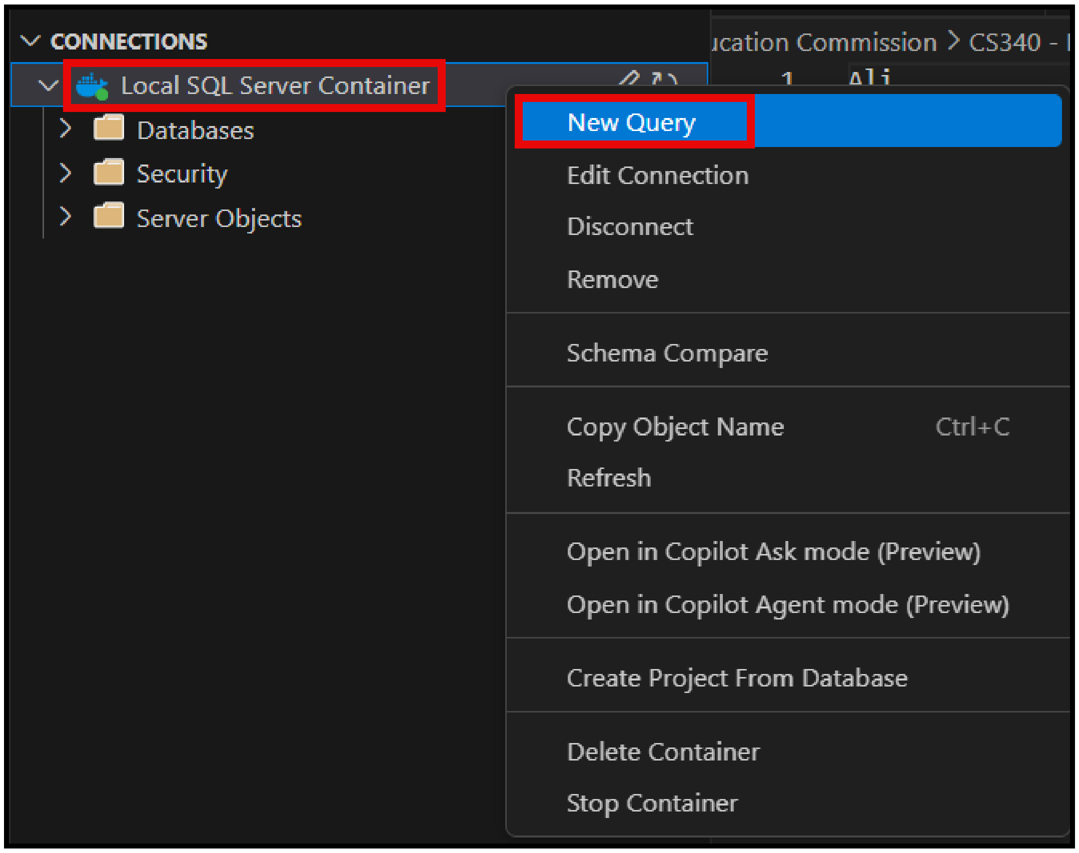
SQL Server Tutorial 2: Relational Model

Create SQL File in Visual Studio Code (vscode):

From vscode, create a new `.sql` file. Right click on your sql server connection (in the below example, the connection is named `Local SQL Server Connection`), and select `New Query`:

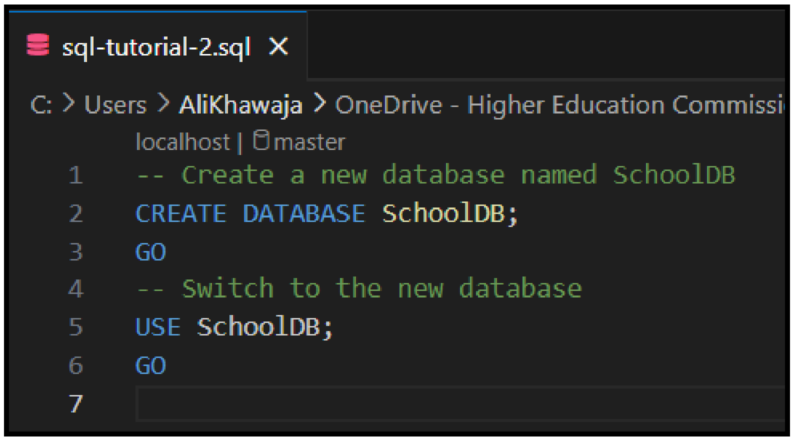


Save the newly created file as `sql-tutorial-2.sql` `(or any name of your preference).`

Creating a Database

To create a new database in SQL Server, use the following SQL commands:

```
CREATE DATABASE SchoolDB;
GO
USE SchoolDB;
GO
```


sql-tutorial-2.sql
C: > Users > AliKhawaja > OneDrive - Higher Education Commissi
localhost | master

```sql
-- Create a new database named SchoolDB
CREATE DATABASE SchoolDB;
GO
-- Switch to the new database
USE SchoolDB;
GO
```

Click on the green arrow to execute the sql commands.

In SQL Server, the `GO` statement is not a SQL command itself, it's a batch separator used by the tools which you use to connect to SQL Server.

It tells the SQL Server client tool to **send the current batch of SQL statements to the server for execution**.

If you have multiple commands in a file in vscode, you can select specific commands and press the green arrow to execute only those selected commands without the need of `GO` statement.

2. Creating a Schema

Schemas help organize database objects. Here's how to create a schema named 'Academic':

```
CREATE SCHEMA Academic AUTHORIZATION dbo;
```

EXPLANATION

```
CREATE SCHEMA Academic
```

This creates a new schema named Academic.

```
AUTHORIZATION dbo
```

This assigns ownership of the schema to the dbo (database owner) user. It means the **dbo** will have control over the schema and its objects.

***Note**: We don't need to discuss security at this point. We'll have detailed discussion about security, roles, and users in the later part of the course.*

***Note**: You can also just type the commands without go; select the specific command completely and press the green execute arrow icon. If you want to run the whole file with all the commands, then its better to put GO between them, to ensure previous command is sent and successfully executed before the next one is sent.*

3. Creating Tables with Constraints

Students Table

```
CREATE TABLE Academic.Students (
    StudentID INT PRIMARY KEY,
    FirstName NVARCHAR(50) NOT NULL,
    LastName NVARCHAR(50) NOT NULL,
    DateOfBirth DATE CHECK (DateOfBirth < GETDATE()),
    Email NVARCHAR(100) UNIQUE,
    Gender CHAR(1) DEFAULT 'M',
    CONSTRAINT CK_Gender CHECK (Gender IN ('M', 'F'))
);
```

Courses Table

```
CREATE TABLE Academic.Courses (
    CourseID INT PRIMARY KEY,
    CourseName NVARCHAR(100) NOT NULL,
    Credits INT CHECK (Credits BETWEEN 1 AND 5)
);
```

Enrollments Table (Relationship Table)

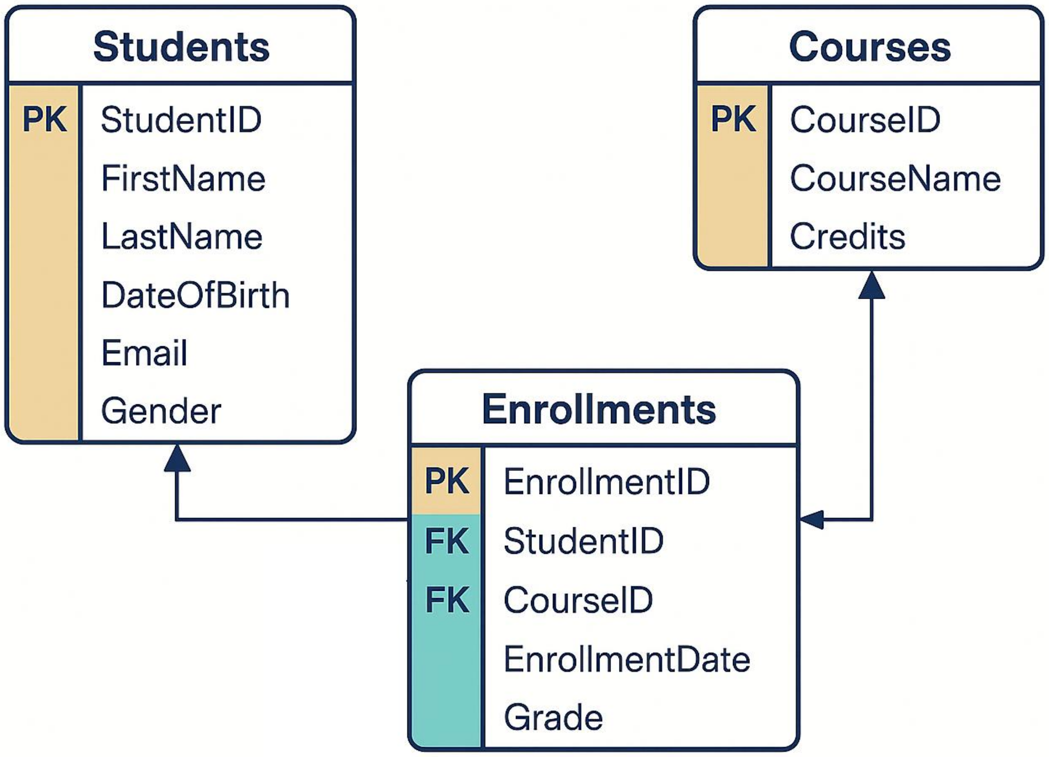
```
CREATE TABLE Academic.Enrollments (
    EnrollmentID INT PRIMARY KEY,
    StudentID INT FOREIGN KEY REFERENCES Academic.Students(StudentID),
    CourseID INT FOREIGN KEY REFERENCES Academic.Courses(CourseID),
    EnrollmentDate DATE DEFAULT GETDATE(),
    Grade CHAR(1) CHECK (Grade IN ('A', 'B', 'C', 'D', 'F'))
);
```

4. Summary of Constraints Used

- **PRIMARY KEY**: Uniquely identifies each row.
- **FOREIGN KEY**: Establishes relationships between tables.
- **NOT NULL**: Ensures a column must have a value.
- **UNIQUE**: Ensures all values in a column are different.
- **CHECK**: Validates data based on a condition.
- **DEFAULT**: Provides a default value if none is supplied.

5. Entity-Relationship (ER) Diagram

The following diagram illustrates the relationships between the Students, Courses, and Enrollments tables:


Students
PK
StudentID
FirstName
LastName
DateOfBirth
Email
Gender
Courses
PK
CourseID
CourseName
Credits
Enrollments
PK
EnrollmentID
FK
StudentID
FK
CourseID
EnrollmentDate
Grade