SQL (structure query language)

query language => not like normal programming language like c++ , python , ..etc , but its query languages that deal with database

Entity Relationship Diagram (ERD)

Database: A Collection of Related Data.
Database Management System (DBMS): A Software that facilitates the creation and maintenance of a computerized database.
Database System: DBMS with the data itself (Software + Database).

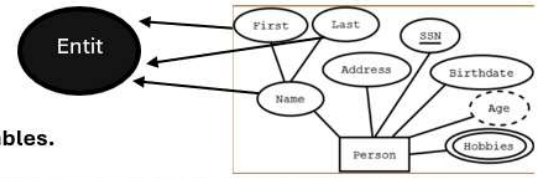
Why We Use Database Rather Than File Based System

	Database	File Based System
Security	Databases make it really hard for unauthorized people to get in, reducing the chance of a breach by more than 95%.	Regular files can be easily accessed by unauthorized people.
Data Redundancy Reduction	With a database, you keep things organized so you don't repeat information too much.	Without a database, you might have the same information stored many times.
Data Consistency	if you change your address in one place, it automatically updates everywhere else it's stored.	if you change your address in one place, it will not automatically updates everywhere else it's stored.
Separation and Isolation of Data	Databases keep different types of data separate, so they're easier to manage. For example, your personal information is kept separate from your shopping history, making it simpler to update one without messing up the other.	data is often mixed together, making it hard to find or change specific information without affecting other parts.

An Entity Relationship Diagram (ERD)

simple picture that shows how different things in a database are related to each other.

- It uses shapes like boxes and lines to show what kinds of information are stored and how they connect.
- It helps people understand the structure of a database without needing to know all the technical details.
- It identifies information required by the business by displaying the relevant Entities and Relationships between them.
- Entity : set of **attributes**
- entities are represented as tables,

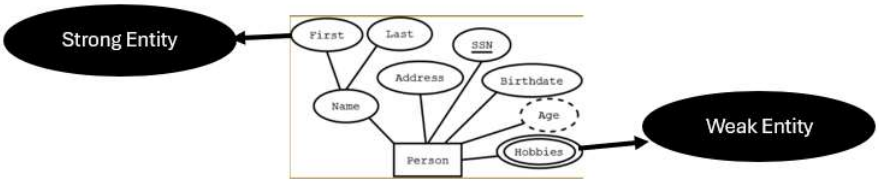


attributes are represented as columns within those tables.

- There should be always relationships between entities and there shouldn't be an entity without any relationship.

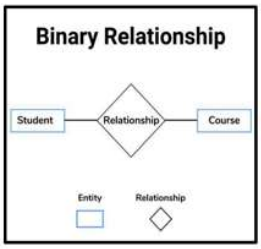
Types Of Attributes

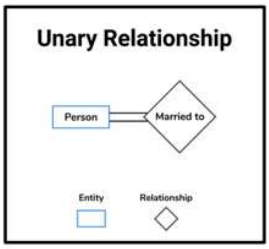
Single or Simple Attributes	These are basic pieces of information. **Example an a person's name or age.**
Multi-Valued Attribute	This is when something can have more than one value, **Example someone's hobbies.**
Composite Attribute	It's when an attribute is made up of smaller parts, **Example a name made up of a first name and a last name.**
Derived Attribute	This is an attribute that can be figured out or calculated from other attributes. **Example: calculate someone's age from their birthdate.**
Candidate Keys	special attributes or combinations of attributes that **uniquely identify** each entity within a database **Example Social Security Numbers (SSNs)**

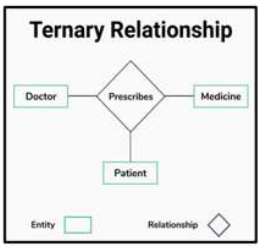


What Is The Different Between Strong Entity and Weak Entity	
strong Entity	**Weak Entity**
It's an entity that can be uniquely identified by its attributes, meaning it doesn't depend on another entity for its existence.	It's an entity that depends on another entity for its existence and cannot be uniquely identified by its own attributes alone. It needs a relationship with a strong entity to make it uniquely identifiable
Example: a person can be uniquely identified by their Social Security Number (SSN).	**Example**: A "Comment" on a blog post, dependent on the "Post" entity for its existence.

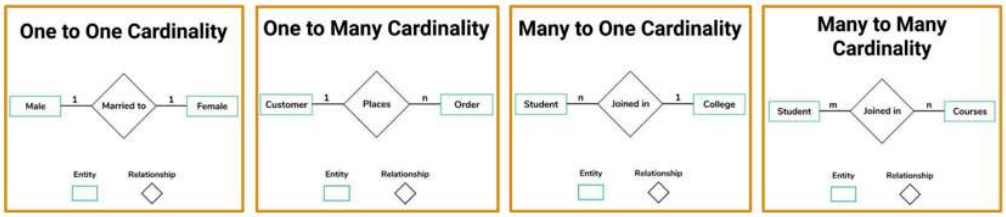
Database Relationships		
Relationships between databases mean how different pieces of information in the database are connected or related to each other.		
Binary Relationship	**Unary Relationship**	**Ternary Relationship**
"Binary" means "two." there are two different entities involved, like a student and a class. Imagine two separate groups, like students and classes. Each student belongs to one class, and each class has many students. So, it's a relationship between these two different groups: students and classes.	"Unary" means "one." In a unary relationship, there is only one entity involved, which relates to itself, like an employee having a relationship with their manager, who is also an employee. Think of it as something relating to itself. For instance, in a company, each employee might have a relationship with their manager, who is also an employee. It's like an entity (employee) is related to itself (another employee).	This is a relationship involving T**hree Different Entities.**
A student (thing 1) belongs to a class (thing 2). Each student attends only one class, and each class has many students.	An employee (thing 1) has a relationship with their manager (also thing 1). Each employee has one manager, who is also an employee.	the relationship between a customer, a product, and an order in an online store, where a customer orders multiple products in a single order.







Cardinality			
It specifies the maximum number of relationships.			
One to One	**One to Many**	**Many to One**	**Many to Many**
Each entity in the first group is related to exactly one entity in the second group, and vice versa.	Each entity in the first group is related to one or more entities in the second group, but each entity in the second group is related to only one entity in the first group.	each entity in the first group is related to only one entity in the second group, but each entity in the second group can be related to one or more entities in the first group.	Many entities in the first group can be related to many entities in the second group, and vice versa.
Each person has one passport, and each passport belongs to one person.	Each country has many cities, but each city belongs to only one country.	Many students attend one school, but each student belongs to only one school.	Many students can enroll in many courses, and each course can have many students.



Participation	
Participation in a database relationship refers to whether each entity in a relationship is required to have a corresponding entity in the related entity set or not.	
Total Participation	**Partial Participation**
Every entity in one set must participate in the relationship	Entities in one set may or may not participate in the relationship
every student must be enrolled in at least one class.	means a student might be enrolled in zero, one, or multiple classes.

For each relationship, We need to identify:

Tables:

1. Table is responsible for storing data in the database. Database tables consist of rows(records) and columns.
2. Every Table should have a primary key (maybe one column or more than one column).

Primary Key:

1. Unique Key. (All values in the column are different).
2. Not Null. (Enforces the field to have a value).

Mapping Entity Relationship Diagram ERD to Tables		
Mapping Strong Entity organizing real-world things into tables in a database, with each thing having its own row and each detail about it having its own column.	Single or Separate attributes	These are individual pieces of information stored separately. For example, in a table for students, you might have separate columns for "First Name," "Last Name," and "Age." Each piece of information has its own column.
	Composite Attributes	Composite attributes are made up of multiple parts, but they're stored as a single piece of information. For instance, a full name is composed of a first name and a last name. Instead of storing the first name and last name separately, you'd have one column for the full name.
	Primary key	We choose a primary key from a unique combination of attributes in the table. If there are multiple options, we pick the one that takes up less space in storage.
	Multi valued attribute	If an entity has a multi-valued attribute, like hobbies, we create a separate table for it. Each hobby becomes its own row in this table, and it's linked back to the person through their primary key. This way, we can track multiple hobbies for each person without repeating information in the main table.
	Derived Attribute	like age calculated from a birthdate, are usually not stored in the database. Instead, we calculate them each time we need them. But if we use a derived attribute frequently, we might decide to store it to improve performance.
Mapping Weak Entity making a table for something that needs another thing to make sense.	imagine you have something that depends on something else to be identified. For example, in a hotel booking system, a booking depends on a room. So, you create a table for bookings, but since each booking depends on a specific room, you include the room's information in the booking table. This way, each booking is uniquely identified within the context of the room it's for.	
knowing how to map strong and weak entities enables us to design databases that are well-structured, maintainable, and performant, which is crucial for effective data management and application development.		

Mapping Entity Relationship Diagram ERD to Tables	
Mapping One to Many Relationship	Adding PK of one side as a FK in Many side (PK of customer as FK in order) Order(**order_id**, order_date, customer_id) • order_id: This column stores a unique identifier for each order. • order_date: This column stores the date when the order was made. • customer_id: This column stores the identifier of the customer who placed the order. This column serves as a foreign key, linking each order to the customer who made it.
Mapping Many to Many Relationship	• We take the PK for the participating Entities and make them as FK in a new table representing this relationship. • The combination of both of FKs will be the PK of this new table. • In case there is an attribute on the Relationship itself, it will be added to this table alongside with both FKs. joinedInTable(student_id, course_id, joining_date).
Mapping One to One Relationship	♣ It depends on the Participation Type. May: Indicates that a relationship between two entities may or may not exist. Must: Indicates that a relationship between two entities must exist. Must Must: relationship between two entities is absolutely mandatory and cannot be ignored or omitted under any circumstances.
Mapping Ternary Relationship	Mapping a ternary relationship means creating a table to represent connections between three different entities in a database.

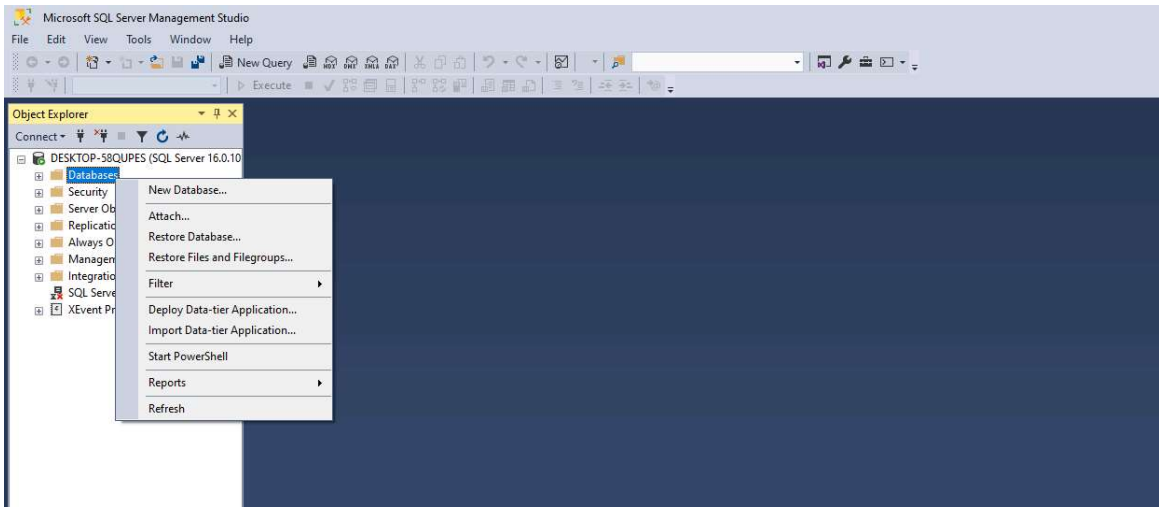
Data Definition Language (DDL):

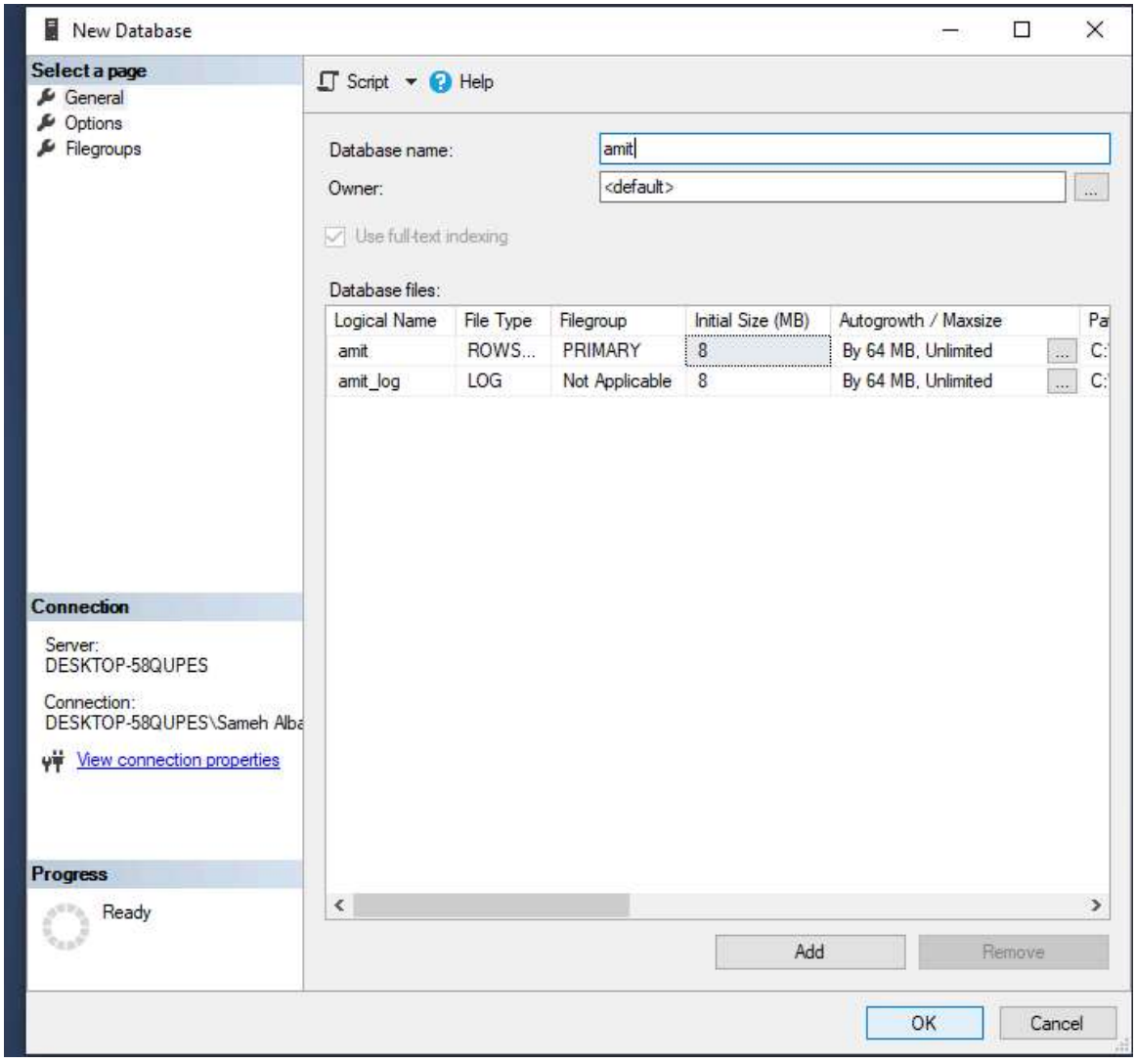
it's used to create and modify the structure of database objects in a database.

1. create table or database

create database

open sql server maangemnt studio

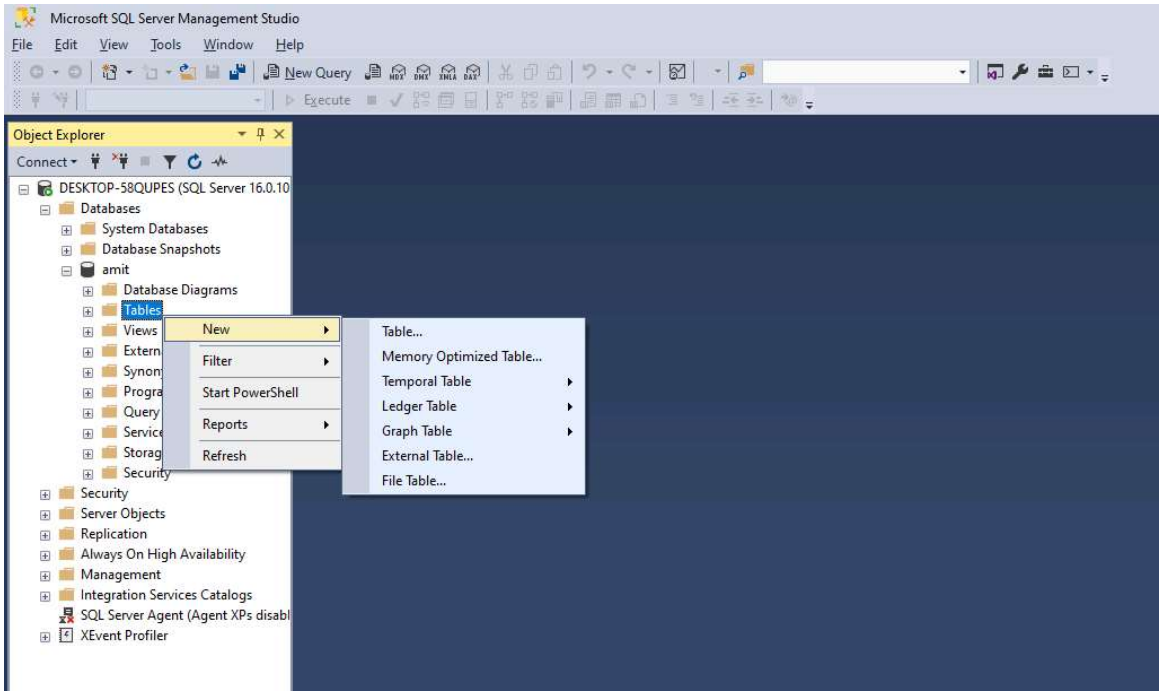


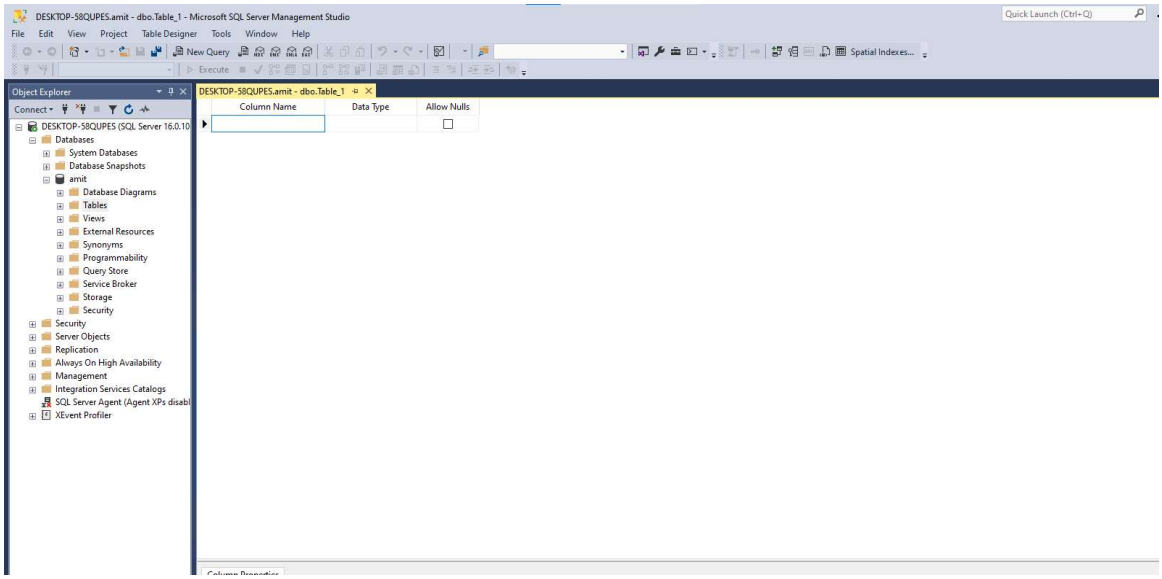


or

```
CREATE DATABASE amit;
```

create table





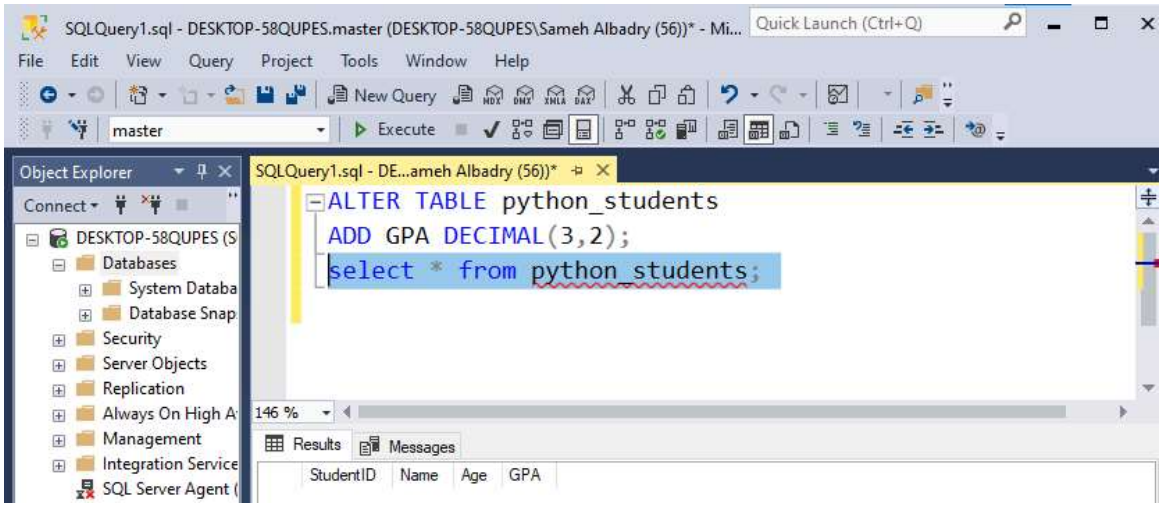
or

```
CREATE TABLE Students (
    StudentID INT PRIMARY KEY,
    Name VARCHAR(50),
    Age INT
);
```

2. alter

It is used to add, delete, modify or rename columns in an existing table.

1. add column:

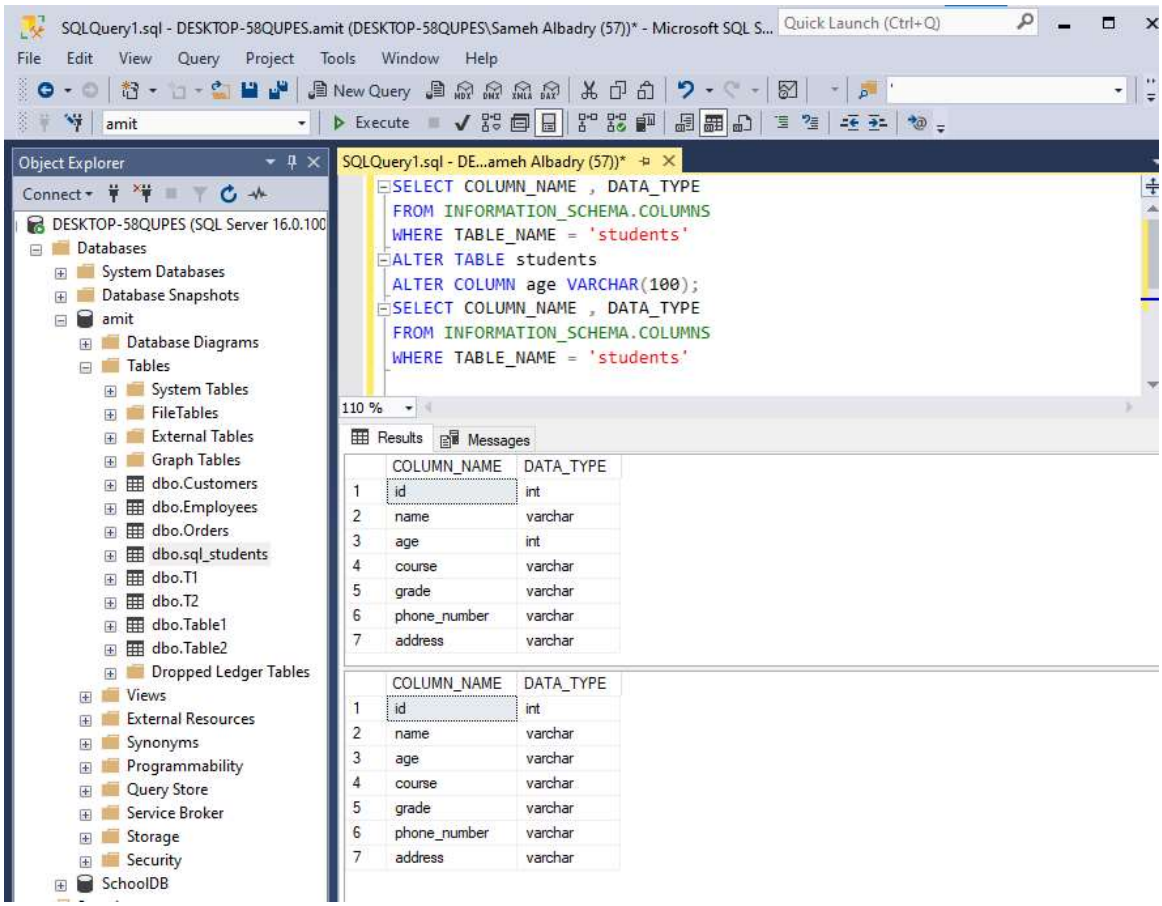


2. add row:

```sql
CREATE TABLE students (
    id INT PRIMARY KEY IDENTITY,
    name VARCHAR(50),
    age INT,
    course VARCHAR(50),
    grade VARCHAR(2),
    phone_number VARCHAR(15),
    address VARCHAR(100)
);
-- Insert sample data into students table (if needed)
INSERT INTO students (name, age, course, grade, phone_number, address)
VALUES ('amit', 20, 'power bi', 'B+', '011111111', 'xxxxxxxxxxx'),
       ('learning', 30, 'python', 'A+', '022222222', 'yyyyyyyyyyy'),
       ('hello', 20, 'sql', 'C-', '03333333333', 'zzzzzzzzzzzzz');
```

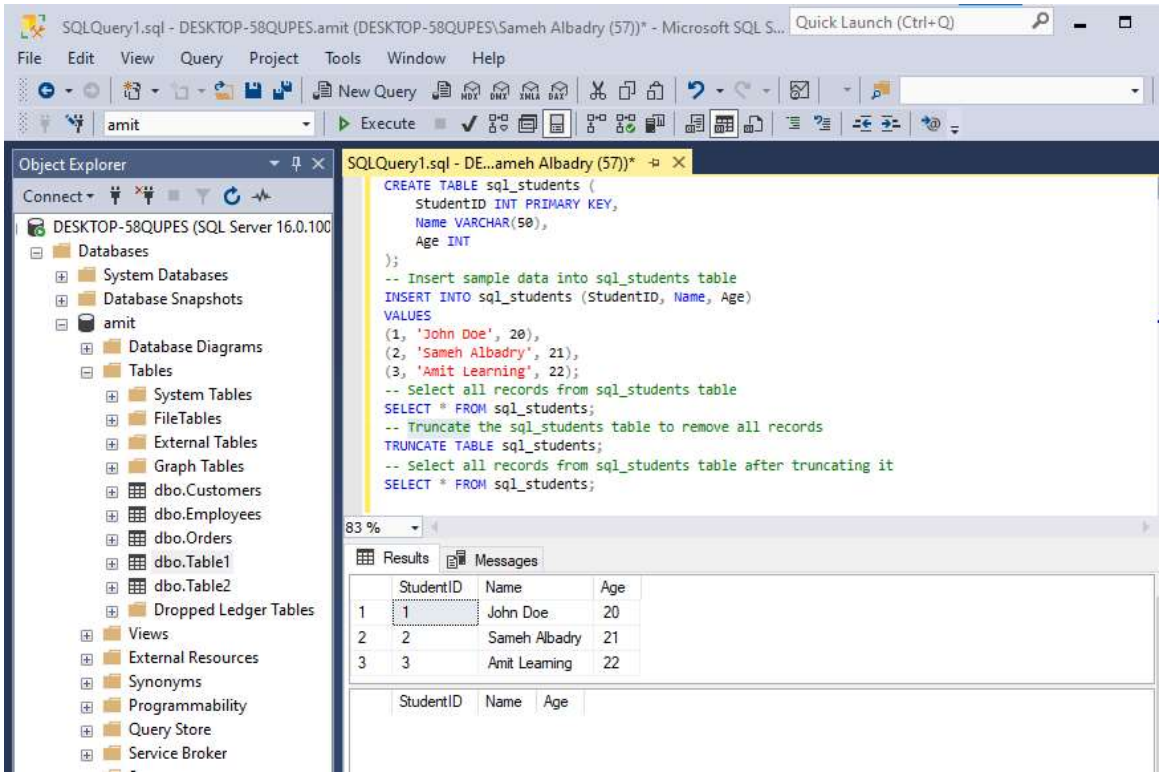
3. Modify Column (Change data type):

It’s used to change the data type of a column.



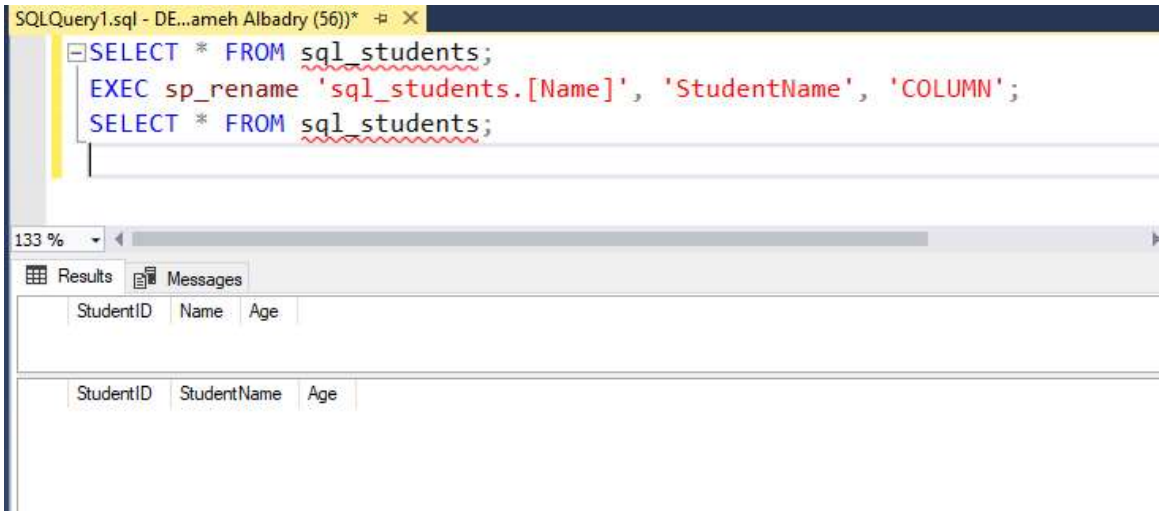
4. truncate

used to delete the data inside a table, but not the table itself.



5. rename

It's used to RENAME an existing COLUMN in a database.



6. delete table:

It's used to delete a column in the table.

```
DROP TABLE python_students;
```

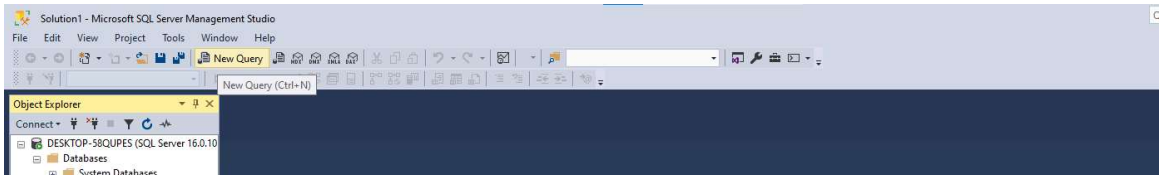
7. delete database:

```
DROP DATABASE python_students;
```

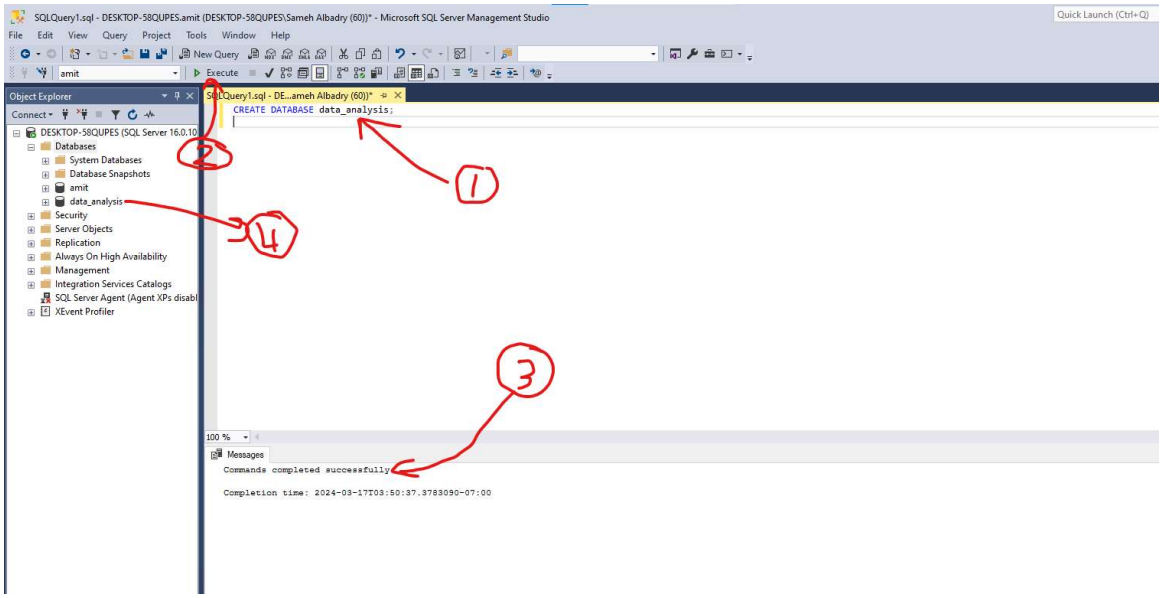
what is datatypes in database ?

Integer	Int	1,10,-5,100
	SMALLINT	50,20,-10
	BIGINT	10000000,20000000
Decimal/Numeric	DECIMAL	3.14, 10.5, -0.75
	NUMERIC	123.456, -987.654
	FLOAT	3.14159, -0.007
	REAL	1.23, -4.56
	DOUBLE	12.345, -67.89
Character Strings	CHAR	'John', 'Alice', 'Bob'(without fixed-length strings)
	VARCHAR	'OpenAI', 'SQL', 'Database' (fixed-length strings)
	TEXT	'Lorem ipsum dolor sit amet', 'Hello World'
Date and Time	DATE	'2024-03-16', '1990-05-25'
	TIME	'13:30:00', '23:59:59'
	TIMESTAMP	'2024-03-16 13:30:00', '1990-05-25 08:15:00'
Boolean	BOOLEAN	True , False
	BIT	0,1

create database and table using query language



to create database



create table inside this database

step 1: select database you want to create table

```
USE data_analysis;
```

step 2: create table

```
CREATE TABLE students (
    id INT PRIMARY KEY IDENTITY,
    name VARCHAR(50),
    age INT,
    course VARCHAR(50),
    grade VARCHAR(2),
    phone_number VARCHAR(15),
    address VARCHAR(100)
);
```

what is primary key

- The primary key is like a unique ID for each row in a table.
- It makes sure that each row has its own special identity.
- It helps to find and organize data quickly.
- It's like a fingerprint for each row, ensuring it's different from all others.

what is Foreign key

- The foreign key is like a connection between two tables in a database.
- It helps one table understand and link to another table.
- It's like a bridge that connects information from one table to another.
- It ensures that the data in one table can relate to the data in another table.

Data Manipulation Lnaguage (DML):

1. insert

It's used to insert a record in the table.

```
SET IDENTITY_INSERT students ON;
INSERT INTO students (id, name, age, course, grade, phone_number, address)
VALUES (1, 'amit', 20, 'powerbi', 'B+', '0111111111', 'xxxxxxxxxx');
```

Results Messages

	id	name	age	course	grade	phone_number	address
1	1	amit	20	powerbi	B+	0111111111	xxxxxxxxxx

```
);
*/
/*
SET IDENTITY_INSERT students ON;
INSERT INTO students (id, name, age, course, grade, phone_number, address)
VALUES (1, 'amit', 20, 'powerbi', 'B+', '0111111111', 'xxxxxxxxxx');
*/
INSERT INTO students (id, name, age, course, grade, phone_number, address)
VALUES (2, 'learning', 30, 'python', 'A+', '022222222', 'yyyyyyyyyyyyy');

INSERT INTO students (id, name, age, course, grade, phone_number, address)
VALUES (3, 'hello', 20, 'sql', 'c-', '03333333333', 'zzzzzzzzzzzzzzz');
```

00 %

Messages

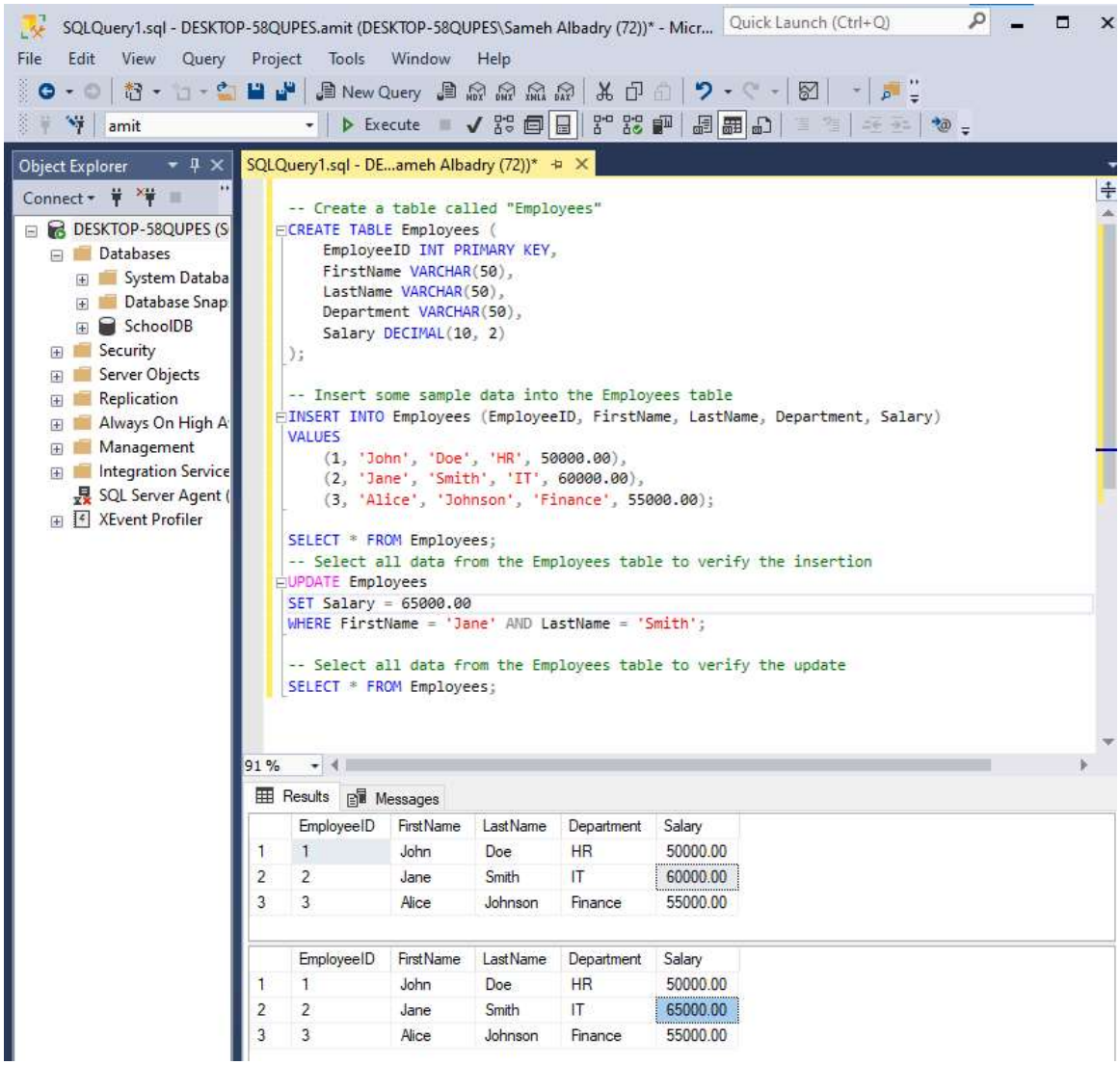
```
(1 row affected)

(1 row affected)

Completion time: 2024-03-17T04:19:06.4764572-07:00
```

Results Messages

	id	name	age	course	grade	phone_number	address
1	1	amit	20	powerbi	B+	0111111111	xxxxxxxxxx
2	2	learning	30	python	A+	022222222	yyyyyyyyyyyyy
3	3	hello	20	sql	c-	03333333333	zzzzzzzzzzzzzzz

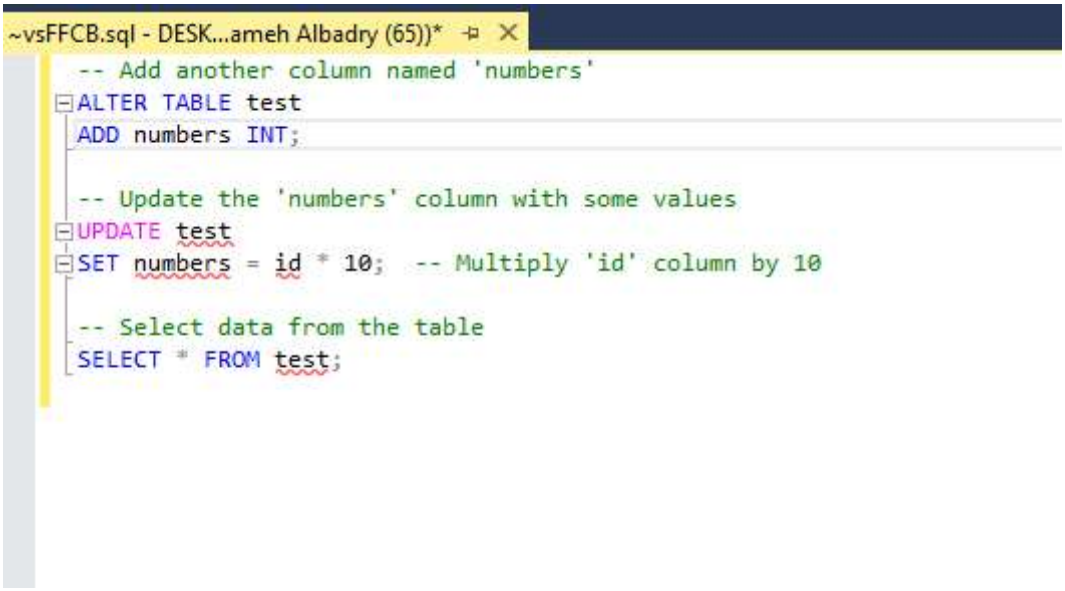
2. update

It's used to modify the existing records in a table.



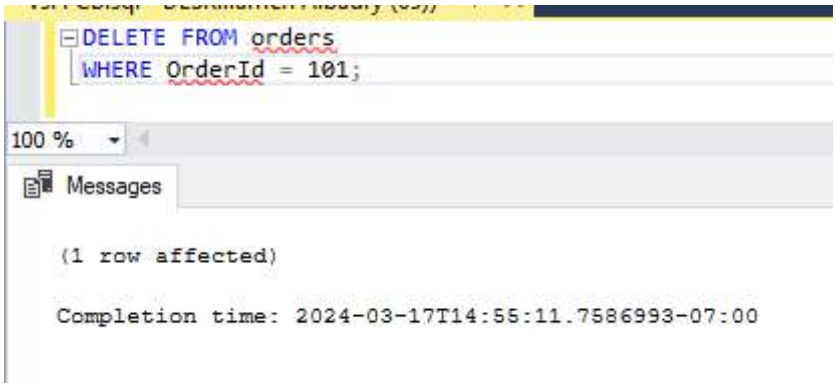
Note: If WHERE Clause isn't written, All records in the table will be updated.

edit your table



3. delete

delete row



delete column

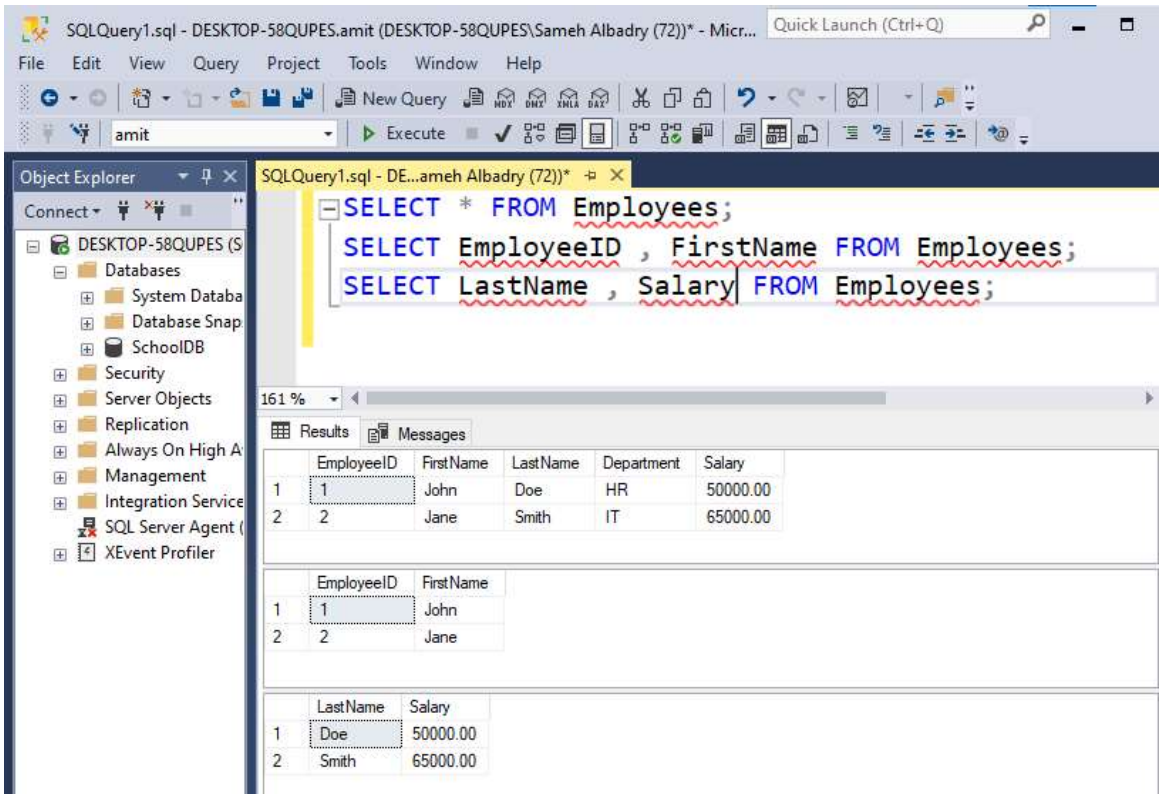
```
ALTER TABLE table_name
DROP COLUMN column_name;
```

Note: If WHERE Clause isn't written, All records in the table will be deleted.

Difference between DELETE and TRUNCATE

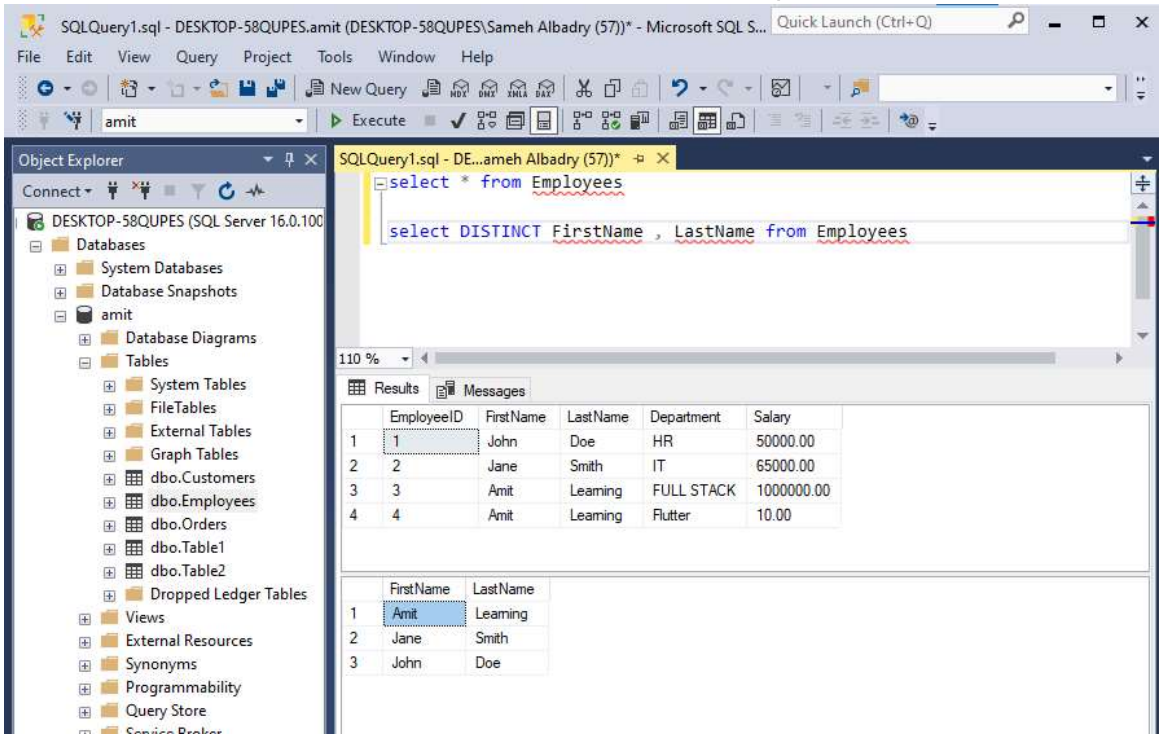
DELETE	TRUNCATE
Syntax: DELETE FROM Table_Name; or DELETE FROM Table_Name WHERE Condition;	Syntax: TRUNCATE TABLE Table_Name;
Delete all the data and keeps the structure of the table (If WHERE Clause isn't used)	Delete all the data and keeps the structure of the table.
Removes the rows one by one so it's slower.	It actually Deletes the table at once and creates a new one. So it's faster.
WHERE Clause can be used	WHERE Clause can't be user(No conditions)

4. select



select distinct

returns values that aren't duplicated for both of the columns together



operations applied in where caluse

Comparison Operators

SQL Comparison Operators

Operator	Description
=	Equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
<>	Not equal to

1. Equality Operator

SELECT * FROM table_name WHERE column1 = value;

2. Inequality Operator

SELECT * FROM table_name WHERE column1 <> value;

3. Comparison Operators

SELECT * FROM table_name WHERE column1 > value;

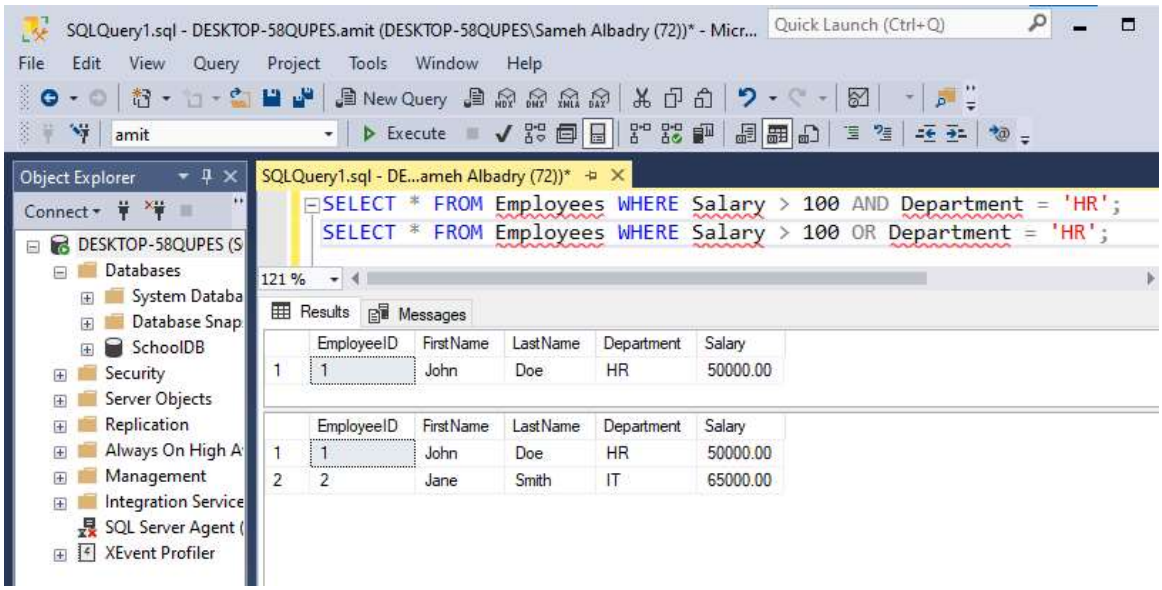
4. IS NULL Operator

SELECT * FROM table_name WHERE column1 IS NULL;

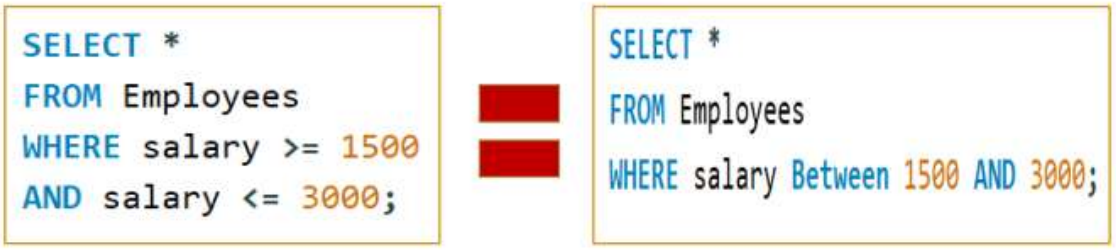
5. Logical Operators (and, or , not)

SELECT * FROM table_name WHERE column1 > value1 AND column2 < value2;

logic operators

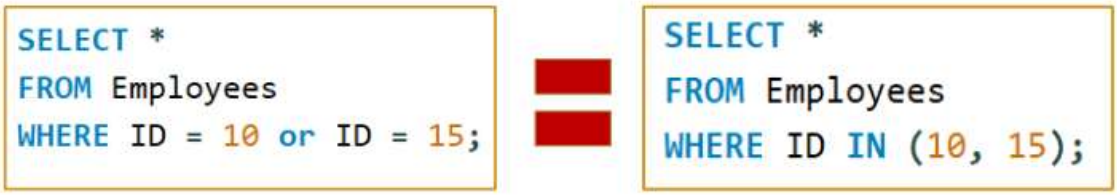


3. BETWEEN Operator



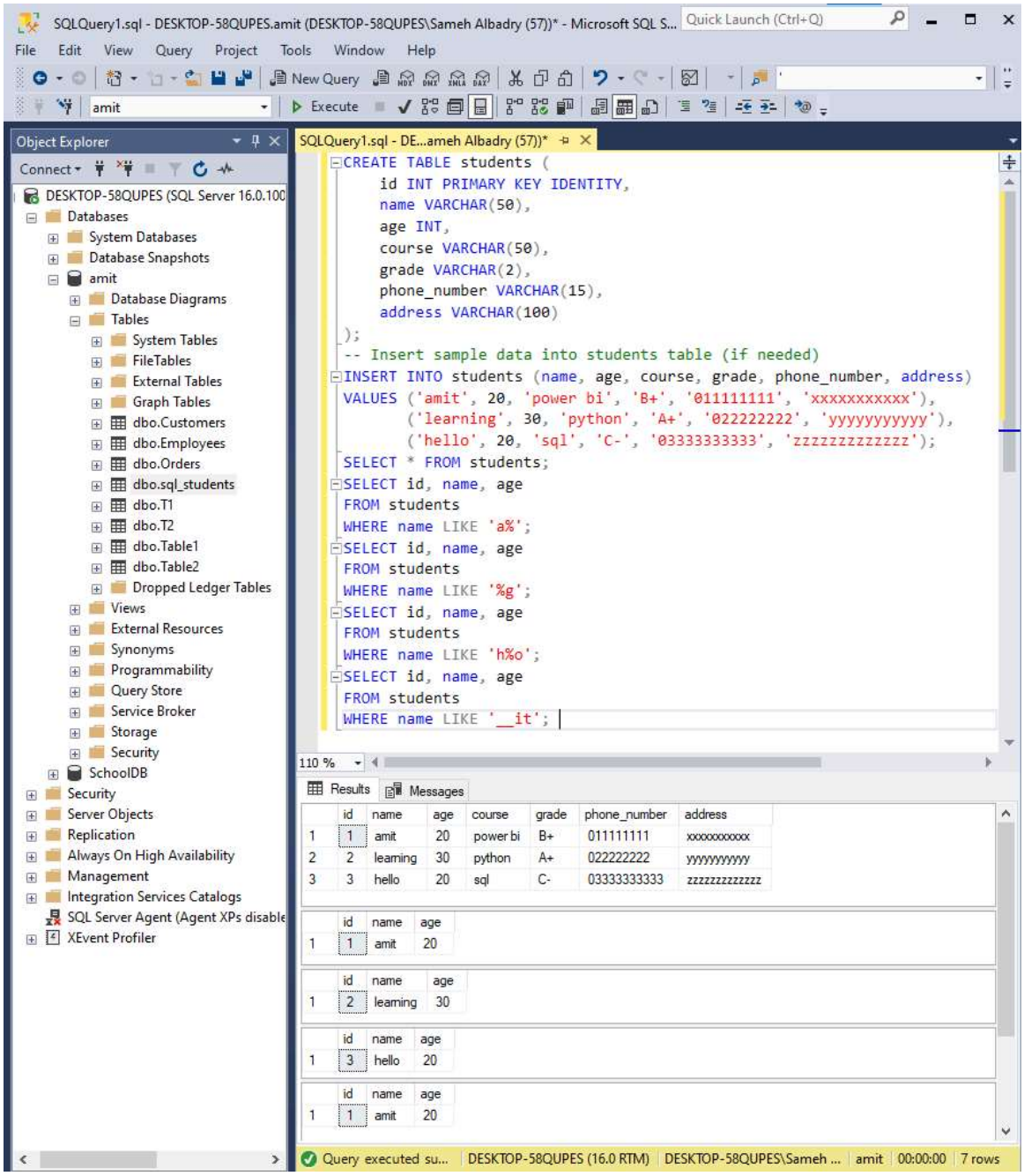
SELECT * FROM table_name WHERE column1 BETWEEN value1 AND value2;

4. IN Operator



SELECT * FROM table_name WHERE column1 IN (value1, value2, value3);

5. LIKE Operator:



6. not

NOT

```
SELECT *
FROM students
WHERE ID NOT IN (13, 100);
```

```
SELECT *
FROM students
WHERE First_Name NOT LIKE '__h%';
```

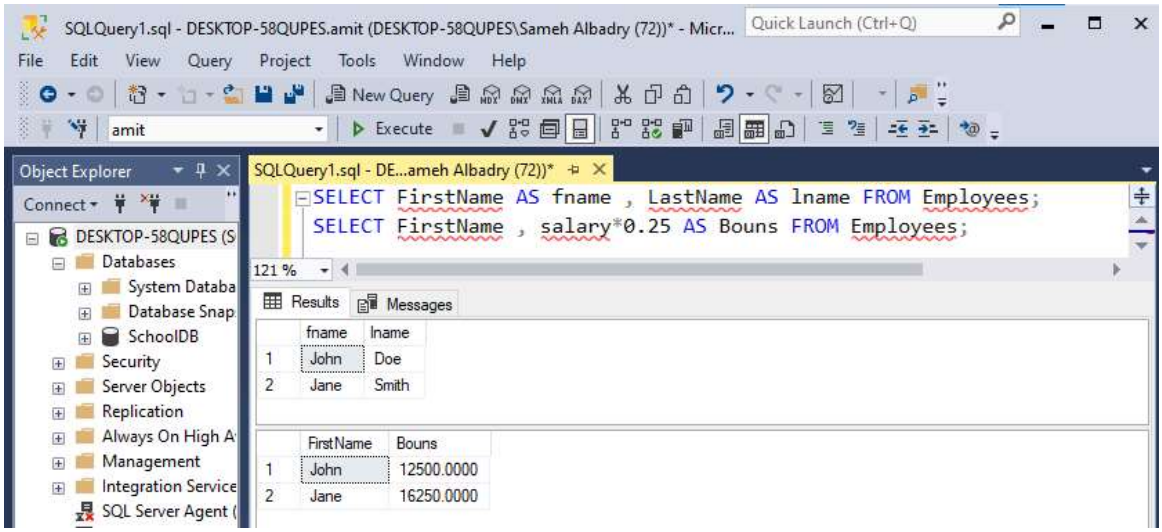
Alias

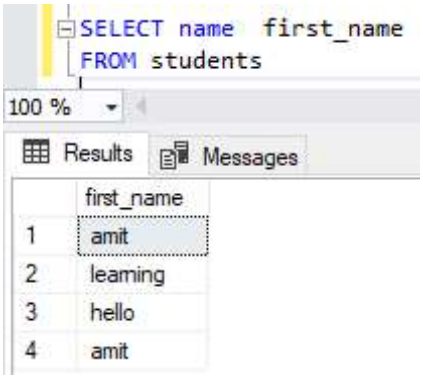
SQL aliases are used to give a table, or a column in a table, a temporary name.

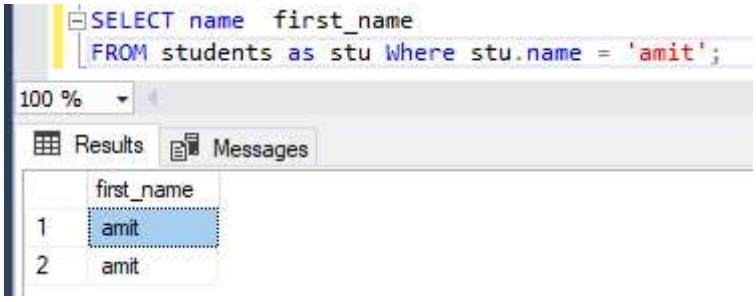
Aliases are often used to make column names more readable.

An alias only exists for the duration of that query.

An alias is created with the `AS` keyword.



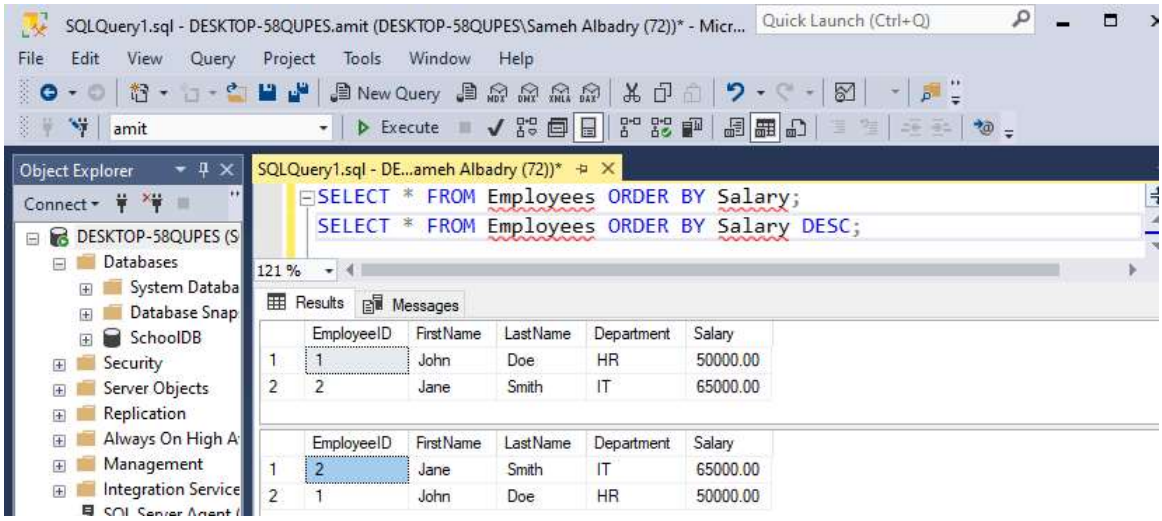




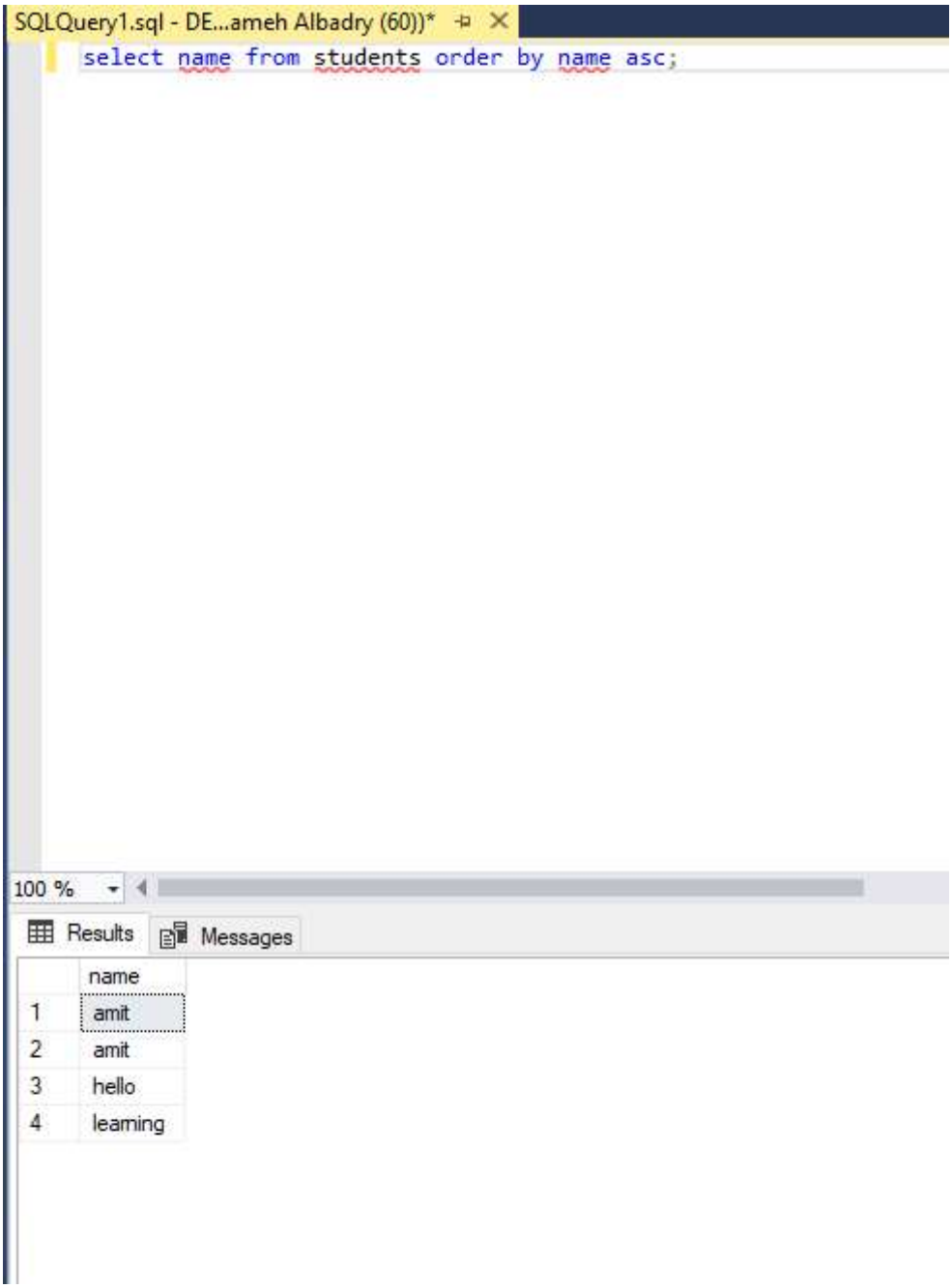
Order BY

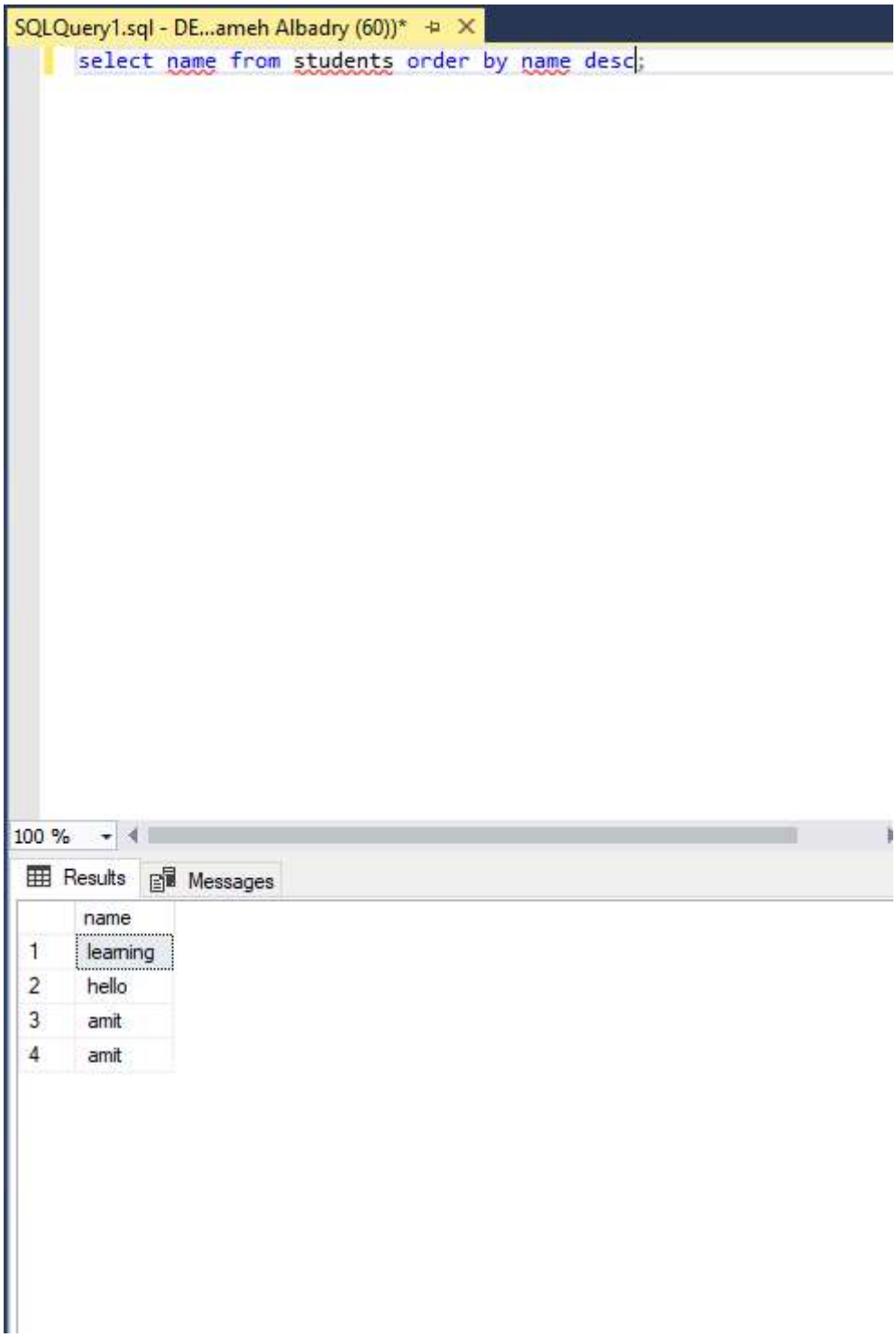
The `ORDER BY` keyword is used to sort the result-set in ascending or descending order.

The `ORDER BY` keyword sorts the records in ascending order by default. To sort the records in descending order, use the `DESC` keyword.



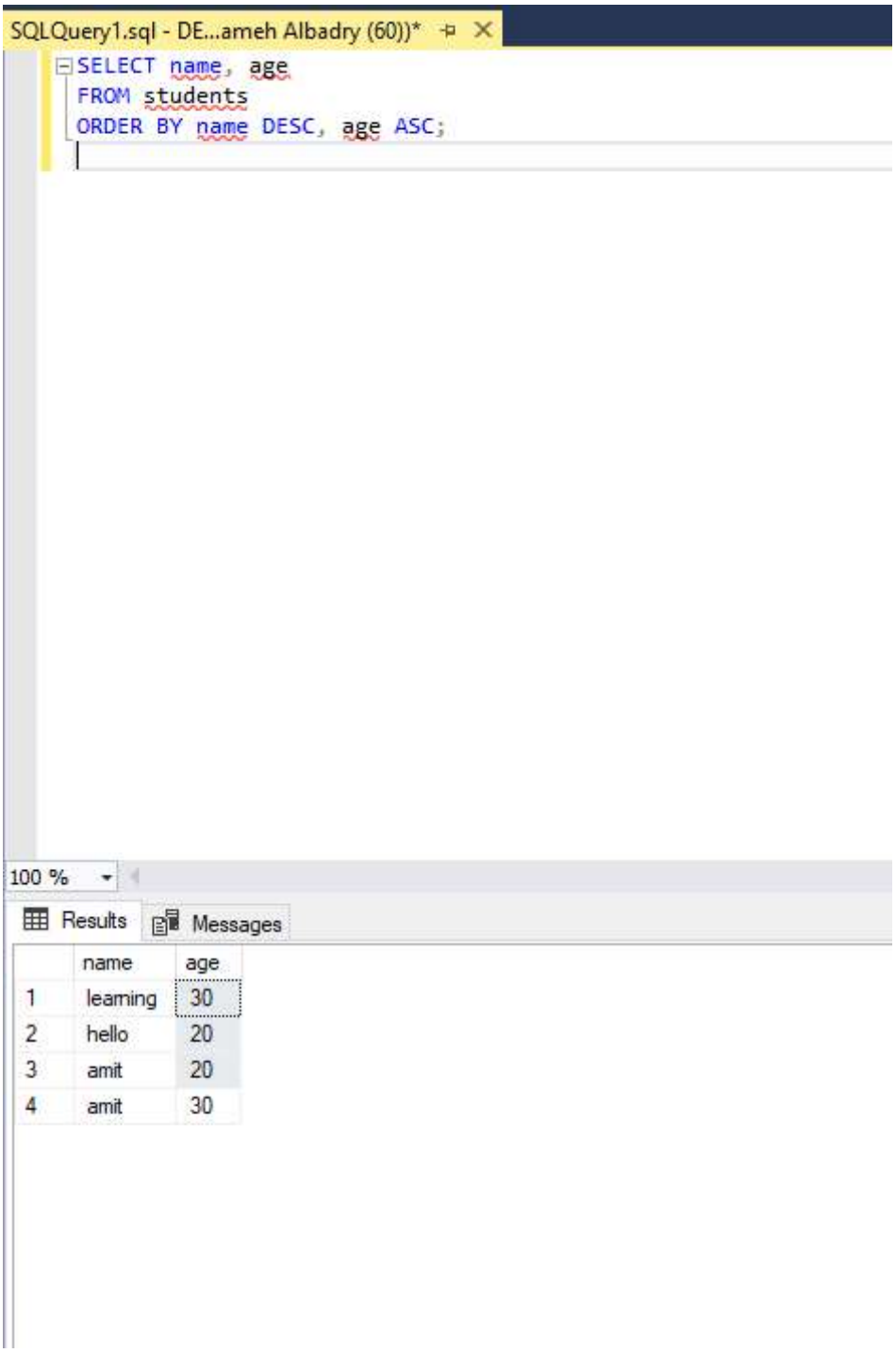
lets see this situation





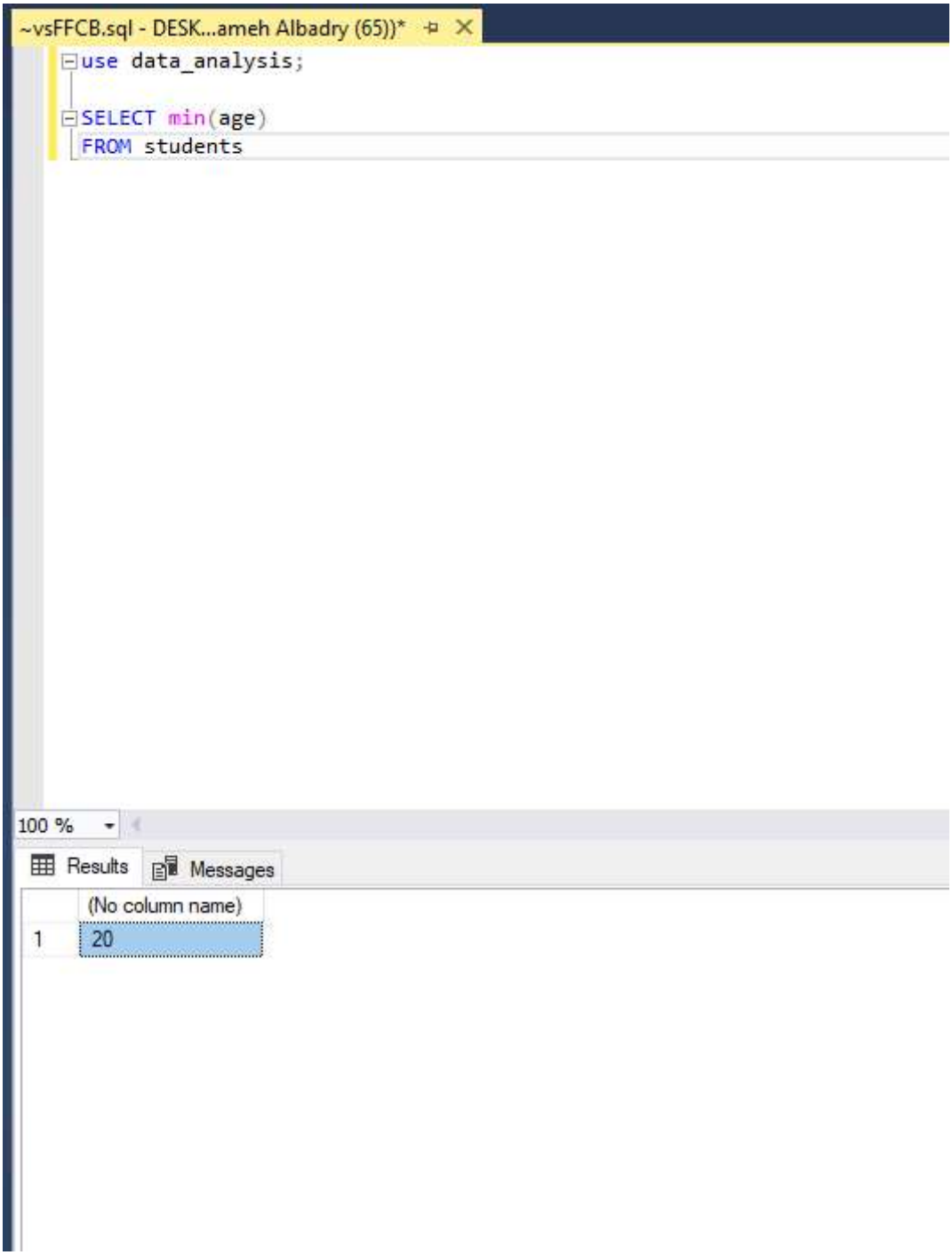
The age column isn't ordered in the query result because the SQL statement sorts by name first in descending order (DESC), and then by age in ascending order (ASC). So, for any rows that have the same name, the rows will be sorted by age with the youngest age first.

الترتيب اولويه للي اتكتب الاول

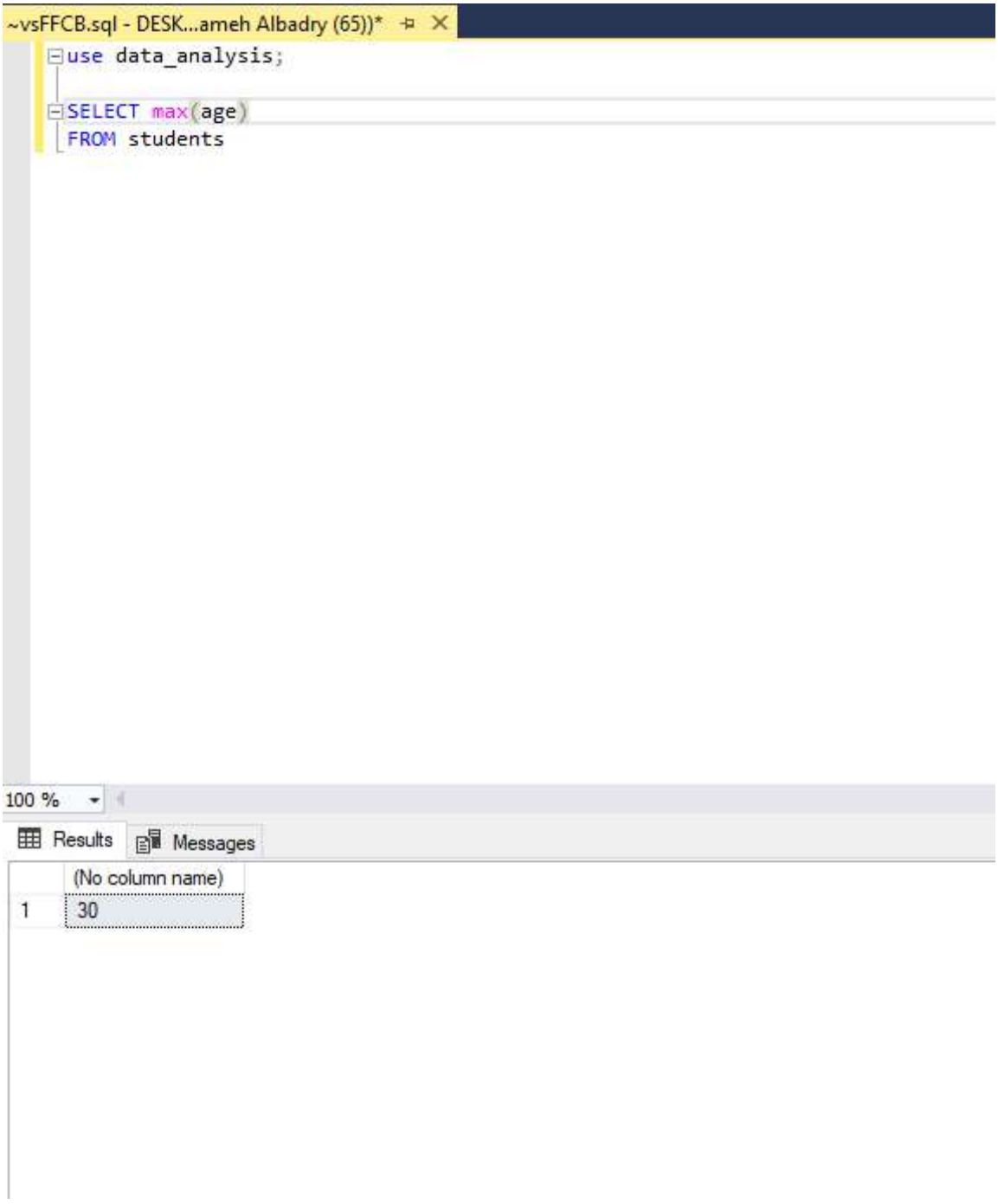


AGGREGATE functions

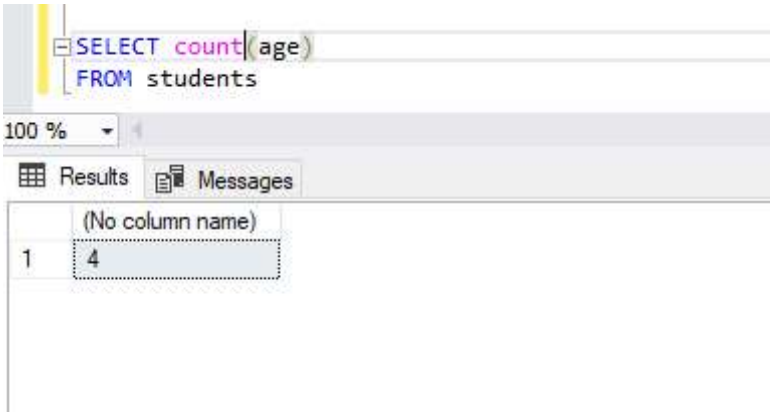
min value in column



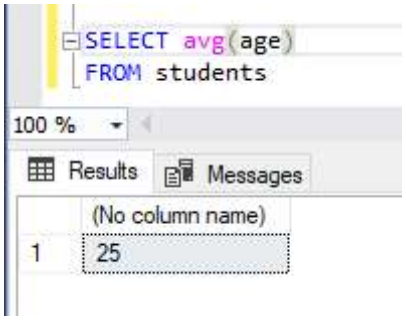
max number in column



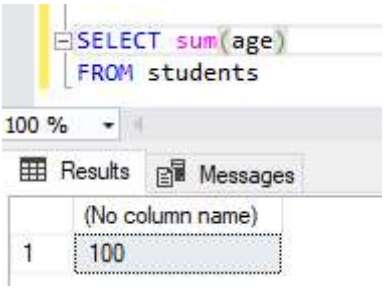
count value in column



calculate average in column

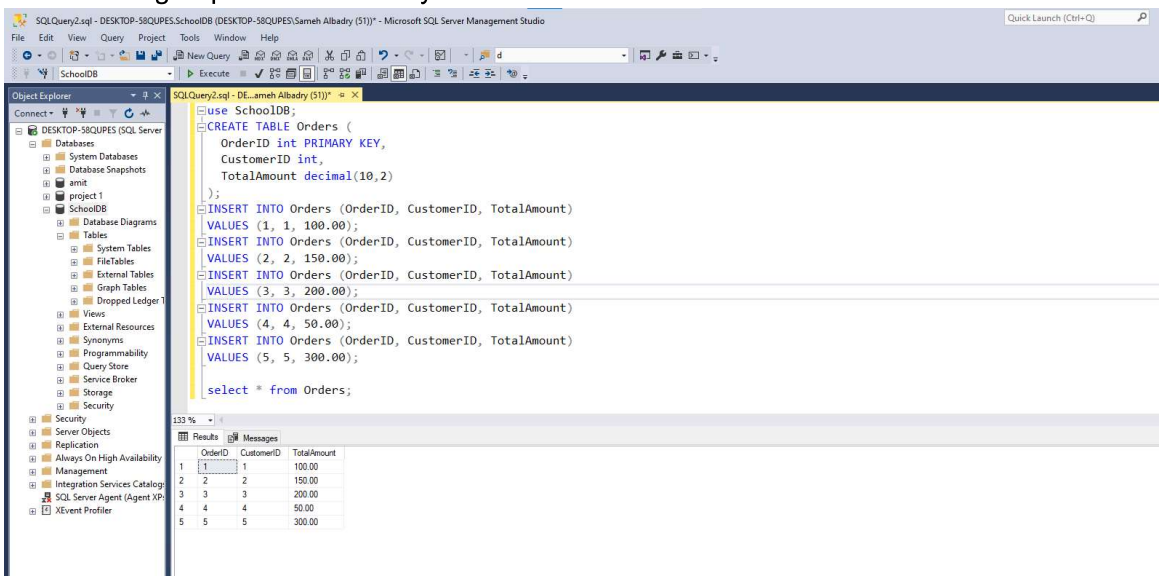


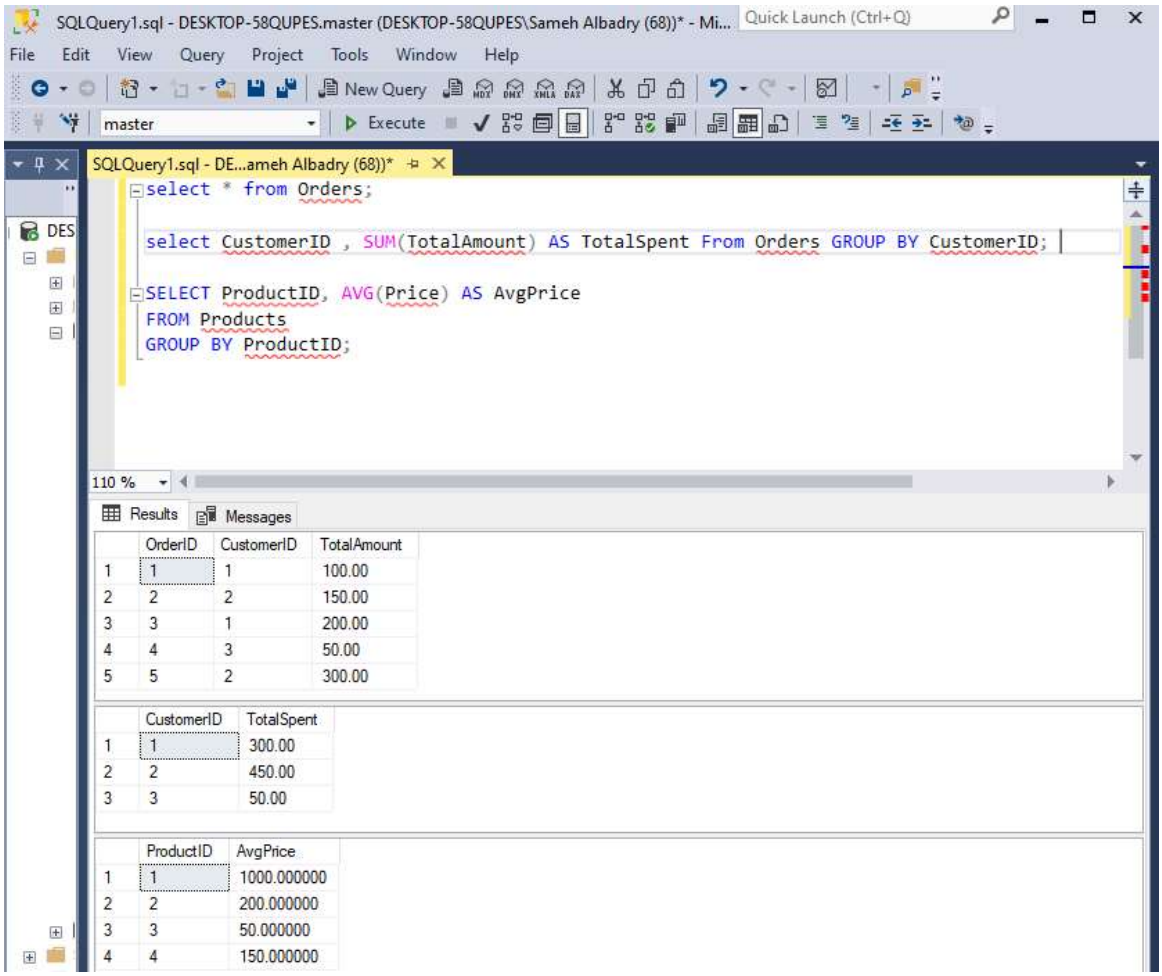
calculate sum in column



group by

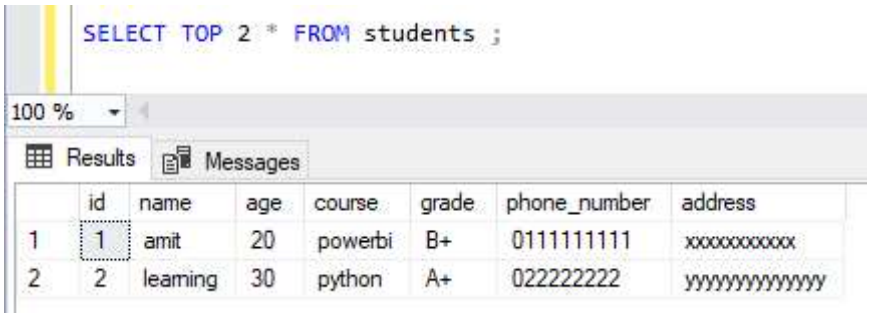
it groups rows that have the same values in a certain column. It is often used with aggregate functions to group the result-set by one or more columns.



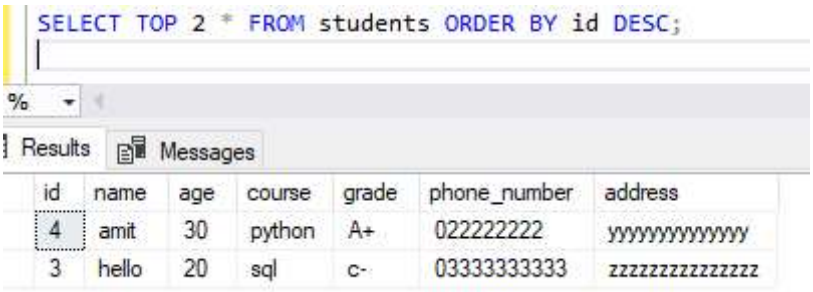


TOP

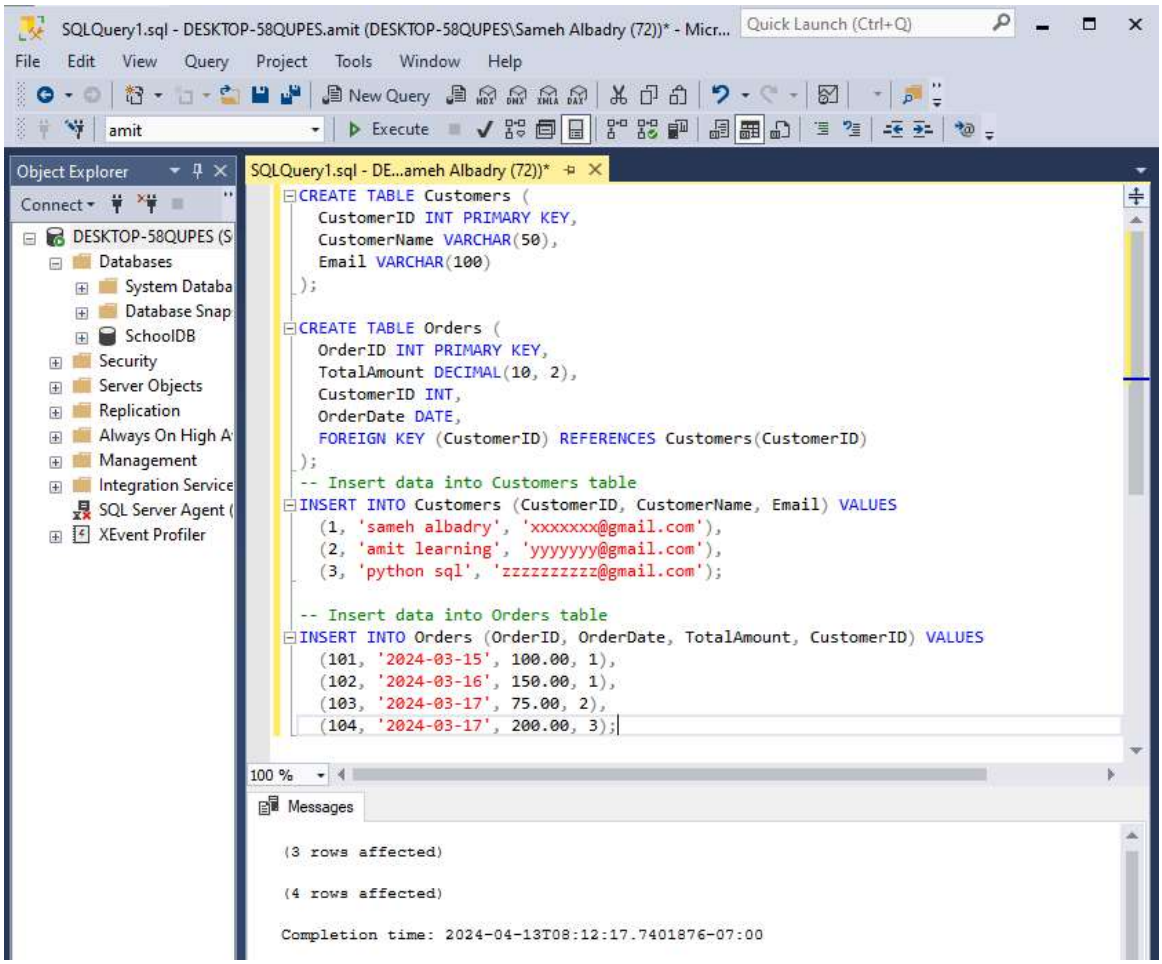
frst two rows



last two rows



get start for joins:



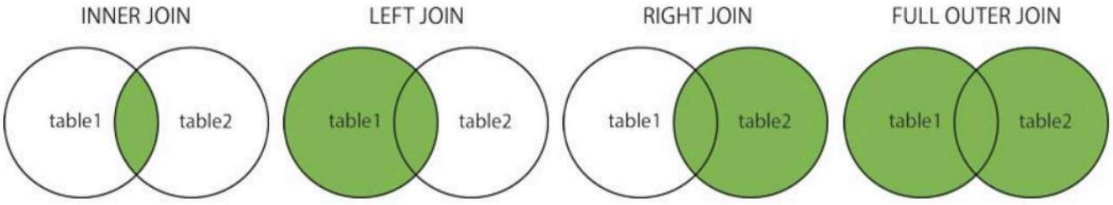
join

A JOIN clause is used to combine rows from two or more tables, based on a related column between them.

Different Types of SQL JOINs

Here are the different types of the JOINs in SQL:

- `(INNER) JOIN`: Returns records that have matching values in both tables
- `LEFT (OUTER) JOIN`: Returns all records from the left table, and the matched records from the right table
- `RIGHT (OUTER) JOIN`: Returns all records from the right table, and the matched records from the left table
- `FULL (OUTER) JOIN`: Returns all records when there is a match in either left or right table



Join						
INNER	**OUTER**					
	LEFT		**RIGHT**		**FULL**	
	Inclusive	Exclusive	Inclusive	Exclusive	Inclusive	Exclusive

INNER JOIN

returns only the rows where there is a match in both tables.

```
CREATE TABLE employees (
    id INT PRIMARY KEY,
    name VARCHAR(50),
    department_id INT,
    salary DECIMAL(10, 2)
);
CREATE TABLE departments (
    id INT PRIMARY KEY,
    name VARCHAR(50)
);
-- Insert values into the employees table
INSERT INTO employees (id, name, department_id, salary)
VALUES
    (1, 'John Doe', 1, 50000.00),
    (2, 'Jane Smith', 1, 60000.00),
    (3, 'Mark Johnson', 2, 55000.00),
    (4, 'Emily Davis', 2, 52000.00);
INSERT INTO employees (id, name, department_id, salary)
VALUES
    (5, 'Mark Johnson', 8, 55000.00),
    (6, 'Emily Davis', 7, 52000.00);
-- Insert values into the departments table
INSERT INTO departments (id, name)
VALUES
    (1, 'Engineering'),
    (2, 'Marketing');
INSERT INTO departments (id, name)
VALUES
    (3, 'Engineering'),
    (4, 'Marketing');

select * from employees;
select * from departments;
SELECT
    e.name AS employee_name,
    d.name AS department_name,
    e.salary
FROM
    employees e
INNER JOIN
    departments d ON e.department_id = d.id;
```

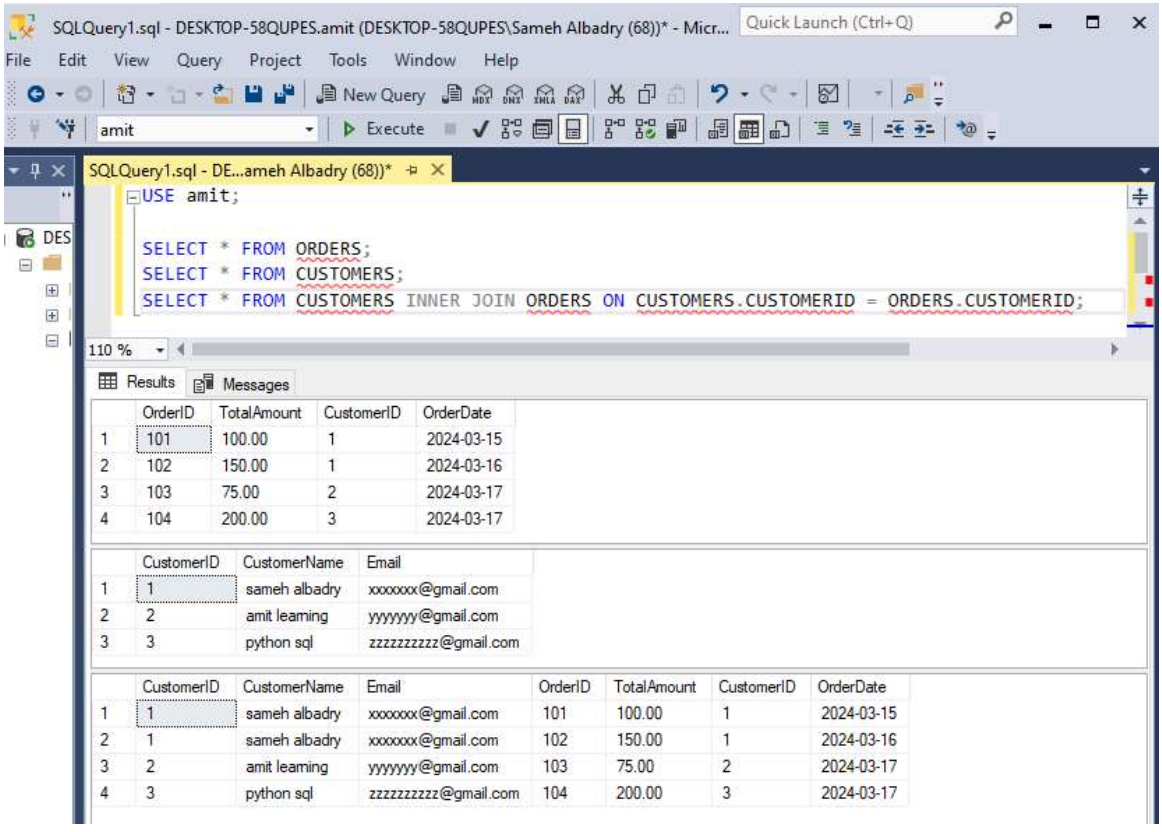
68 %

Results | Messages

	id	name	department_id	salary
1	1	John Doe	1	50000.00
2	2	Jane Smith	1	60000.00
3	3	Mark Johnson	2	55000.00
4	4	Emily Davis	2	52000.00
5	5	Mark Johnson	8	55000.00
6	6	Emily Davis	7	52000.00

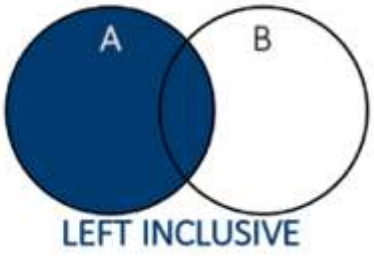
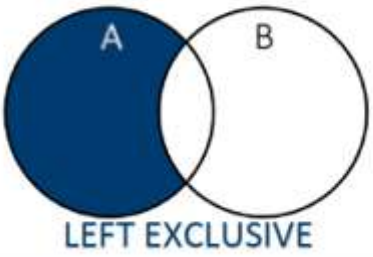
	id	name
1	1	Engineering
2	2	Marketing
3	3	Engineering
4	4	Marketing

	employee_name	department_name	salary
1	John Doe	Engineering	50000.00
2	Jane Smith	Engineering	60000.00
3	Mark Johnson	Marketing	55000.00
4	Emily Davis	Marketing	52000.00



left join

returns all rows from the left table (sales), and the matched rows from the right table (clients). If there's no match, NULL values are returned from the right side.

<table>
<tr><th colspan="2">Left Join</th></tr>
<tr><th>Inclusive</th><th>Exclusive</th></tr>
<tr>
<td> 
</td>
<td> 
</td>
</tr>
</table>

```sql
CREATE TABLE Sales (
  SaleID INT PRIMARY KEY,
  ClientID INT,
  SaleDate DATE,
  TotalAmount DECIMAL (10, 2)
);

CREATE TABLE Clients (
  ClientID INT PRIMARY KEY,
  ClientName VARCHAR(100),
  Email VARCHAR(100)
);

-- Insert sample data into Sales table
INSERT INTO Sales (SaleID, ClientID, SaleDate, TotalAmount) VALUES
(1, 101, '2024-04-09', 150.50),
(2, 102, '2024-04-02', 200.75),
(3, 103, '2024-04-03', 100.00),
(4, 104, '2024-04-04', 75.25),
(5, NULL, '2024-04-05', 50.00); -- Unmatched row

-- Insert sample data into Clients table
INSERT INTO Clients (ClientID, ClientName, Email) VALUES
(101, 'John Doe', 'john@example.com'),
(102, 'Alice Smith', 'alice@example.com'),
(103, 'Bob Johnson', 'bob@example.com'),
(105, 'Bob Johnson', 'bob@example.com');

-- Left Inclusive Join
SELECT
    *
FROM
    Sales
LEFT JOIN
    Clients ON Sales.ClientID = Clients.ClientID;

-- Left Exclusive Join
SELECT
    *
FROM
    Sales
LEFT JOIN
    Clients ON Sales.ClientID = Clients.ClientID
WHERE
    Clients.ClientID IS NULL;
```

75 %

Results | Messages

	SaleID	ClientID	SaleDate	TotalAmount	ClientID	ClientName	Email
1	1	101	2024-04-09	150.50	101	John Doe	john@example.com
2	2	102	2024-04-02	200.75	102	Alice Smith	alice@example.com
3	3	103	2024-04-03	100.00	103	Bob Johnson	bob@example.com
4	4	104	2024-04-04	75.25	NULL	NULL	NULL
5	5	NULL	2024-04-05	50.00	NULL	NULL	NULL

	SaleID	ClientID	SaleDate	TotalAmount	ClientID	ClientName	Email
1	4	104	2024-04-04	75.25	NULL	NULL	NULL
2	5	NULL	2024-04-05	50.00	NULL	NULL	NULL

SQLQuery1.sql - DESKTOP-58QUPES.amit (DESKTOP-58QUPES\Sameh Albadry (68))* - Micr...

Quick Launch (Ctrl+Q)

File Edit View Query Project Tools Window Help

New Query | Execute | amit

SQLQuery1.sql - DE...ameh Albadry (68))*

```
-- Create Sales table
CREATE TABLE Sales (
    SaleID INT PRIMARY KEY,
    ClientID INT,
    SaleDate DATE,
    TotalAmount DECIMAL(10, 2)
);
-- Create Clients table
CREATE TABLE Clients (
    ClientID INT PRIMARY KEY,
    ClientName VARCHAR(100),
    Email VARCHAR(100)
);
-- Insert sample data into Sales table
INSERT INTO Sales (SaleID, ClientID, SaleDate, TotalAmount) VALUES
(1, 101, '2024-04-01', 150.50),
(2, 102, '2024-04-02', 200.75),
(3, 103, '2024-04-03', 100.00),
(4, 104, '2024-04-04', 75.25),
(5, NULL, '2024-04-05', 50.00); -- Unmatched row

-- Insert sample data into Clients table
INSERT INTO Clients (ClientID, ClientName, Email) VALUES
(101, 'John Doe', 'john@example.com'),
(102, 'Alice Smith', 'alice@example.com'),
(103, 'Bob Johnson', 'bob@example.com');
-- Left Inclusive Join
SELECT *
FROM Sales
LEFT JOIN Clients ON Sales.ClientID = Clients.ClientID;

--Left Exclusive Join
SELECT *
FROM Sales
LEFT JOIN Clients ON Sales.ClientID = Clients.ClientID
WHERE Clients.ClientID IS NOT NULL;
```

91 %

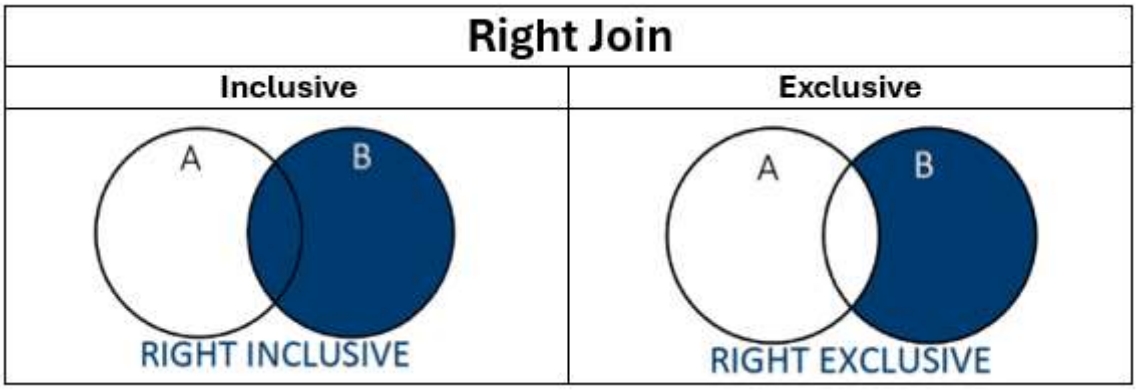
Results | Messages

	SaleID	ClientID	SaleDate	TotalAmount	ClientID	ClientName	Email
1	1	101	2024-04-01	150.50	101	John Doe	john@example.com
2	2	102	2024-04-02	200.75	102	Alice Smith	alice@example.com
3	3	103	2024-04-03	100.00	103	Bob Johnson	bob@example.com
4	4	104	2024-04-04	75.25	NULL	NULL	NULL
5	5	NULL	2024-04-05	50.00	NULL	NULL	NULL

	SaleID	ClientID	SaleDate	TotalAmount	ClientID	ClientName	Email
1	1	101	2024-04-01	150.50	101	John Doe	john@example.com
2	2	102	2024-04-02	200.75	102	Alice Smith	alice@example.com
3	3	103	2024-04-03	100.00	103	Bob Johnson	bob@example.com

right join

returns all rows from the right table (clients), and the matched rows from the left table (sales). If there's no match, NULL values are returned from the left side.



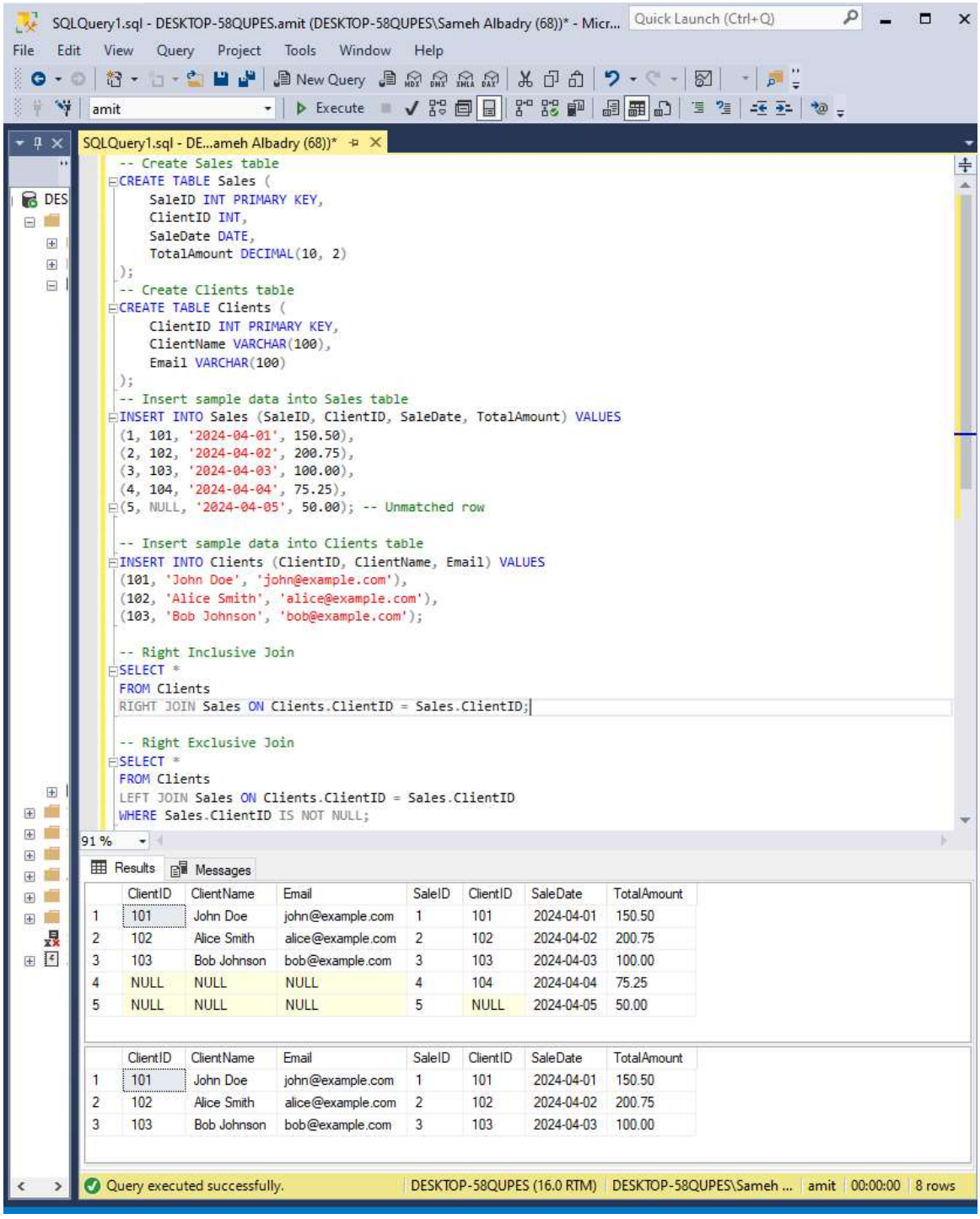

```sql
CREATE TABLE Sales (
    SaleID INT PRIMARY KEY,
    ClientID INT,
    SaleDate DATE,
    TotalAmount DECIMAL (10, 2)
);

CREATE TABLE Clients (
    ClientID INT PRIMARY KEY,
    ClientName VARCHAR(100),
    Email VARCHAR(100)
);

-- Insert sample data into Sales table
INSERT INTO Sales (SaleID, ClientID, SaleDate, TotalAmount) VALUES
(1, 101, '2024-04-09', 150.50),
(2, 102, '2024-04-02', 200.75),
(3, 103, '2024-04-03', 100.00),
(4, 104, '2024-04-04', 75.25),
(5, NULL, '2024-04-05', 50.00); -- Unmatched row

-- Insert sample data into Clients table
INSERT INTO Clients (ClientID, ClientName, Email) VALUES
(101, 'John Doe', 'john@example.com'),
(102, 'Alice Smith', 'alice@example.com'),
(103, 'Bob Johnson', 'bob@example.com'),
(105, 'Bob Johnson', 'bob@example.com');

-- Right Inclusive Join
SELECT
    *
FROM
    Sales
RIGHT JOIN
    Clients ON Sales.ClientID = Clients.ClientID;

-- Right Exclusive Join
SELECT
    *
FROM
    Sales
RIGHT JOIN
    Clients ON Sales.ClientID = Clients.ClientID
WHERE
    Sales.ClientID IS NULL;
```

5 %

Results | Messages

	SaleID	ClientID	SaleDate	TotalAmount	ClientID	ClientName	Email
1	1	101	2024-04-09	150.50	101	John Doe	john@example.com
2	2	102	2024-04-02	200.75	102	Alice Smith	alice@example.com
3	3	103	2024-04-03	100.00	103	Bob Johnson	bob@example.com
4	NULL	NULL	NULL	NULL	105	Bob Johnson	bob@example.com

	SaleID	ClientID	SaleDate	TotalAmount	ClientID	ClientName	Email
1	NULL	NULL	NULL	NULL	105	Bob Johnson	bob@example.com



self join

A self join is a regular join, but the table is joined with itself.

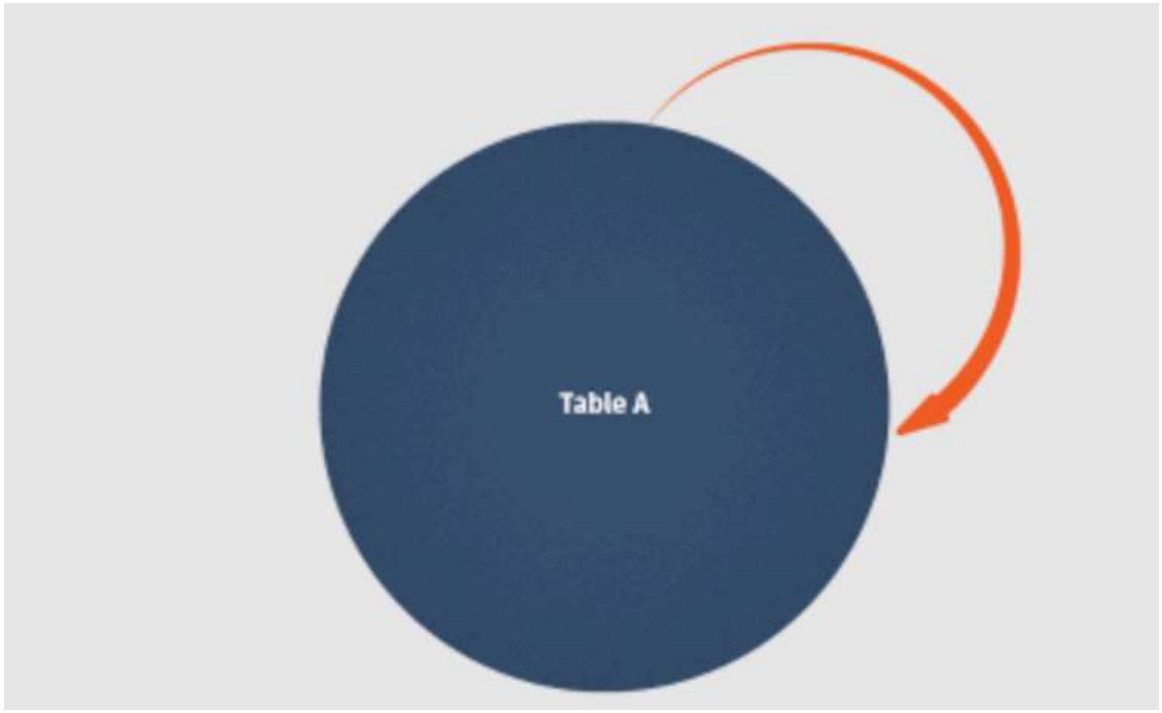
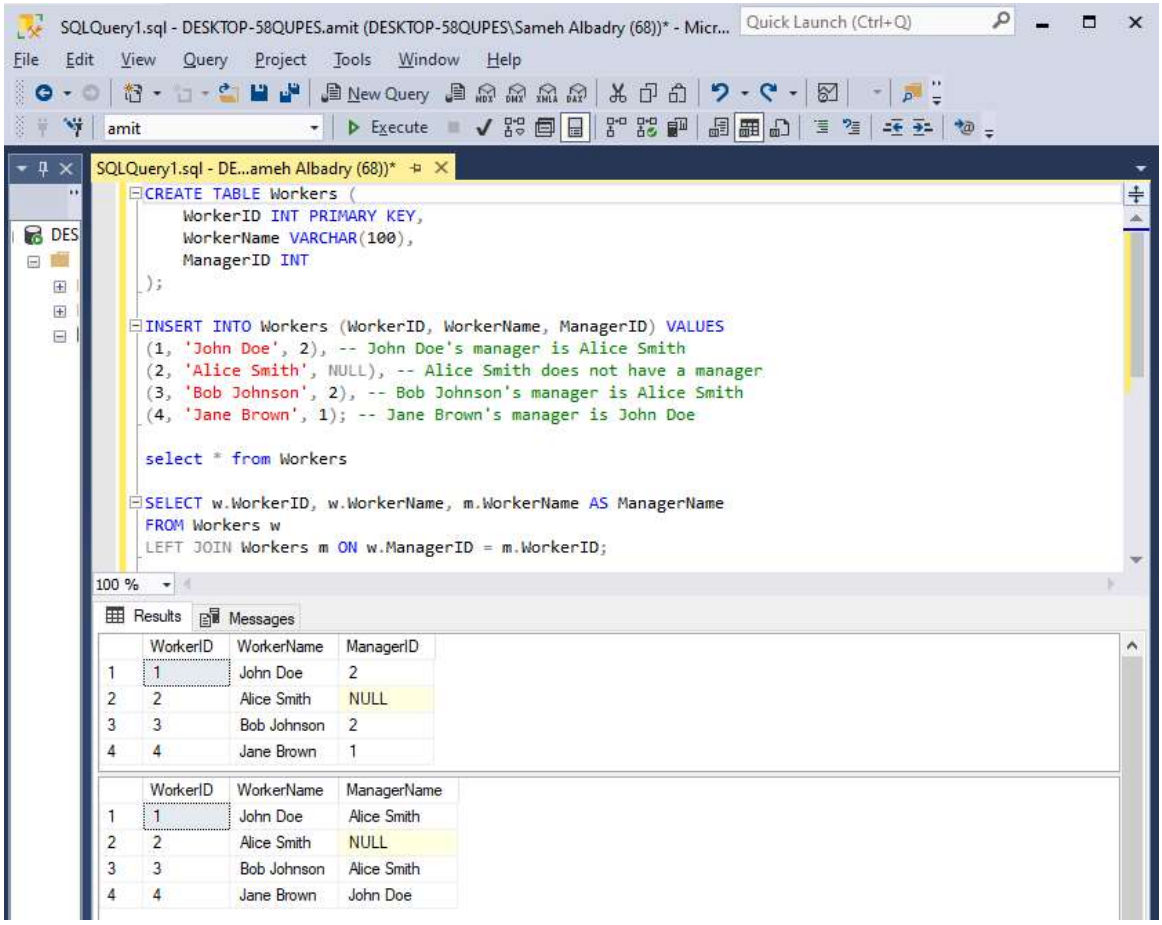
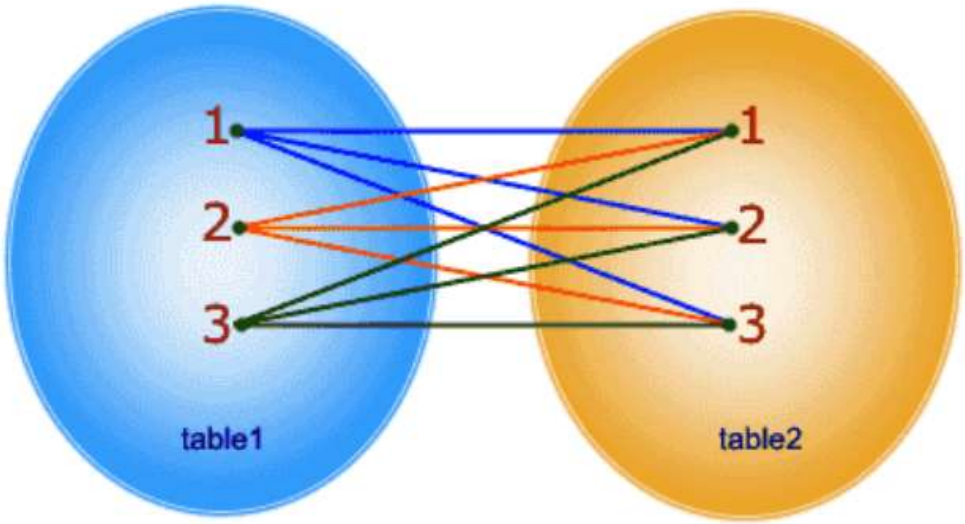


Table A

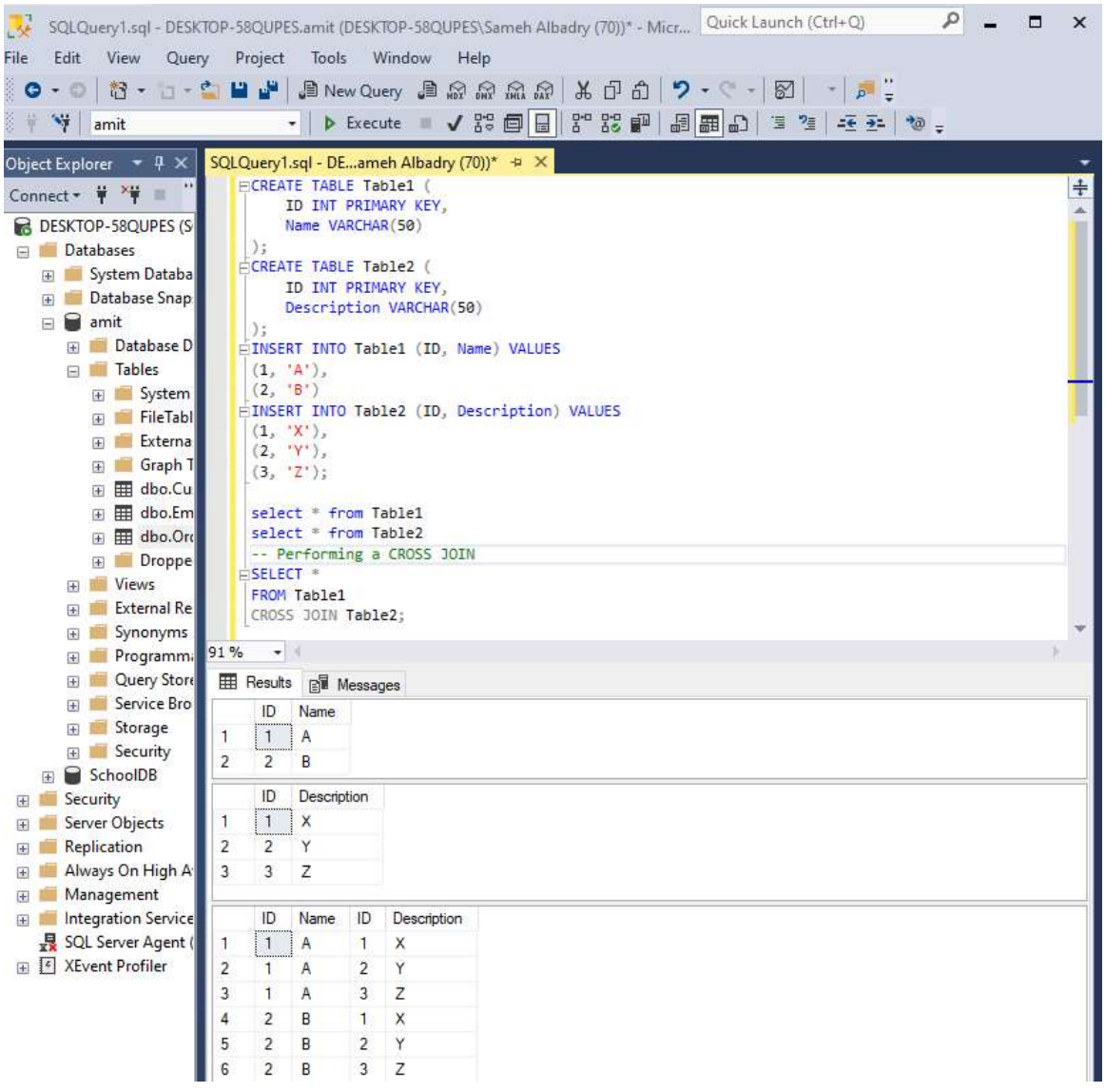

SQLQuery1.sql - DESKTOP-58QUPES.amit (DESKTOP-58QUPES\Sameh Albadry (68))* - Micr...
Quick Launch (Ctrl+Q)
File Edit View Query Project Tools Window Help
New Query
amit
Execute
SQLQuery1.sql - DE...ameh Albadry (68))*
CREATE TABLE Workers (
WorkerID INT PRIMARY KEY,
WorkerName VARCHAR(100),
ManagerID INT
);
INSERT INTO Workers (WorkerID, WorkerName, ManagerID) VALUES
(1, 'John Doe', 2), -- John Doe's manager is Alice Smith
(2, 'Alice Smith', NULL), -- Alice Smith does not have a manager
(3, 'Bob Johnson', 2), -- Bob Johnson's manager is Alice Smith
(4, 'Jane Brown', 1); -- Jane Brown's manager is John Doe
select * from Workers
SELECT w.WorkerID, w.WorkerName, m.WorkerName AS ManagerName
FROM Workers w
LEFT JOIN Workers m ON w.ManagerID = m.WorkerID;
100 %
Results
Messages
WorkerID WorkerName ManagerID
1 1 John Doe 2
2 2 Alice Smith NULL
3 3 Bob Johnson 2
4 4 Jane Brown 1
WorkerID WorkerName ManagerName
1 1 John Doe Alice Smith
2 2 Alice Smith NULL
3 3 Bob Johnson Alice Smith
4 4 Jane Brown John Doe

cross join

SELECT * FROM table1 CROSS JOIN table2;

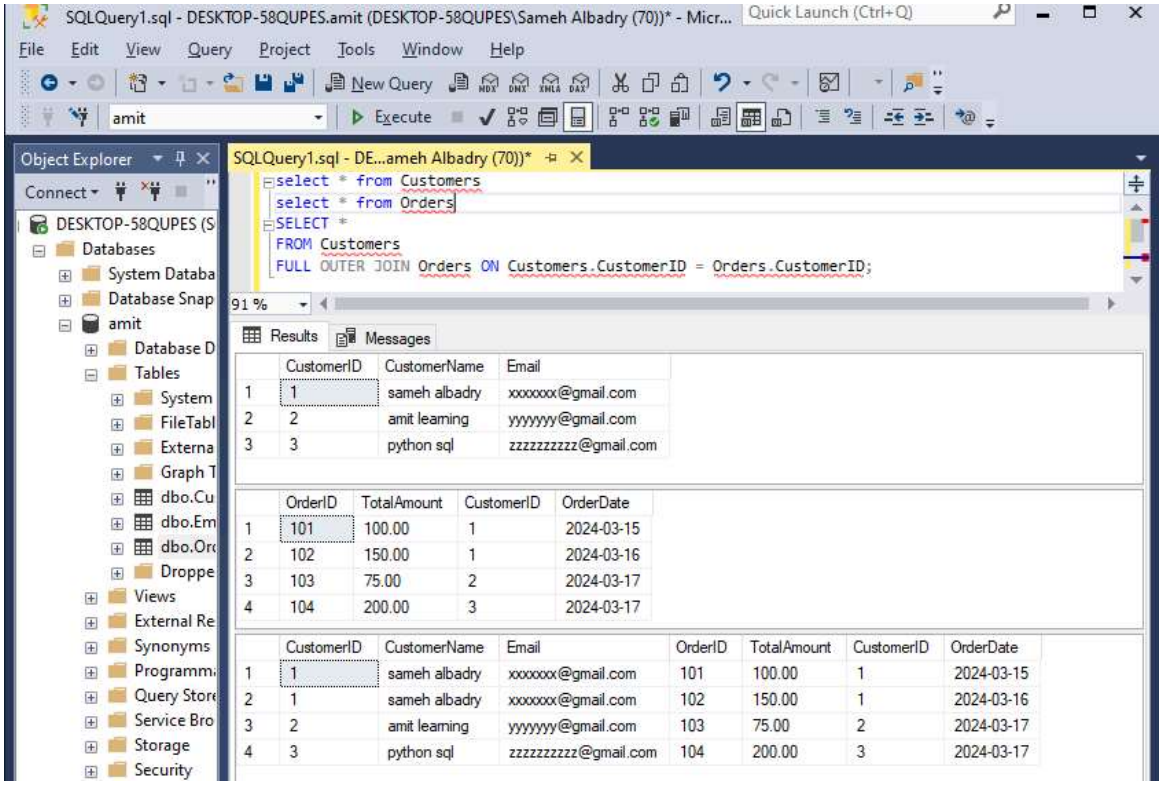


In CROSS JOIN, each row from 1st table joins with all the rows of another table.
If 1st table contain x rows and y rows in 2nd one the result set will be x * y rows.



Full Outer Join

This returns all rows when there is a match in either table. If there's no match, NULL values are returned on both sides.



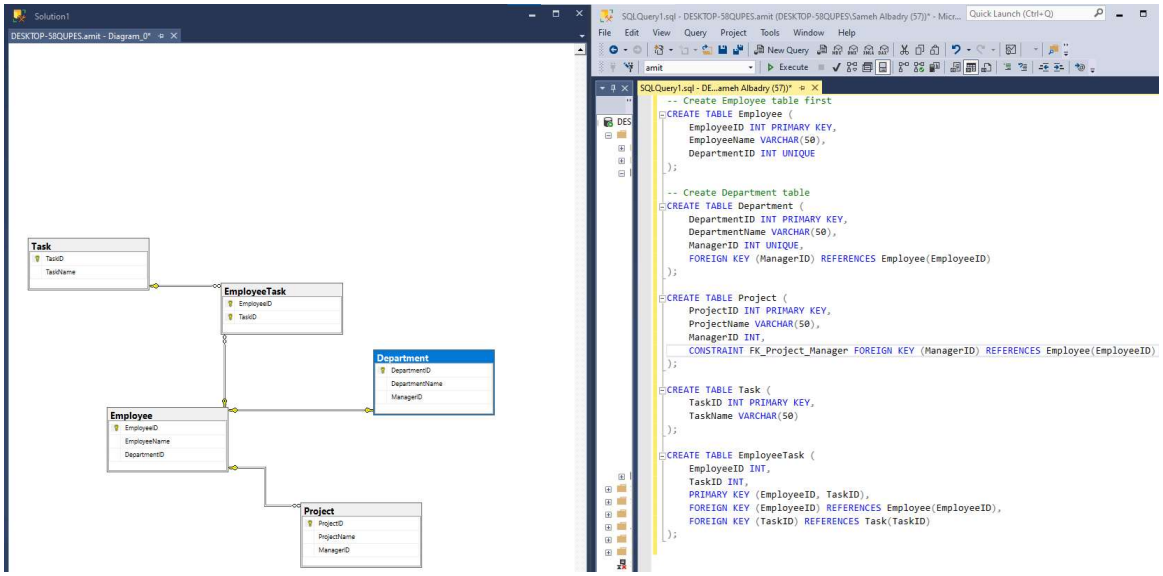
diffrent between full outer join and union

UNION: It takes the results of two queries and puts them together into one result set. It only keeps distinct rows, removing duplicates.

FULL OUTER JOIN: It combines the results of two tables, keeping all rows from both tables. It fills in NULL values where there is no match between the tables.

```
 -- Create Table1
CREATE TABLE T1 (
    ID INT PRIMARY KEY,
    Name VARCHAR(50)
);
-- Insert sample data into Table1
INSERT INTO T1 (ID, Name) VALUES
(1, 'John'),
```

Entity Relationship Diagrams


Solution1
DESKTOP-58QUPES.amit - Diagram_0*
Task
TaskID
TaskName
EmployeeTask
EmployeeID
TaskID
Department
DepartmentID
DepartmentName
ManagerID
Employee
EmployeeID
EmployeeName
DepartmentID
Project
ProjectID
ProjectName
ManagerID
SQLQuery1.sql - DESKTOP-58QUPES.amit (DESKTOP-58QUPES\Sameh Albadry (57))* - Micr...
Quick Launch (Ctrl+Q)
File Edit View Query Project Tools Window Help
New Query
Execute
amit
SQLQuery1.sql - DE...ameh Albadry (57))*
-- Create Employee table first
CREATE TABLE Employee (
EmployeeID INT PRIMARY KEY,
EmployeeName VARCHAR(50),
DepartmentID INT UNIQUE
);
-- Create Department table
CREATE TABLE Department (
DepartmentID INT PRIMARY KEY,
DepartmentName VARCHAR(50),
ManagerID INT UNIQUE,
FOREIGN KEY (ManagerID) REFERENCES Employee(EmployeeID)
);
CREATE TABLE Project (
ProjectID INT PRIMARY KEY,
ProjectName VARCHAR(50),
ManagerID INT,
CONSTRAINT FK_Project_Manager FOREIGN KEY (ManagerID) REFERENCES Employee(EmployeeID)
);
CREATE TABLE Task (
TaskID INT PRIMARY KEY,
TaskName VARCHAR(50)
);
CREATE TABLE EmployeeTask (
EmployeeID INT,
TaskID INT,
PRIMARY KEY (EmployeeID, TaskID),
FOREIGN KEY (EmployeeID) REFERENCES Employee(EmployeeID),
FOREIGN KEY (TaskID) REFERENCES Task(TaskID)
);

In [ ]: