar:** Run, Cancel, Disconnect, Change Connection, SQLTutorial, Explain, Enable SQLCMD.
- Code Editor:** Contains the following T-SQL script:

```
1  DROP TABLE IF EXISTS test;
2
3  -- Create a table by defining a PRIMARY KEY constraint on multiple columns/Composite Key
4  CREATE TABLE test (
5      id INT NOT NULL,
6      First_name VARCHAR(20),
7      Last_name VARCHAR(20) NOT NULL,
8      Age INT,
9  );
10
11 INSERT INTO test (id, First_name, Last_name, Age) VALUES (1, 'John', 'Smith', 32);
12 INSERT INTO test (id, First_name, Last_name, Age) VALUES (2, 'Shan', 'Sharm', 38);
13 INSERT INTO test (id, First_name, Last_name, Age) VALUES (3, 'Jim', 'Tang', 42);
14 SELECT * FROM test;
15
16 -- Create Primary/Composite key while table is already created
17 ALTER TABLE test
18     ADD CONSTRAINT ID_Last_name PRIMARY KEY (id, Last_name);
19
20 -- Disable Primary key
21 ALTER INDEX ID_Last_name ON test
22 DISABLE;
23
24 -- Enable Primary key
25 ALTER INDEX ID_Last_name ON test
26 REBUILD;
27
28 -- Drop a PRIMARY KEY constraint
29 ALTER TABLE test
30     DROP CONSTRAINT ID_Last_name;
```
- Bottom pane (Results):** Displays the data from the "test" table:

	id	First_name	Last_name	Age
1	1	John	Smith	32
2	2	Shan	Sharm	38
3	3	Jim	Tang	42

Foreign Key Constraint On Create Table

- A FOREIGN KEY links two tables together
- Foreign key in one table (child table) refers to the Primary key in another table (referenced or parent table)
- Define FOREIGN KEY constraint when the table is created.
- Concept is shown below with example tables **Customer**, and **Orders**.
- Run the following command as shown in below figure →

The screenshot shows the SSMS interface with the following details:

- Connections:** Shows a connection to "localhost, <default> (sa)".
- Databases:** Shows databases "System Databases", "LA_Airline", "Music", "My.Database", and "SQL_Tutorial".
- Tables:** Under "SQL_Tutorial", there are tables "dbo.Customer", "dbo.item", "dbo.Orders", and "dbo.Columns".
- Keys:** Under "dbo.Orders", there are two keys: "PK_Orders_F1E4639..." and "FK_Orders_Customer...". The "FK_Orders_Customer..." key is highlighted with a red box.
- Script:** The script pane contains T-SQL code for creating the "Customer" and "Orders" tables and inserting data. The line "FOREIGN KEY (Customer_ID) REFERENCES customer(Customer_ID)" is highlighted with a red box.
- Results:** The results pane shows the inserted data for the "Customer" table:

Customer_ID	Name	Address	City	State	Zip	Email
1	Sean Smith	110 Convoy St	San Diego	CA	92123	seansmith@work.com
2	Som Sharma	121 Pepper Tree Ln	Poway	CA	92064	somsharma@work.com
3	Jag Gos	235 Costa Verde	Irvine	CA	92112	jagos@work.com
4	Lili Golzs	211 Dorian St	San Francisco	CA	93678	liligolzs@work.com
5	Ryan Tomas	123 Indian St	Los Angeles	CA	91328	ryantomas@work.com

- Messages:** The messages pane is empty.

Multi-Column Foreign Key Constraint On Create Table

- Define multi-column FOREIGN KEY constraint when the table is created.
- Concept is shown below with example tables **Customer**, and **Orders**.
- Run the following command as shown in below figure →

The screenshot shows the SSMS interface with the following details:

- Servers:** localhost, master (sa)
- Tables:** dbo.Customer, dbo.item, dbo.Orders
- Keys:** PK_Orders_F1E4639... (highlighted with a red box), FK_CustomerID_Order (highlighted with a red box)
- SQL Query:** A script for creating the Orders table and defining a multi-column foreign key constraint. The relevant part is highlighted with a red box:

```
CREATE TABLE Orders (
    Order_ID int NOT NULL PRIMARY KEY,
    Order_Number int NOT NULL,
    Customer_ID int,
    CONSTRAINT FK_CustomerID_Order FOREIGN KEY (Customer_ID) REFERENCES customer(Customer_ID)
```
- Results:** A table showing customer data:

	Customer_ID	Name	Address	City	State	Zip	Email
1	1	Sean Smith	110 Convoy St	San Diego	CA	92123	seansmith@work.com
2	2	Som Sharma	121 Pepper Tree Ln	Poway	CA	92064	somsharma@work.com
3	3	Jag Gos	235 Costa Verde	Irvine	CA	92112	jagos@work.com
4	4	Lili Golzs	211 Dorian St	San Francisco	CA	93678	liligolzs@work.com
5	5	Ryan Tomas	123 Indian St	Los Angeles	CA	91328	ryantomas@work.com
- Messages:** A table showing order data:

	Order_ID	Order_Number	Customer_ID
1	11	30	1
2	22	31	2
3	33	32	3
4	44	33	4
5	55	34	5

Foreign Key Constraint On Existing Table

- Define FOREIGN KEY constraint existing table.
- Concept is shown below with example tables **Customer**, and **Orders**.
- Run the following command as shown in below figure →

The screenshot shows the SSMS interface with the following details:

- Servers:** localhost, <default> (sa), localhost, master (sa), System Databases, My_Database, SQL_Tutorial, Tables, Customer, item, Orders, Columns, Keys (highlighted with a red box).
- SQL Query Window:** SQLQuery_2 - localhost.SQL_Tutorial (sa). The code is as follows:

```
27     Customer_ID int,
28 );
29
30 -- Insert values in the table orders
31 -- Explicitly pass the values for Customer_ID column in Order table
32 -- Foreign Key will not automatically inserted
33 INSERT INTO orders (Order_ID, Order_Number, Customer_ID) VALUES (11, 30, 1);
34 INSERT INTO orders (Order_ID, Order_Number, Customer_ID) VALUES (22, 31, 2);
35 INSERT INTO orders (Order_ID, Order_Number, Customer_ID) VALUES (33, 32, 3);
36 INSERT INTO orders (Order_ID, Order_Number, Customer_ID) VALUES (44, 33, 4);
37 INSERT INTO orders (Order_ID, Order_Number, Customer_ID) VALUES (55, 34, 5);
38 SELECT * FROM Orders;
39
40 -- Create Foreign key constraint on Customer_ID column when Orders table is already created
41 ALTER TABLE Orders
42 ADD FOREIGN KEY (Customer_ID) REFERENCES Customer(Customer_ID);
```

Results Window: Displays two tables of data:

	Customer_ID	Name	Address	City	State	Zip	Email
1	1	Sean Smith	110 Convoy St	San Diego	CA	92123	seansmith@work.com
2	2	Som Sharma	121 Pepper Tree Ln	Poway	CA	92064	somsharma@work.com
3	3	Jag Gos	235 Costa Verde	Irvine	CA	92112	jagos@work.com
4	4	Lili Golzs	211 Dorian St	San Francisco	CA	93678	liligolzs@work.com
5	5	Ryan Tomas	123 Indian St	Los Angeles	CA	91328	ryantomas@work.com

	Order_ID	Order_Number	Customer_ID
1	11	30	1
2	22	31	2
3	33	32	3
4	44	33	4
5	55	34	5

Multi-Column Foreign Key Constraint On Existing Table

- Define multi-column FOREIGN KEY constraint on existing table.
- Concept is shown below with example tables **Customer**, and **Orders**.
- Run the following command as shown in below figure →

The screenshot shows the SSMS interface with the following details:

- Servers:** localhost, master (sa)
- Databases:** System Databases, LA_Airline, Music, My_Database, SQL_Tutorial
- Tables:** dbo.Customer, dbo.item, dbo.Orders
- Keys:** PK_Orders_F1E4639... (highlighted with a red box), FK_Customer_ID_Order (highlighted with a red box)
- SQL Query:** The code pane contains T-SQL commands to insert data into the Orders table and create a multi-column foreign key constraint named FK_Customer_ID_Order on the Customer_ID column of the Orders table, referencing the Customer_ID column in the Customer table.
- Results:** A table showing customer information:

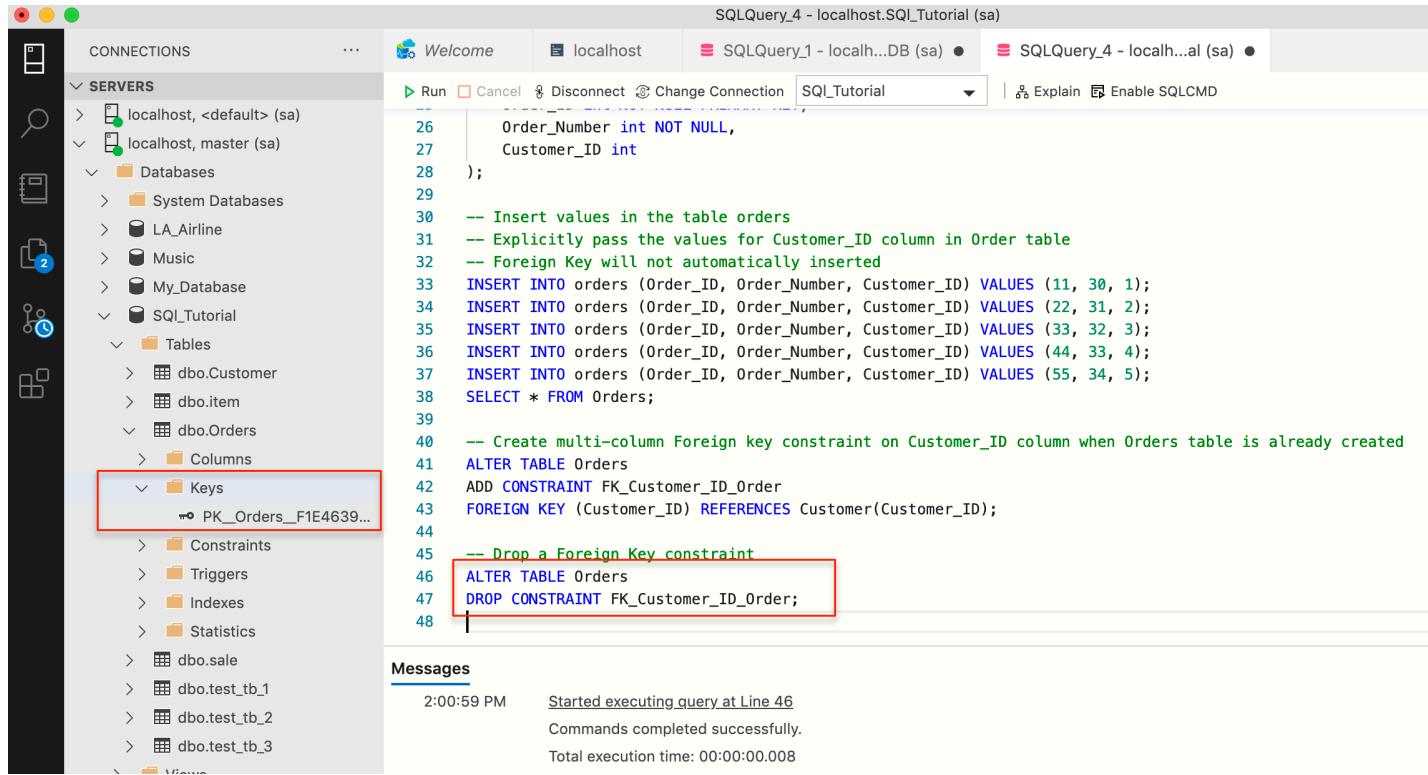
	Customer_ID	Name	Address	City	State	Zip	Email
1	1	Sean Smith	110 Convoy St	San Diego	CA	92123	seansmith@work.com
2	2	Som Sharma	121 Pepper Tree Ln	Poway	CA	92064	somsharma@work.com
3	3	Jag Gos	235 Costa Verde	Irvine	CA	92112	jagos@work.com
4	4	Lili Golzs	211 Dorian St	San Francisco	CA	93678	liligolzs@work.com
5	5	Ryan Tomas	123 Indian St	Los Angeles	CA	91328	ryantomas@work.com

- Messages:** A table showing order information:

	Order_ID	Order_Number	Customer_ID
1	11	30	1
2	22	31	2
3	33	32	3
4	44	33	4
5	55	34	5

Drop A Foreign Key Constraint

- Drop a FOREIGN KEY constraint
- Run the following command as shown in below figure →

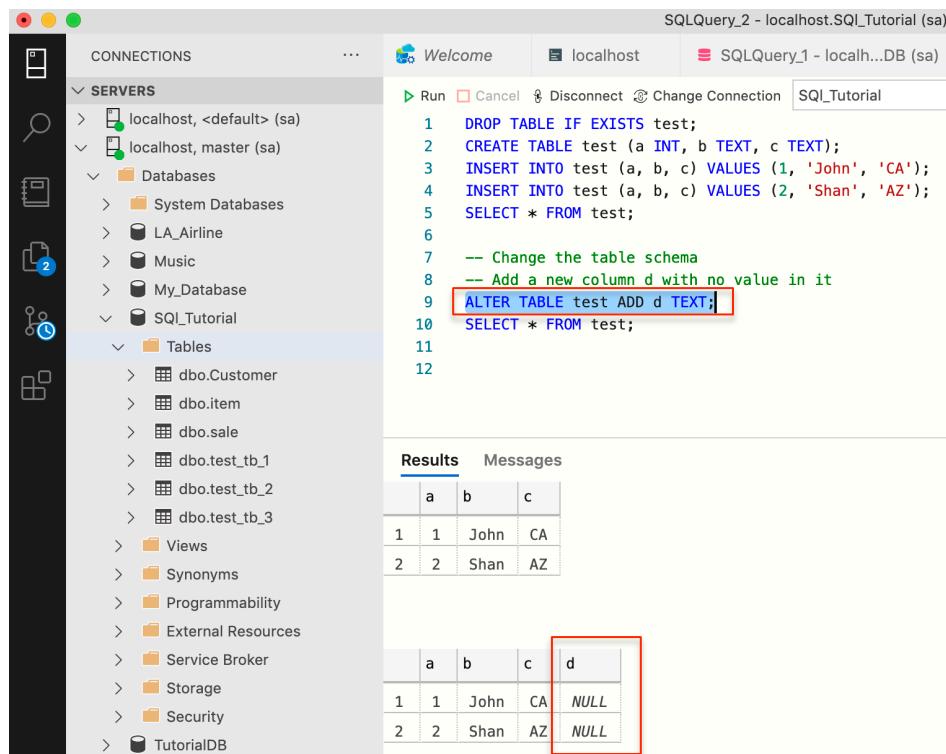


```
SQLQuery_4 - localhost.SQL_Tutorial (sa)
CONNECTIONS ... Welcome localhost SQLQuery_1 - localhost...DB (sa) SQLQuery_4 - localhost...al (sa) SQL_Tutorial Explain Enable SQLCMD
26     Order_Number int NOT NULL,
27     Customer_ID int
28 );
29
30 -- Insert values in the table orders
31 -- Explicitly pass the values for Customer_ID column in Order table
32 -- Foreign Key will not automatically inserted
33 INSERT INTO orders (Order_ID, Order_Number, Customer_ID) VALUES (11, 30, 1);
34 INSERT INTO orders (Order_ID, Order_Number, Customer_ID) VALUES (22, 31, 2);
35 INSERT INTO orders (Order_ID, Order_Number, Customer_ID) VALUES (33, 32, 3);
36 INSERT INTO orders (Order_ID, Order_Number, Customer_ID) VALUES (44, 33, 4);
37 INSERT INTO orders (Order_ID, Order_Number, Customer_ID) VALUES (55, 34, 5);
38 SELECT * FROM Orders;
39
40 -- Create multi-column Foreign key constraint on Customer_ID column when Orders table is already created
41 ALTER TABLE Orders
42 ADD CONSTRAINT FK_Customer_ID_Order
43 FOREIGN KEY (Customer_ID) REFERENCES Customer(Customer_ID);
44
45 -- Drop a Foreign Key constraint
46 ALTER TABLE Orders
47 DROP CONSTRAINT FK_Customer_ID_Order;
48

Messages
2:00:59 PM Started executing query at Line 46
Commands completed successfully.
Total execution time: 00:00:00.008
```

Changing A Schema: Add A New Column

- Change a table schema after it's already been defined and populated with data
- Using ALTER statement followed by ADD clause
- In the following example, we added d column
- Run the following command as shown in below figure →



The screenshot shows the SSMS interface with the following details:

- Servers:** localhost, master (sa)
- Databases:** System Databases, LA_Airline, Music, My_Database, SQL_Tutorial
- Tables:** dbo.Customer, dbo.item, dbo.sale, dbo.test_tb_1, dbo.test_tb_2, dbo.test_tb_3
- Scripting:** A script window titled "SQLQuery_2 - localhost.SQL_Tutorial (sa)" contains the following SQL code:

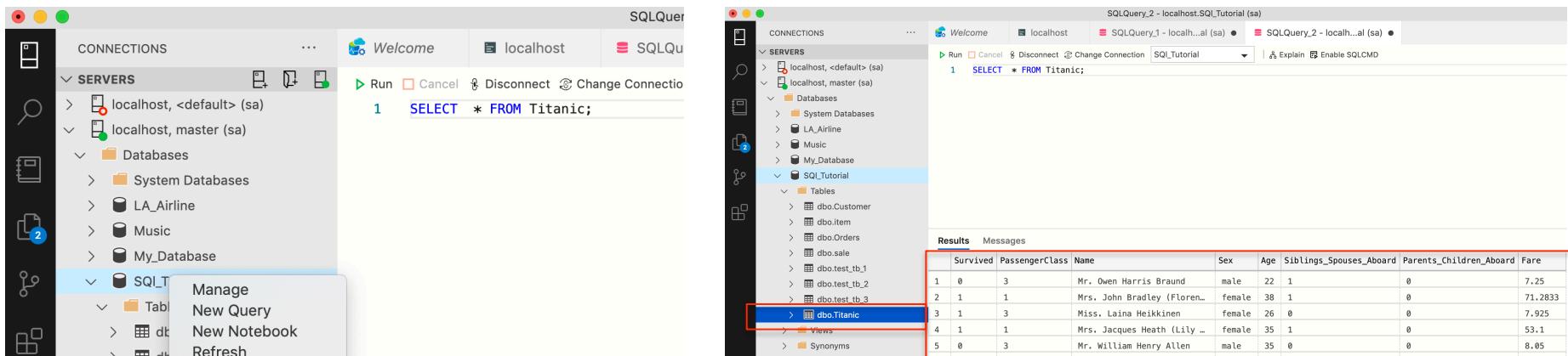
```
1  DROP TABLE IF EXISTS test;
2  CREATE TABLE test (a INT, b TEXT, c TEXT);
3  INSERT INTO test (a, b, c) VALUES (1, 'John', 'CA');
4  INSERT INTO test (a, b, c) VALUES (2, 'Shan', 'AZ');
5  SELECT * FROM test;
6
7  -- Change the table schema
8  -- Add a new column d with no value in it
9  ALTER TABLE test ADD d TEXT;
10 SELECT * FROM test;
11
12
```
- Results:** The results grid shows two rows of data from the "test" table:

	a	b	c
1	1	John	CA
2	2	Shan	AZ
- Messages:** The messages grid shows the output of the ALTER TABLE command:

	a	b	c	d
1	1	John	CA	NULL
2	2	Shan	AZ	NULL

Import Files Into SQL Server

- Install the **SQL Server Import Extension** (click [here](#) to see installation steps)
- The **SQL Server Import Extension** converts .txt and .csv files into a SQL table.
- Once installation is complete, start **Import Wizard** → Right-click on the **database** → click **Import Wizard**
- Importing a file → on **Import flat file wizard** → select a file by clicking **Browse**
- Keep clicking **Next** to proceed → *on modify columns columns can be changed*
- Click on **Import data** → click **Done** to finish the task
- Refresh target **Database** and run **SELECT** query on the table name to retrieve the data



Imported titanic.csv file using Import Wizard

	Survived	PassengerClass	Name
1	0	3	Mr. Owen Harris
2	1	1	Mrs. John Bradley (Floren...
3	1	3	Miss. Laina Heikkinen

Saving Query Results

- In Azure Data Studio, to save query results as CSV, Excel, and JSON, click on the file icons on the right side of the result window.

A screenshot of the Azure Data Studio interface. On the left is a tree view of database connections and objects. The main area shows the results of a SQL query:

```
3 SELECT Survived, PassengerClass, Name, Sex, Age FROM Titanic
4 WHERE Age > 50 AND Sex = 'Female' ORDER BY Age;
```

	Survived	PassengerClass	Name	Sex	Age
1	1	1	Mrs. John C (Anna Andrews... 2	Female	51
2	1	1	Mrs. Walter Bertram (M... 3	Female	52
3	1	1	Mrs. Charles Melville (C... 4	Female	52
4	1	1	Mrs. Edward Dale (Charlot... 5	Female	53
5	1	2	Mrs. Elizabeth (Eliza Nee... 6	Female	54
6	1	1	Miss. Elizabeth Mussey Eu... 7	Female	54
7	1	1	Mrs. Martin (Elizabeth Lu... 8	Female	55
8	1	2	Mrs. (Mary D Kingcome) He... 9	Female	56
9	1	1	Mrs. Thomas Jr (Lily Alex... -	Female	57
-	0	2	Mrs. (Mary) Mack -	Female	58
-	1	1	Miss. Elizabeth Bonnell -	Female	58
-	1	1	Miss. Elise Lurette -	Female	58
-	1	1	Mrs. William Thompson (Ed... -	Female	58
-	1	1	Mrs. Frank Manley (Anna S... -	Female	60
-	0	3	Miss. (Marion Ogden) Mean... -	Female	62
-	1	1	Mrs. George Nelson (Marth... -	Female	62
-	1	3	Mrs. (Hedwig) Turkula -	Female	63
-	1	1	Miss. Kornelia Theodosia ... -	Female	63

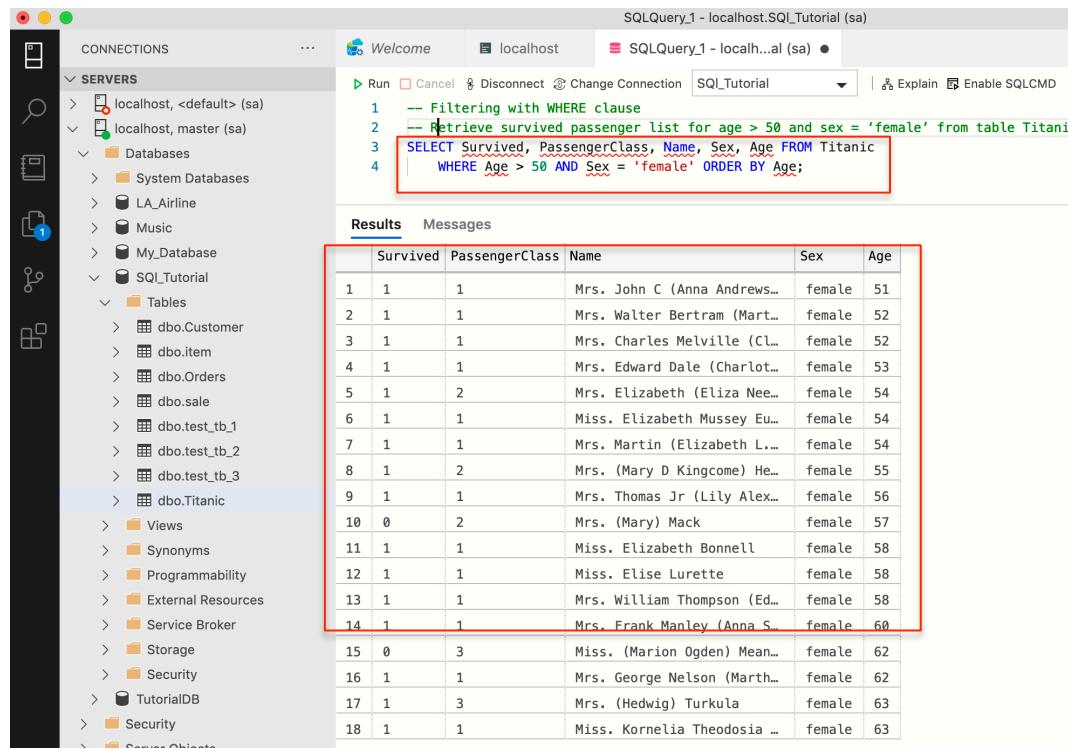
- To save a subset of the results directly → select the rows, columns, or block of interest (click and drag to select within the result) → then Right click → A dialog will appear where to save the selection

A screenshot of the Azure Data Studio interface, similar to the first one but with a context menu open over a selected subset of rows. The menu options include:

- Select All
- Save As CSV [40.8KB]
- Save As Excel [40.8KB]
- Save As JSON [4K]
- Save As XML [4K]
- Copy
- Copy With Headers

Filtering Data: WHERE Clause

- Filtering data using **WHERE** clause
- **WHERE** clause allows to retrieve specific rows instead of entire table
- For e.g., we want to retrieve survived passenger list for age > 50 and sex = 'female' from table Titanic
- Run the following command as shown in below figure →



The screenshot shows the SSMS interface with the following details:

- Left Sidebar (Object Explorer):** Shows the database structure under "SERVERS". The "Tables" node under "SQL_Tutorial" is expanded, showing "dbo.Customer", "dbo.item", "dbo.Orders", "dbo.sale", "dbo.test_tb_1", "dbo.test_tb_2", "dbo.test_tb_3", "dbo.Titanic", "Views", "Synonyms", "Programmability", "External Resources", "Service Broker", "Storage", "Security", "TutorialDB", and "Security" again.
- Top Bar:** Displays the connection information: "localhost, master (sa)" and the query window title "SQLQuery_1 - localhost.SQL_Tutorial (sa)".
- Query Window:** Contains the following T-SQL code:

```
1 -- Filtering with WHERE clause
2 -- Retrieve survived passenger list for age > 50 and sex = 'female' from table Titanic
3 SELECT Survived, PassengerClass, Name, Sex, Age FROM Titanic
4 WHERE Age > 50 AND Sex = 'female' ORDER BY Age;
```

The last two lines of the query are highlighted with a red box.
- Results Grid:** Shows the filtered data from the query. The columns are "Survived", "PassengerClass", "Name", "Sex", and "Age". The data includes 18 rows of passengers, all of whom are female and have an age greater than 50.

	Survived	PassengerClass	Name	Sex	Age
1	1	1	Mrs. John C (Anna Andrews...)	female	51
2	1	1	Mrs. Walter Bertram (Mart...	female	52
3	1	1	Mrs. Charles Melville (Cl...	female	52
4	1	1	Mrs. Edward Dale (Charlot...	female	53
5	1	2	Mrs. Elizabeth (Eliza Nee...	female	54
6	1	1	Miss. Elizabeth Mussey Eu...	female	54
7	1	1	Mrs. Martin (Elizabeth L...	female	54
8	1	2	Mrs. (Mary D Kingcome) He...	female	55
9	1	1	Mrs. Thomas Jr (Lily Alex...	female	56
10	0	2	Mrs. (Mary) Mack	female	57
11	1	1	Miss. Elizabeth Bonnell	female	58
12	1	1	Miss. Elise Lurette	female	58
13	1	1	Mrs. William Thompson (Ed...	female	58
14	1	1	Mrs. Frank Manley (Anna S...	female	60
15	0	3	Miss. (Marion Ogden) Mean...	female	62
16	1	1	Mrs. George Nelson (Marth...	female	62
17	1	3	Mrs. (Hedwig) Turkula	female	63
18	1	1	Miss. Kornelia Theodosia ...	female	63

Filtering Data: LIKE Operator

- Filtering data using **LIKE** operator by constructing a wildcard (% sign)
- For e.g., we want to retrieve Name that has the word ‘William’ in it from table Titanic
- Run the following command as shown in below figure ➔

The screenshot shows the SSMS interface with the following details:

- Left Sidebar:** Shows the "CONNECTIONS" section with "localhost, <default> (sa)" selected, and the "SERVERS" tree view which includes "localhost, master (sa)", "Databases", "System Databases", and "SQL_Tutorial" (selected), which contains "Tables", "Views", "Synonyms", "Programmability", and "External Resources".
- Top Bar:** Displays the connection information "localhost" and the database "SQLQuery_2 - localhost.SQL_Tutorial (sa)".
- Toolbar:** Includes "Run", "Cancel", "Disconnect", "Change Connection", "Explain", and other options.
- Query Editor:** Contains the following T-SQL code:

```
1 -- Filtering using LIKE operator
2 -- Retrieve Name that has the word 'William' in it from table Titanic
3 SELECT Survived, PassengerClass, Name, Sex, Age FROM Titanic
4 WHERE Name LIKE '%William%';
```
- Results Grid:** Shows the output of the query with columns: Survived, PassengerClass, Name, Sex, and Age. The "Name" column is highlighted with a red border, showing rows where the name contains "William".

	Survived	PassengerClass	Name	Sex	Age
1	0	3	Mr. William Henry Allen	male	35
2	0	3	Mr. William Henry Saund...	male	20
3	1	2	Mr. Charles Eugene Willia...	male	23
4	1	1	Mr. William Thompson Slop...	male	28
5	1	1	Mrs. William Augustus (Ma...	female	48
6	0	2	Mrs. William John Robert ...	female	27
7	0	3	Mr. William John Rogers	male	30
8	0	3	Master. William Frederick...	male	11
9	0	3	Mr. William Neal Ford	male	16
10	1	1	Mr. William Bertram Green...	male	23
11	0	2	Mr. William John Robert T...	male	29

Filtering Data: IN Operator

- IN operator is used to select results that match values in a list
- For e.g., we want to retrieve survived passenger list where **Sex** column match with **female** from table Titanic
- Run the following command as shown in below figure →

The screenshot shows the SSMS interface with the following details:

- Servers:** localhost, master (sa) is selected.
- Databases:** Databases, System Databases, LA_Airline, Music, My_Database, and SQL_Tutorial are listed.
- Tables:** dbo.Customer, dbo.item, dbo.Orders, dbo.sale, dbo.test_tb_1, dbo.test_tb_2, dbo.test_tb_3, and dbo.Titanic are listed under SQL_Tutorial.
- Query Window:** The query `SELECT Survived, PassengerClass, Name, Sex, Age FROM Titanic WHERE Sex IN ('female') ORDER BY Name;` is entered. The WHERE clause is highlighted with a red box.
- Results Grid:** The output shows 10 rows of data from the Titanic table, filtered by sex='female'. The columns are Survived, PassengerClass, Name, Sex, and Age. The Sex column for all rows is highlighted with a red box.

	Survived	PassengerClass	Name	Sex	Age
1	1	1	Dr. Alice (Farnham) Leader	female	49
2	1	1	Lady. (Lucille Christiana...)	female	48
3	0	3	Miss. (Marion Ogden) Mean...	female	62
4	1	3	Miss. Adele Kiamie Najib	female	15
5	0	3	Miss. Agda Thorilda Vikto...	female	25
6	1	3	Miss. Agnes McCoy	female	28
7	1	1	Miss. Albina Bazzani	female	32
8	1	1	Miss. Alice Cleaver	female	22
9	1	1	Miss. Alice Elizabeth For...	female	24
10	1	2	Miss. Alice Herman	female	24

Removing Duplicates

- Retrieve unique results from a column by removing duplicates using **SELECT DISTINCT** statement
- To retrieve unique disease from table **Disease** run the following command as shown in below figure →

The screenshot shows the SSMS interface with the following details:

- Servers:** localhost, master (sa)
- Databases:** System Databases, LA_Airline, Music, My_Database, SQL_Tutorial
- Tables:** dbo.Customer, dbo.Disease (highlighted with a red box), dbo.item, dbo.Orders, dbo.sale, dbo.test_tb_1, dbo.test_tb_2, dbo.test_tb_3, dbo.Titanic, Views, Synonyms, Programmability, External Resources, Service Broker, Storage, Security, TutorialDB, Security
- SQL Query:** `1 SELECT DISTINCT Disease FROM Disease;`
- Results:** A table showing unique diseases from the Disease table.

	Disease
1	Ehrlichiosis
2	Ciguatera Fish Poisoning
3	Leptospirosis
4	Coccidioidomycosis
5	Flavivirus Infection of U...
6	Shiga toxin-producing E. ...
7	Salmonellosis
8	Trichinosis
9	Paratyphoid Fever
10	Shiga Toxin Positive Fec...
11	Malaria
12	Streptococcal Infection (...
13	Botulism, Other

Ordering Results: ORDER BY Clause

- Sorting results (in ascending or descending order) on single or multiple columns using **ORDER BY** clause
- In the following example we want to sort **Disease** column in descending order,
- and within that sort **County** column in ascending order, and sort **Year** in descending order→

The screenshot shows the SQL Server Management Studio interface. On the left, the Object Explorer displays the database structure, including servers, databases, tables, and other objects. In the center, the SQL Query Editor window contains the following SQL code:

```
1 SELECT * FROM Disease ORDER BY Disease DESC, County ASC, Year DESC;
```

The results grid shows the output of the query. The data is ordered by Disease in descending order, then by County in ascending order, and finally by Year in descending order. The results include columns: Disease, County, Year, Sex, Cases, Population, Lower_95_CI, Upper_95_CI, and Rate. The first few rows of the result set are:

	Disease	County	Year	Sex	Cases	Population	Lower_95_CI	Upper_95_CI	Rate
1	Zika Virus Infection	ALAMEDA	2018	FEMALE	7	846926	0.332	1.703	0.827*
2	Zika Virus Infection	ALAMEDA	2018	MALE	0	815849	0	0.452	-
3	Zika Virus Infection	ALAMEDA	2018	TOTAL	7	1662775	0.169	0.867	0.421*
4	Zika Virus Infection	ALAMEDA	2017	FEMALE	8	840889	0.411	1.875	0.951*
5	Zika Virus Infection	ALAMEDA	2017	MALE	2	810430	0.03	0.891	0.247*
6	Zika Virus Infection	ALAMEDA	2017	TOTAL	10	1651319	0.29	1.114	0.606*
7	Zika Virus Infection	ALAMEDA	2016	FEMALE	21	833941	NULL	NULL	-
8	Zika Virus Infection	ALAMEDA	2016	MALE	9	803825	NULL	NULL	-
9	Zika Virus Infection	ALAMEDA	2016	TOTAL	30	1637766	NULL	NULL	-
10	Zika Virus Infection	ALPINE	2018	FEMALE	0	564	0	651.922	-
11	Zika Virus Infection	ALPINE	2018	MALE	0	577	0	637.281	-
12	Zika Virus Infection	ALPINE	2018	TOTAL	0	1141	0	322.78	-
13	Zika Virus Infection	ALPINE	2017	FEMALE	0	565	0	650.772	-
14	Zika Virus Infection	ALPINE	2017	MALE	0	581	0	632.908	-
15	Zika Virus Infection	ALPINE	2017	TOTAL	0	1146	0	321.374	-
16	Zika Virus Infection	ALPINE	2016	FEMALE	0	566	NULL	NULL	-
17	Zika Virus Infection	ALPINE	2016	MALE	0	585	NULL	NULL	-

Grouping Data: GROUP BY Statement

- The **GROUP BY** statement groups rows that have the same values into summary rows, like “find the number of cases for each disease for the year 2018”
- The **GROUP BY** statement is often used with aggregate functions (COUNT, MAX, MIN, SUM, AVG)
- For example, we want to find the number of cases for each disease for the year 2018 from table **Disease** →

The screenshot shows the SQL Server Management Studio (SSMS) interface. On the left is the Object Explorer tree, which is collapsed. The main area contains a query window with the following SQL code:

```
-- Find the number of cases for each disease for the year 2018
SELECT COUNT(Cases) AS 'Disease_count', Disease
FROM Disease
WHERE Year = 2018
GROUP BY Disease
ORDER BY COUNT(cases) ASC;
```

The results pane below displays the output of the query, showing 15 rows of data. The first 10 rows are highlighted with a red border. The columns are labeled "Disease_count" and "Disease".

	Disease_count	Disease
1	141	Salmonellosis
2	143	Giardiasis
3	145	Campylobacteriosis
4	147	Coccidioidomycosis
5	149	Shiga toxin-producing E. ...
6	163	Cryptosporidiosis
7	163	Shigellosis
8	163	Legionellosis
9	167	Vibrio Infection (non-Cholera)
10	169	Shiga toxin-producing E. coli
11	169	Streptococcal Infection (non-Group A streptococcus)
12	169	Amebiasis
13	169	Yersiniosis
14	169	Lyme Disease
15	171	Dengue Virus Infection

Conditional Expressions

- The conditional expression uses **CASE** keyword followed by **WHEN condition THEN result statement**
- The concept is illustrated in the following example using a **test** table.

The screenshot shows the SSMS interface with the following details:

- Servers:** localhost, <default> (sa), localhost, master (sa), Databases, System Databases, LA_Airline, Music, My_Database, SQL_Tutorial.
- Tables under SQL_Tutorial:** Customer, Disease, item, Orders, sale, test_tb_1, test_tb_2, test_tb_3, Titanic.
- Query Window:** SQLQuery_2 - localhost.SQL_Tutorial (sa)
SQL code:

```
1 DROP TABLE IF EXISTS test;
2 CREATE TABLE test (a INT);
3 INSERT INTO test (a) VALUES (10);
4 INSERT INTO test (a) VALUES (20);
5 INSERT INTO test (a) VALUES (30);
6 SELECT * FROM test;
7
8 -- Conditional expression
9 SELECT
10 CASE
11     WHEN a > 20 THEN 'a is greater than 20'
12     WHEN a < 20 THEN 'a is less than 20'
13     ELSE 'a is equal to 20'
14 END AS a_value
15 FROM test;
```
- Results Pane:** Shows the output of the SELECT statement from the test table.

a
10
20
30
- Messages Pane:** Shows the output of the conditional expression.

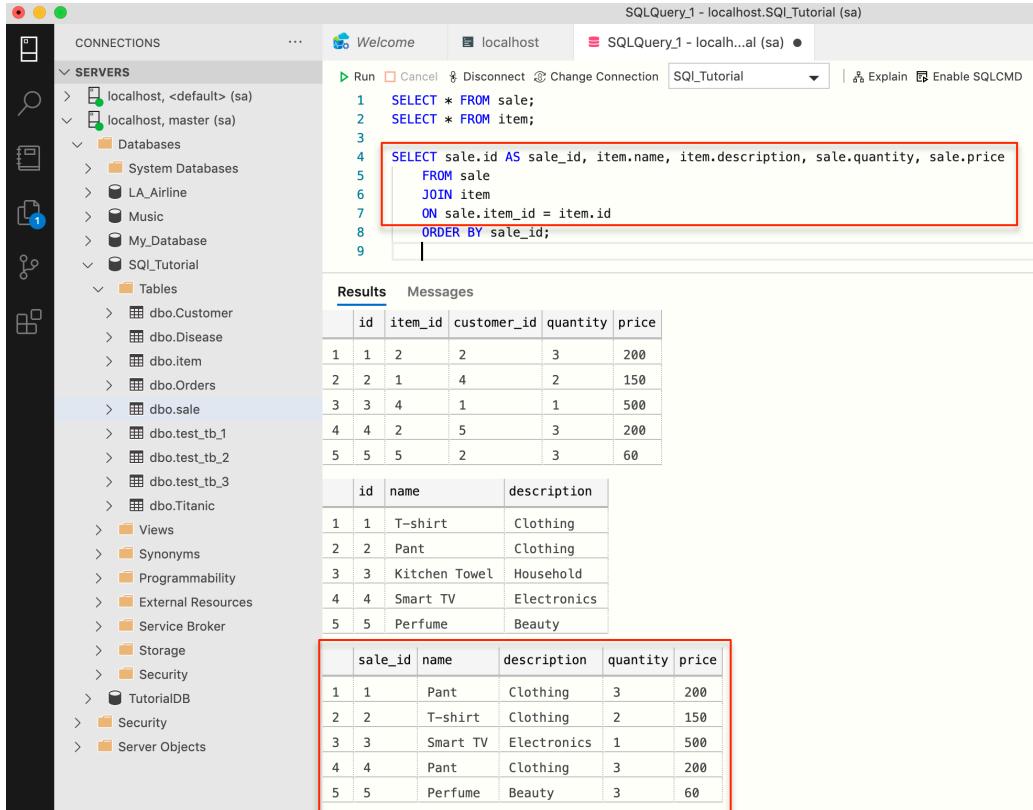
a_value
a is less than 20
a is equal to 20
a is greater than 20

Accessing Related Tables: JOIN Clause

- SQL performs query on related data from multiple tables using **JOIN** clause
- Unique ID columns, automatically generated by Database system, is used to create relationships
- The most common form of join is **inner join** (the default join)
- **Inner join** includes rows from both tables where the join condition is met
- The less common form of join is **outer join: left outer join, right outer join, and full outer join**
- A **left outer join** includes the rows where the condition is met and also includes all the rows from the table on the left, where the condition is not met
- A **right outer join** includes the rows where the condition is met and also includes all the rows from the table on the right, where the condition is not met
- A **full outer join** combines the effects of the left and right joins
- Many database systems do not support right and full joins

JOIN Clause Example

- The concept of **inner join** is illustrated in the following example using **item** and **sale** tables
- In the given example, **JOIN** clause gives the *id*, the quantity, and the *price* from **sale** table; and the *name* and the *description* from the **item** table



The screenshot shows the SSMS interface with a query window and a results grid.

Query Window:

```
SQLQuery_1 - localhost.SQL_Tutorial (sa)
Welcome    localhost    SQLQuery_1 - localhost (sa)
Run Cancel Disconnect Change Connection SQL_Tutorial Explain Enable SQLCMD
1 SELECT * FROM sale;
2 SELECT * FROM item;
3
4 SELECT sale.id AS sale_id, item.name, item.description, sale.quantity, sale.price
5   FROM sale
6     JOIN item
7       ON sale.item_id = item.id
8 ORDER BY sale_id;
9
```

Results Grid:

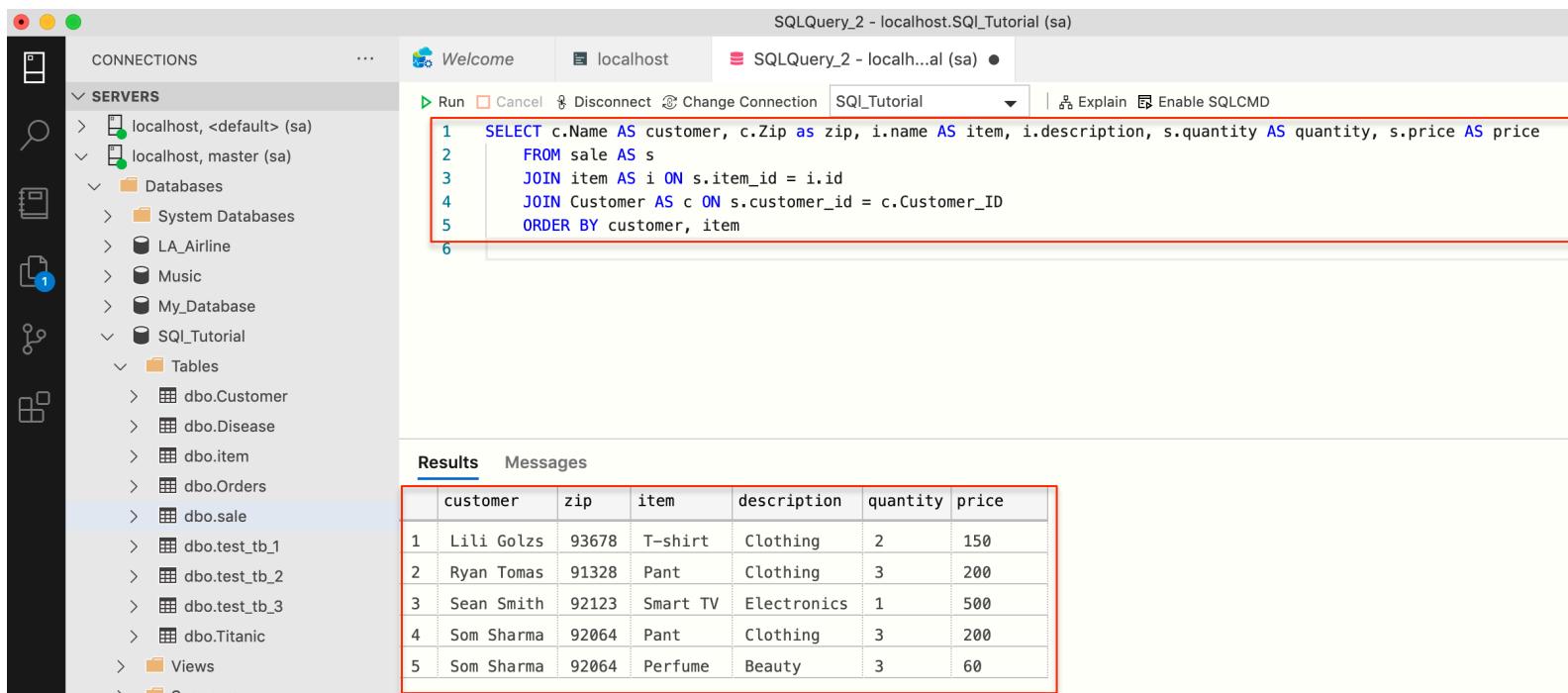
	id	item_id	customer_id	quantity	price
1	1	2	2	3	200
2	2	1	4	2	150
3	3	4	1	1	500
4	4	2	5	3	200
5	5	5	2	3	60

	id	name	description
1	1	T-shirt	Clothing
2	2	Pant	Clothing
3	3	Kitchen Towel	Household
4	4	Smart TV	Electronics
5	5	Perfume	Beauty

	sale_id	name	description	quantity	price
1	1	Pant	Clothing	3	200
2	2	T-shirt	Clothing	2	150
3	3	Smart TV	Electronics	1	500
4	4	Pant	Clothing	3	200
5	5	Perfume	Beauty	3	60

Join Multiple Tables

- In practice, it's very common to have tables with many-to-many relationships
- The many-to-many relationships are implemented using a junction table
- The concept is illustrated in the following example using **customer**, **item** and **sale** tables
- In the given example, **JOIN** clause gives the *customer name* and *zip* from **customer** table; the *name* and the *description* from the **item** table; and the *quantity* and the *price* from **sale** table



The screenshot shows the SSMS interface with the following details:

- Left Sidebar:** Shows the "CONNECTIONS" section with "localhost, <default> (sa)" selected. Below it, the "SERVERS" tree includes "localhost, master (sa)", "Databases", "System Databases", "LA_Airline", "Music", "My_Database", and "SQL_Tutorial" database, which contains "Tables" like "dbo.Customer", "dbo.Disease", "dbo.item", "dbo.Orders", "dbo.sale", "dbo.test_tb_1", "dbo.test_tb_2", "dbo.test_tb_3", and "dbo.Titanic".
- Top Bar:** Displays "SQLQuery_2 - localhost.SQL_Tutorial (sa)" and various tabs like "Run", "Cancel", "Disconnect", "Change Connection", "SQL Tutorial", "Explain", and "Enable SQLCMD".
- Query Window:** Contains the following T-SQL code:

```
1  SELECT c.Name AS customer, c.Zip as zip, i.name AS item, i.description, s.quantity AS quantity, s.price AS price
2      FROM sale AS s
3      JOIN item AS i ON s.item_id = i.id
4      JOIN Customer AS c ON s.customer_id = c.Customer_ID
5      ORDER BY customer, item
6
```
- Results Grid:** Shows the output of the query with columns: customer, zip, item, description, quantity, and price. The data is as follows:

	customer	zip	item	description	quantity	price
1	Lili Golzs	93678	T-shirt	Clothing	2	150
2	Ryan Tomas	91328	Pant	Clothing	3	200
3	Sean Smith	92123	Smart TV	Electronics	1	500
4	Som Sharma	92064	Pant	Clothing	3	200
5	Som Sharma	92064	Perfume	Beauty	3	60

LEFT JOIN Example

The screenshot shows the SSMS interface with the following details:

- Left Panel (Object Explorer):** Shows the database structure. Under "SERVERS", "localhost, master (sa)" is expanded, showing "Databases" (System Databases, LA_Airline, Music, My_Database), "Tables" (dbo.Customer), and "Columns". Under "Tables", "dbo.Customer" is expanded, showing columns: Customer_ID (PK, int, ...), Name (varchar(20), null), Address (text, null), City (varchar(20), null), State (varchar(20), null), Zip (int, null), Email (text, null). Other nodes like "Keys", "Constraints", and "Triggers" are also listed.
- Top Bar:** Displays the connection information: "Welcome" (localhost), "SQLQuery_1 - localhost.SQL_Tutorial (sa)".
- Toolbar:** Includes buttons for Run, Cancel, Disconnect, Change Connection (set to "SQL Tutorial"), Explain, and Enable SQLCMD.
- Query Editor:** Contains the following T-SQL code:

```
1  INSERT INTO Customer (Customer_ID, Name, City) VALUES (6, 'Jake', 'Irvine');
2  SELECT * FROM Customer;
3
4  SELECT c.Name AS customer, c.Zip as zip, i.name AS item, i.description, s.quantity AS quantity, s.price AS price
5    FROM Customer AS c
6    LEFT JOIN sale AS s ON s.customer_id = c.Customer_ID
7    LEFT JOIN item AS i ON s.item_id = i.id
8  ORDER BY customer, item
9
```

The code is highlighted with a red box.
- Results Grid:** Shows the output of the query. The first two rows (Jag Gos and Jake) have their entire row highlighted with a red box. The rest of the data is visible below.

	customer	zip	item	description	quantity	price
1	Jag Gos	92112	NULL	NULL	NULL	NULL
2	Jake	NULL	NULL	NULL	NULL	NULL
3	Lili Golzs	93678	T-shirt	Clothing	2	150
4	Ryan Tomas	91328	Pant	Clothing	3	200
5	Sean Smith	92123	Smart TV	Electronics	1	500
6	Som Sharma	92064	Pant	Clothing	3	200
7	Som Sharma	92064	Perfume	Beauty	3	60

String Functions

LEN(string):	Returns the length of the given string
DATALENGTH(string):	Returns string length including trailing spaces in SQL server
SUBSTRING(string, start, length) :	Returns some characters from a given string
TRIM(string):	Removes leading and trailing spaces from a string.
ASCII(string)	Returns the ASCII value of the first character of the string
CHAR(code)	Returns the character based on the ASCII code
CHARINDEX(substring, string, start)	Searches for a substring in a string and returns the position
CONCAT(string1, string2, string3,...)	Add two or more strings together
CONCAT_WS(separator, string1, string2,...)	Add two or more strings together with separator
SOUNDEX(expression)	Returns a four-character code to evaluate the similarity of two expressions
DIFFERENCE(expression1, expression2)	Compares two SOUNDEX values and return an integer
REVERSE(string)	Reverses a string

Getting Length Of Strings

➤ Length functions:

- **LEN(string)** for SQL Server Or **LENGTH(string)** for other database system; returns the length of the given string
- **DATALENGTH(string)** returns string length including trailing spaces in SQL server
- **COL_LENGTH(table_name, col_name)** returns the maximum size in bytes in SQL server

The screenshot shows the SSMS interface with a query window titled "SQLQuery_1 - localhost.SQL_Tutorial (sa)". The left pane displays the database structure under "SERVERS". The query window contains the following code:

```
1 -- Trailing spaces are excluded (returns 10)
2 SELECT LEN('SQL Server ');
3
4 -- Trailing spaces are included (returns 11)
5 SELECT DATALENGTH('SQL Server ');
6
7 -- Trying DATELENGTH with a Unicode string (returns 22)
8 SELECT DATALENGTH(N'SQL Server ');
9
10 -- Using DATALENGTH to get the length in characters of a Unicode string (returns 11)
11 SELECT DATALENGTH(N'SQL Server ')/2;
12
13 -- Get the maximum column size in bytes (returns 20)
14 SELECT COL_LENGTH('Customer', 'Name') FROM Customer;
```

The results pane shows the output of the first query, which is highlighted with a red box:

(No column name)
1 10

Getting Part Of Strings

- The function **SUBSTRING(string, start, length)** for SQL Server and **SUBSTR(string, start, length)** for other database system; returns some characters from a given string

The screenshot shows the SSMS interface with the following details:

- Left Sidebar:** Shows the Object Explorer with the tree structure of servers, databases, tables, views, synonyms, etc. The database "Music" is currently selected.
- Top Bar:** Displays the connection information: "localhost, master (sa)" and the query window title "SQLQuery_2 - localhost.SQL_Tutorial (sa)".
- Toolbar:** Includes buttons for Run, Cancel, Disconnect, Change Connection, and a dropdown for the current database.
- Query Window:** Contains the SQL command: `SELECT SUBSTRING('SQL Server', 1, 3) AS substring;`. The entire command is highlighted with a red box.
- Results Tab:** Shows the output of the query in a table:

	substring
1	SQL

The entire table is also highlighted with a red box.
- Messages Tab:** Currently empty.

Removing Spaces

- The function **TRIM(string)** removes leading and trailing spaces from a string
- **LTRIM(string)** removes leading spaces from a string
- **RTRIM(string)** removes trailing spaces from a string

The screenshot shows the SSMS interface with the following details:

- Left pane (Object Explorer):** Shows the database structure under the "SERVERS" node, including databases like "localhost, <default> (sa)", "localhost, master (sa)", "Databases", "System Databases", "LA_Airline", "Music", and "SQL_Tutorial".
- Top bar:** Displays the connection information: "SQLQuery_1 - localhost.SQL_Tutorial (sa)".
- Toolbar:** Includes icons for Run, Cancel, Disconnect, Change Connection, Explain, and Enable SQLCMD.
- Query window:** Contains the following T-SQL code:

```
1 SELECT TRIM(' SQL Tutorial ') AS trim;
2 SELECT LTRIM(' SQL Tutorial ') AS L_trim;
3 SELECT RTRIM('SQL Tutorial ') AS R_trim;
```
- Results pane:** Displays three result sets:

trim
SQL Tutorial

L_trim
SQL Tutorial

R_trim
SQL Tutorial

Numeric Functions

ABS(<i>number</i>):	Returns the absolute value of a number
AVG (<i>expression</i>):	Returns the average value of an expression
COUNT(<i>expression</i>) :	Returns the number of records returned by a select query
FLOOR(<i>number</i>):	Returns the largest integer value that is smaller than or equal to a number
MAX(<i>expression</i>)	Returns the maximum value in a set of values
POWER(<i>a, b</i>)	Returns the value of a number raised to the power of another; <i>a</i> -base, <i>b</i> -exponent
RAND(<i>seed</i>)	Returns a random number ≥ 0 and < 1
ROUND(<i>number, decimals, operation</i>)	Rounds a number to a specified number of decimal places, <i>number</i> —to be rounded, <i>decimals</i> —the number of decimal places to round number to, <i>operation</i> —(opt, default 0), if 0 it rounds the result to the number of decimal, if another value than 0, it truncates the result to the number of decimals.
SQRT(<i>number</i>)	Returns the square root of a number
SQUARE(<i>number</i>)	Returns the square of a number
SUM(<i>expression</i>)	Calculates the sum of a set of values

Integer Division

The screenshot shows the SSMS interface with the following details:

- Servers:** localhost, master (sa)
- Databases:** System Databases, LA_Airline, Music, My_Database, SQL_Tutorial
- Tables:** CarSale, Customer, Disease, item, Orders, sale, test, test_tb_1, test_tb_2, test_tb_3, Titanic, Views, Synonyms, Programmability, External Resources, Service Broker.

SQL Query Window: SQLQuery_2 - localhost.SQL_Tutorial (sa)

```
1 -- When divide an integer number by an integer number, result is an integer
2 -- (Divide 1 / 2 gives 0)
3 SELECT 1 / 2 AS result_integer;
4
5 -- When divide a real number by an integer number, result is a real number
6 -- (Divide 1.0 / 2 gives 0.5)
7 SELECT 1.0 / 2 AS result_real;
8
9 -- Using CAST function by casting one of the operand as real gives the real number
10 -- (Divide CAST(1 AS REAL) / 2 gives 0.5)
11 SELECT CAST(1 AS REAL) / 2 AS result_real;
12
13 -- (Divide 13 / 2 gives integer value 6)
14 SELECT 13 / 2 AS result_integer;
15
16 -- The modulus operator returns the remainder
17 -- In the following example, 13 / 2 gives integer value 6 and 13 % 2 gives the remainder
18 SELECT 13 / 2 AS result_integer, 13 % 2 AS remainder;
```

Results Tab:

result_real
1 0.5

result_integer
1 6

result_integer	remainder
1 6	1

Rounding Numbers

The screenshot shows the SSMS interface with the following details:

- Object Explorer (Left):** Shows the database structure under "localhost, master (sa)" and "SQL_Tutorial".
- SQL Query Window (Center):** Displays the following T-SQL code:

```
1 -- (Result 5.5678)
2 SELECT 5.5678;
3
4 -- Rounds a number to a specified number of decimal places, number-to be rounded,
5 -- decimals-the number of decimal places to round number to, operation- (opt, default 0),
6 -- if 0 it rounds the result to the number of decimal, if another value than 0, it truncates
7 -- the result to the number of decimals.
8
9 -- (ROUND(5.5678) returns 5.5600)
10 SELECT ROUND(5.5678, 2, 2);
11
```
- Results Window (Bottom):** Shows the output of the query. The first row is "5.5678" and the second row is "5.5600". Both rows are highlighted with a red border.

Date And Time Functions

CURRENT_TIMESTAMP:	Returns the current date and time
YEAR(date):	Returns the year part for a specified date
MONTH(date):	Returns the month part for a specified date
DAY(date):	Returns the day part for a specified date
GETDATE():	Returns the current database system date and time (in 'YYYY-MM-DD hh:mm:ss:mmm' format)
ISDATE(expression):	Checks an expression and returns 1 if it is a valid date, otherwise returns 0
SYSDATETIME():	Returns the date and time of the computer where the SQL server is running
DATEADD(interval, number, date):	Adds a time/date interval to a date and then returns the date
DATEDIFF(interval, date1, date2)	Returns differences between two dates
DATEFROMPARTS(year, month, day)	Returns a date from the specified parts (year, month, and day)
DATEPART(interval, date)	Returns a specified part of a date
DATENAME(interval, date)	Returns a specified part of a date

Date Time Function Examples

The screenshot shows the SSMS interface with a query window displaying several examples of date and time functions.

Query Window Content:

```
1 -- Returns the current date and time
2 SELECT CURRENT_TIMESTAMP AS Date_Time;
3
4 -- Returns the part for a specified date
5 SELECT YEAR('2018/05/31') AS Date, MONTH('2018/05/31') AS Month, DAY('2018/05/31') AS day;
6
7 -- Returns the part for a specified date
8 SELECT DATENAME(yy, '2018/05/31') AS Date,
9      DATENAME(mm, '2018/05/31') AS Month,
10     DATENAME(day, '2018/05/31') AS day;
11
12 -- Adds a time/date interval to a date and then returns the date
13 SELECT DATEADD(month, 3, '2018/05/31') AS Date_Add;
```

Results Grids:

- Date_Time**:

1	2020-03-06 22:12:26.463
---	-------------------------
- Date**:

	Date	Month	day
1	2018	5	31
- Month**:

	Date	Month	day
1	2018	May	31
- Date_Add**:

1	2018-08-31 00:00:00.000
---	-------------------------

Aggregating Data

- Aggregating data is the information derived from more than one row at a time
- In the table **Disease**, we use **COUNT()** aggregate function to know how many cases are present in each disease
- **COUNT()** returns a single value from a query that spans many rows of data
- The **GROUP BY** clause groups the results before calling the aggregation function **COUNT()**
- The **HAVING** clause retrieves results for a specified condition, in this case, retrieves results for **COUNT(*) > 2000**
- And finally, execute **ORDER BY Count DESC, Disease**, to retrieve disease count in descending order, and **Disease** in ascending order within **Count**.

The screenshot shows the SSMS interface with the following details:

- Left pane (Object Explorer):** Shows the database structure. Under "Databases", "SQL_Tutorial" is expanded, showing "Tables" (with "dbo.CarSale", "dbo.Customer", "dbo.Disease", "dbo.item", "dbo.Orders", "dbo.sale", "dbo.test"), and "dbo.test_tb_1" (with "Columns", "Keys", "Constraints", "Triggers", "Indexes").
- Top bar:** "Welcome" tab is selected. Other tabs include "localhost" and "SQLQuery_1 - localhost.SQL_Tutorial (sa)".
- Toolbar:** Includes "Run", "Cancel", "Disconnect", "Change Connection", "SQL_Tutorial", "Explain", and "Enable SQLCMD".
- Text Editor (Query Window):** Contains the following T-SQL code:

```
1 -- In the below example, we use COUNT() aggregate function to know how many cases are present in each disease
2 -- The GROUP BY clause groups the results before calling the aggregation function COUNT()
3 -- The HAVING clause retrieves results for a specified condition, in this case, retrieves results for COUNT(*) > 2000
4 -- And finally, execute ORDER BY Count DESC, Disease, to retrieve disease count in descending order,
5 -- and Disease in ascending order within Count.
6
7 SELECT Disease, COUNT(*) AS Count
8   FROM Disease
9   GROUP BY Disease
10  HAVING COUNT(*) >= 2000
11
12  ORDER BY Count DESC, Disease;
```
- Results Window:** Displays the query results in a table:

Disease	Count
1 Amebiasis	3186
2 Anthrax	3186
3 Babesiosis	3186
4 Botulism, Foodborne	3186
5 Botulism, Other	3186
6 Botulism, Wound	3186
7 Brucellosis	3186
8 Campylobacteriosis	3186

Concatenating Column Data

- The following statement uses the **CONCAT()** function to concatenate values in the **First_Name** and **Last_Name** columns of the table **Student**

The screenshot shows the SSMS interface with the following details:

- Left Sidebar:** Shows the "CONNECTIONS" section with "localhost, <default> (sa)" selected. Below it, the "SERVERS" tree shows "localhost, master (sa)", "Databases" (with "System Databases" expanded), and "SQL_Tutorial" (with "Tables" expanded, showing "dbo.CarSale", "dbo.Customer", etc.).
- Top Bar:** Displays "Welcome", "localhost", "SQLQuery_1 - localhost.SQL_Tutorial (sa)", and "SQL_Tutorial".
- Query Window:** Contains the following SQL code:

```
1 /*The CONCAT() function to concatenate values
2 in the First_Name and Last_Name columns of the table Student*/
3 SELECT ID, First_Name, Last_Name, Score,
4       CONCAT(First_Name, ' ', Last_Name) Full_Name
5 FROM Student
6 ORDER BY Full_Name;
```

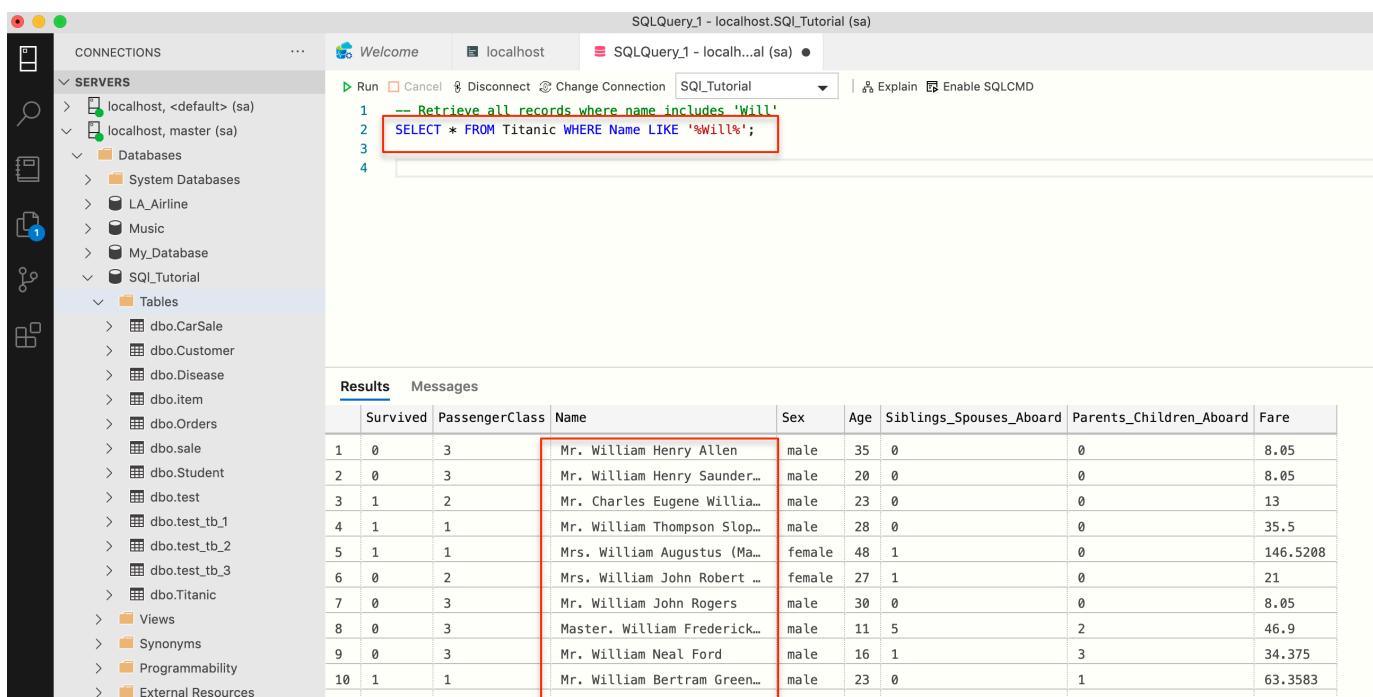
The code is highlighted with a red box.
- Results Grid:** Shows the output of the query:

ID	First_Name	Last_Name	Score	Full_Name
1	4	Jags	Smith	Jags Smith
2	1	Jake	Smith	Jake Smith
3	3	James	Smith	James Smith
4	2	John	Smith	John Smith

The "Full_Name" column is also highlighted with a red box.

Searching For Patterns: Wildcard

- SQL query uses wildcard characters to search a pattern
- The **LIKE** and **NOT** operators are used with **WHERE** clause to search a pattern
- For example, using the wildcard “A%” we can search any string starting with a capital A
- Using the wildcard “%A%” we can search all results that contain a capital A
- Using the _ wildcard we can search any string at specific location; for e.g., ‘_A%’ gives the results only when C is in the second position



The screenshot shows the SSMS interface with the following details:

- Connections:** Shows connections to localhost, including default (sa) and master (sa). Under localhost, there are databases: System Databases, LA_Airline, Music, My_Database, and SQL_Tutorial. SQL_Tutorial contains Tables: dbo.CarSale, dbo.Customer, dbo.Disease, dbo.item, dbo.Orders, dbo.sale, dbo.Student, dbo.test, dbo.test_tb_1, dbo.test_tb_2, dbo.test_tb_3, and dbo.Titanic.
- SQL Query Window:** The title bar says "SQLQuery_1 - localhost.SQL_Tutorial (sa)". The query pane contains:

```
-- Retrieve all records where name includes 'Will'
SELECT * FROM Titanic WHERE Name LIKE '%Will%';
```
- Results Grid:** The results pane displays the output of the query. The columns are Survived, PassengerClass, Name, Sex, Age, Siblings_Spouses_Aboard, Parents_Children_Aboard, and Fare. The "Name" column is highlighted with a red border, and several rows containing "Will" in the name are also highlighted with a red border.

Survived	PassengerClass	Name	Sex	Age	Siblings_Spouses_Aboard	Parents_Children_Aboard	Fare
1	0	Mr. William Henry Allen	male	35	0	0	8.05
2	0	Mr. William Henry Saund...	male	20	0	0	8.05
3	1	Mr. Charles Eugene Willia...	male	23	0	0	13
4	1	Mr. William Thompson Slop...	male	28	0	0	35.5
5	1	Mrs. William Augustus (Ma...	female	48	1	0	146.5208
6	0	Mrs. William John Robert ...	female	27	1	0	21
7	0	Mr. William John Rogers	male	30	0	0	8.05
8	0	Master William Frederick...	male	11	5	2	46.9
9	0	Mr. William Neal Ford	male	16	1	3	34.375
10	1	Mr. William Bertram Green...	male	23	0	1	63.3583

UNION Operator

- The UNION operator combines the results of two or more SELECT statements
- Each SELECT statement within UNION must have the same number of columns
- The columns must also have similar data types
- The columns in each SELECT statement must also be in the same order
- UNION operator selects only distinct values by default. To allow duplicate values, use UNION ALL
- UNION Syntax : **SELECT column(s) FROM table1 UNION column(s) FROM table2**

UNION Operator Example

The screenshot shows the SSMS interface with the following details:

- Connections:** Shows a connection to "localhost, master (sa)".
- Servers:** Shows the "localhost, master (sa)" server node expanded, displaying databases like "LA_Airline", "Music", "My_Database", and "SQL_Tutorial".
- Query Editor:** The query window contains the following T-SQL code:

```
7  INSERT INTO Suppliers (Supplier_id, Supplier_Name, City) VALUES (104, 'DD', 'Los Angeles');
8  SELECT * FROM Suppliers;
9
10 /* The below statement returns the cities (only distinct values)
11 from both the tables Customer and Suppliers*/
12 SELECT City FROM Customer
13 UNION
14 SELECT City FROM Suppliers
15 ORDER BY City;
```
- Results:** The results tab displays two tables. The first table, "Customer", has columns: Customer_ID, Name, Address, City, State, Zip, Email. The second table, "Suppliers", has columns: Supplier_id, Supplier_name, City.
- Messages:** The messages tab shows the output of the UNION query, listing cities: Irvine, Los Angeles, Orange County, Poway, San Diego, and San Francisco.

	Customer_ID	Name	Address	City	State	Zip	Email
1	1	Sean Smith	110 Convoy St	San Diego	CA	92123	seansmith@work.com
2	2	Som Sharma	121 Pepper Tree Ln	Poway	CA	92064	somsharma@work.com
3	3	Jag Gos	235 Costa Verde	Irvine	CA	92112	jagos@work.com
4	4	Lili Golzs	211 Dorian St	San Francisco	CA	93678	liligolzs@work.com
5	5	Ryan Tomas	123 Indian St	Los Angeles	CA	91328	ryantomas@work.com
6	6	Jake	NULL	Irvine	NULL	NULL	NULL

	Supplier_id	Supplier_name	City
1	101	AA	Orange County
2	102	BB	Irvine
3	103	CC	San Diego
4	104	DD	Los Angeles

	City
1	Irvine
2	Los Angeles
3	Orange County
4	Poway
5	San Diego
6	San Francisco

Transactions

- A transaction is a group of operations that are handled as one unit of work If one operation fails, the entire group of operations fail. Transaction Commands:
 - **BEGIN TRANSACTION** – to begin a transaction
 - **COMMIT** - to end and save a transaction
 - **ROLLBACK** – to rollback the changes

The screenshot shows the SSMS interface with a query window titled "SQLQuery_1 - localhost.SQL_Tutorial (sa)". The left pane displays the database structure under "SERVERS". The right pane shows a script window with numbered SQL statements. A red box highlights the transaction logic from statement 26 to 30. Below the script is a "Results" tab showing the state of two tables after the transaction.

```
23  
24 -- Begin transaction with BEGIN TRANSACTION statement  
25 -- End transaction with COMMIT statement  
26 BEGIN TRANSACTION;  
27 INSERT INTO widgetSales (id, inv_id, quantity, price) VALUES (1, 1, 5, 200);  
28 UPDATE widgetInventory SET onhand = (onhand - 5) WHERE id = 1;  
29 COMMIT;  
30  
31 -- Rollback a transaction with ROLLBACK statement  
32 BEGIN TRANSACTION;  
33 INSERT INTO widgetInventory (id, description, onhand) VALUES (4, 'pencil', 20);  
34 ROLLBACK;  
35 SELECT * FROM widgetInventory;  
36 |
```

Results tab content:

	id	description	onhand
1	1	rock	5
2	2	paper	20
3	3	scissors	20

Results tab content (widgetSales table):

	id	inv_id	quantity	price
1	1	1	5	200

Summary

- Click on [Github](#) for example codes discussed in this presentation.
- **References:**
 1. LinkedIn video by Bill Weinman
 2. SQL Cookbook by Anthony Molinaro
 3. SQL Notes for Professionals
 4. [W3schools.com](#)