

## Lab 1: Installation of MySQL(structured query language)

### Introduction

MySQL is one of the most widely used open-source relational database management systems (RDBMS). It uses Structured Query Language (SQL) to manage and manipulate data efficiently. The purpose of this lab is to install MySQL on the system and become familiar with the basic setup required to run and manage a database. Understanding how to install and configure MySQL is the first step in learning database management and working with SQL commands.

### Steps while downloading and installing MySQL

**Step1:** Download Mysql installer from community downloads choosing developer option.

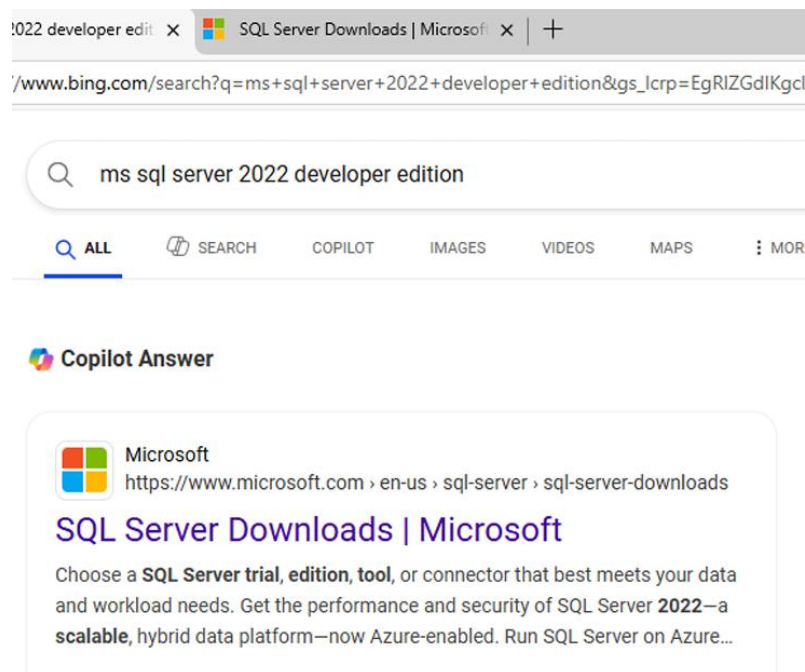


Fig 1.1: Bing search result for "MS SQL Server 2022 Developer Edition" showing the official Microsoft download link.

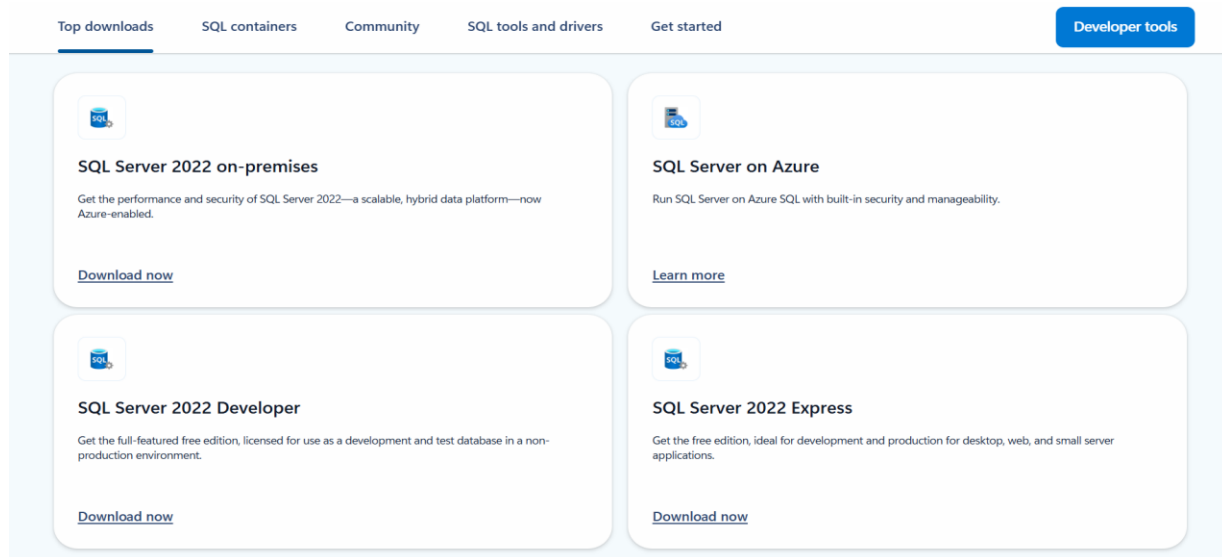


Fig1.2: The figure shows various SQL Server 2022 editions available for download, including On-premises, Developer, Express, and Azure versions from the official Microsoft website.

**Step2:** Download and install the basic option in from developer edition.

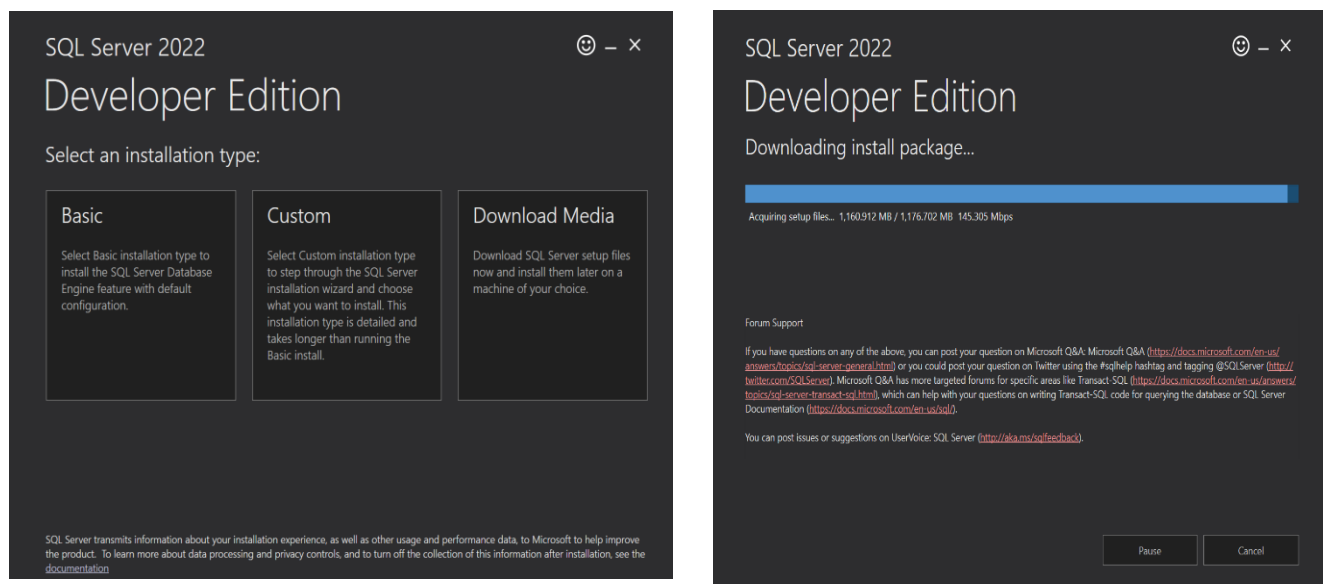



Fig 1.3: The figure displays the installation screen for SQL Server 2022 Developer Edition, offering three options: Basic, Custom, and Download Media.

Step 3: Install ssms(SQL Server Management Studio).

## Step 2 - Determine which version of SQL Server Management Studio to install

Decide which version of SSMS to install. The most common options are:

- The latest release of SQL Server Management Studio 21 that is hosted on Microsoft servers. To install this version, select the following link. The installer downloads a small *bootstrapper* to your *Downloads* folder.

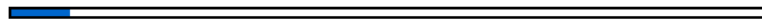
[Download SSMS 21](#) 

- If you already have SQL Server Management Studio 21 installed, you can [install another version alongside it](#).
- You can download a bootstrapper or installer for a specific version from the [Release history](#) page and use it to install another version of SSMS.

### Visual Studio Installer

Getting the Visual Studio Installer ready.

Downloading: 2.52 MB of 31.39 MB 822.22 KB/sec



Installing



Cancel

Fig 1.4: This step shows how to choose and download the right version of SQL Server Management Studio, with a link to get the latest SSMS 21 and installing it.

**Step4:** Download and install SQL server management and launch it.

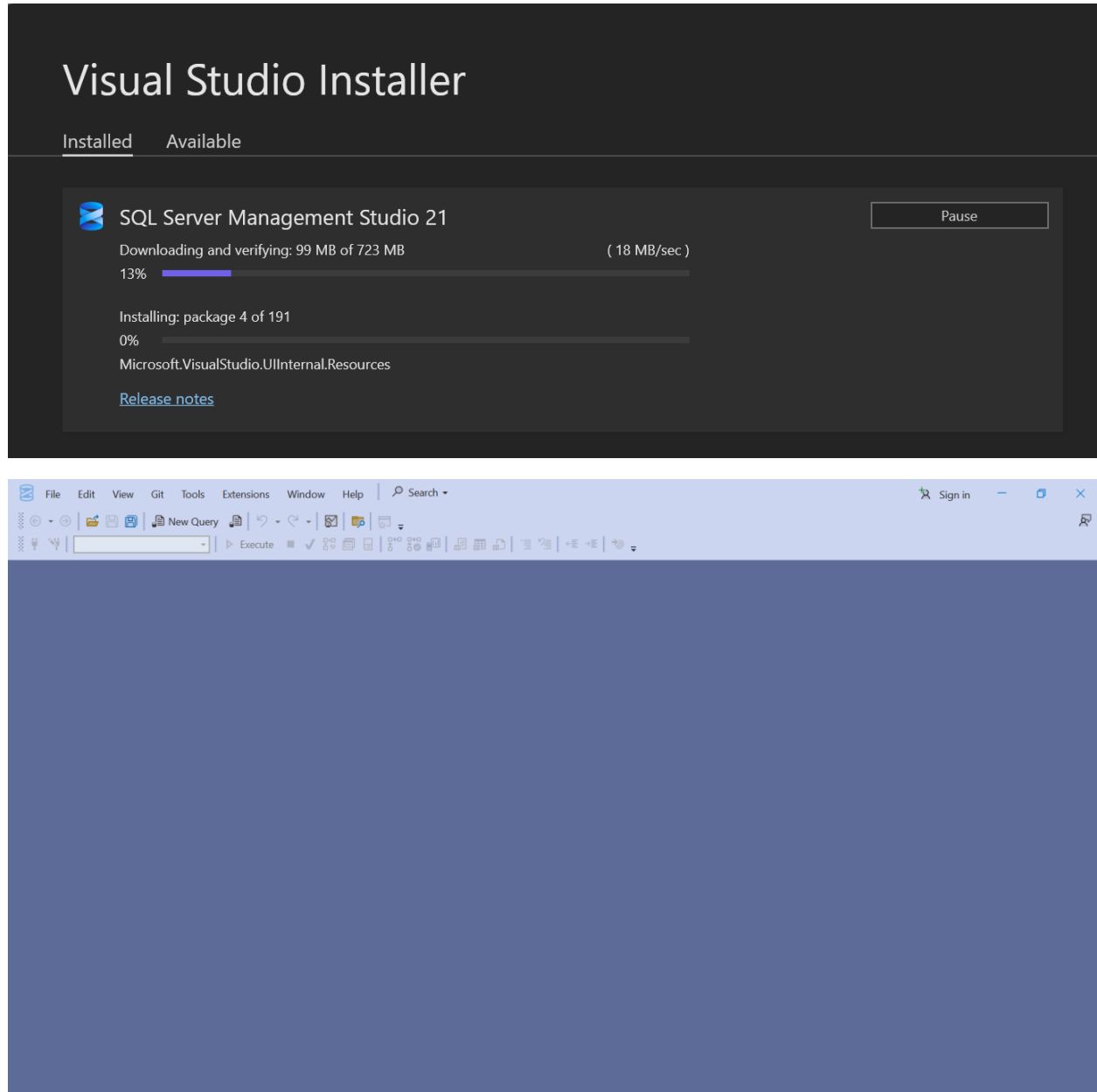


Fig 1.5: The Visual Studio Installer is downloading and installing SQL Server Management Studio 21 and The SSMS 21 interface is open, showing the main workspace with toolbar options like File, Edit, View, and New Query.

## Conclusion

In this lab, we successfully installed MySQL and ensured that it runs properly on our system. This setup provides the foundation for practicing SQL commands and developing database-driven applications. By completing this lab, we are now ready to explore the core functionalities of MySQL and perform operations such as creating tables, inserting data, and running queries.

## Lab 2: Creating a Database and tables using query

### Introduction

In this lab, we learn how to create a **database** and define **tables** using SQL queries. Understanding how to structure a database is essential for organizing data efficiently. Using the CREATE DATABASE and CREATE TABLE commands, we can design a relational database that suits the needs of any application. This lab focuses on writing basic SQL commands to set up a database and its tables from scratch.

### Creating a database and connecting to local server

**Step1:** Establish a connection to a server name the server local inside the parenthesis.

**Step2:** Right click and click create a new database and name the database lab 1.

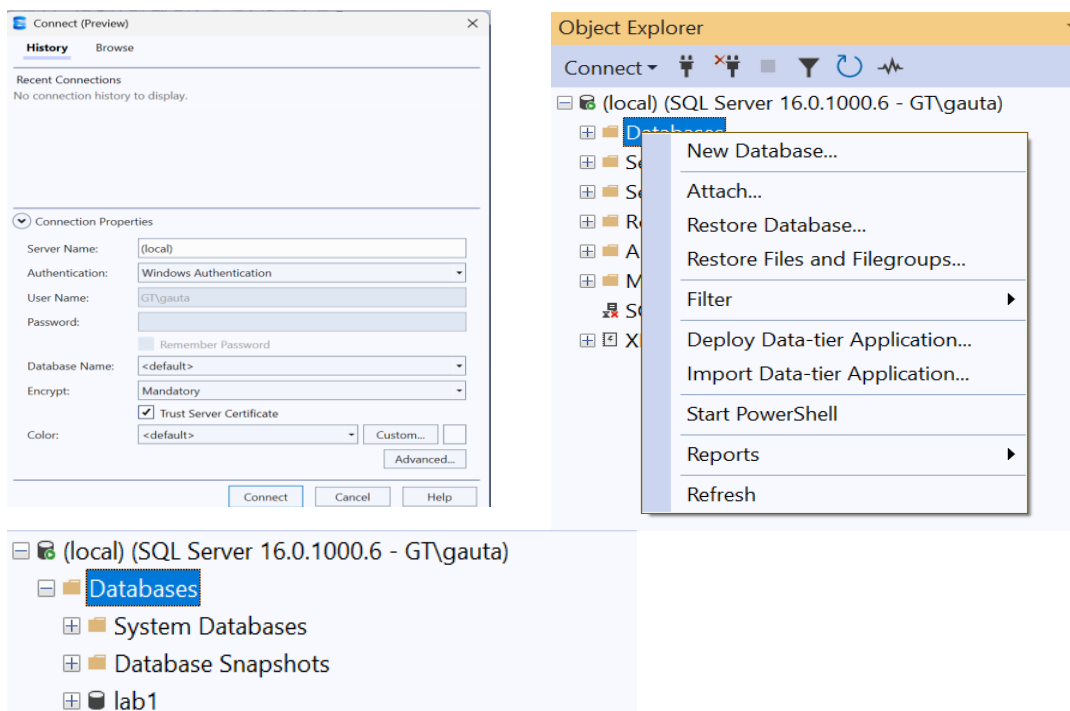


fig2.1: The images show connecting to a local SQL Server in SSMS, accessing the database options, and viewing the existing "lab1" database.

## Creating a table for relational database.

### Steps for creating tables and showing its diagrammatic view

Step 1: Right click and click create new query.

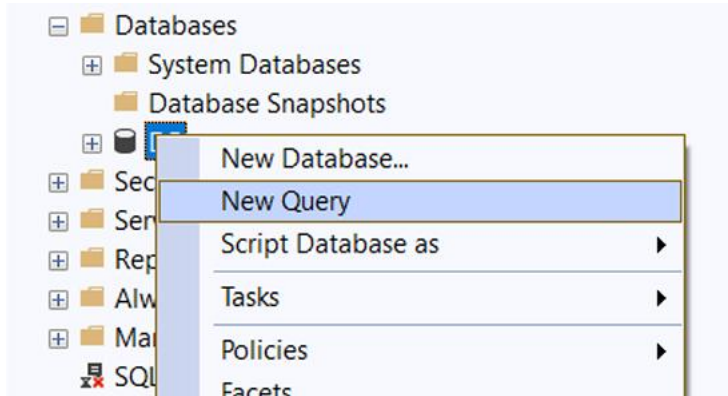


fig2.2: The image shows the context menu in SSMS for starting a **New Query** or creating a **New Database** under the "Databases" section.

**Step2:** Write the given query given below and execute the code.

```
CREATE TABLE StudentDetails
```

```
(
```

```
Rollno Int,
```

```
StudentName Varchar(100)
```

```
)
```

```
CREATE TABLE SubjectDetails
```

```
(
```

```
SubjectID INT,
```

```
SubjectName VARCHAR(100)
```

```
)
```

```
CREATE TABLE MarksheetDetails
```

```
(
```

```
Rollno INT,
```

```
SubjectID INT,
```

```
Marks INT)
```

**Step3:** Right click and click new database diagram and add the necessary tables.

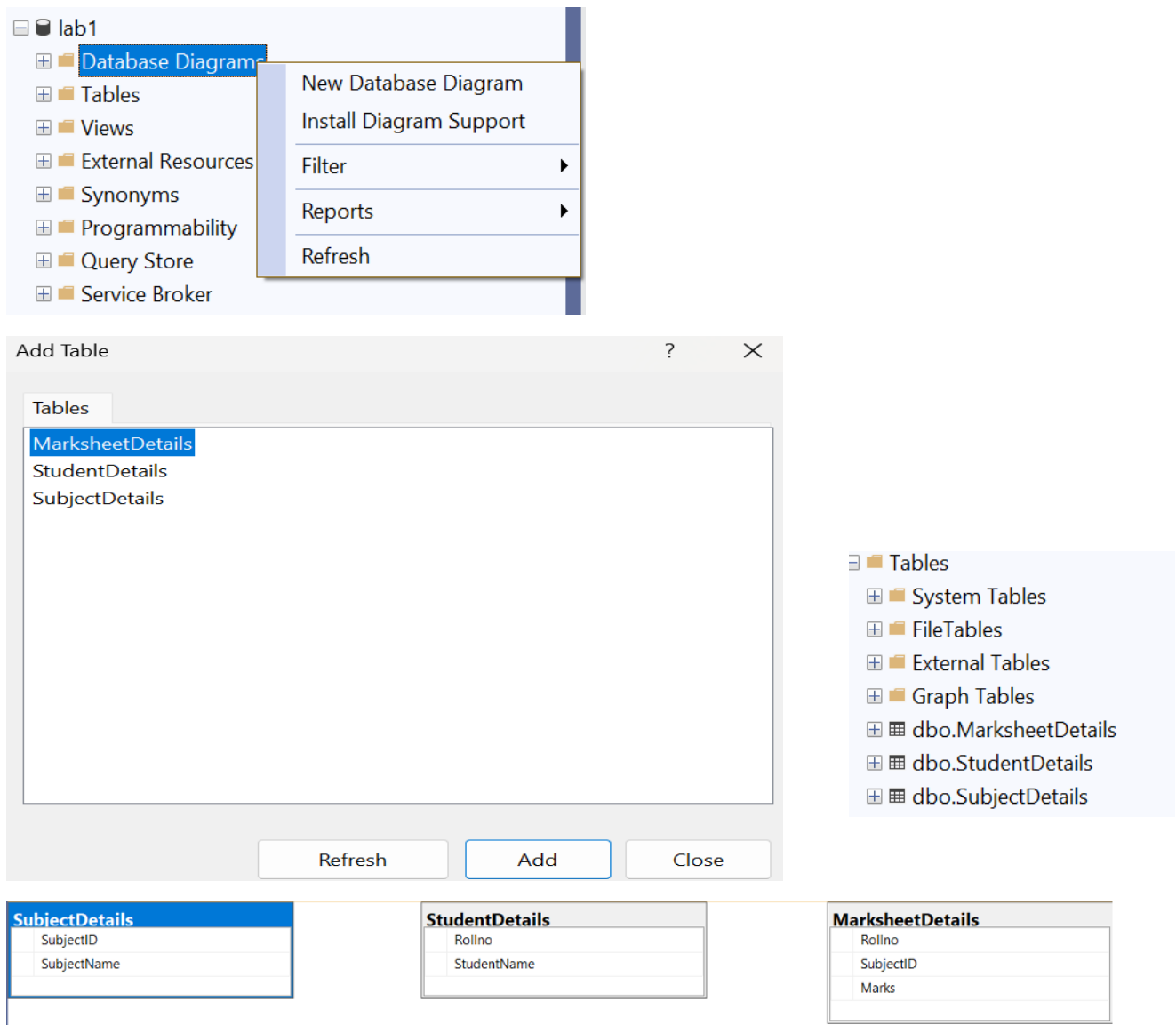


Fig 2.3: These images show the creation of a **Database Diagram** in SSMS using tables: `StudentDetails`, `SubjectDetails`, and `MarksheetDetails`. It visualizes the table structures and their columns, helping understand the relationships between student records, subjects, and marks.

## Conclusion

Through this lab, we successfully created a database and multiple tables using SQL queries. This practical experience helped us understand how to define table structures with appropriate data types and constraints. Learning to create databases and tables manually provides a solid foundation for managing real-world data in relational database systems.



## Lab 3: Showing Keys and various functions of DDL.

### Keys in SQL

Keys in SQL are used to uniquely identify rows in a table and to create relationships between tables. They help maintain data accuracy and integrity.

#### Common types of keys:

- **Primary Key:** Uniquely identifies each record in a table. It cannot be NULL or duplicated.
- **Foreign Key:** A field in one table that refers to the **primary key** in another table. It creates a link between two tables.
- **Candidate Key:** A column (or set of columns) that can uniquely identify rows in a table. One candidate key becomes the **primary key**.
- **Composite Key:** A key made up of two or more columns used together to uniquely identify a record.
- **Unique Key:** Ensures all values in a column are different, like a primary key, but it can accept a NULL value.

### Database Language in SQL

SQL (Structured Query Language) includes different types of commands grouped as **database languages**. These are used to create, manage, and manipulate data in a database.

#### Types of Database Languages:

1. **DDL (Data Definition Language)** – Defines the structure of the database.  
Commands: CREATE, ALTER, DROP, TRUNCATE
2. **DML (Data Manipulation Language)** – Deals with data manipulation.  
Commands: SELECT, INSERT, UPDATE, DELETE
3. **DCL (Data Control Language)** – Controls access to data.  
Commands: GRANT, REVOKE
4. **TCL (Transaction Control Language)** – Manages transactions in the database.  
Commands: COMMIT, ROLLBACK, SAVEPOINT



## Some Functions in DDL

**CREATE():** Used to create a new table or database.

**ALTER():** Used to change the structure of an existing table (e.g., add or remove columns).

**DROP():** Used to delete a table or database completely.

## Steps for showing keys and DDL language in a relational database

**Step1:** Create a new query and write the following code which includes primary key and composite primary key and DDL functions then executing it.

```
-- Dropping table if it exists

DROP TABLE IF EXISTS StudentDetails;
DROP TABLE IF EXISTS SubjectDetails;
DROP TABLE IF EXISTS MarksheetDetails;
GO

--CreatingStudentDetailstable

CREATE TABLE StudentDetails
(
    RollNo INT NOT NULL,
    StudentName VARCHAR(100),
    PRIMARY KEY (RollNo)
);

-- Creating SubjectDetails table

CREATE TABLE SubjectDetails
(
    SubjectId INT NOT NULL,
    SubjectName VARCHAR(100),
    PRIMARY KEY (SubjectId)
);

-- Creating MarksheetDetails table

CREATE TABLE MarksheetDetails
(
    RollNo INT NOT NULL,
    SubjectId INT NOT NULL,
    Marks INT,
```

```
FOREIGN KEY (RollNo) REFERENCES StudentDetails(RollNo),
FOREIGN KEY (SubjectId) REFERENCES SubjectDetails(SubjectId)
);
```

**Step2:** Showing the diagrammatic view of relational tables including the database tables.



Fig 3.1: The image shows a database diagram in SSMS with three tables—StudentDetails, SubjectDetails, and MarksheetDetails—visualizing keys and relationships.

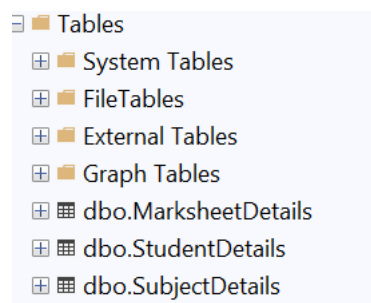


Fig 3.2: The image shows three user-created tables—MarksheetDetails, StudentDetails, and SubjectDetails—listed under the "Tables" section in SSMS.

## Conclusion

In this lab, we applied various DDL commands to create, alter, and drop database objects, and we learned how keys are used to uniquely identify and relate records in tables. This hands-on practice helped us understand the structural foundation of relational databases and how to manage them effectively using SQL.



## Lab 4: Insertion, deletion and update of Data into Relational Tables Using Queries with Foreign Keys

### Introduction:

In this lab, we focus on **DML (Data Manipulation Language)** commands, which are used to manage the data within relational database tables. These commands do not modify the table structure but allow users to:

- **SELECT** – Retrieve data from tables.
- **INSERT** – Add new records into a table.
- **UPDATE** – Modify existing records.
- **DELETE** – Remove records from a table.

The lab demonstrates the use of the **INSERT** and **SELECT** commands to insert and retrieve student-related data from multiple interrelated tables, while ensuring **referential integrity** using **foreign keys**.

### Step 1: Creating Tables with Foreign Keys

Three relational tables are created using SQL Server Management Studio (SSMS):

1. StudentDetails – Stores student information.
2. SubjectDetails – Stores subject information.
3. MarksheetDetails – Stores marks obtained by students, linked to the other two tables using foreign keys.

This design enforces relational integrity and ensures that entries in MarksheetDetails cannot exist unless corresponding records exist in StudentDetails and SubjectDetails.

#### *SQL Source Code:*

```
-- Dropping tables if they exist
DROP TABLE IF EXISTS StudentDetails;
DROP TABLE IF EXISTS SubjectDetails;
DROP TABLE IF EXISTS MarksheetDetails;
GO
```

```
-- Creating StudentDetails table
CREATE TABLE StudentDetails (
    RollNo INT NOT NULL,
    StudentName VARCHAR(100),
```

```

PRIMARY KEY (RollNo)
);

-- Creating SubjectDetails table
CREATE TABLE SubjectDetails (
    SubjectId INT NOT NULL,
    SubjectName VARCHAR(100),
    PRIMARY KEY (SubjectId)
);

-- Creating MarksheetDetails table
CREATE TABLE MarksheetDetails (
    RollNo INT NOT NULL,
    SubjectId INT NOT NULL,
    Marks FLOAT,
    FOREIGN KEY (RollNo) REFERENCES StudentDetails(RollNo),
    FOREIGN KEY (SubjectId) REFERENCES SubjectDetails(SubjectId)
);

```

### *Inserting Data into Tables:*

```

-- Inserting students
INSERT INTO StudentDetails (RollNo, StudentName) VALUES
(1, 'SARFRAJ'),
(2, 'UDIT'),
(3, 'SOHEAL');

-- Inserting subjects
INSERT INTO SubjectDetails (SubjectId, SubjectName) VALUES
(101, 'MATHS'),
(102, 'C++'),
(103, 'DSA');

-- Inserting marks
INSERT INTO MarksheetDetails (RollNo, SubjectId, Marks) VALUES
(1, 101, 88),
(1, 102, 78),
(1, 103, 92),
(2, 101, 82),
(2, 102, 88),
(3, 101, 90),
(3, 103, 86);

```

## **Step 2: Retrieving Data Using SELECT Queries**

After inserting the data, we use **SELECT** queries to verify and view the inserted records.

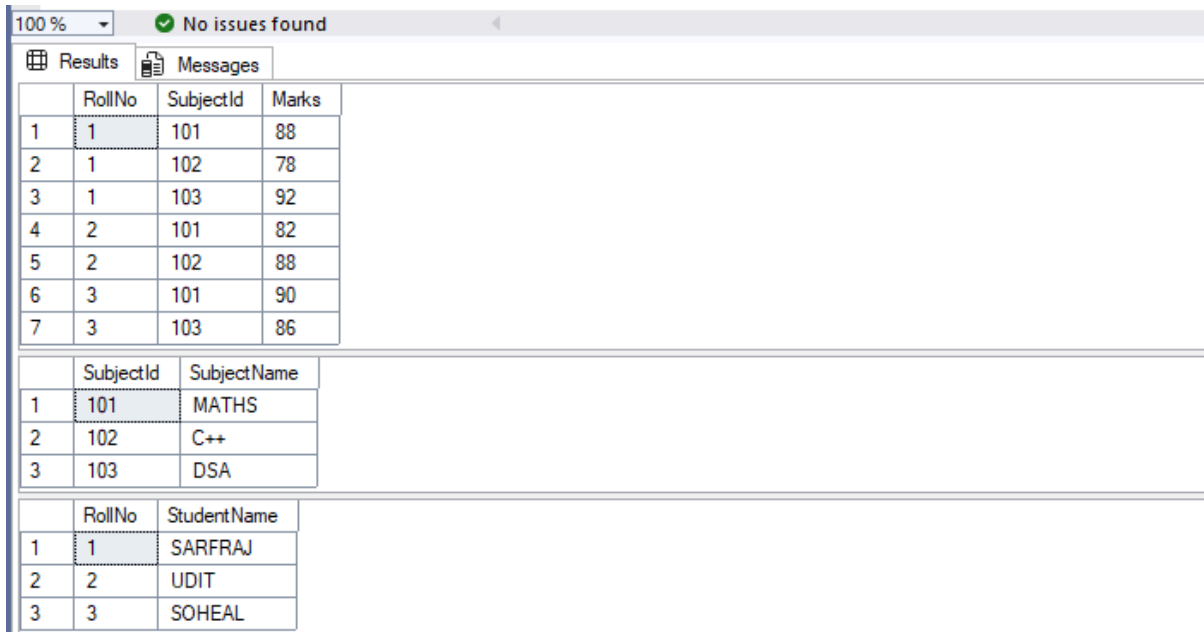
### *SQL Source Code:*

```

SELECT * FROM MarksheetDetails;
SELECT * FROM SubjectDetails;
SELECT * FROM StudentDetails;

```

## Figure Description:



The screenshot displays the results of three SQL SELECT queries. The first table, 'Marks', shows 7 rows of student marks. The second table, 'Subject', shows 3 rows of subject names. The third table, 'Student', shows 3 rows of student names.

	RollNo	SubjectId	Marks
1	1	101	88
2	1	102	78
3	1	103	92
4	2	101	82
5	2	102	88
6	3	101	90
7	3	103	86

	SubjectId	SubjectName
1	101	MATHS
2	102	C++
3	103	DSA

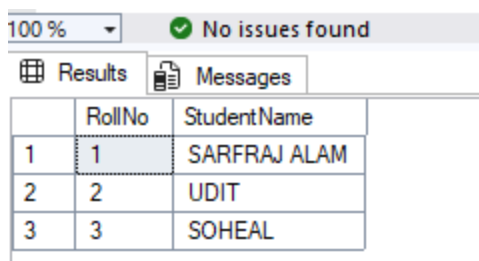
	RollNo	StudentName
1	1	SARFRAJ
2	2	UDIT
3	3	SOHEAL

**Fig 4.1:** The figure displays the output of SQL SELECT queries showing the contents of the StudentDetails, SubjectDetails, and MarksheetDetails tables after inserting data. It confirms that referential integrity is maintained across the related tables using foreign key constraints.

**Step3:** To perform an update operation in a database table using the UPDATE SQL command.

*SQL Source Code:*

```
--update
UPDATE StudentDetails
SET StudentName = 'SARFRAJ ALAM'
WHERE RollNo = 1;
select * from StudentDetails
```



The screenshot shows the result of the UPDATE query. The 'StudentDetails' table now has 'SARFRAJ ALAM' as the student name for RollNo 1.

	RollNo	StudentName
1	1	SARFRAJ ALAM
2	2	UDIT
3	3	SOHEAL

**Fig 4.2:** The output shows that the StudentName for RollNo = 1 has been successfully updated to SARFRAJ ALAM, while other records remain unchanged.

**Step4:** To delete a row of data with primary detail of RollNo=1.

```
--delete
delete from MarksheetDetails
where RollNo=1;
select * from MarksheetDetails;
--primary key is refrenced as foreign key in
```

```
delete from StudentDetails
where RollNo=1;
```

100 % No issues found

Results Messages

	RollNo	SubjectId	Marks
1	2	101	82
2	2	102	88
3	3	101	90
4	3	103	86

	RollNo	StudentName
1	2	UDIT
2	3	SOHEAL

**Fig 4.3:** The output shows that the record with RollNo = 1 has been successfully deleted from both MarksheetDetails and StudentDetails. The remaining data confirms that the delete operation worked as expected without violating any foreign key constraints.

Similarly

## Step 1: Creating Tables with Foreign Keys

Three relational tables are created in SQL Server Management Studio (SSMS):

1. **ProductDetails** – Stores product information including name, price, discount, and description.
2. **CustomerDetails** – Stores customer information such as name, email, mobile, DOB, and address.
3. **TransactionDetails** – Stores transaction records, linked to ProductDetails and CustomerDetails using foreign keys to maintain relational integrity.

This design ensures that every transaction must reference existing customer and product records, thus enforcing referential integrity.

SQL Source Code:

```
-- Drop in correct dependency order
DROP TABLE IF EXISTS TransactionDetails;
DROP TABLE IF EXISTS ProductDetails;
DROP TABLE IF EXISTS CustomerDetails;

-- Create tables
CREATE TABLE ProductDetails (
    ProductID INT NOT NULL,
    ProductName VARCHAR(100),
    Price FLOAT,
    Discount FLOAT,
    Descriptionproduct varchar(100),
    PRIMARY KEY (ProductID)
```

```

);

CREATE TABLE CustomerDetails (
    CustomerID INT NOT NULL,
    CustomerName VARCHAR(100),
    Email VARCHAR(100),
    Mobile BIGINT,
    DOB DATE,
    Address VARCHAR(100),
    PRIMARY KEY (CustomerID)
);

CREATE TABLE TransactionDetails (
    TransactionID INT NOT NULL,
    Traxndatetime DATETIME,
    CustomerID INT,
    ProductID INT,
    Paymentstatus CHAR,
    PRIMARY KEY (TransactionID),
    FOREIGN KEY (CustomerID) REFERENCES CustomerDetails(CustomerID),
    FOREIGN KEY (ProductID) REFERENCES ProductDetails(ProductID)
);

```

## Step 2: Inserting Data into Tables

Insert sample records to validate table structure and relationships.

```

INSERT INTO ProductDetails(ProductID, ProductName, Price, Discount,
Descriptionproduct) VALUES
(101, 'Coke', 70, 3.5, 'Chilled soft drink'),
(102, 'Fanta', 72, 3.5, 'soft drink'),
(103, 'Sprite', 74, 3.5, 'Carbonated soft drink');

INSERT INTO CustomerDetails (CustomerID, CustomerName, Email, Mobile, DOB, Address)
VALUES
(1, 'Sarfracj', 'sarfracj@example.com', 9812345678, '2000-05-10', 'Kathmandu'),
(2, 'Udit', 'Udit@example.com', 9811122233, '1999-07-12', 'Pokhara'),
(3, 'Soheal', 'soheal@example.com', 9800012345, '2001-01-01', 'Biratnagar');

INSERT INTO TransactionDetails (TransactionID, Traxndatetime, CustomerID, ProductID,
Paymentstatus) VALUES
(1, '2025-07-15 10:30:00', 1, 101, 'Y'),
(2, '2025-07-15 11:00:00', 2, 101, 'N'),
(3, '2025-07-15 12:15:00', 3, 101, 'Y');

```

## Step 3: Retrieving Data Using SELECT Queries

Verify the inserted data by querying all records from each table.

Sql code:

```

SELECT * FROM ProductDetails;
SELECT * FROM CustomerDetails;
SELECT * FROM TransactionDetails;

```

Results

Messages

	ProductID	ProductName	Price	Discount	Descriptionproduct
1	101	Coke	70	3.5	Chilled soft drink
2	102	Fanta	72	3.5	soft drink
3	103	Sprite	74	3.5	Carbonated soft drink

	CustomerID	CustomerName	Email	Mobile	DOB	Address
1	1	Sanjog	sanjog@example.com	9812345678	2000-05-10	Kathmandu
2	2	Swaggy	swaggy@example.com	9811122233	1999-07-12	Pokhara
3	3	Chapri	chapri@example.com	9800012345	2001-01-01	Biratnagar

	TransactionID	Traxndatetime	CustomerID	ProductID	Paymentstatus
1	1	2025-07-15 10:30:00.000	1	101	Y
2	2	2025-07-15 11:00:00.000	2	101	N
3	3	2025-07-15 12:15:00.000	3	101	Y

✔ Query executed successfully.

🔒 (local) (16.0 RTM)

GT\gauta (73)

**Fig 4.4:** Displays the output of SQL SELECT queries showing the contents of ProductDetails, CustomerDetails, and TransactionDetails tables after inserting data. It confirms that referential integrity is maintained by foreign key constraints linking transactions to valid customers and products.

## Conclusion:

This lab demonstrated the practical use of **DML commands** (INSERT, SELECT, UPDATE, and DELETE) in a relational database. By designing tables with **foreign key constraints**, we enforced **referential integrity** and ensured that all operations followed relational rules. The results verified the proper insertion, modification, and deletion of records while maintaining data consistency across the tables.



