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

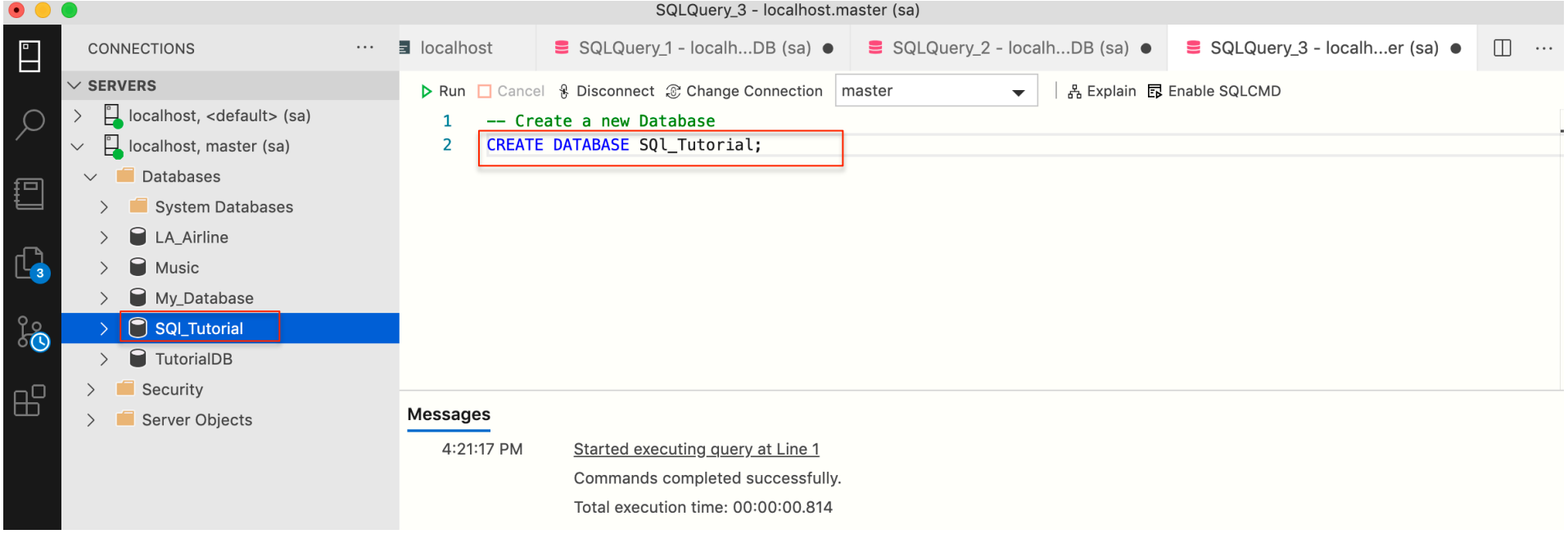
- **Installation of Microsoft SQL Server 2017 on macOS Mojave version 10.14.4**

1. Download and install **Docke**r → 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

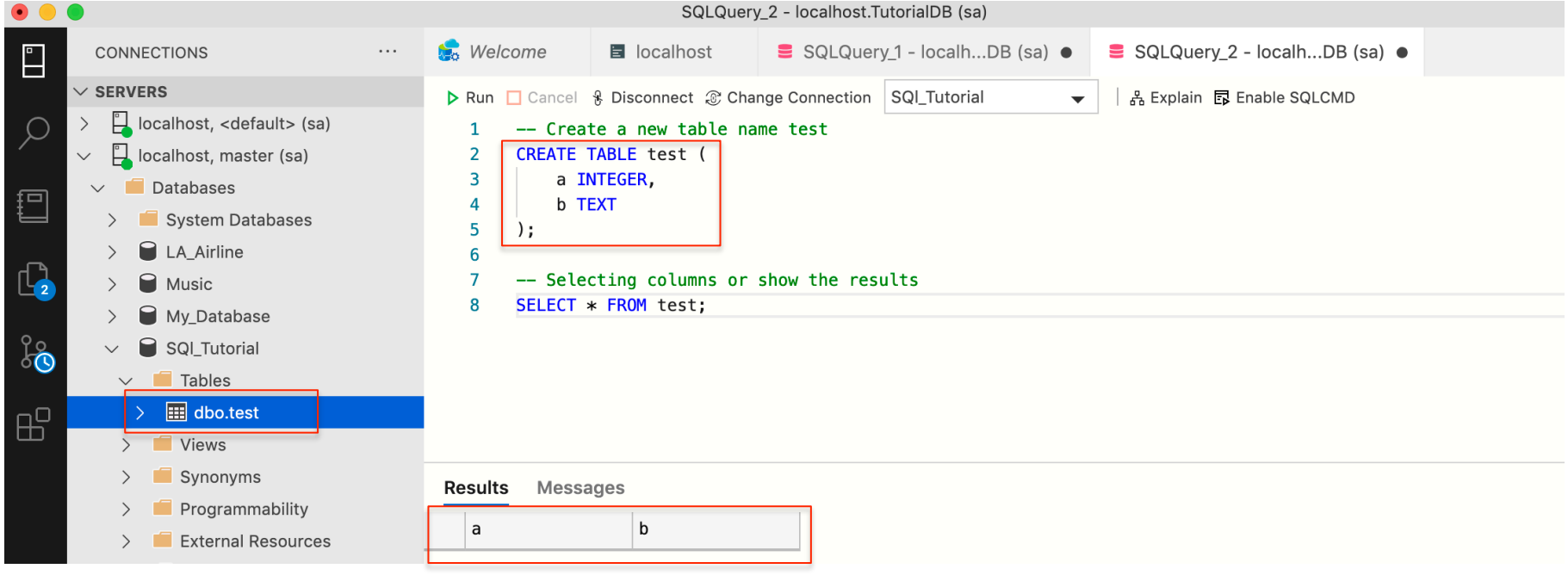
- Start **Azure Data Studio** (see instructions on installation and setup slide)
- Create a new database
  - Right-click on server, **localhost** → select **New Query -->** run following SQL command

    CREATE DATABASE *database_name*;



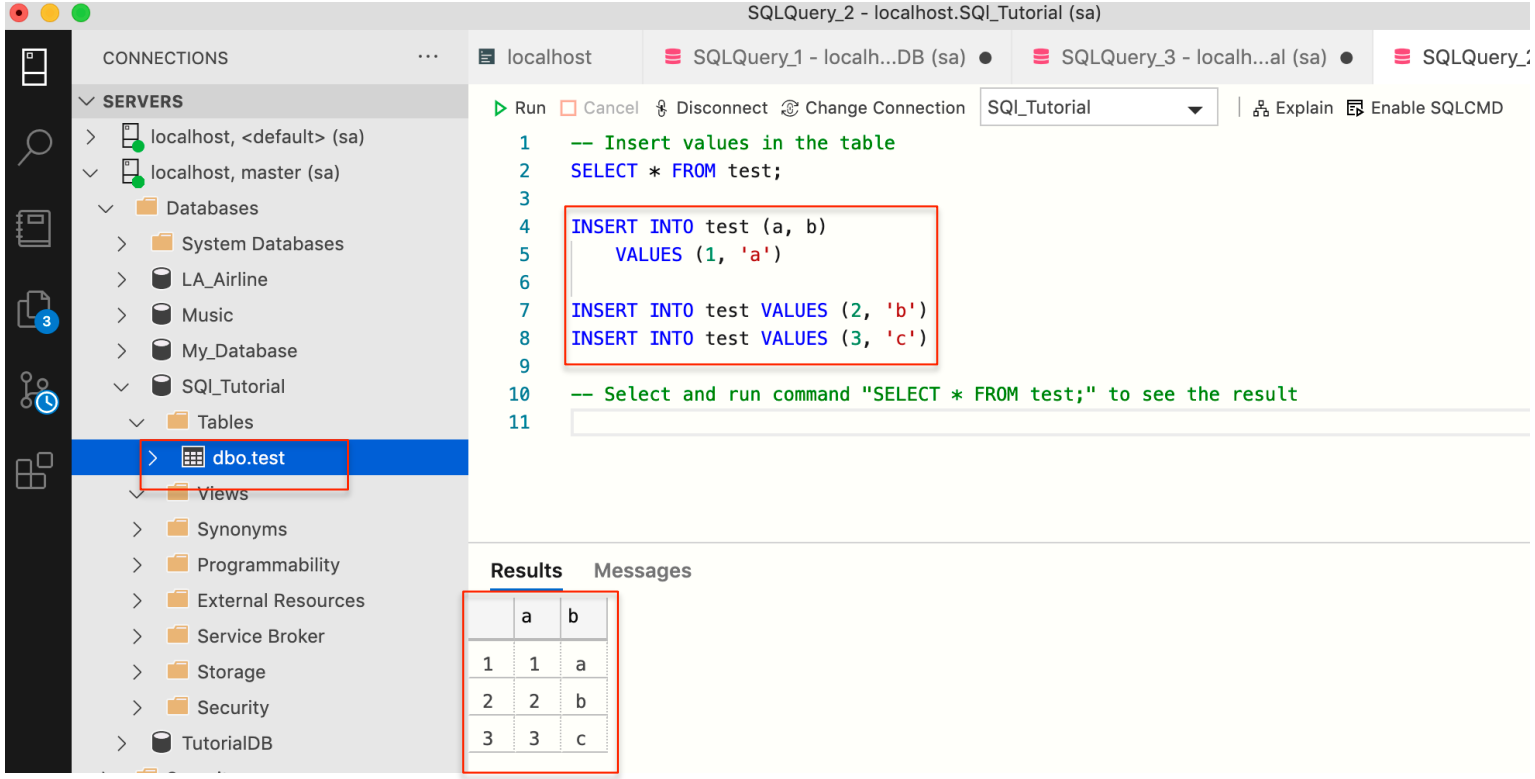
Creating a Table

- 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 →



Inserting Values

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



Example Tables: Customer

- DB **SQl_Tutorial**, Table **Customer:**

```sql
CREATE TABLE Customer (
    Customer_ID INT,
    Name VARCHAR(20),
    Address TEXT,
    City VARCHAR(20),
    State VARCHAR(20),
    Zip INT,
    Email TEXT
) ;

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**:

```sql
-- Inserting values in the table Customer
SELECT * FROM Customer;

INSERT INTO Customer VALUES (1, 'Sean Smith', '110 Convoy St', 'San Diego', 'CA', 92123, 'seansmith@work.com')
INSERT INTO Customer VALUES (2, 'Som Sharma', '121 Pepper Tree Ln', 'Poway', 'CA', 92064, 'somsharma@work.com')
INSERT INTO Customer VALUES (3, 'Jag Gos', '235 Costa Verde', 'Irvine', 'CA', 92112, 'jagos@work.com')
INSERT INTO Customer VALUES (4, 'Lili Golzs', '211 Dorian St', 'San Francisco', 'CA', 93678, 'liligolzs@work.com')
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:**

```sql
CREATE TABLE item (
    id INT,
    name VARCHAR(20),
    description TEXT
);

SELECT * FROM item;
```

- DB **SQl_Tutorial,** Table **sale:**

```sql
CREATE TABLE sale (
    id INT,
    item_id INT,
    customer_id INT,
    quantity INT,
    price INT
);

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**:

```sql
-- Inserting data into table item
SELECT * FROM item;

INSERT INTO item VALUES (1, 'T-shirt', 'Clothing');
INSERT INTO item VALUES (2, 'Pant', 'Clothing');
INSERT INTO item VALUES (3, 'Kitchen Towel', 'Household');
INSERT INTO item VALUES (4, 'Smart TV', 'Electronics');
INSERT INTO item VALUES (5, 'Perfume', 'Beauty');
```

- Inserting values in table **sale**:

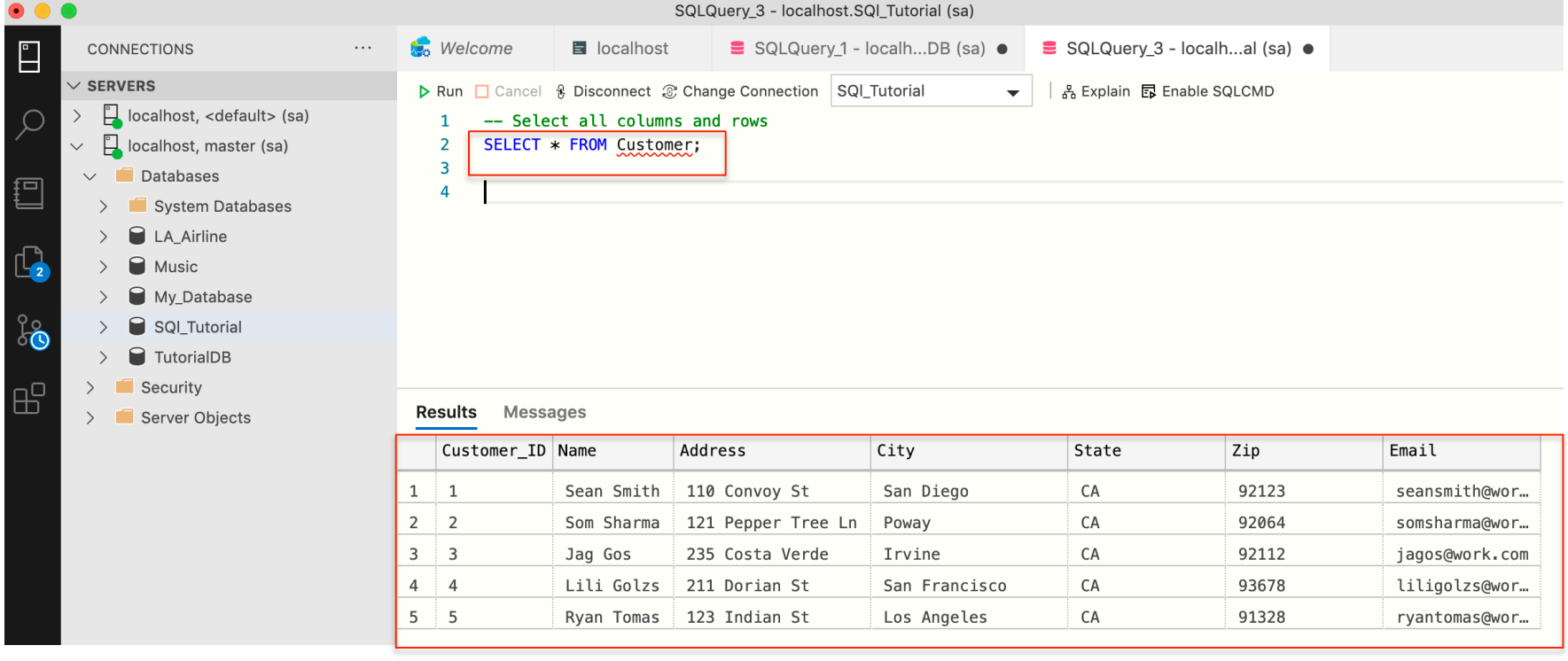
```sql
-- Inserting data into table sale
SELECT * FROM sale;

INSERT INTO sale VALUES (1, 2, 2, 3, 200);
INSERT INTO sale VALUES (2, 1, 4, 2, 150);
INSERT INTO sale VALUES (3, 4, 1, 1, 500);
INSERT INTO sale VALUES (4, 2, 5, 3, 200);
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

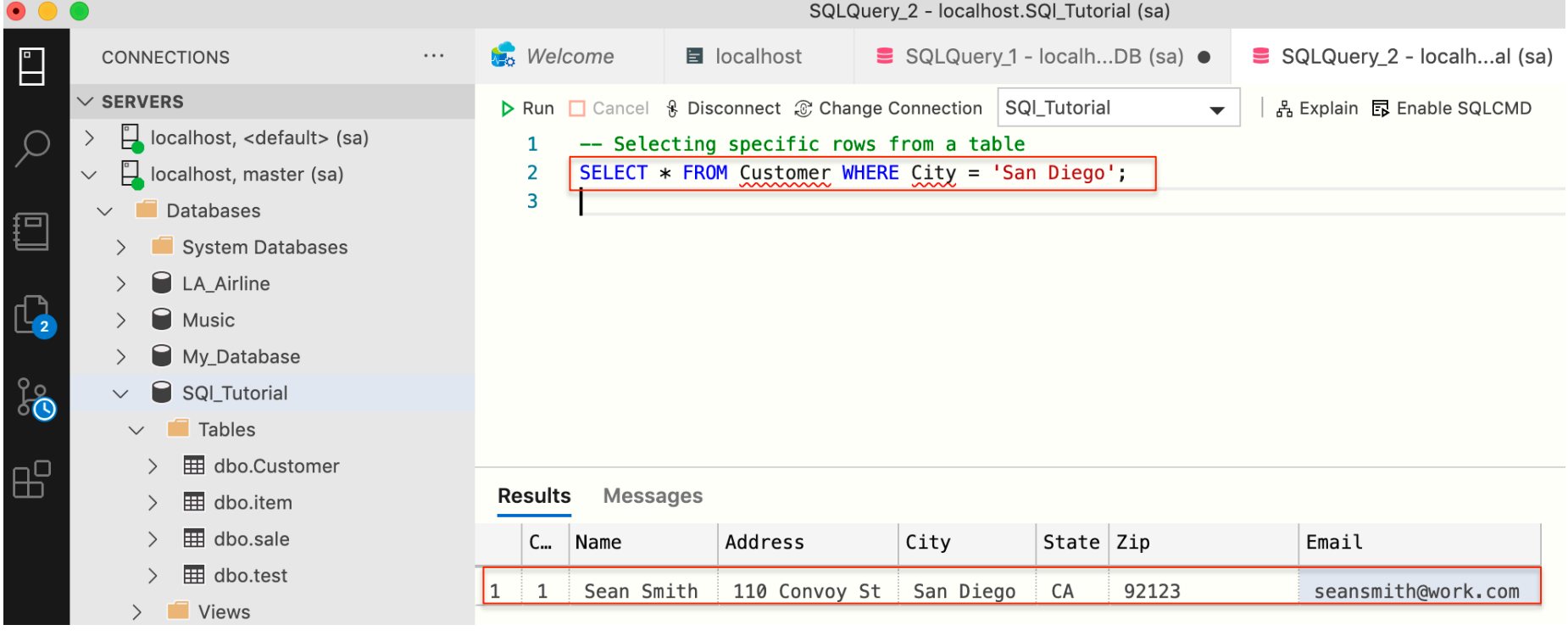
Retrieving All Columns And Rows

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



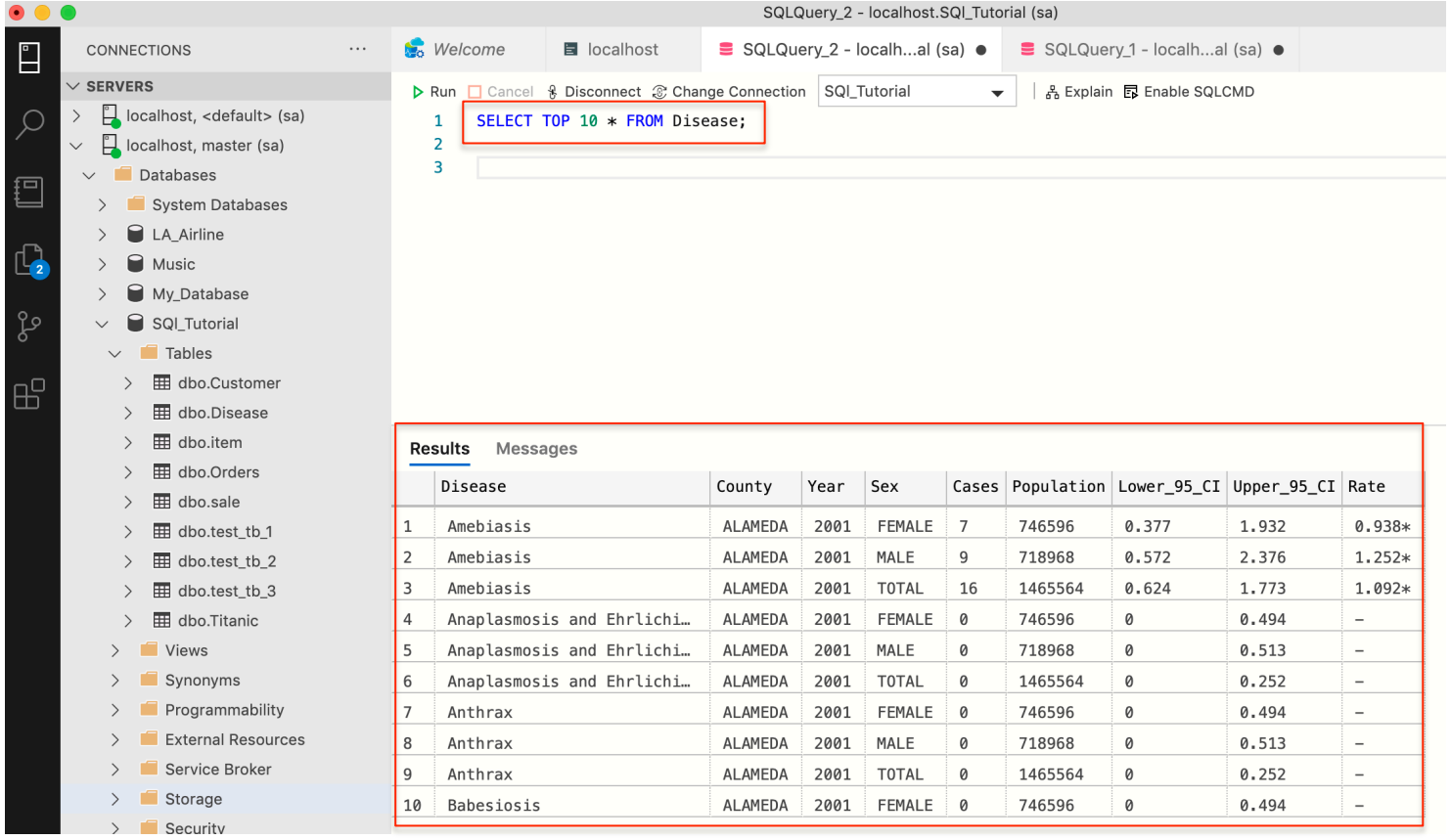
Retrieving Subset Of Rows

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



Retrieving Top N number Rows

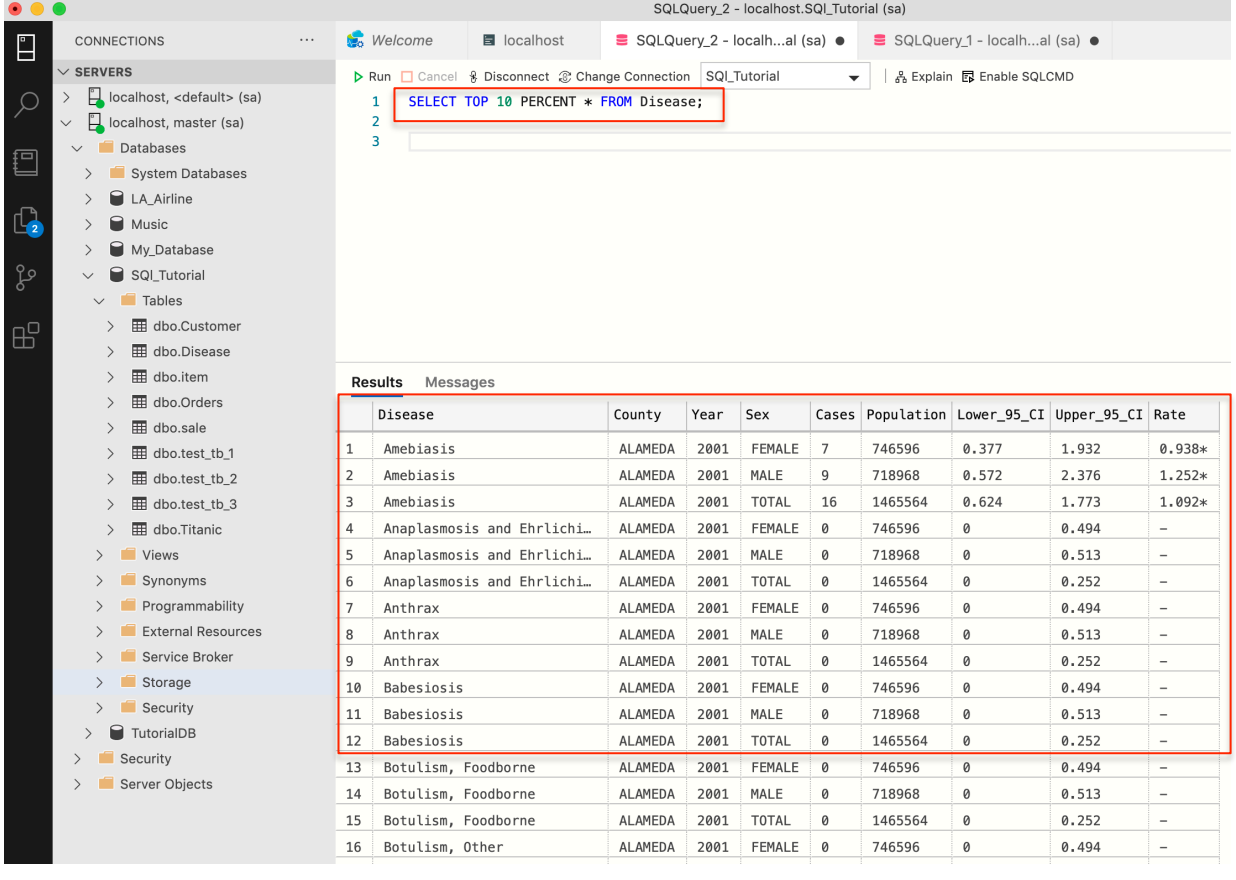
- Retrieving top N number rows from table **Disease** using **SELECT TOP** statement followed by number of rows
- Run the following command as shown in below figure →



	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 Ehrlichi…	ALAMEDA	2001	FEMALE	0	746596	0	0.494	–
5	Anaplasmosis and Ehrlichi…	ALAMEDA	2001	MALE	0	718968	0	0.513	–
6	Anaplasmosis and Ehrlichi…	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

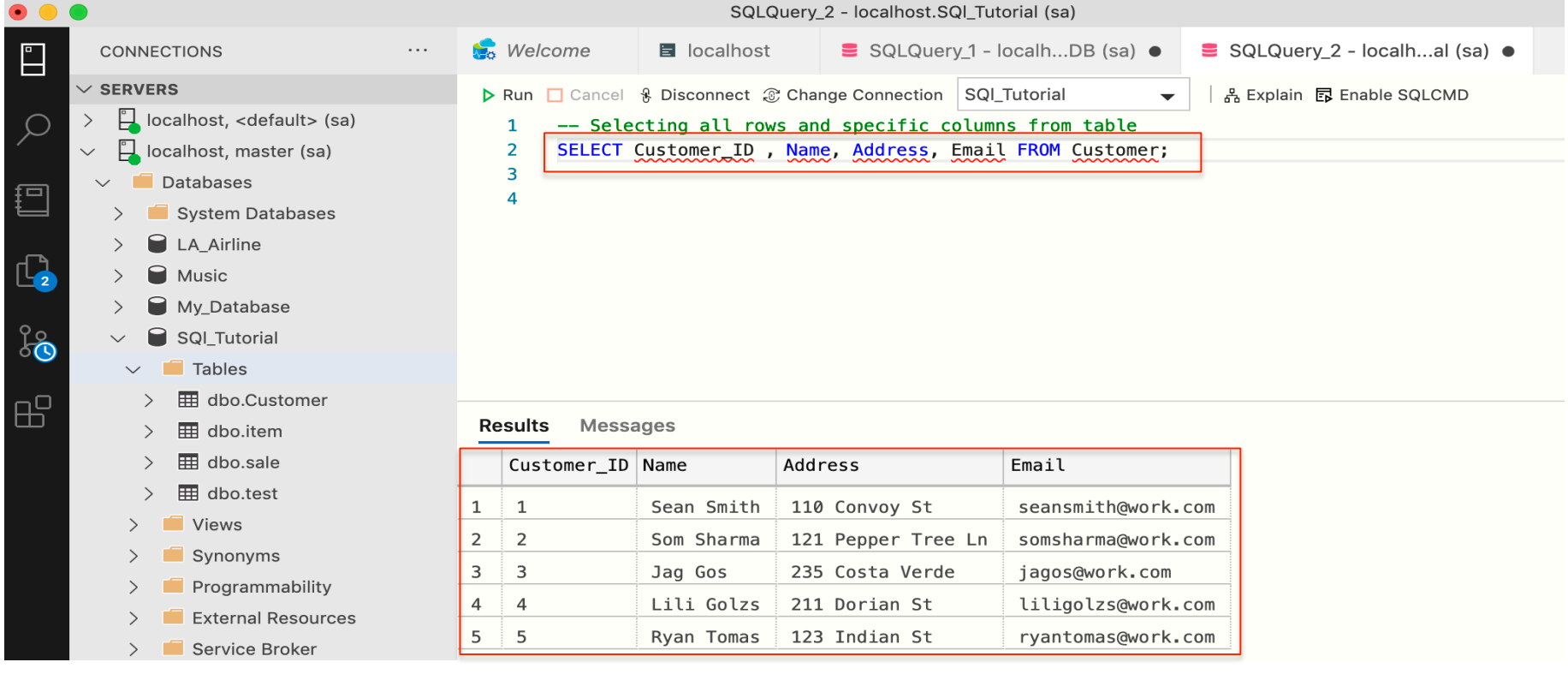
- Retrieving top x percent rows from table **Disease** using **SELECT TOP x PERCENT** statement
- Run the following command as shown in below figure →



	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 Ehrlichi…	ALAMEDA	2001	FEMALE	0	746596	0	0.494	–
5	Anaplasmosis and Ehrlichi…	ALAMEDA	2001	MALE	0	718968	0	0.513	–
6	Anaplasmosis and Ehrlichi…	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	–

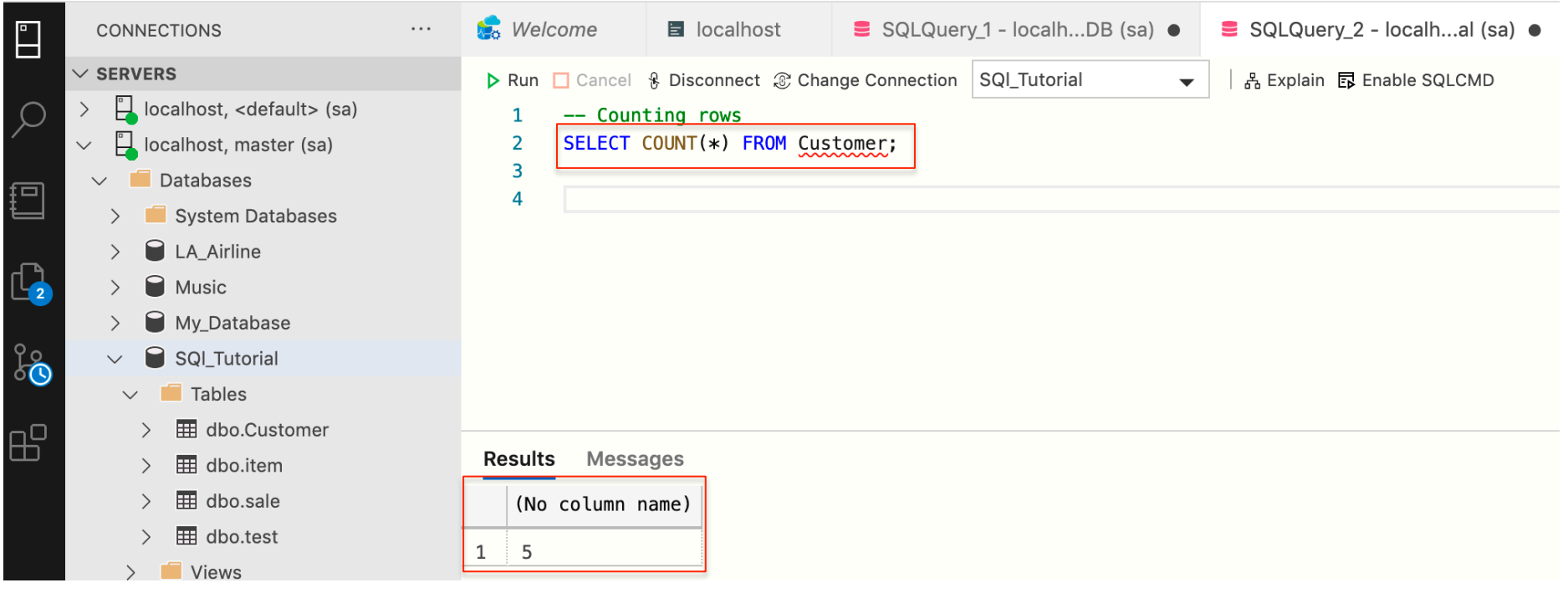
Retrieving Subset Of Columns

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



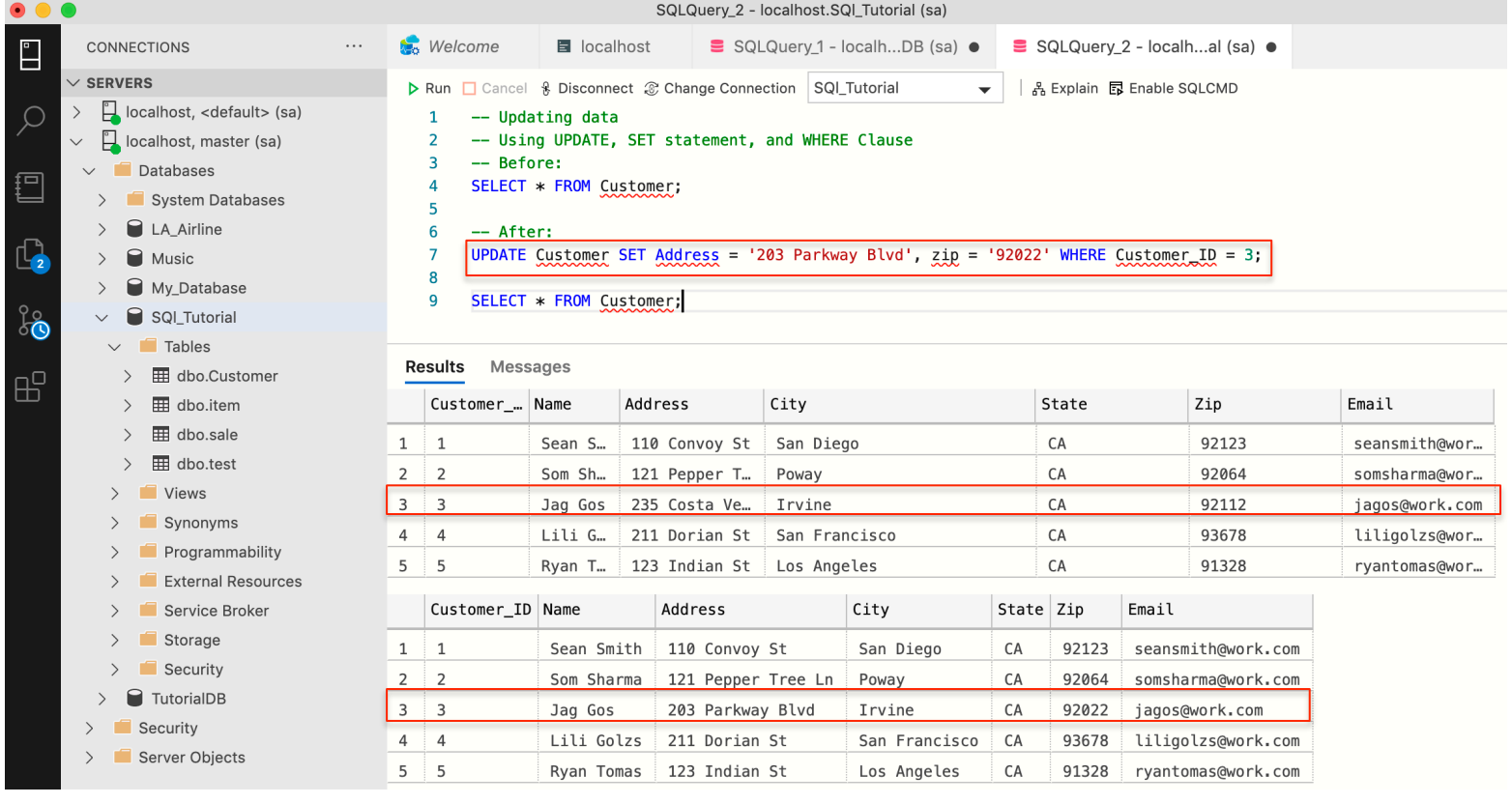
Counting Rows

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



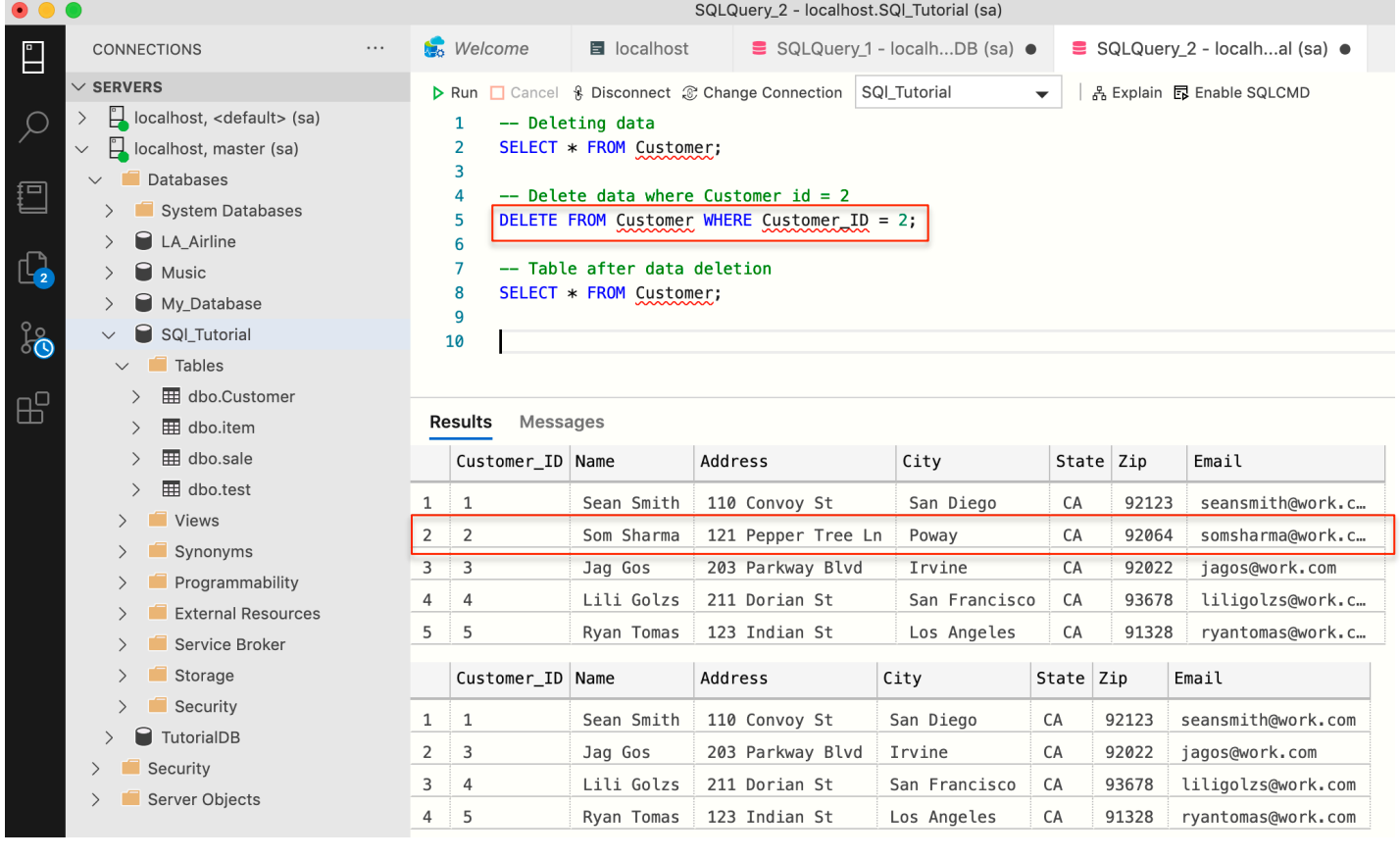
Updating Data

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



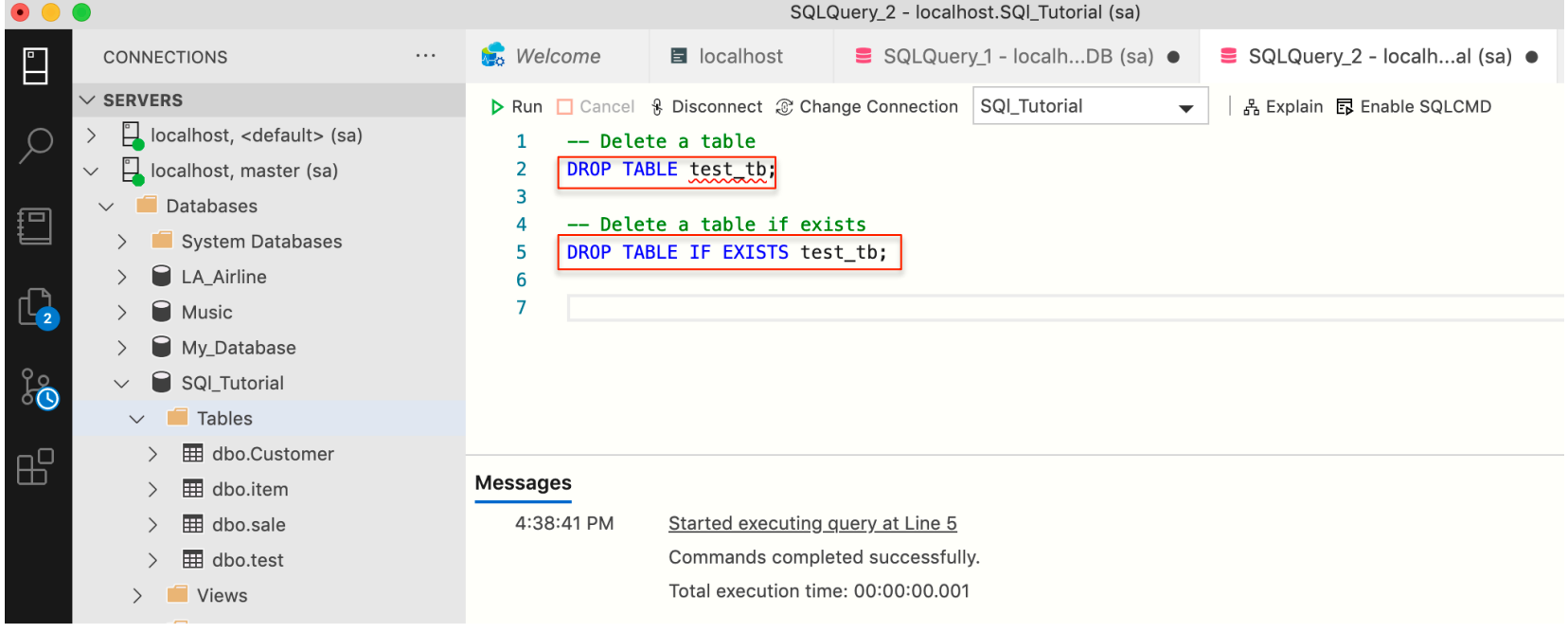
Deleting Data

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



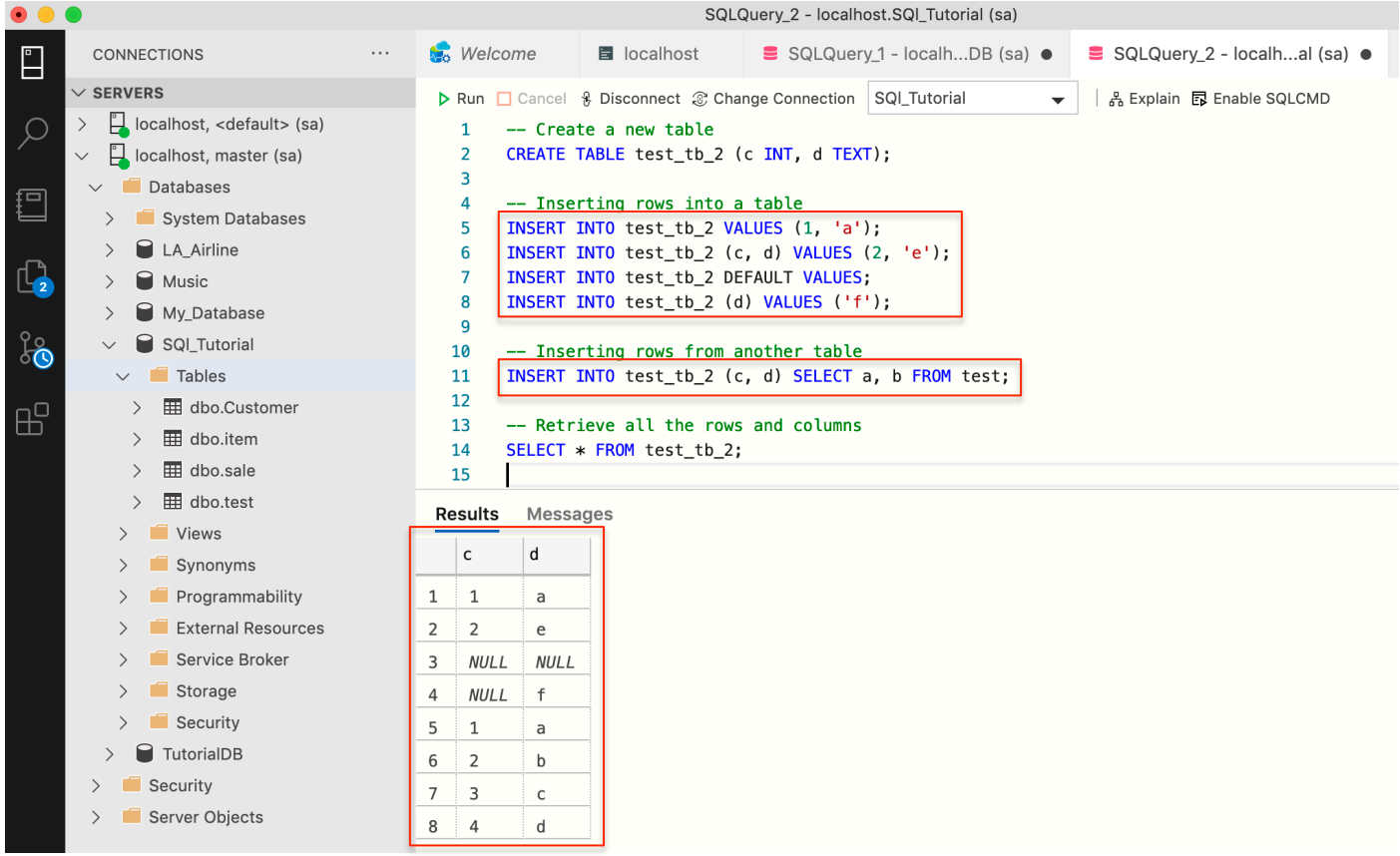
Deleting Table

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



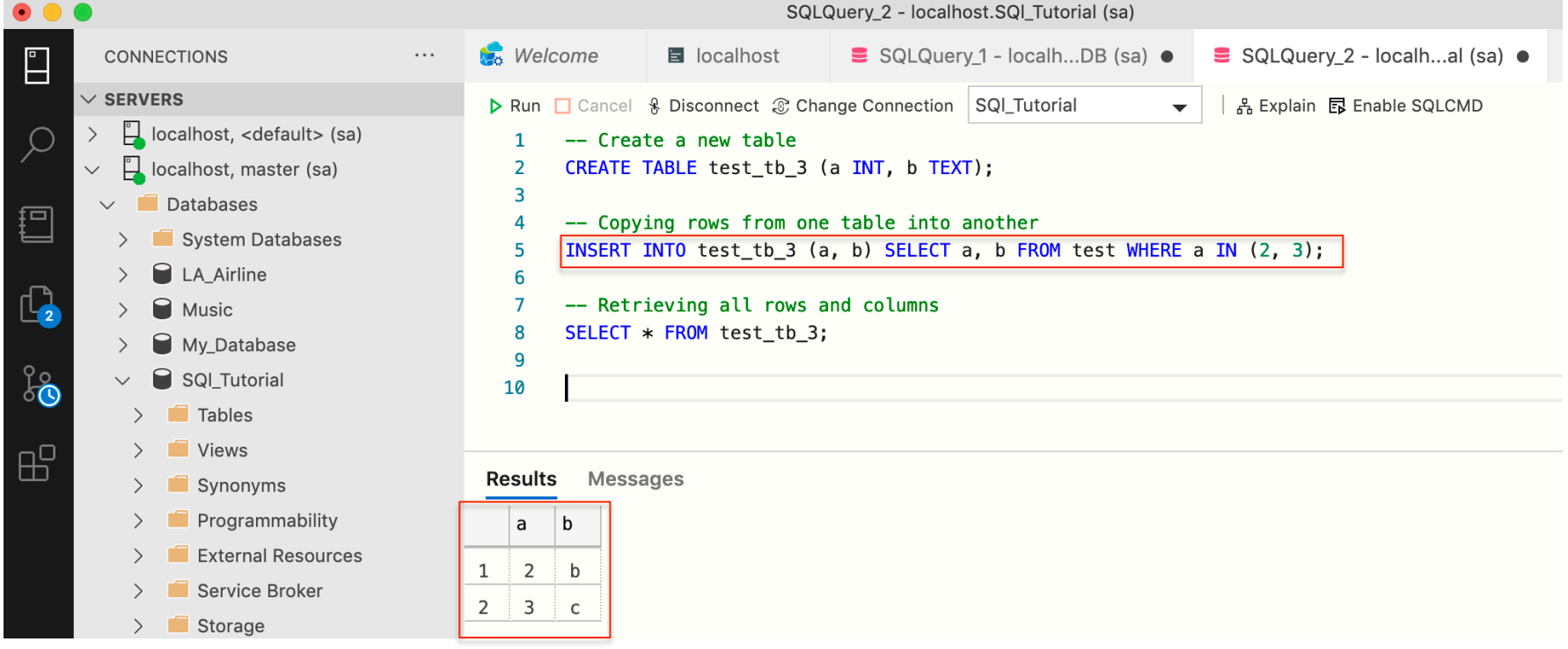
Inserting Rows

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



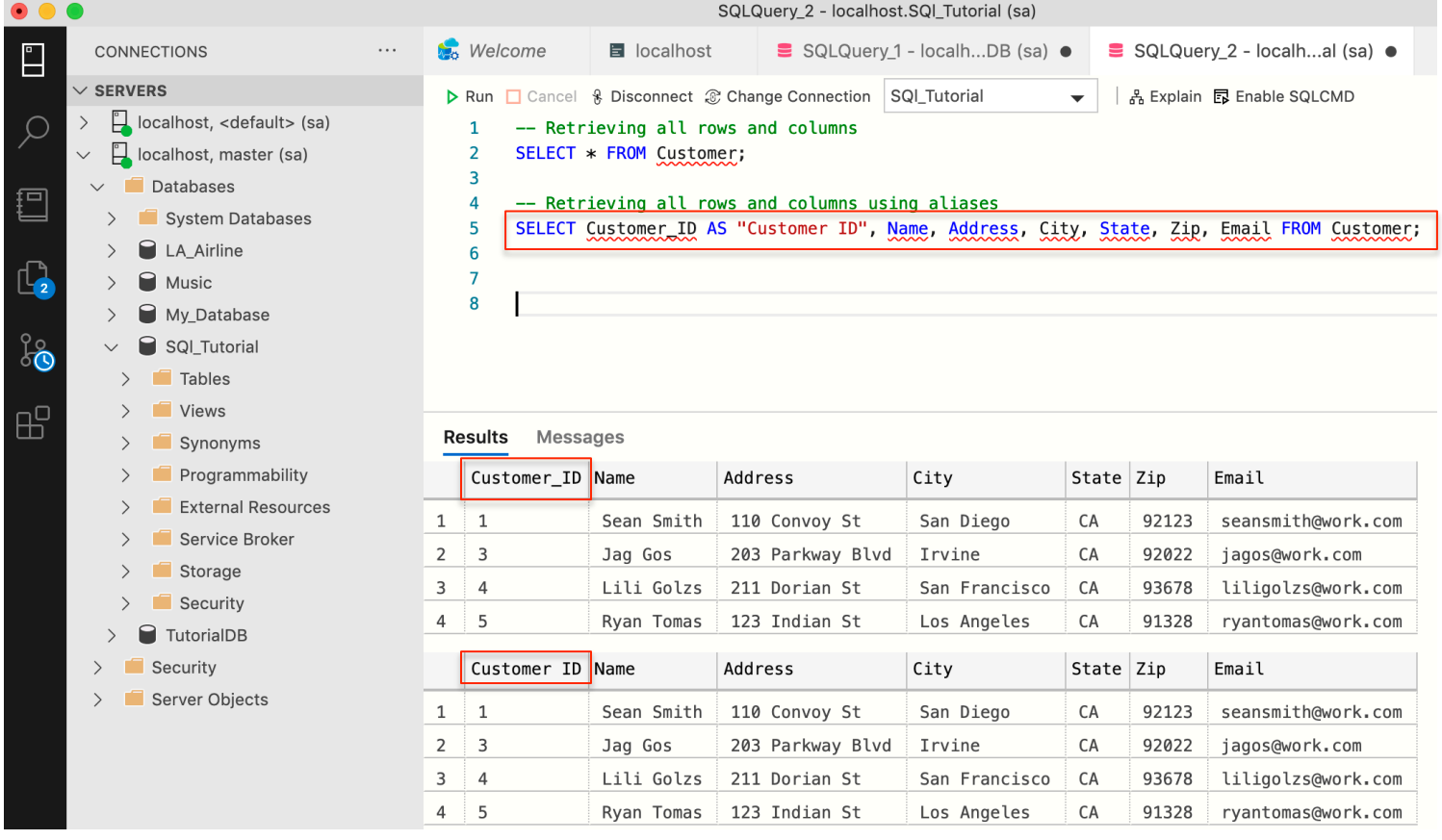
Copying Rows From One Table Into Another

- 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 →



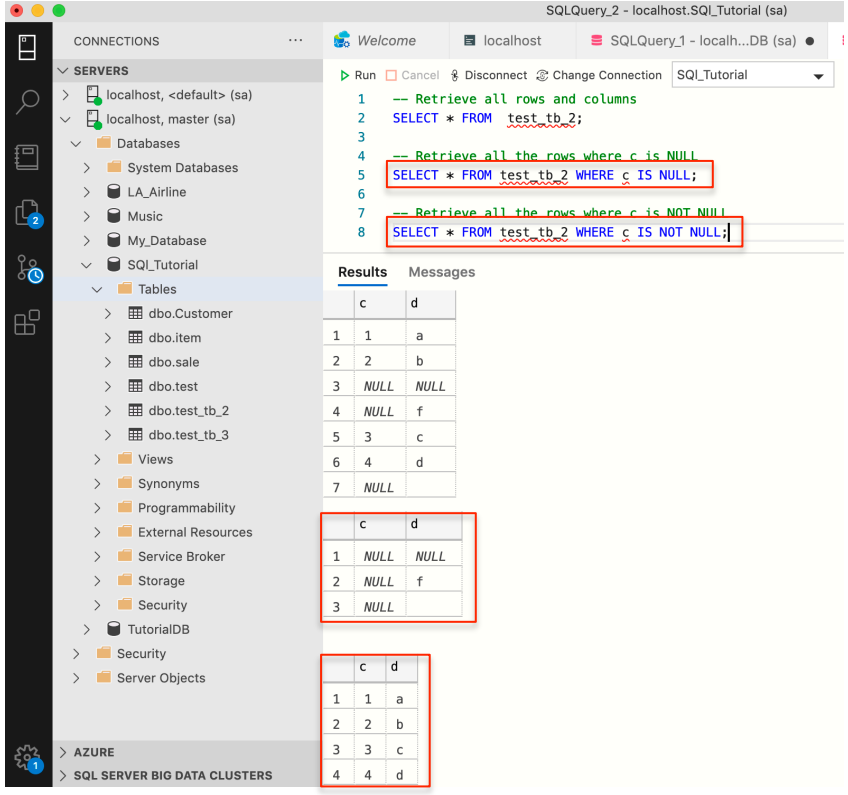
Using Aliases

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



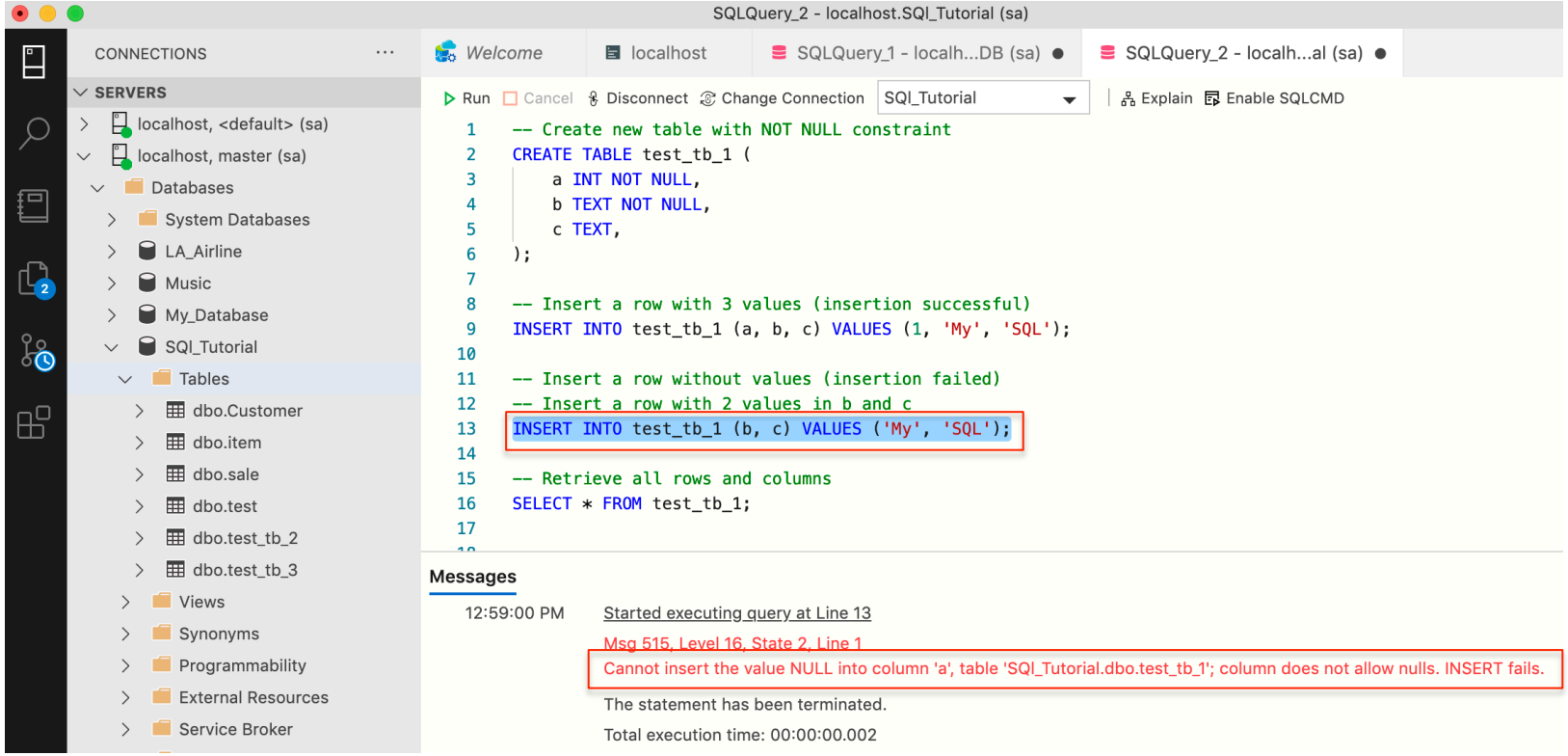
The NULL Value

- 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 →



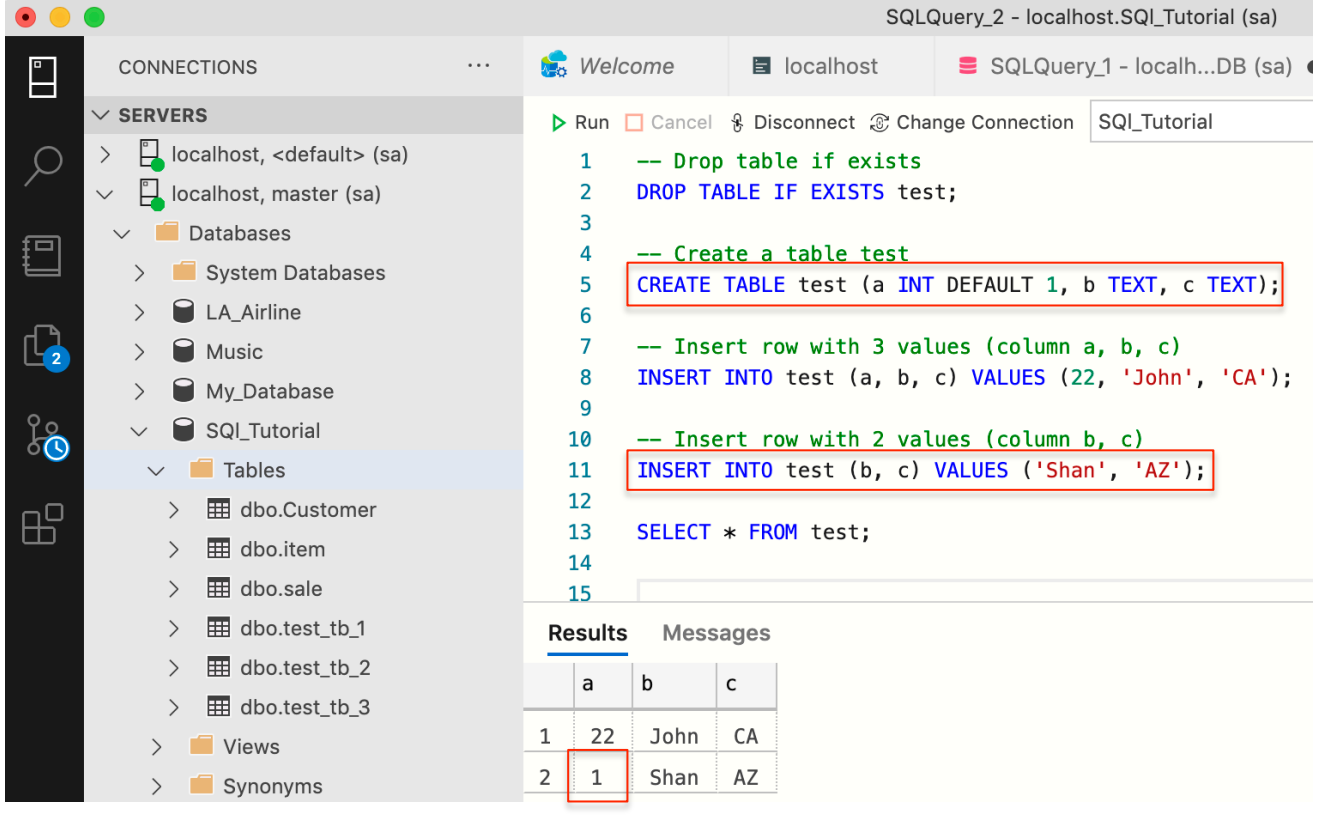
Constraining columns: NOT NULL

- 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 →



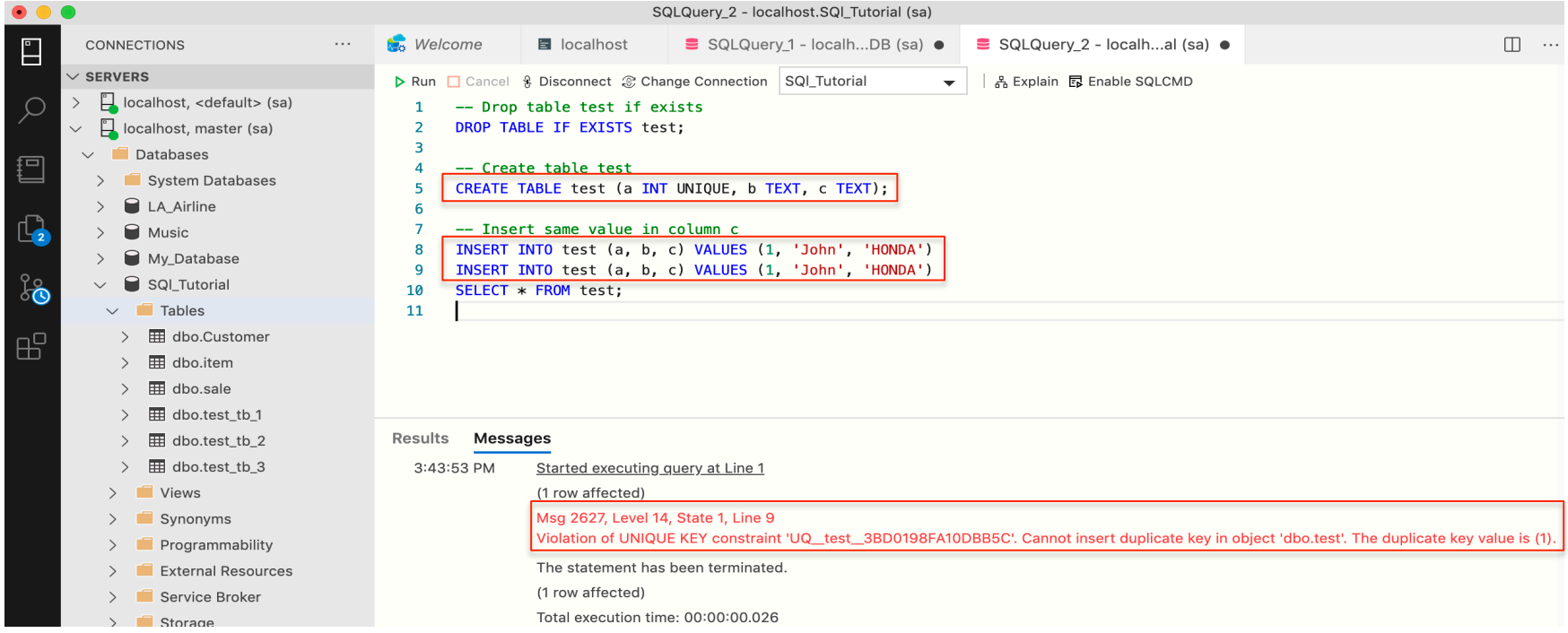
Constraining columns: DEFAULT

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



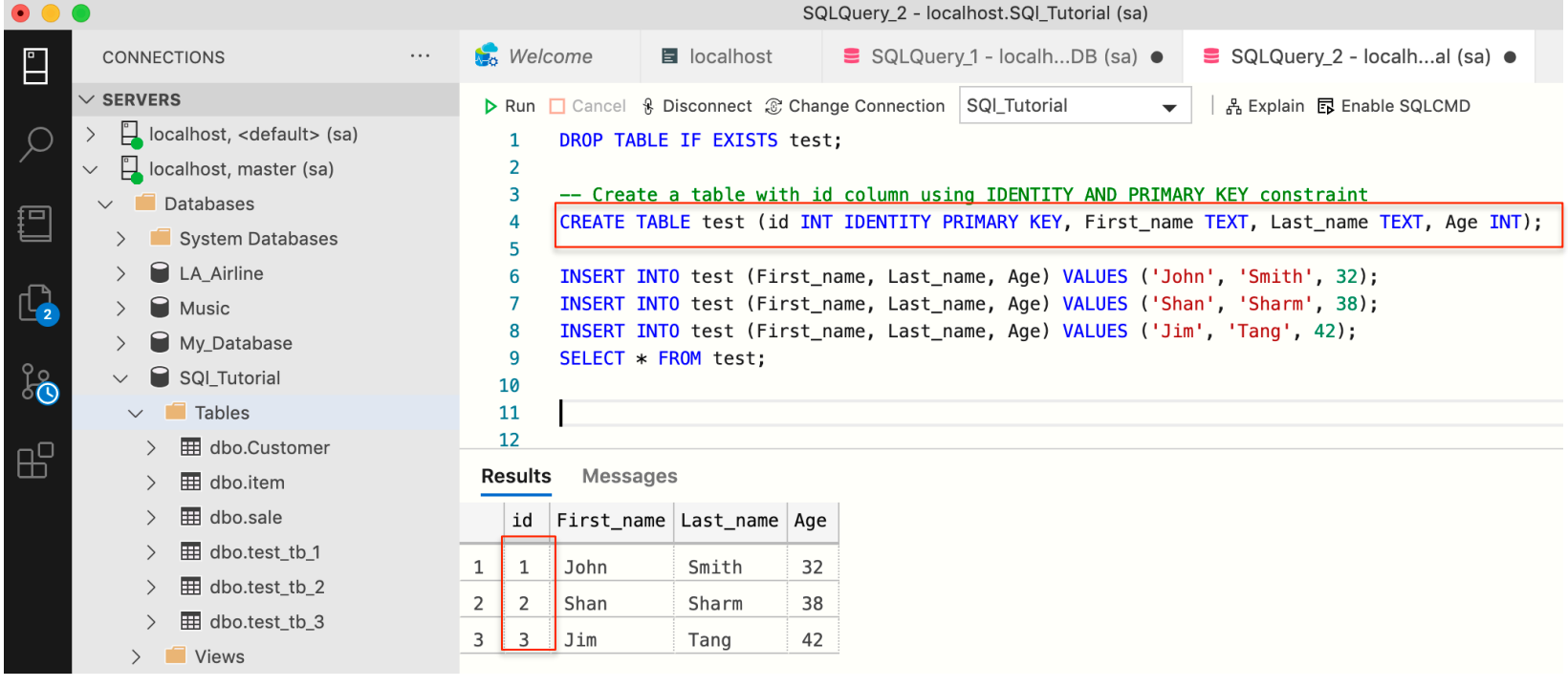
Constraining columns: UNIQUE

- 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 →



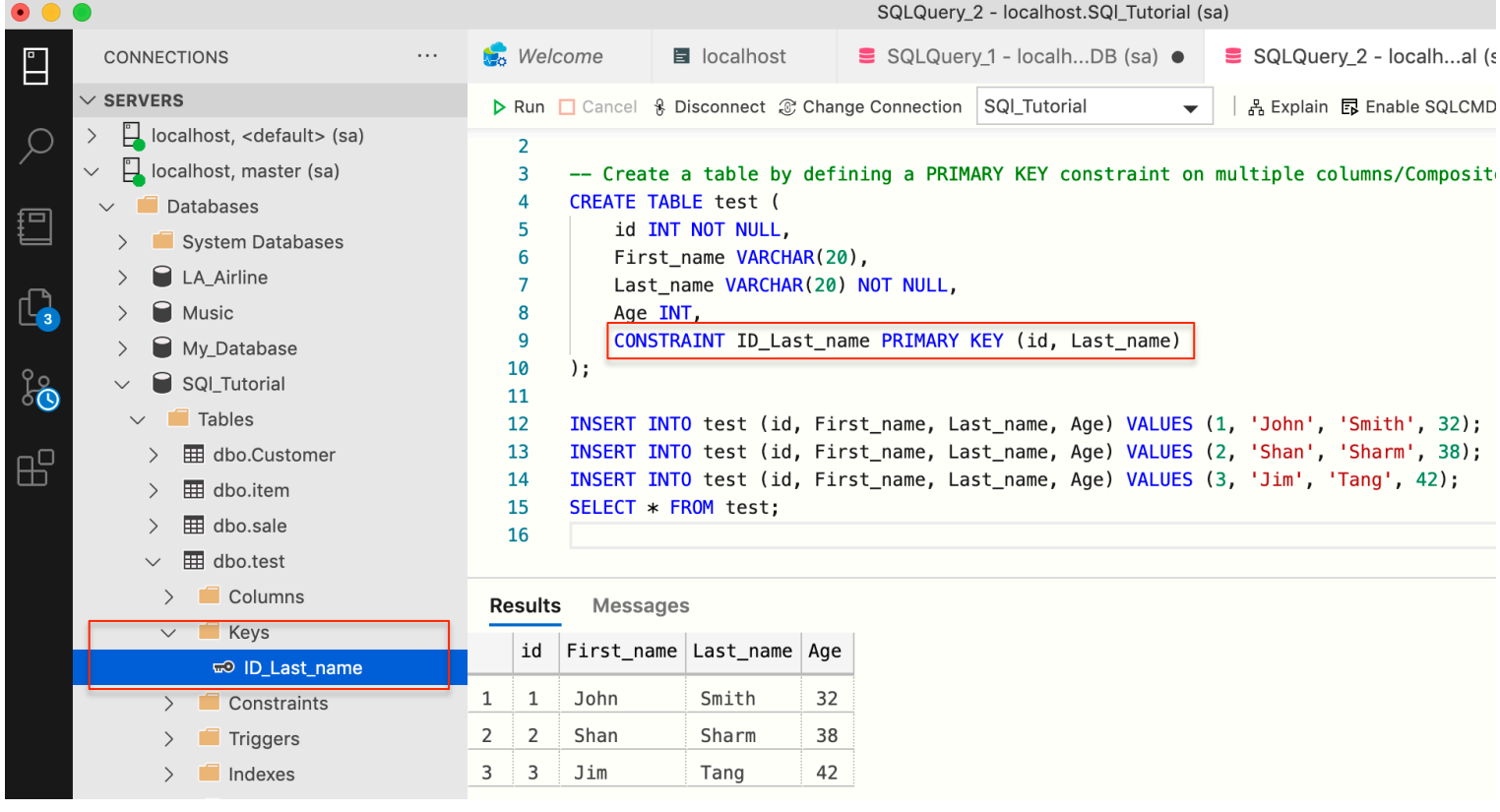
Primary Key Constraint On Create Table

- 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 →



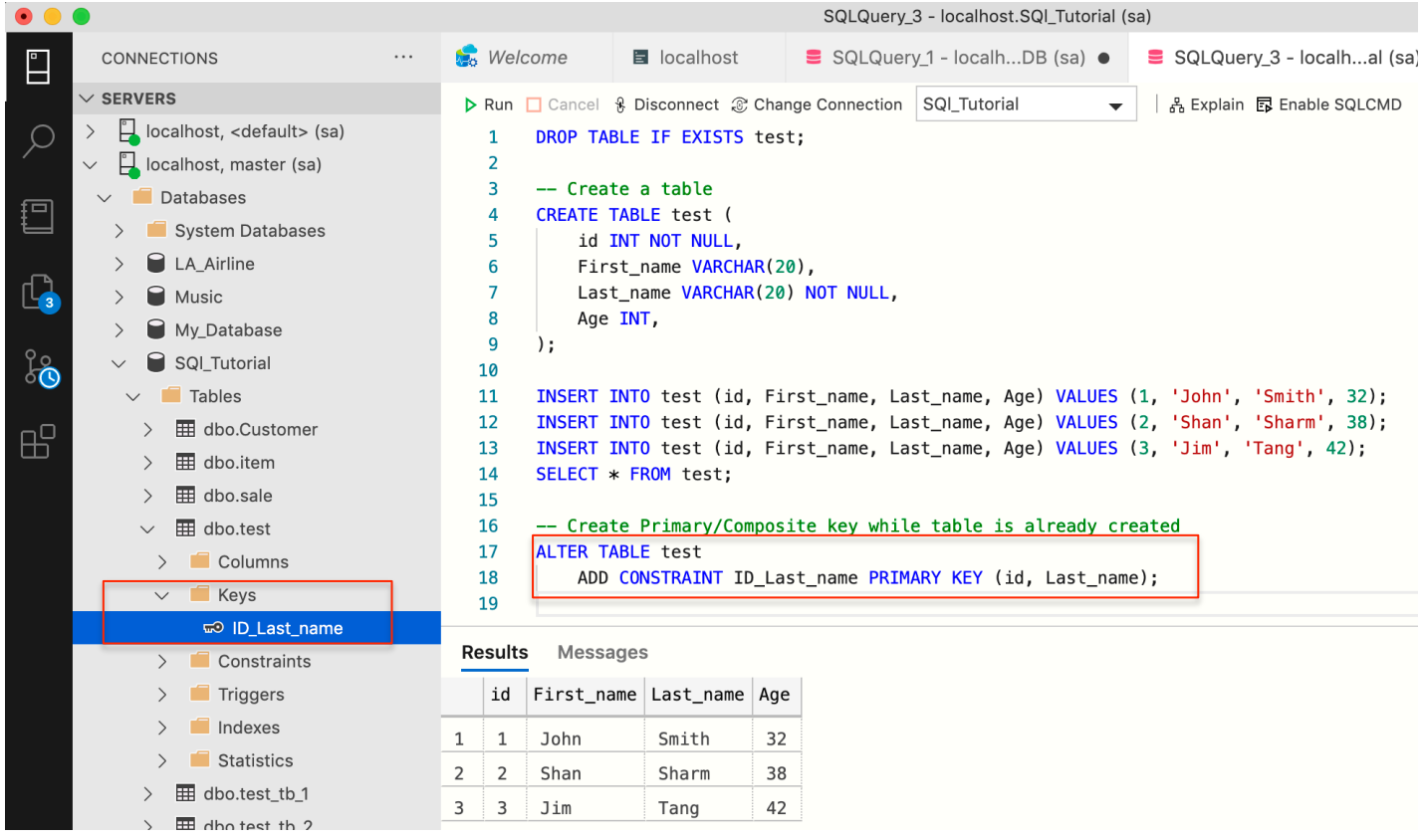
Multi-Column Primary Key Constraint On Create Table

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



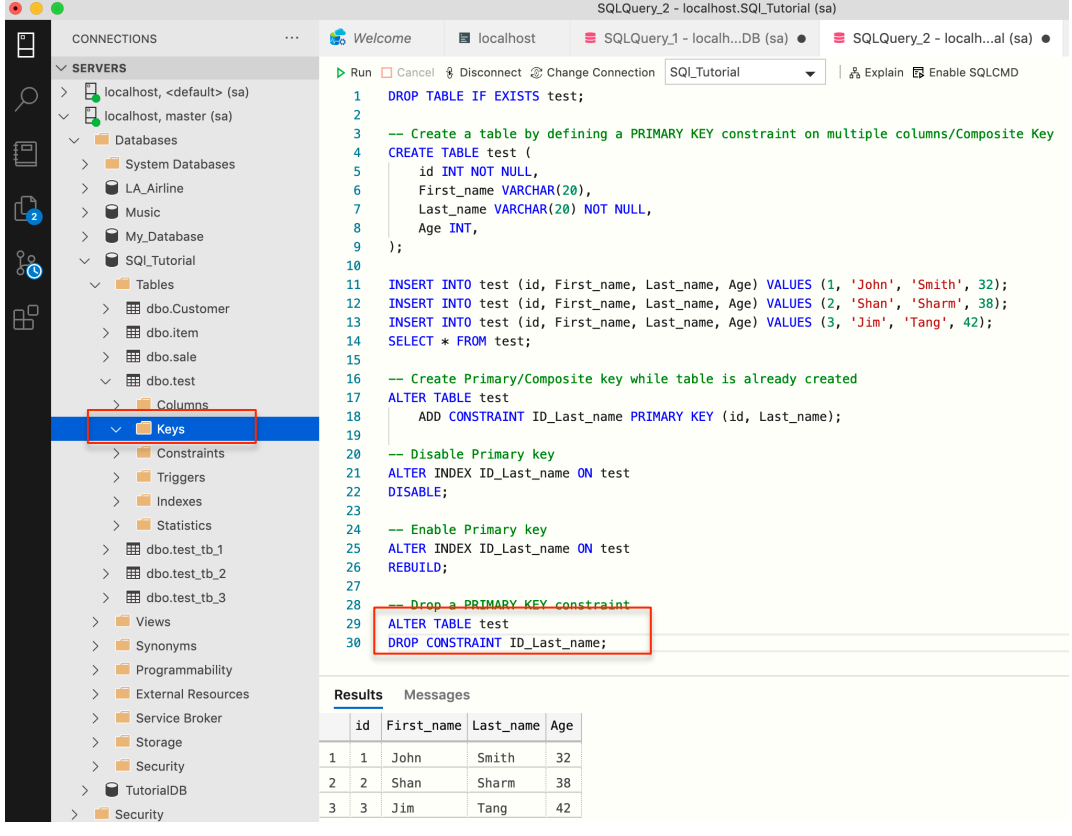
Multi-Column Primary Key Constraint On Existing Table

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



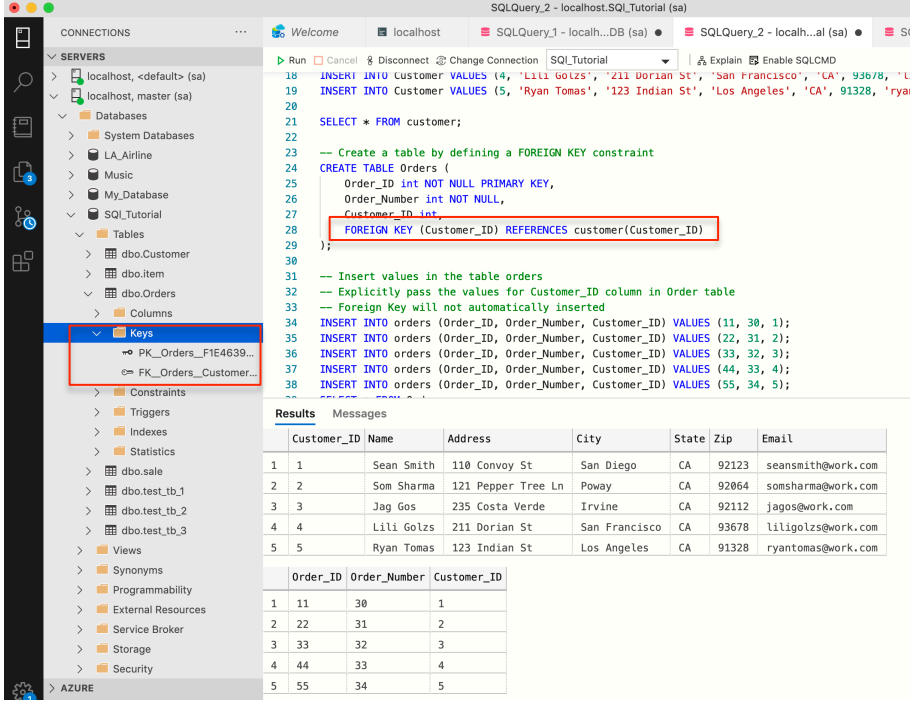
Disable, Enable and Drop A Primary Key Constraint

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



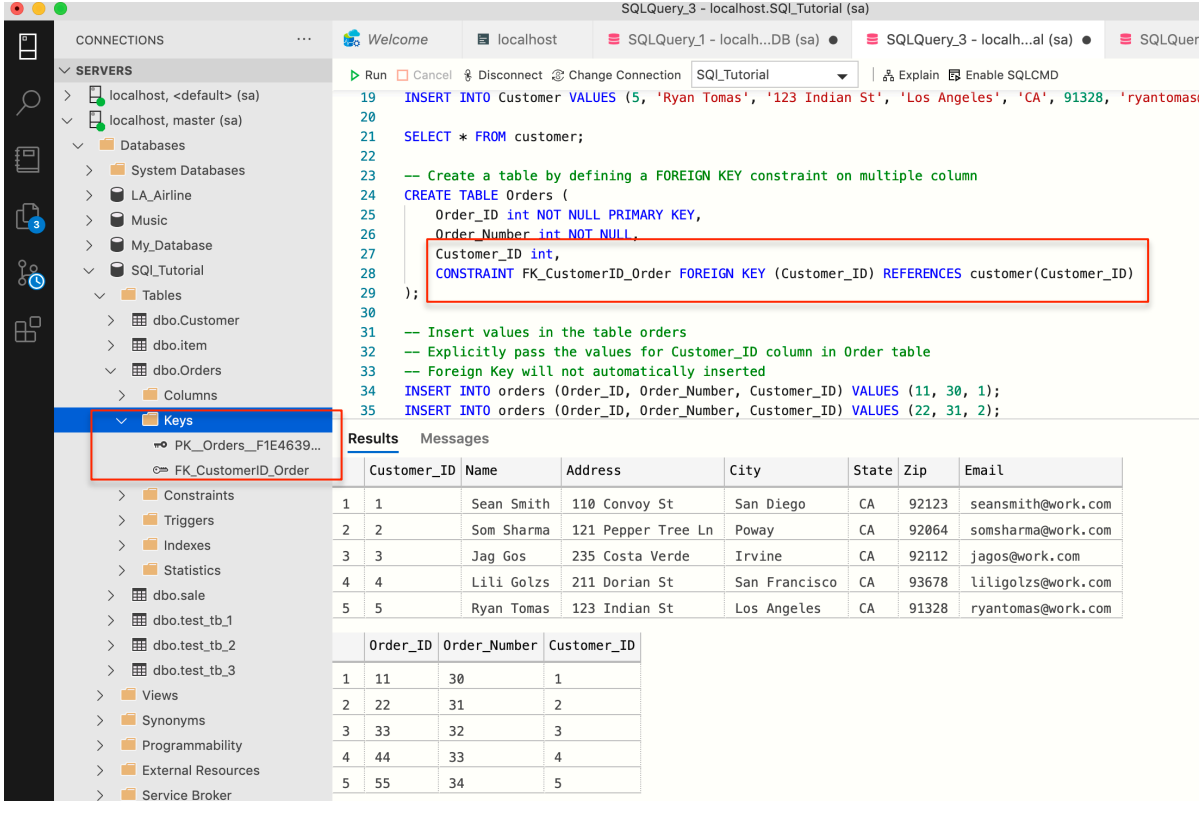
Foreign Key Constraint On Create Table

- 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 →



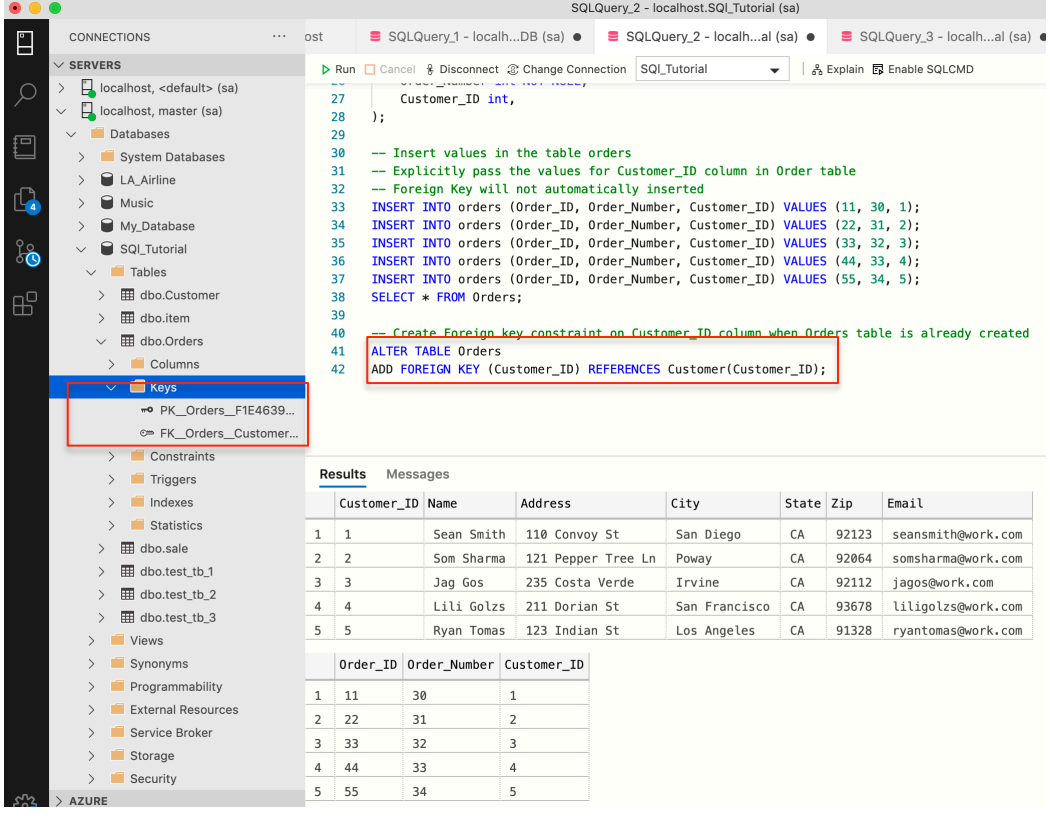
Multi-Column Foreign Key Constraint On Create Table

- 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 →



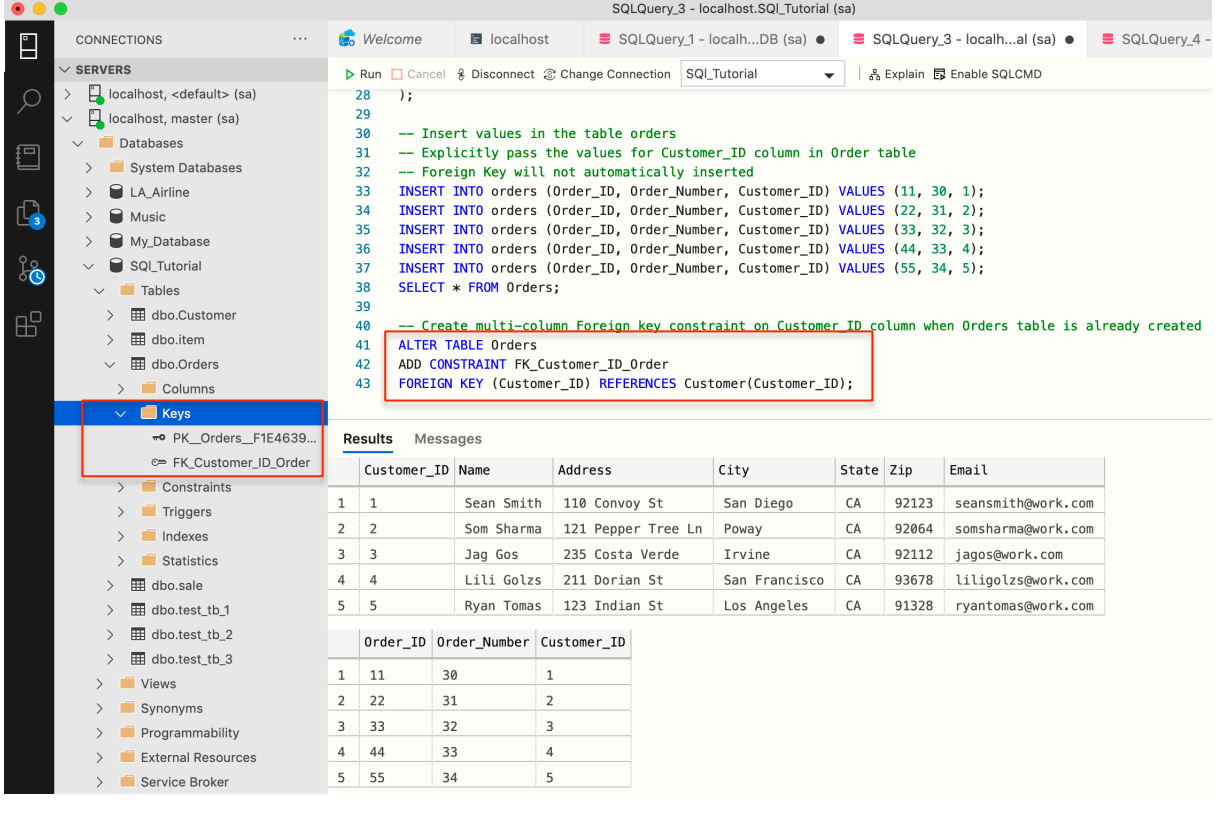
Foreign Key Constraint On Existing Table

- 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 →



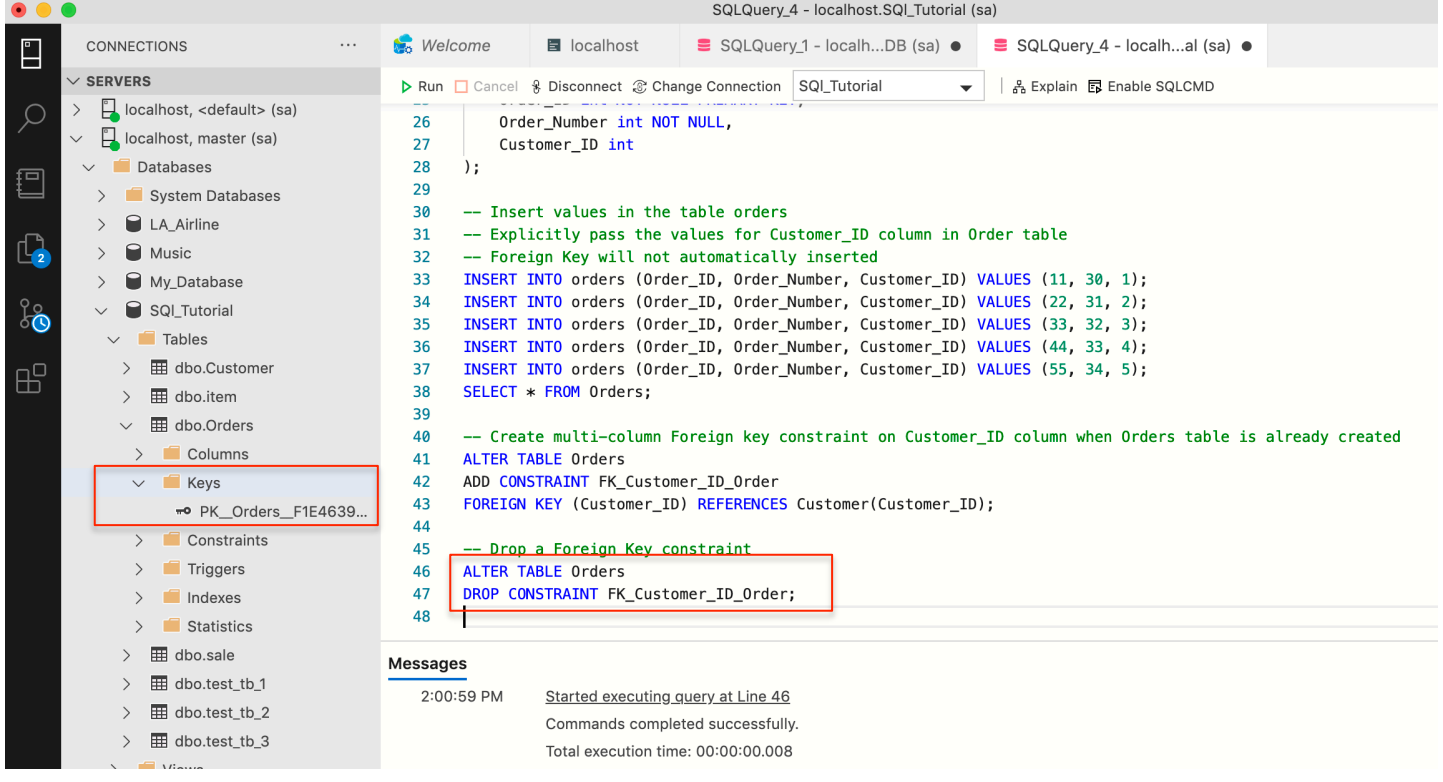
Multi-Column Foreign Key Constraint On Existing Table

- 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 →



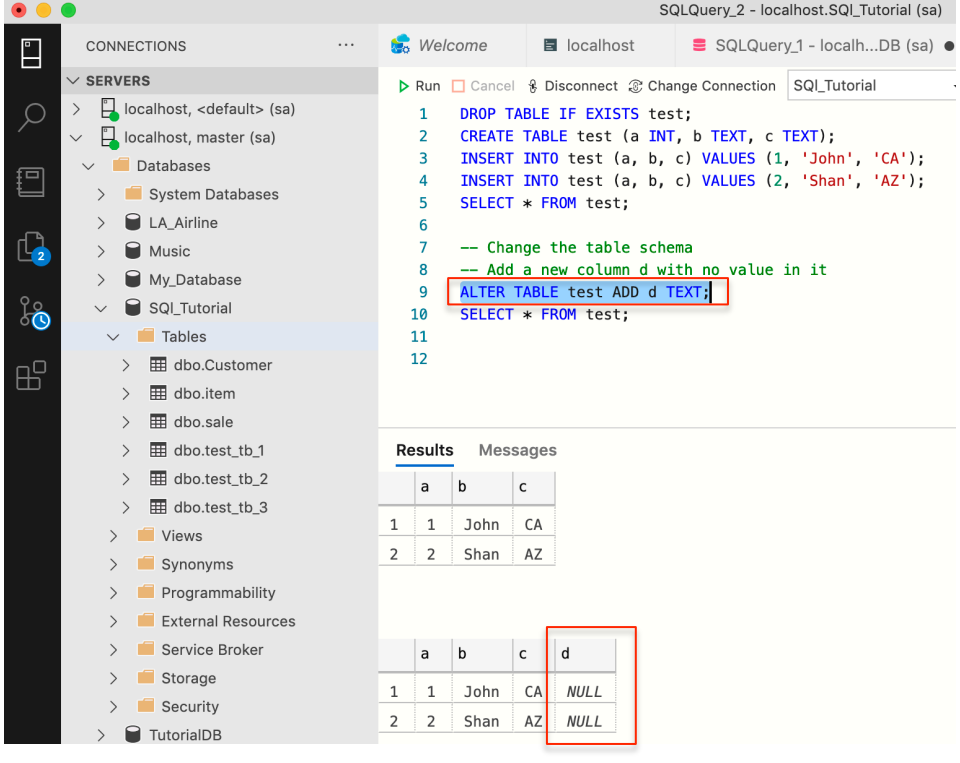
Drop A Foreign Key Constraint

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



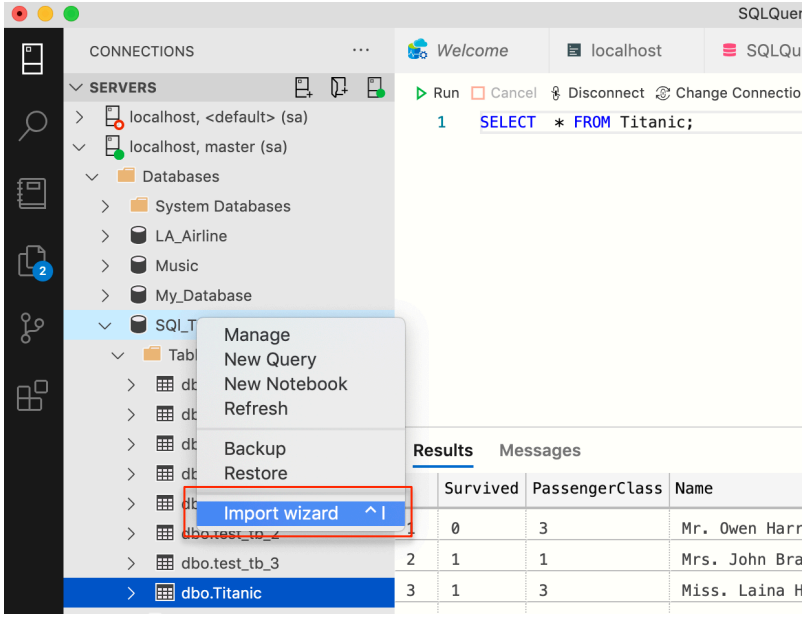
Changing A Schema: Add A New Column

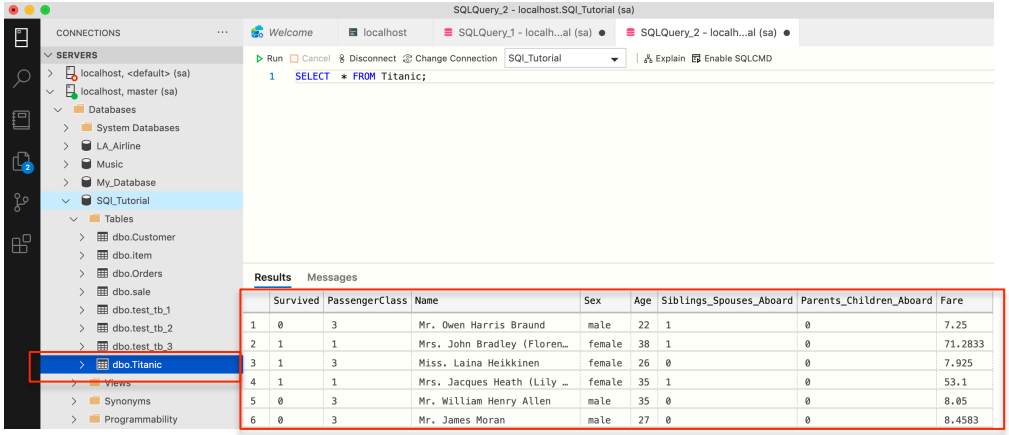
- Change a table schema after it's already been defined and populated with data
- Using ALTER statement followed by ADD clause
- In the following example, we added d column
- Run the following command as shown in below figure →



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

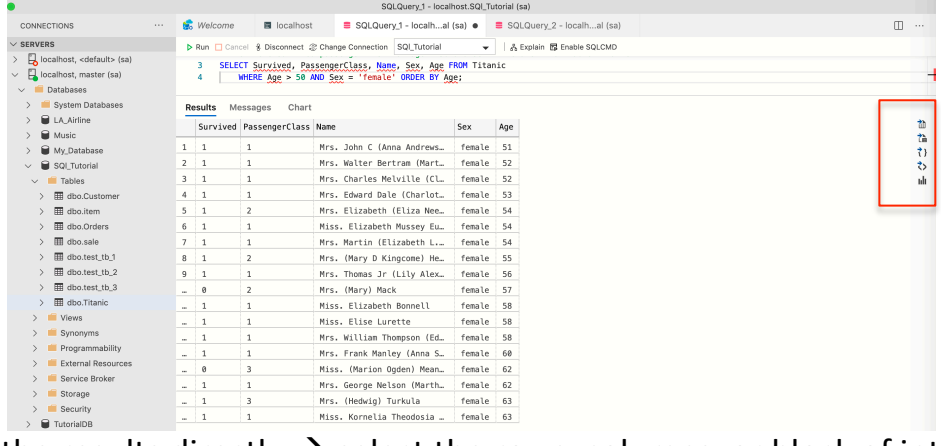




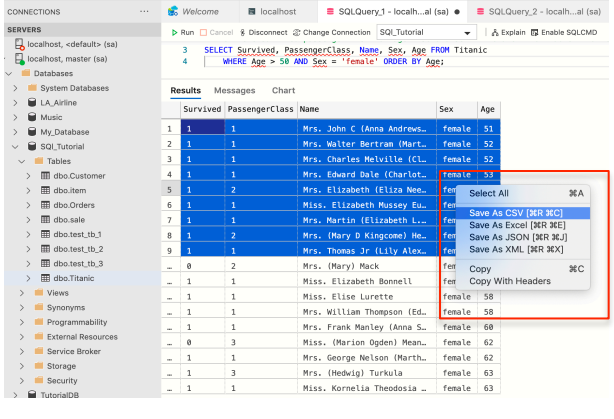
Imported titanic.csv file using Import Wizard

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.

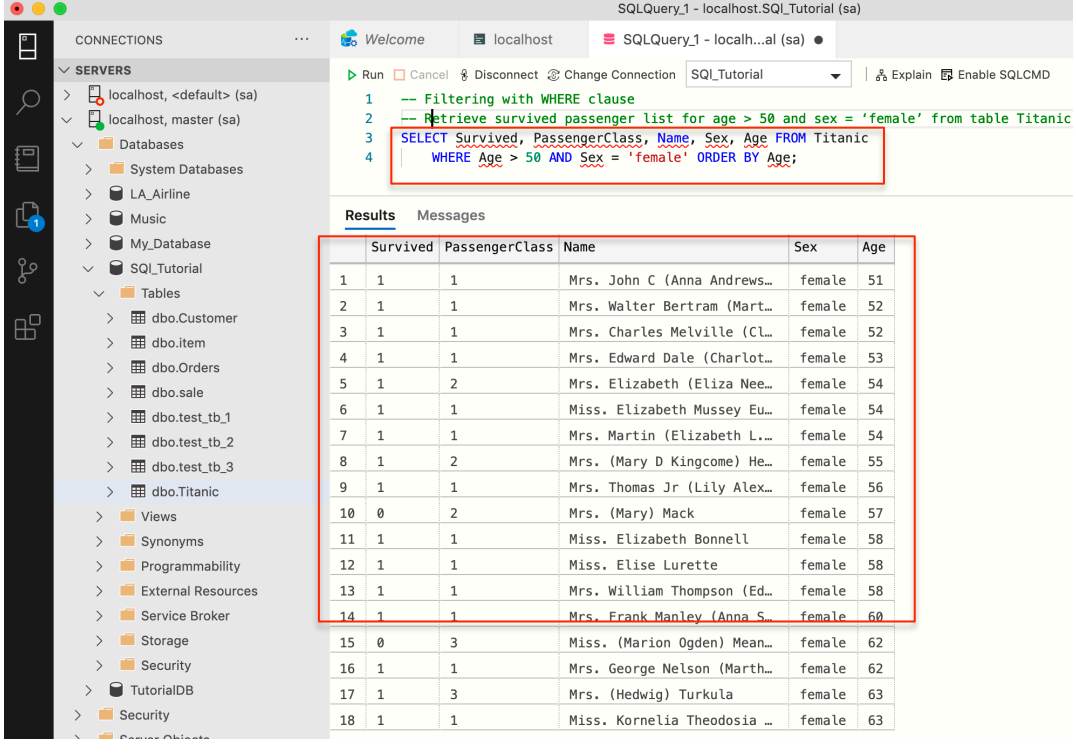


- To save a subset of the results directly → select the rows, columns, or block of interest (click and drag to select within the result)→ then Right click → A dialog will appear where to save the selection



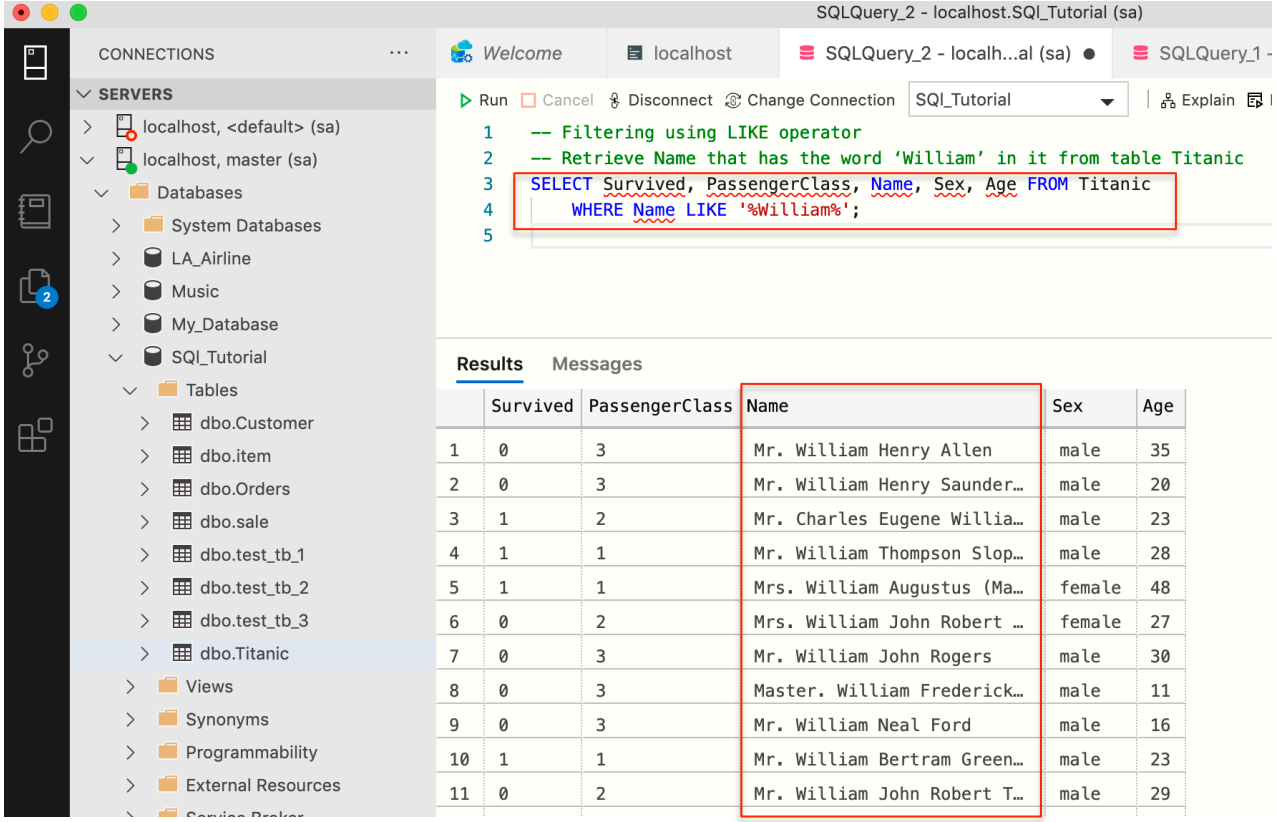
Filtering Data: WHERE Clause

- 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 →



Filtering Data: LIKE Operator

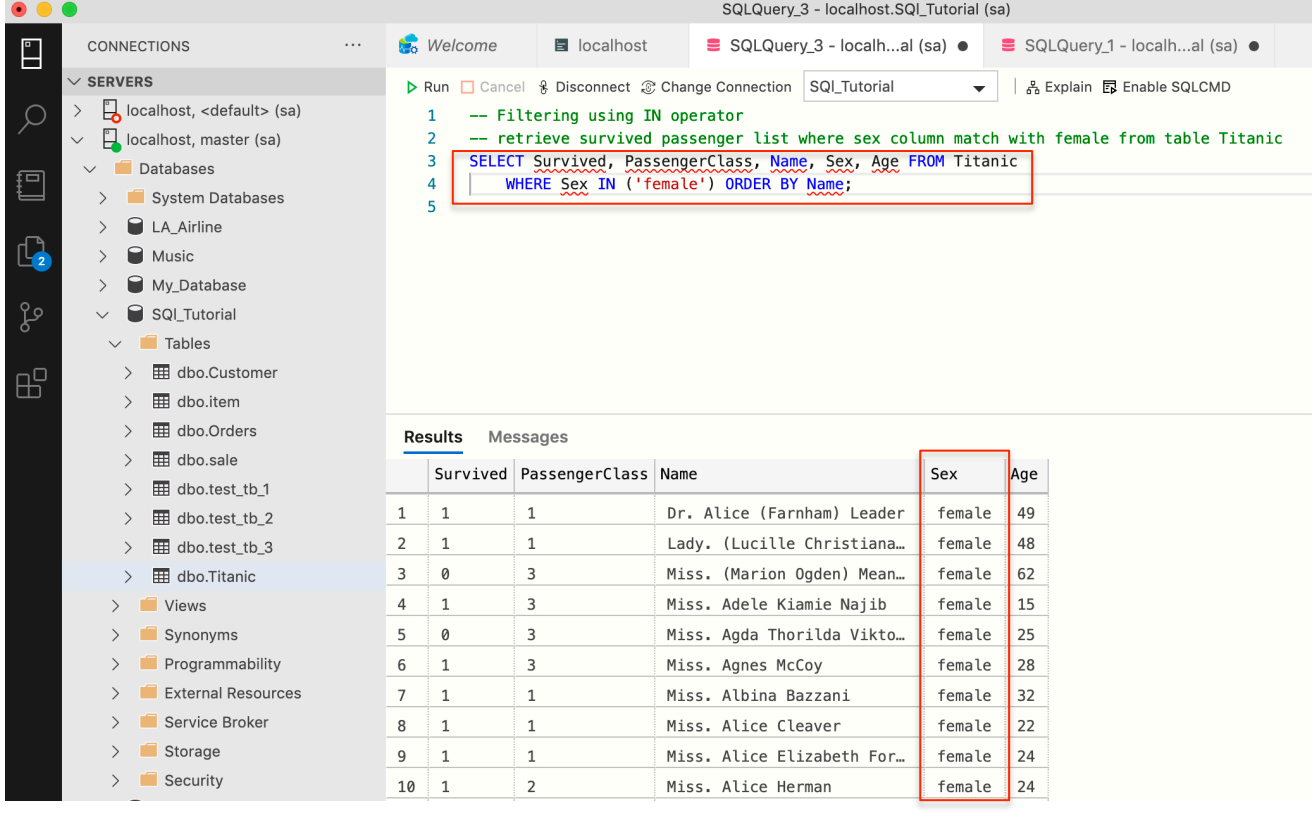
- Filtering data using **LIKE** operator by constructing a wildcard (% sign)
- For e.g., we want to retrieve Name that has the word 'William' in it from table Titanic
- Run the following command as shown in below figure →



	Survived	PassengerClass	Name	Sex	Age
1	0	3	Mr. William Henry Allen	male	35
2	0	3	Mr. William Henry Saunder…	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

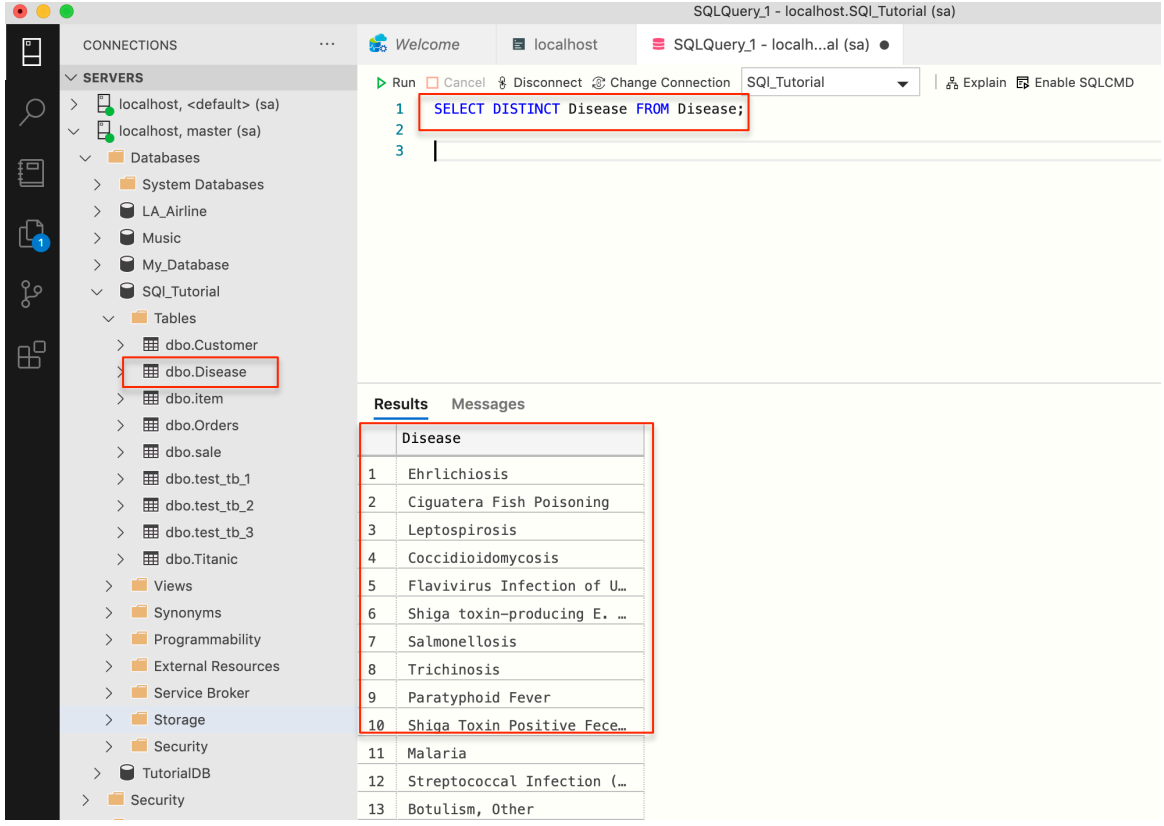
Filtering Data: IN Operator

- **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 →



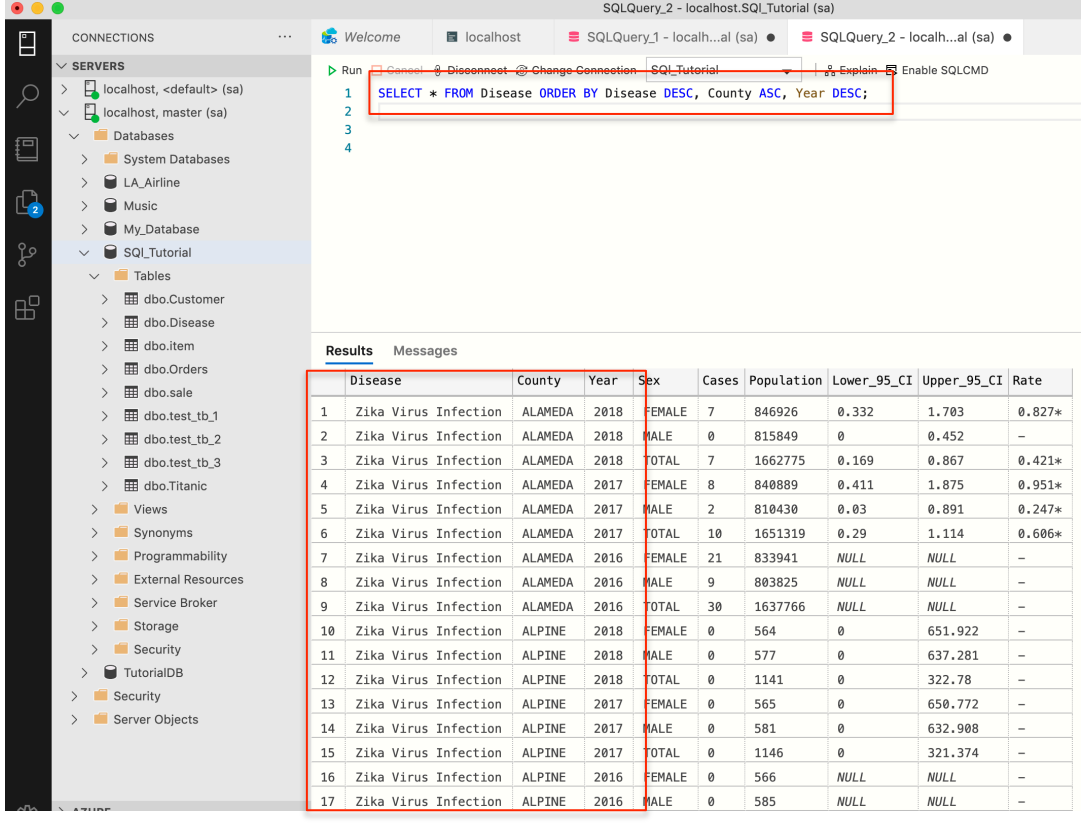
Removing Duplicates

- 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 →



Ordering Results: ORDER BY Clause

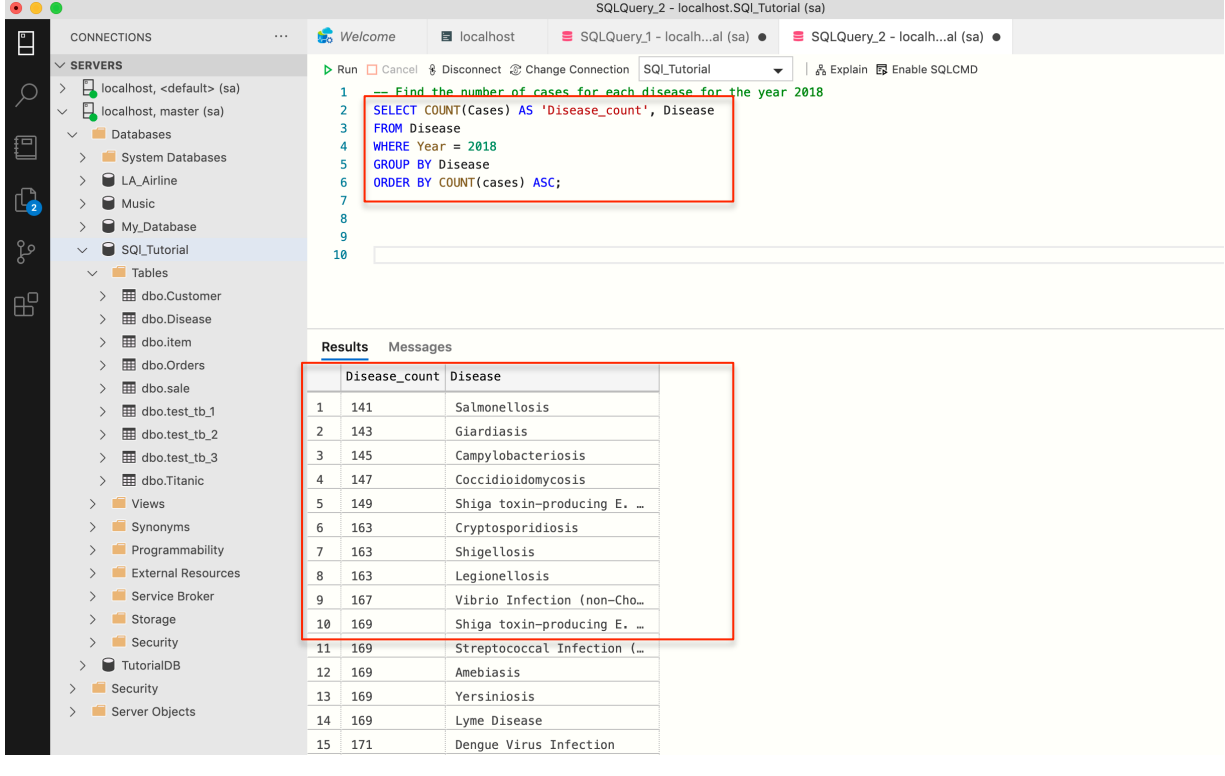
- Sorting results (in ascending or descending order) on single or multiple columns using **ORDER BY** clause
- In the following example we want to sort **Disease** column in descending order,
- and within that sort **County** column in ascending order, and sort **Year** in descending order→



	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*	–

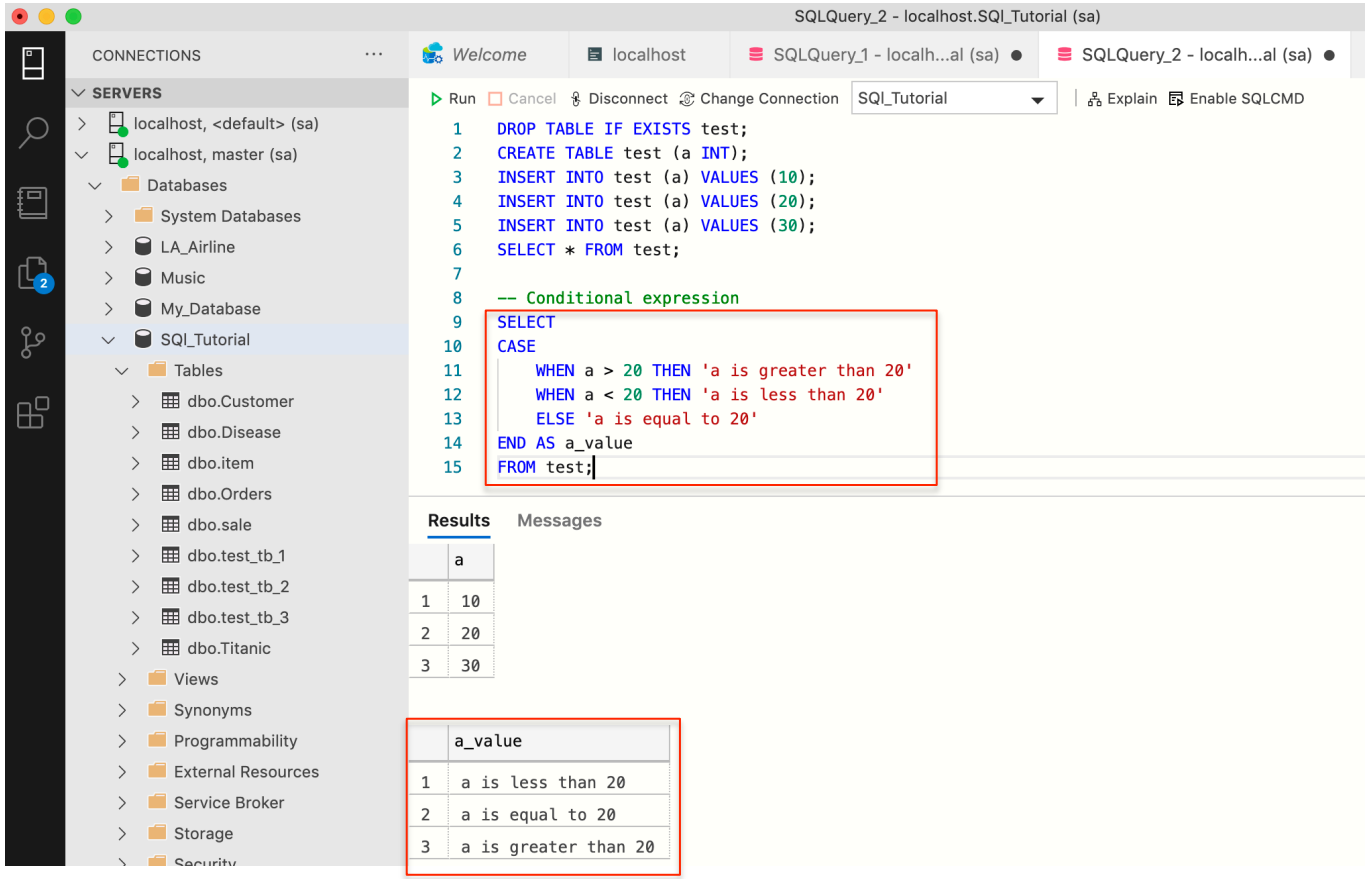
Grouping Data: GROUP BY Statement

- 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** →



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.

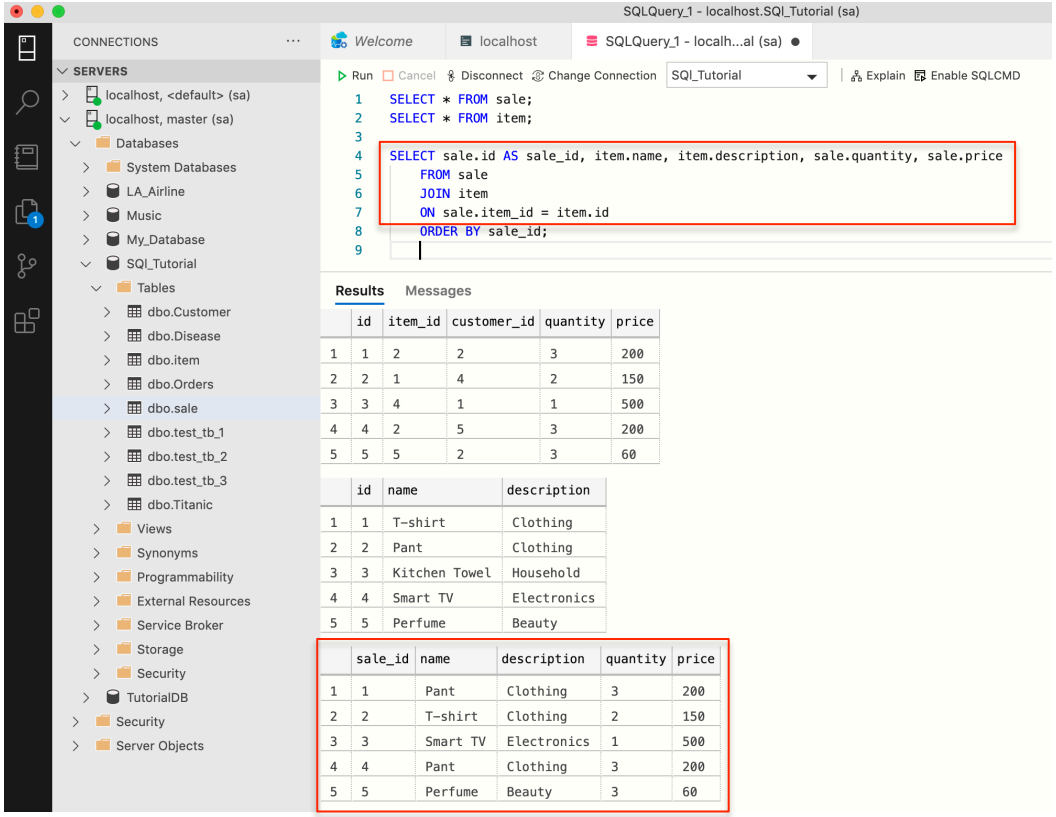


Accessing Related Tables: JOIN Clause

- 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

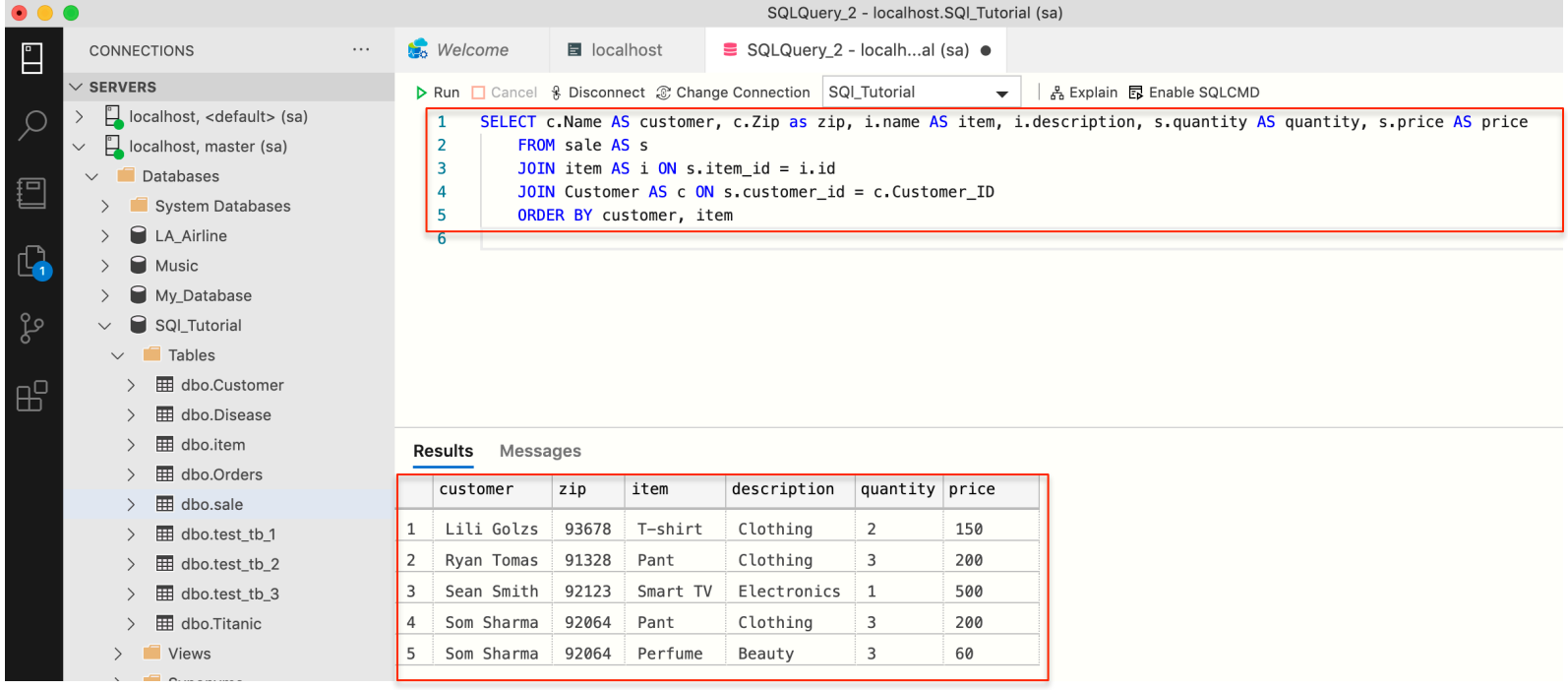
JOIN Clause Example

- 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

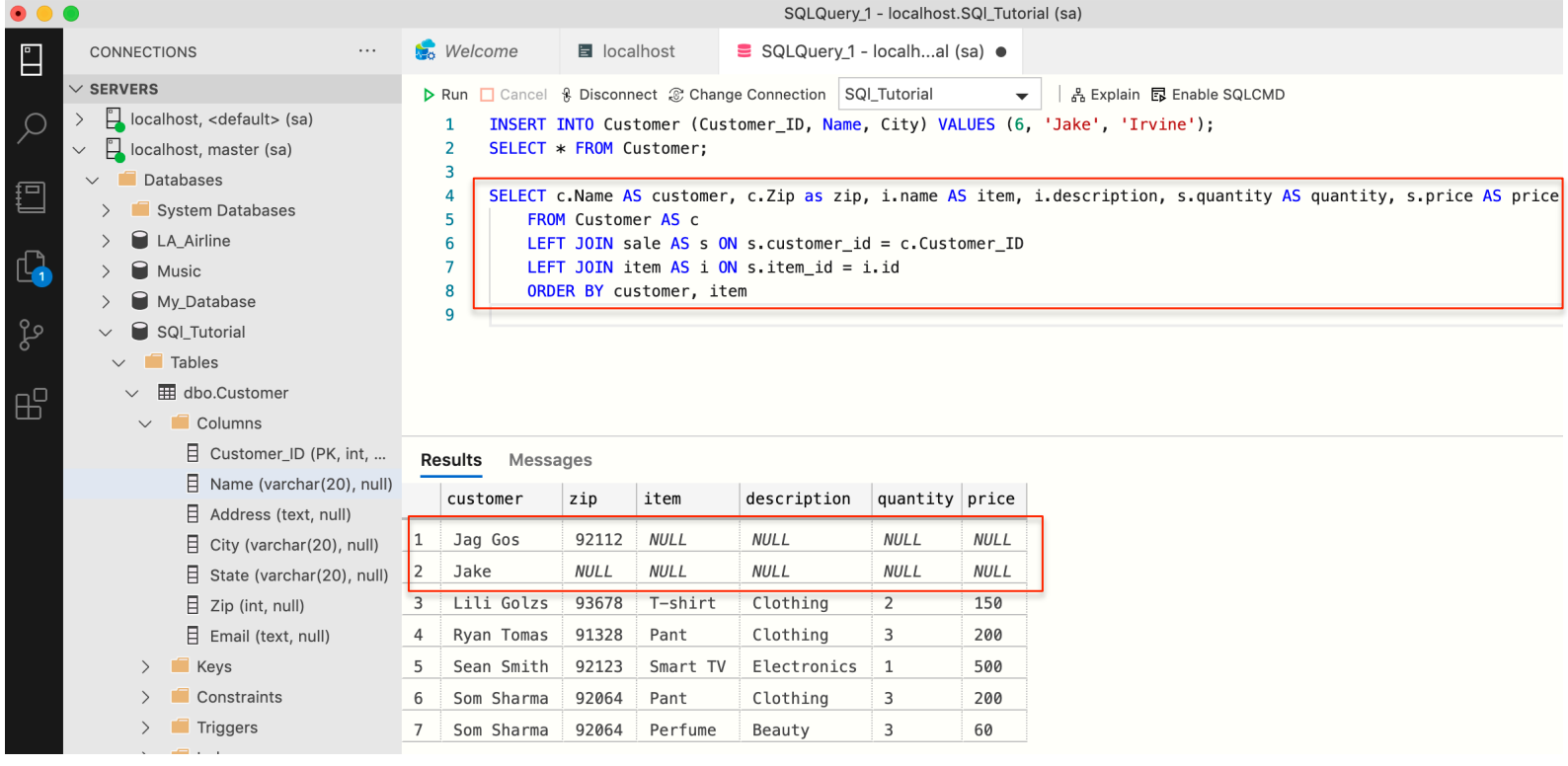


Join Multiple Tables

- 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



LEFT JOIN Example



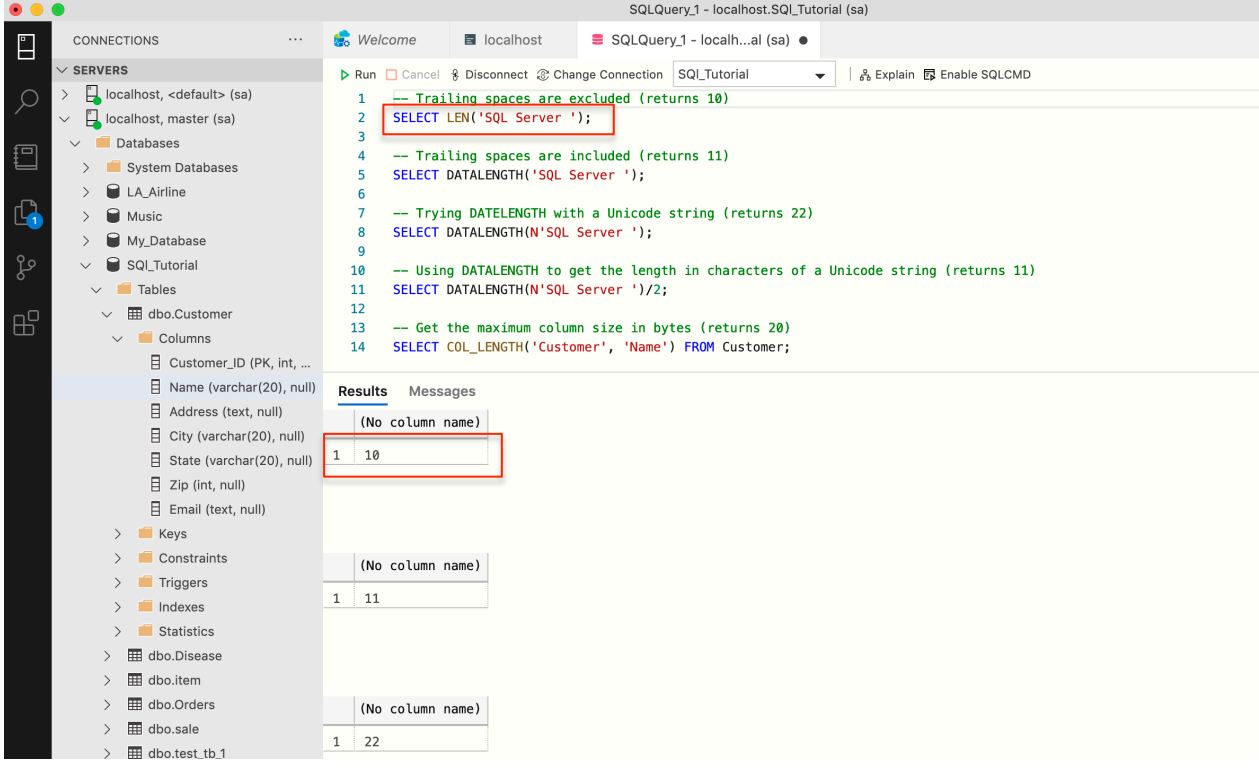
	customer	zip	item	description	quantity	price
1	Jag Gos	92112	*NULL*	*NULL*	*NULL*	*NULL*
2	Jake	*NULL*	*NULL*	*NULL*	*NULL*	*NULL*
3	Lili Golzs	93678	T-shirt	Clothing	2	150
4	Ryan Tomas	91328	Pant	Clothing	3	200
5	Sean Smith	92123	Smart TV	Electronics	1	500
6	Som Sharma	92064	Pant	Clothing	3	200
7	Som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

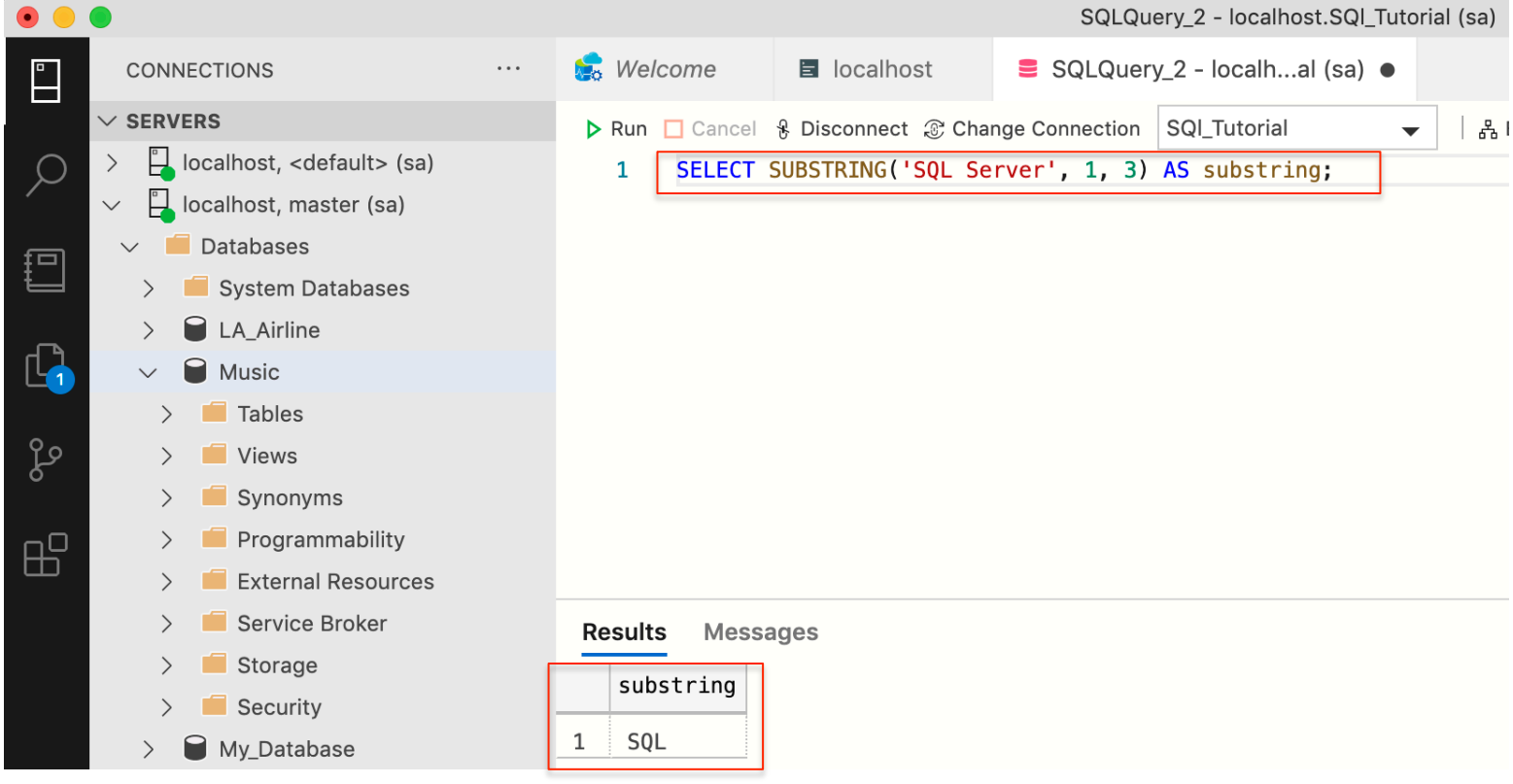
Getting Length Of Strings

- 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



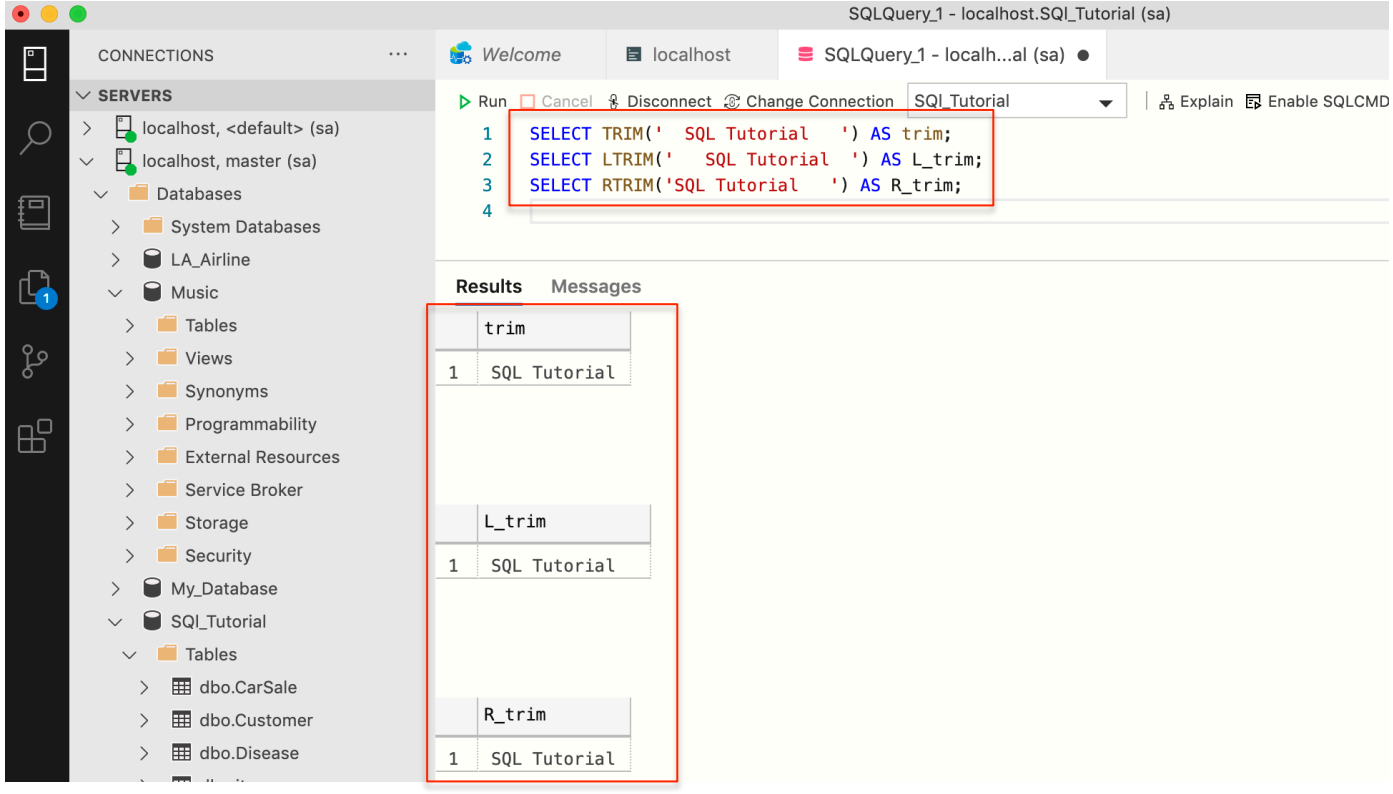
Getting Part Of Strings

- 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



Removing Spaces

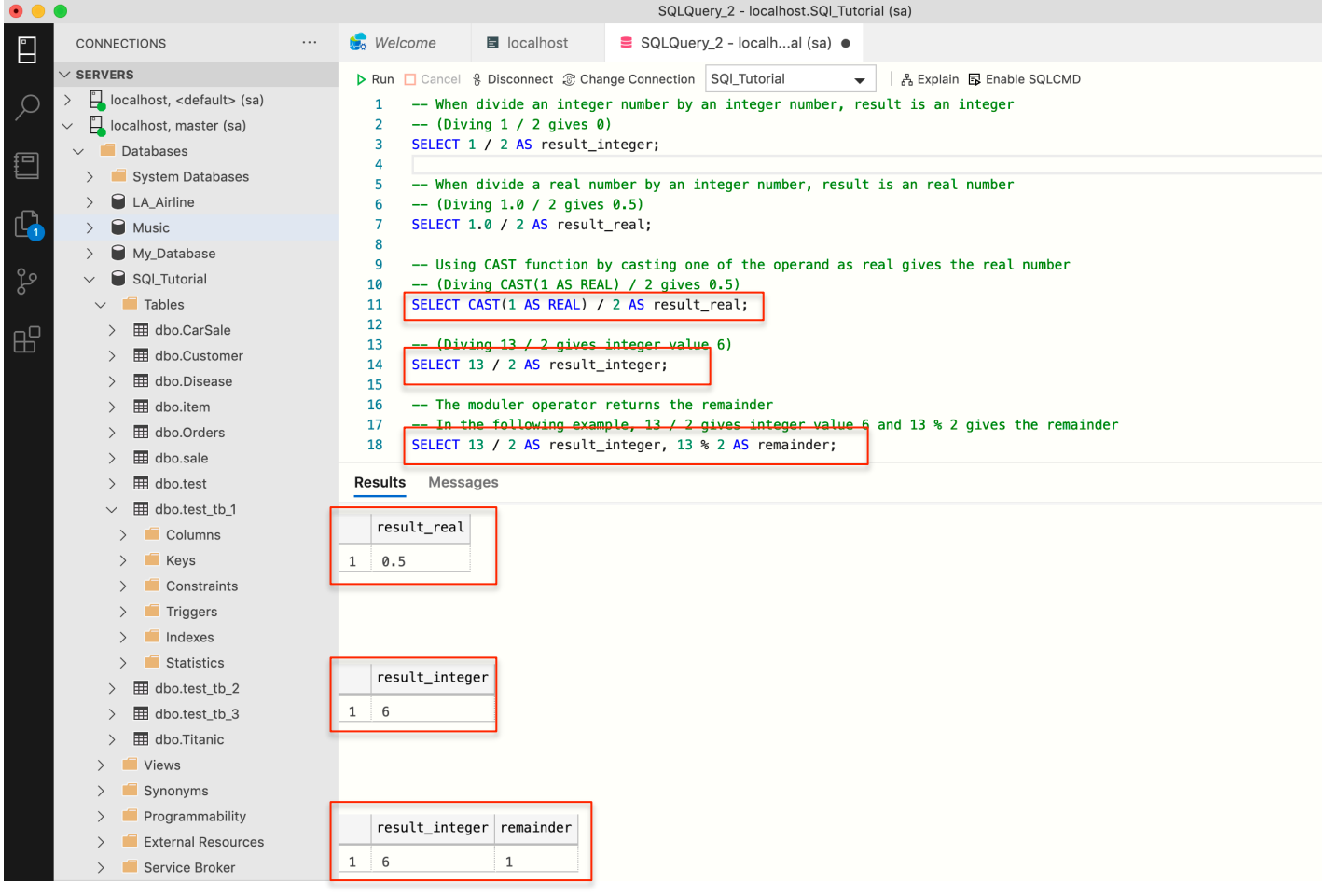
- The function **TRIM**(*string*) removes leading and trailing spaces from a string
- **LTRIM**(*string*) removes leading spaces from a string
- **TRIM**(*string*) removes trailing spaces from a string



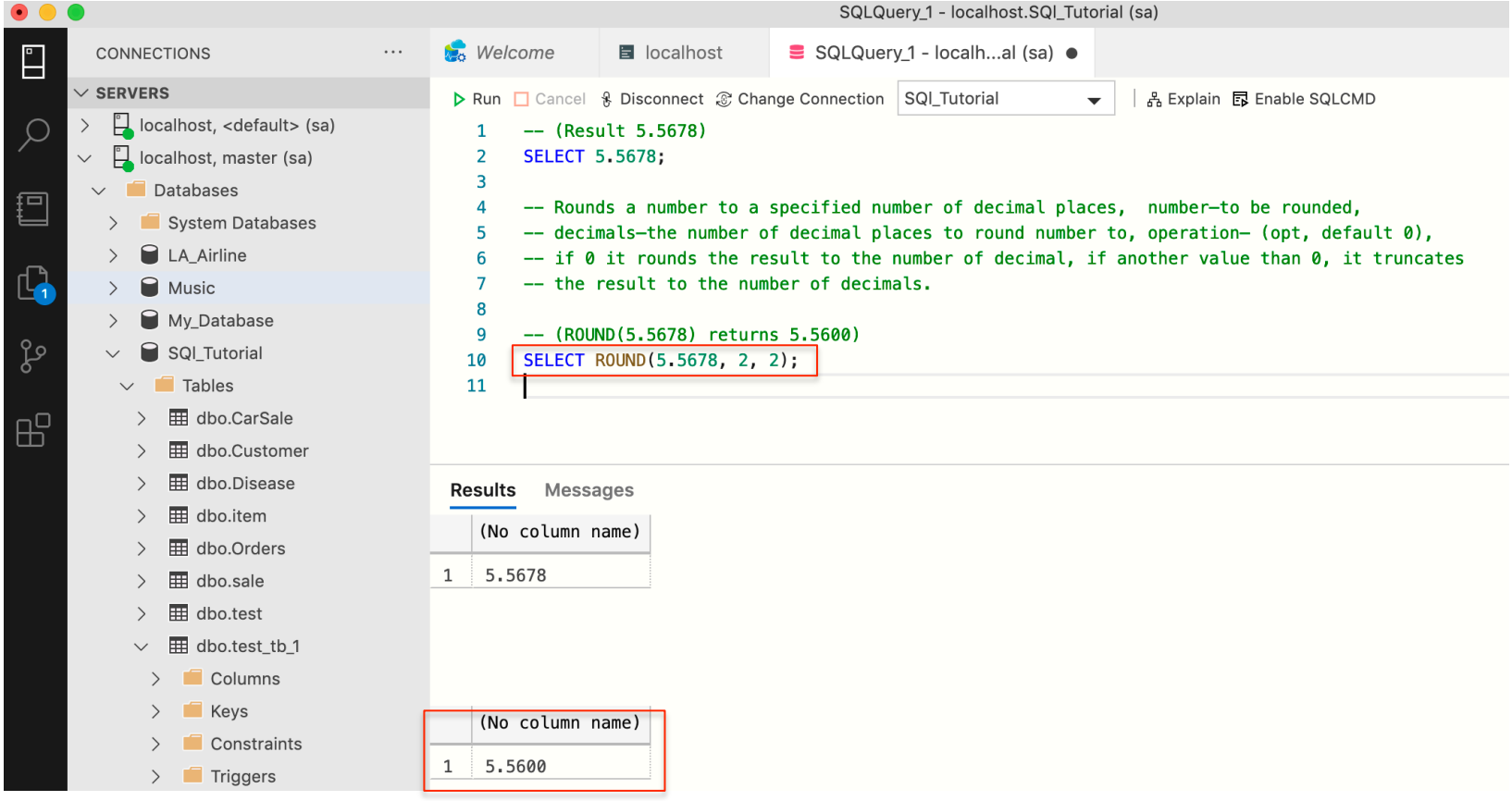
Numeric Functions

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

Integer Division



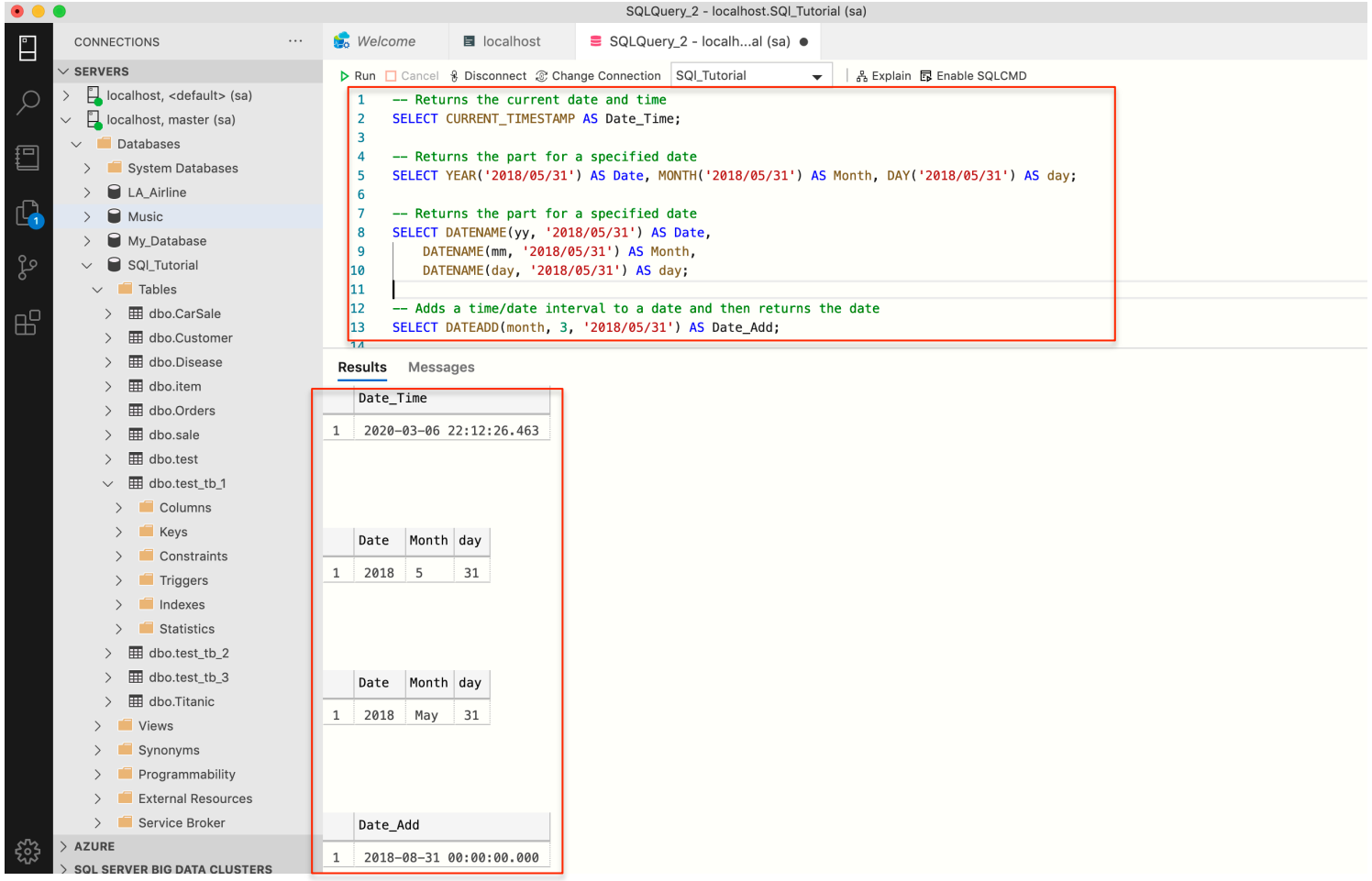
Rounding Numbers



Date And Time Functions

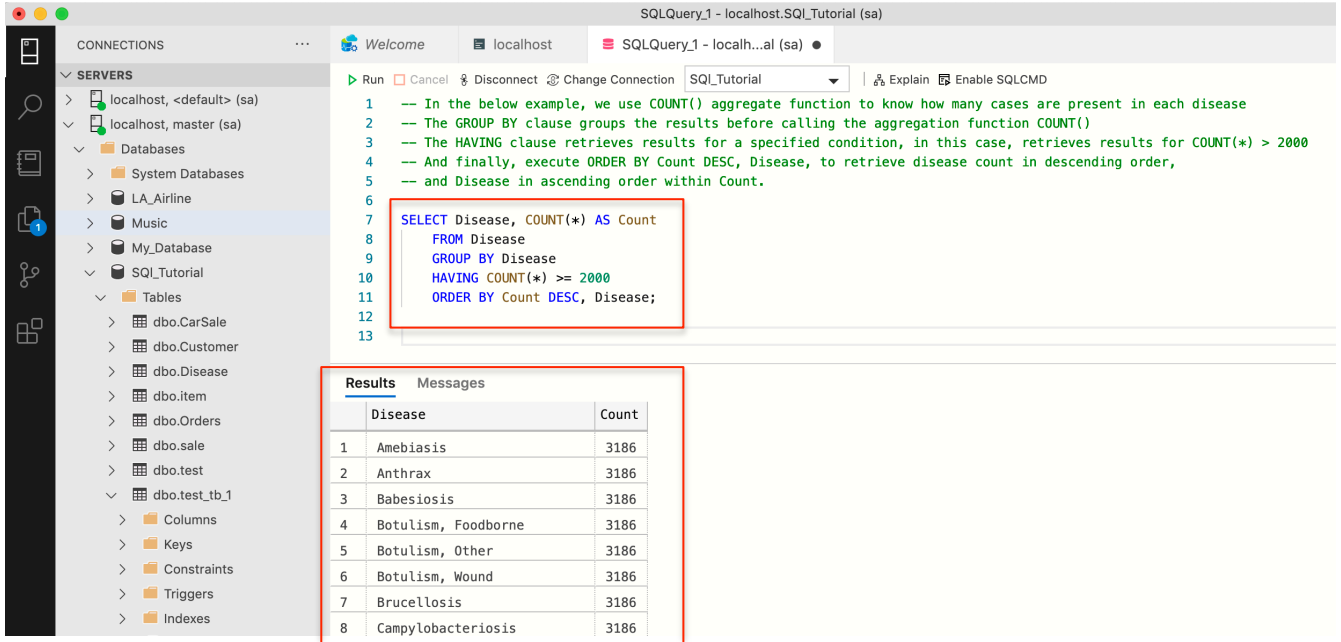
CURRENT_TIMESTAMP:	Returns the current date and time
YEAR(*date*):	Returns the year part for a specified date
MONTH(*date*):	Returns the month part for a specified date
DAY(*date*):	Returns the day part for a specified date
GETDATE():	Returns the current database system date and time (in 'YYYY-MM-DD 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



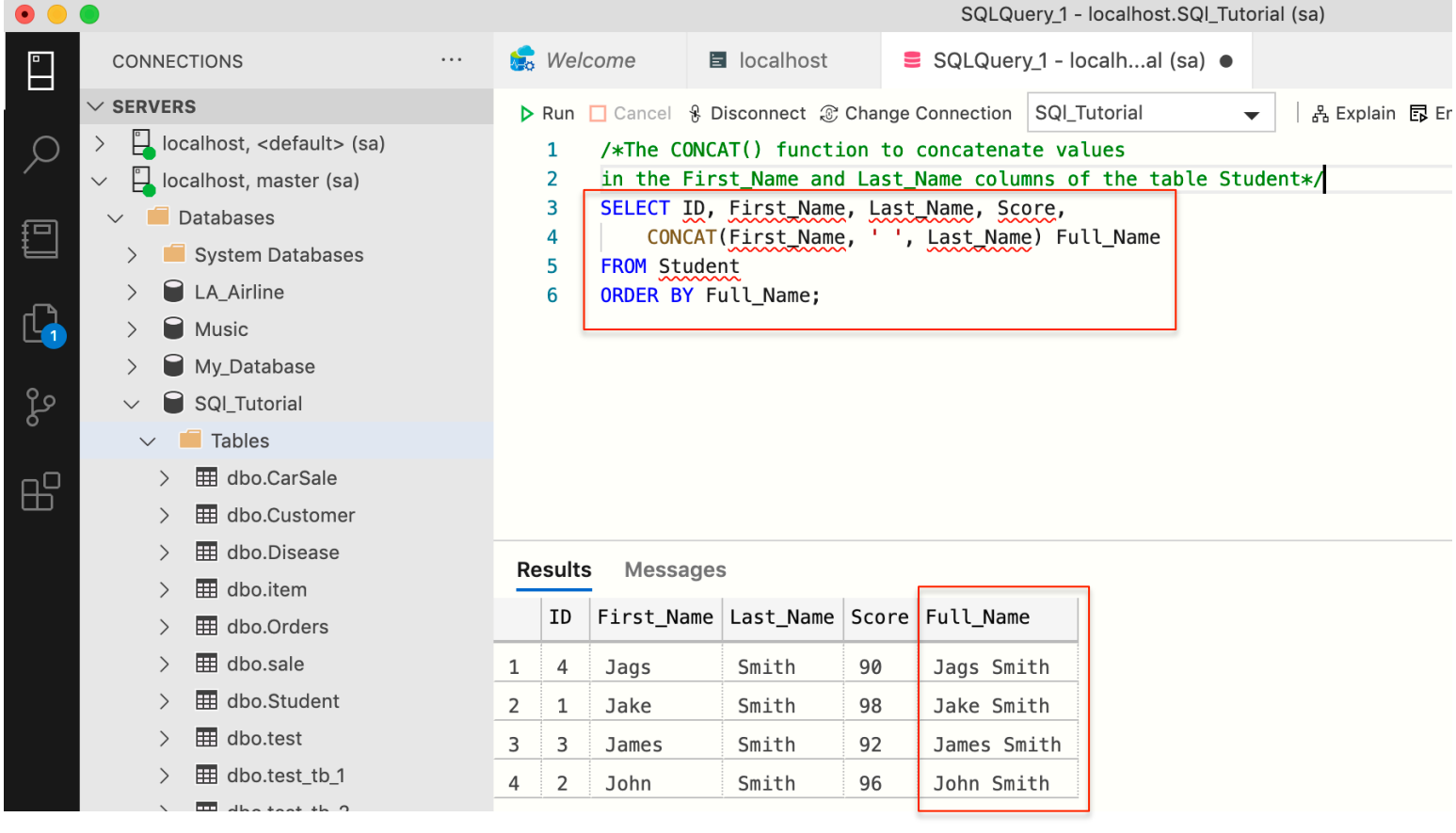
Aggregating Data

- 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**.



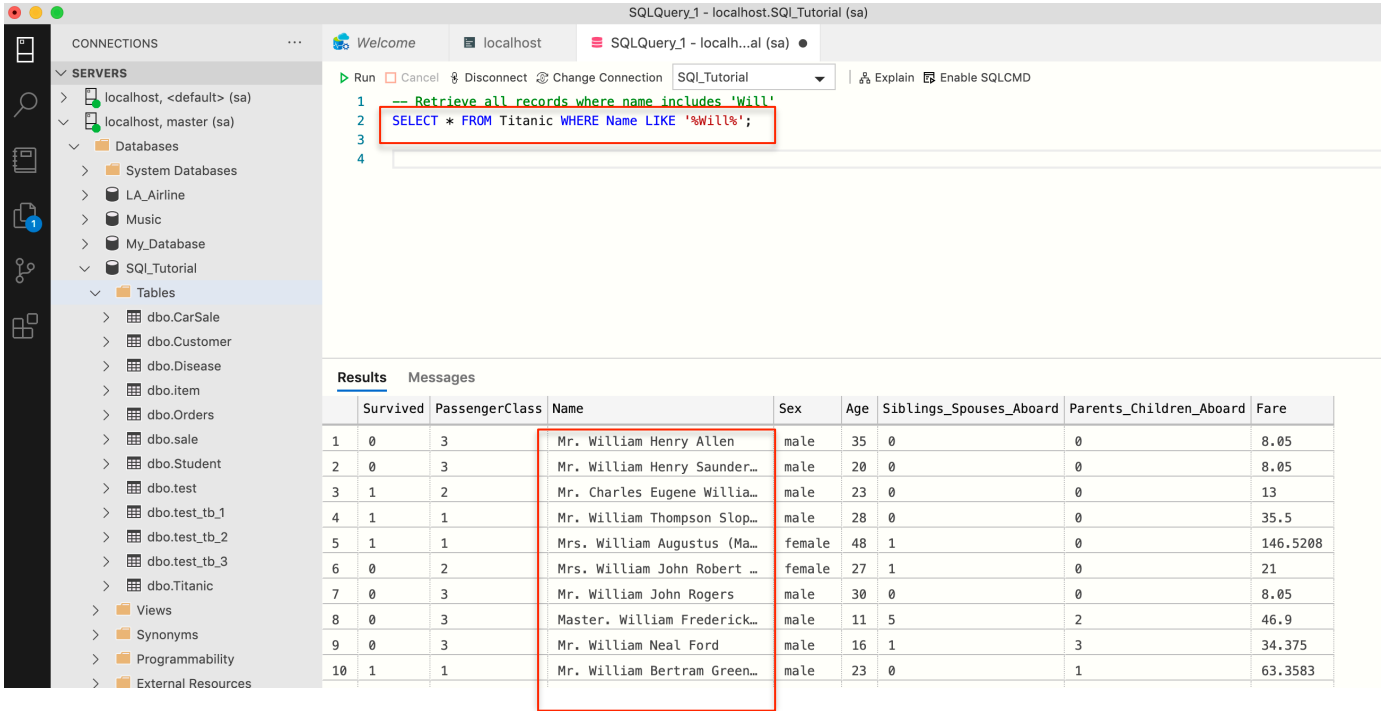
Concatenating Column Data

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



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

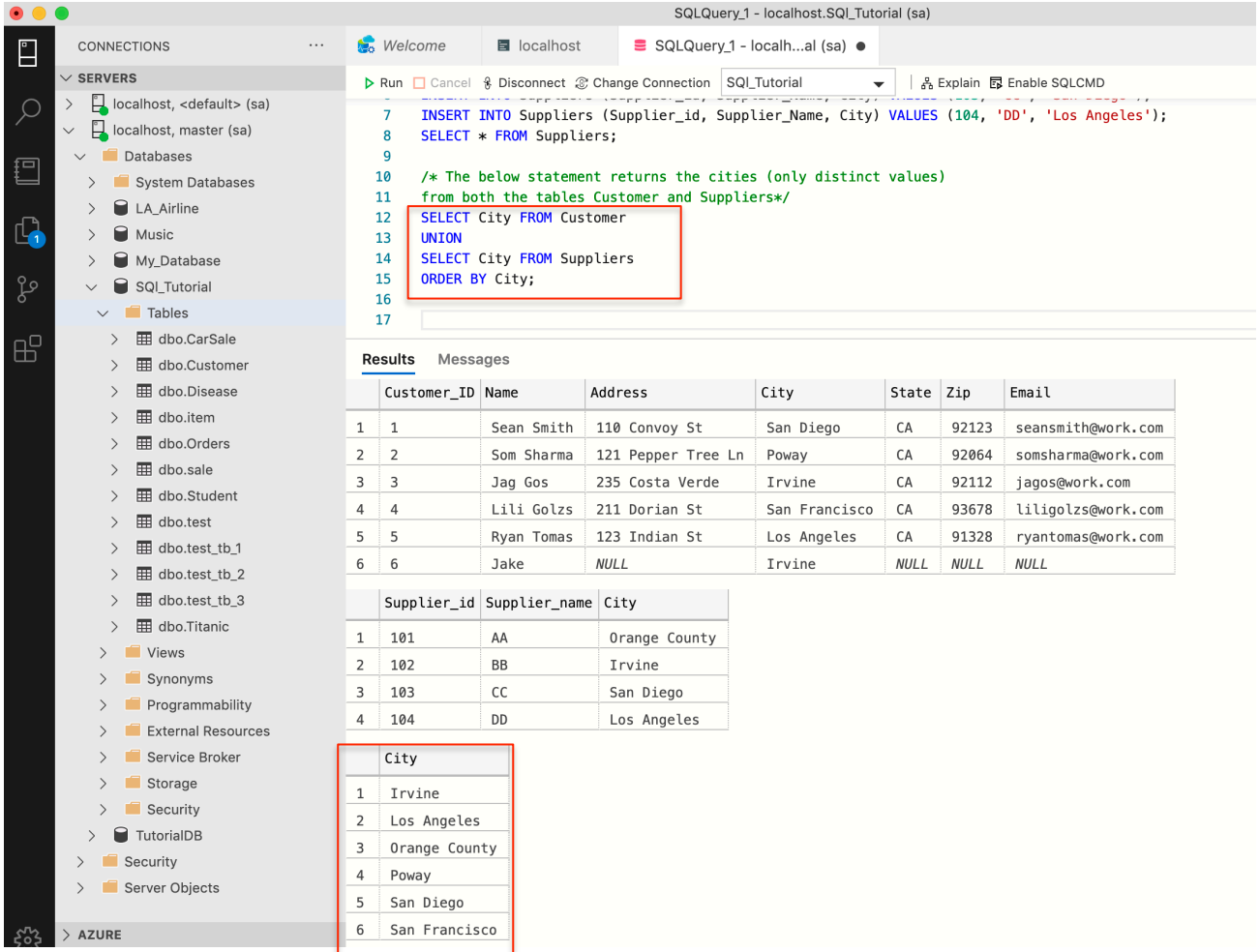


	Survived	PassengerClass	Name	Sex	Age	Siblings_Spouses_Aboard	Parents_Children_Aboard	Fare
1	0	3	Mr. William Henry Allen	male	35	0	0	8.05
2	0	3	Mr. William Henry Saunder…	male	20	0	0	8.05
3	1	2	Mr. Charles Eugene Willia…	male	23	0	0	13
4	1	1	Mr. William Thompson Slop…	male	28	0	0	35.5
5	1	1	Mrs. William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

UNION Operator

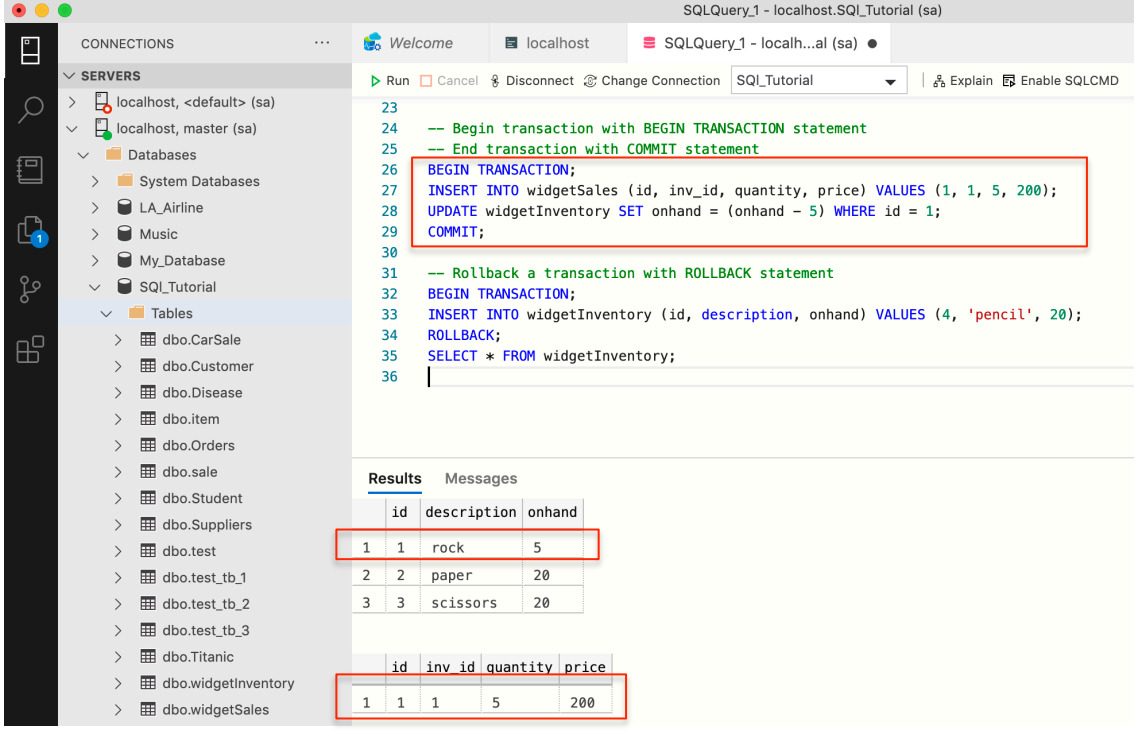
- The UNION operator combines the results of two or more SELECT statements
- Each SELECT statement within UNION must have the same number of columns
- The columns must also have similar data types
- The columns in each SELECT statement must also be in the same order
- UNION operator selects only distinct 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



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



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