Lecture 20: SQL Introduction (DDL)

Tables, Constraints, and Schema Evolution

Z2004: Database Management Systems



IIT Madras Zanzibar

Monday, 20 April 2026

Lecture Outline

Learning Objectives

What you should be able to do after this lecture

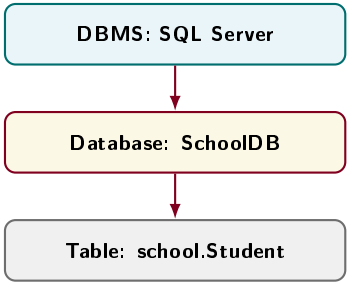
1. Explain SQL **sublanguages** (DDL/DML/DCL/TCL) and choose correct **data types** in SQL Server.
2. Use **SSMS** to connect, run scripts, and interpret error messages.
3. Create a **database**, optionally create a **schema**, and create **tables**.
4. Enforce correctness using **constraints**: `PRIMARY KEY`, `FOREIGN KEY`, `UNIQUE`, `NOT NULL`, `CHECK`, `DEFAULT`.
5. Modify schema with `ALTER TABLE` and understand `DROP` vs `TRUNCATE` vs `DELETE`.

Tooling: Microsoft SQL Server + **SQL Server Management Studio (SSMS)**.

DBMS vs Database vs Table

- **DBMS**: the software (SQL Server) that stores data and runs SQL.
- **Database**: a container managed by the DBMS (e.g., `SchoolDB`).
- **Table**: a structure inside a database that stores rows and columns.
- **Row**: one record (one student).
- **Column**: one attribute (Name, Dept, GPA).

In this lecture: we learn how to **define** these structures using **DDL**.



Relational vocabulary

Math term	SQL term
Relation	Table
Tuple	Row
Attribute	Column
Domain	Allowed values (type/constraint)

The goal of DDL is to define the **domain** correctly using **data types + constraints**.

Example domain rules:

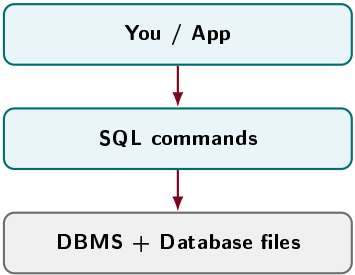
- GPA must be between 0 and 4.
- Dept code must be exactly 5 characters.
- Every student must have a name (NOT NULL).

SQL: Why it exists

SQL is the standard language used to **talk to a relational DBMS**.

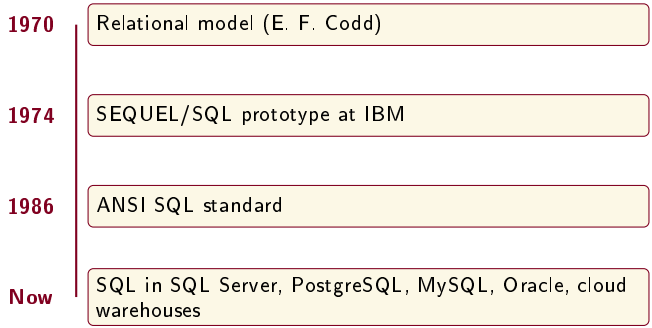
What SQL gives you:

- A **declarative** way to define data structures and ask questions.
- **Constraints** to keep data correct.
- A consistent language across many systems (SQL Server, PostgreSQL, MySQL, ...).



SQL: A brief histor

- 1970: Codd proposes the **relational model**.
- 1974: IBM develops **SEQUEL** (early SQL).
- 1986: SQL becomes an **ANSI standard**; later ISO.
- Today: SQL remains the industry default for relational data.



SQL is a family of commands

- **DDL (Data Definition Language):** define structure
  - `CREATE`, `ALTER`, `DROP`, `TRUNCATE`
- **DML (Data Manipulation Language):** work with rows
  - `SELECT`, `INSERT`, `UPDATE`, `DELETE`
- **DCL (Data Control Language):** permissions
  - `GRANT`, `REVOKE`
- **TCL (Transaction Control Language):** transactions
  - `BEGIN TRAN`, `COMMIT`, `ROLLBACK`

In this lecture (L20), we focus on **DDL**: building the database structure correctly so DML works smoothly later.

SQL Server data types: what they mean

Category	Type	Use it for (example)
Integers	`INT`	counts/IDs: `Age=21`, `SID=1005`
Exact decimals	`DECIMAL(5,2)`	money/grades: `GPA=3.75`, `Price=49.99`
Approx decimals	`FLOAT`	scientific values: `temperature=36.6123`
Text (ASCII)	`VARCHAR(50)`	emails, codes (English-only)
Text (Unicode)	`NVARCHAR(50)`	names with accents/Arabic/Swahili
Fixed text	`CHAR(5)`	fixed-size codes: `'DB101'`
Dates	`DATE`	`'2026-04-16'`
Date+time	`DATETIME2`	`'2026-04-16 09:30:00'`
Boolean-ish	`BIT`	`IsActive=1`

Important: use `DECIMAL` (not `FLOAT`) for money/grades to avoid rounding errors.

Strings: CHAR vs VARCHAR vs NVARCHAR

`CHAR(n)` **(fixed length)**

- Always stores exactly `n` characters (pads with spaces).
- Use for truly fixed codes (e.g., `'TZ'`, `'DB101'`).

`VARCHAR(n)` **(variable length, non-Unicode)**

- Stores up to `n` characters; uses only what you need.
- OK if you are sure text is only English/ASCII.

`NVARCHAR(n)` **(variable length, Unicode)**

- Supports multilingual text (recommended for names).
- Typically uses more storage than `VARCHAR`.

```
-- Compare storage behaviour
DECLARE @a CHAR(5)      = 'DB1';
DECLARE @b VARCHAR(5)   = 'DB1';
DECLARE @c NVARCHAR(5) = N'Znz';

-- CHAR(5) stores 'DB1  '
-- VARCHAR stores 'DB1'
```

UG default: use `NVARCHAR` for names/text unless you have a strong reason not to.

NULL is not 0 and not an empty string

NULL means: *unknown / missing / not provided.*

- `NULL` is **not equal** to anything (even another NULL).
- This is why SQL uses `IS NULL` and `IS NOT NULL`.
- Empty string '' is a real value (a known, empty text).

Why it matters in DDL:

- `NOT NULL` forbids missing values.
- `CHECK` constraints must handle NULL correctly.

```sql
-- NULL comparisons
SELECT CASE WHEN NULL = NULL
  THEN 'equal' ELSE 'not equal'
     END;

-- Correct way
SELECT CASE WHEN NULL IS NULL
  THEN 'is null' ELSE 'not null'
     END;

-- Empty string is different
SELECT CASE WHEN '' = ''
  THEN 'equal' ELSE 'not equal'
     END;
```

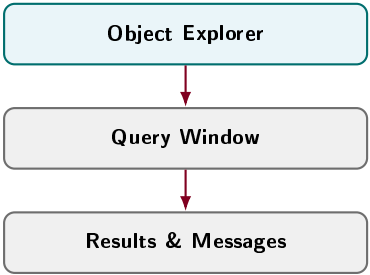
SSMS: What you need to know

Main panes

- **Object Explorer:** databases, tables, columns (refresh often).
- **Query Window:** where you write SQL.
- **Results + Messages:** output tables and error messages.

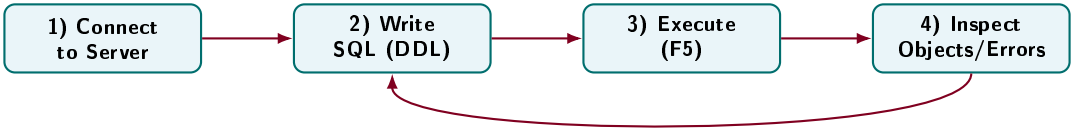
How to debug

- Read the **first error** in the *Messages* tab.
- Fix one error, run again.
- Always confirm the current database: `SELECT DB_NAME();`



SSMS: The Workflow

You will do the same loop again and again:



Tip: if a command fails, read the **exact error line** and fix one issue at a time.

SSMS: Your First Commands

In SSMS, open **New Query** and run:

```
-- What server am I connected to?
SELECT @@SERVERNAME AS ServerName;

-- Which database am I currently using?
SELECT DB_NAME() AS CurrentDB;
```

If your current database isn't what you want, use `USE <dbName>;` (shown next).

Writing a clean SQL script in SSMS

Good scripts are easy to re-run and easy to read.

```
-- 1) Always start by checking where you are
SELECT DB_NAME() AS CurrentDB;

-- 2) Switch to the correct DB before creating tables
USE SchoolDB;

-- 3) Use comments to separate tasks
-- === Create tables ===
-- === Add constraints ===
```

Note: SQL Server also supports `GO` as a batch separator in SSMS, but we keep scripts beginner-simple and avoid it unless needed.

Naming conventions (so your DB stays readable)

Tables/columns

- Tables: singular nouns: `Student`, `Course`, `Enrolment`
- Primary keys: `StudentID` or `SID` (be consistent)
- Foreign keys: same name as referenced PK (e.g., `SID` in `Enrolment`)

Constraints

- `PK_Student`, `FK_Enrolment_Student`, `UQ_Student_Email`, `CHK_Student_GPA`

Why it matters:

- Easier debugging (error messages show constraint names).
- Easier teamwork (everyone understands object purpose).

Step 1: Create a Database (SSMS)

A **database** is a container for tables, views, procedures, and security.

```
-- Create a new database
CREATE DATABASE SchoolDB;

-- Switch into it
USE SchoolDB;
```

In SSMS: Object Explorer → Databases → (Refresh) to see `SchoolDB`.

Step 2: Schemas (dbo vs custom)

A **schema** is a namespace inside a database. Most beginner scripts use dbo.

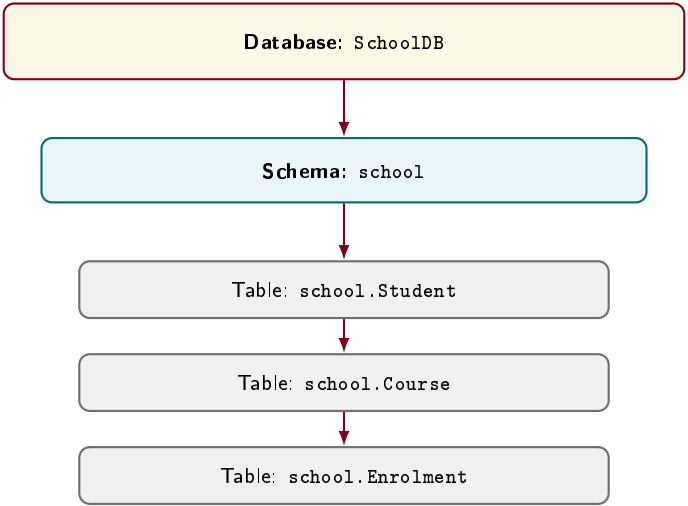
```
-- Create a schema for the course
CREATE SCHEMA school;

-- A table name is: schema.table
-- Example: school.Student
```

Why schemas?

- Organize objects (`school.*`, `hr.*`, `sales.*`).
- Control permissions at schema level.
- Avoid name clashes.

Where do tables live?



Design Rule: Start from the ER Idea

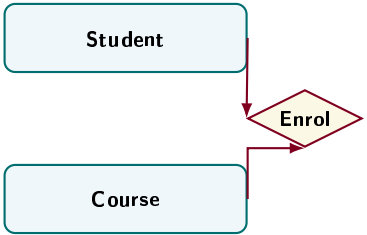
Before writing `CREATE TABLE`, answer:

Entities (tables)

- Student
- Course
- Enrolment (relationship between Student and Course)

Keys

- Student: `SID` (primary key)
- Course: `CID` (primary key)
- Enrolment: (`SID`, `CID`) (composite primary key)



Data Types: The Practical Shortlist

For this course, you can cover 90% of cases with:

- Integers: `INT`, `BIGINT`
- Text: `VARCHAR(n)` (ASCII), `NVARCHAR(n)` (Unicode)
- Money/grades: `DECIMAL(p,s)` (exact)
- Dates: `DATE`, `DATETIME2`
- Flags: `BIT`
- IDs: `IDENTITY(1,1)` (auto-number)

Common mistake: using `FLOAT` for money/marks (rounding errors).

Constraints: your data guardrails

Constraints stop bad data *at the door*.

- **PRIMARY KEY**: unique row ID (and NOT NULL).
- **FOREIGN KEY**: must match a key in another table.
- **UNIQUE**: no duplicates (e.g., Email).
- **NOT NULL**: value is required.
- **CHECK**: rule like GPA range.
- **DEFAULT**: value if user doesn't provide one.

Typical examples:

- GPA: `CHECK (GPA BETWEEN 0 AND 4)`
- Credits: `DEFAULT 3`
- Email: `UNIQUE`

Key idea: constraints are part of the **schema** (DDL), not part of the data (DML).

Natural key vs surrogate key

Natural key (meaningful)

- Example: `CID='DB101'`
- Pros: readable, already exists
- Cons: can change; may be long

Surrogate key (artificial)

- Example: `SID INT IDENTITY(1,1)`
- Pros: small, stable, fast joins
- Cons: needs an extra UNIQUE constraint for natural values (like Email)

Surrogate key example

Student
SID = 1, 2, 3, ...
Name = 'Asha'
Email = 'a@uni.ac.tz'

Create Table 1: Student

```
USE SchoolDB;

CREATE TABLE school.Student (
    SID         INT IDENTITY(1,1) NOT NULL,
    Name        NVARCHAR(100)     NOT NULL,
    Dept        CHAR(5)           NOT NULL,
    GPA         DECIMAL(3,2)      NULL,
    DateJoined  DATE              NOT NULL DEFAULT (CAST(GETDATE() AS DATE)),

    CONSTRAINT PK_Student PRIMARY KEY (SID),
    CONSTRAINT CHK_Student_GPA CHECK (GPA IS NULL OR (GPA BETWEEN 0.00 AND
        4.00))
);
```

Read it top-to-bottom: columns first, then constraints.

Create Table 2: Course

```
CREATE TABLE school.Course (
    CID      CHAR(5)          NOT NULL,
    CName    NVARCHAR(200)    NOT NULL,
    Credits  SMALLINT         NOT NULL DEFAULT (3),

    CONSTRAINT PK_Course PRIMARY KEY (CID),
    CONSTRAINT CHK_Course_Credits CHECK (Credits IN (1,2,3,4))
);
```

We used a natural key for Course (e.g., `DB101`). That's fine when codes are stable.

Create Table 3: Enrolment (junction table)

```
CREATE TABLE school.Enrolment (
    SID    INT      NOT NULL,
    CID    CHAR(5)  NOT NULL,
    Grade  CHAR(2)  NULL,

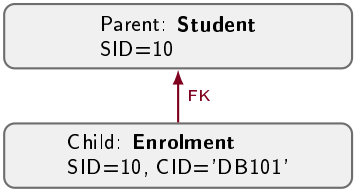
    CONSTRAINT PK_Enrolment PRIMARY KEY (SID, CID),
    CONSTRAINT FK_Enrolment_Student FOREIGN KEY (SID)
        REFERENCES school.Student(SID)
        ON DELETE CASCADE,
    CONSTRAINT FK_Enrolment_Course FOREIGN KEY (CID)
        REFERENCES school.Course(CID)
        ON DELETE CASCADE,
    CONSTRAINT CHK_Enrolment_Grade CHECK (Grade IN ('A','B','C','D','F') OR
        Grade IS NULL)
);
```

Foreign keys: delete/update actions

A foreign key can define what happens when the parent row changes.

- **ON DELETE CASCADE**: delete child rows automatically.
- **ON DELETE NO ACTION**: block delete if children exist (safe default).
- **ON UPDATE NO ACTION**: block updating a referenced key (common).

UG rule of thumb: use `NO ACTION` unless you are very sure you want automatic deletion.



Quick note: keys and indexes

- In SQL Server, **PRIMARY KEY** and **UNIQUE** usually create an **index** automatically.
- Indexes make lookups and joins faster, but they add overhead to inserts/updates.
- We'll study indexes properly later (B-trees, query optimization).

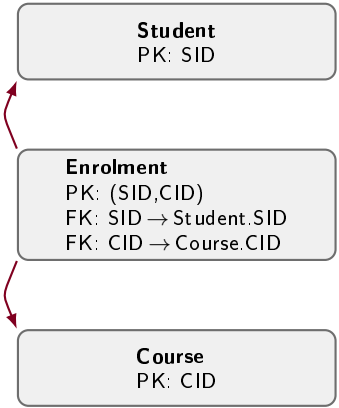
Keys & Constraints

Primary key (PK): uniquely identifies a row.

- Cannot be NULL.
- Can be **single-column** or **composite**.

Foreign key (FK): enforces a valid reference to another table.

- Prevents orphan rows.
- Can cascade deletes/updates (use carefully).



ALTER (DDL): definition & basic syntax

ALTER is a Data Definition Language (DDL) command used to **modify the structure** of an existing database object (most commonly a table) **without deleting it**.

Common `ALTER TABLE` operations:

- **ADD column** (append a new column)
- **DROP COLUMN** (remove a column + its data)
- **ALTER/MODIFY COLUMN** (change type/size/NULLability)
- **ADD/DROP constraint** (PK/FK/UNIQUE/CHECK/DEFAULT)

```sql
-- Add a column
ALTER TABLE table_name
ADD column_name datatype;

-- Drop a column
ALTER TABLE table_name
DROP COLUMN column_name;

-- Change a column (SQL Server)
ALTER TABLE table_name
ALTER COLUMN column_name datatype;

-- Add a constraint (example)
ALTER TABLE table_name
ADD CONSTRAINT UQ_Name
UNIQUE (column_name);
```

ALTER TABLE: Typical changes

```sql
-- Add a new column
ALTER TABLE Student
  ADD Email NVARCHAR(255) NULL;

-- Make Email unique (no duplicates)
ALTER TABLE Student
  ADD CONSTRAINT UQ_Student_Email UNIQUE (Email);

-- Change a column's definition
ALTER TABLE Student
  ALTER COLUMN Name NVARCHAR(150) NOT NULL;
```

Rule: if a column already contains NULLs, you must clean data first before switching to `NOT NULL`.

DROP vs TRUNCATE vs DELETE (one slide you must memorize)

```sql
-- Remove the table object
DROP TABLE Enrolment;

-- Remove all rows, keep the table
TRUNCATE TABLE Student;

-- Remove some rows (with filter)
DELETE FROM Student
WHERE Dept = 'CSE';
```

	DROP	TRUNCATE	DELETE
Removes rows	✓	✓	✓
Keeps table	×	✓	✓
WHERE allowed	×	×	✓
Typical speed	Fast	Very fast	Slower

Safety tip: never run `DROP`/`TRUNCATE` in the wrong database. Always confirm `SELECT DB_NAME();`.

Mini lab: prove your schema works

After creating the tables, insert a few rows and test the FK.

```
USE SchoolDB;

-- Insert a student
INSERT INTO Student (Name, Dept, GPA)
VALUES (N'Asha Ali', 'CSE  ', 3.80);

-- Insert a course
INSERT INTO Course (CID, CName, Credits)
VALUES ('DB101', N'Introduction to Databases', 3);

-- Enrol the student (must succeed if FK is correct)
INSERT INTO Enrolment (SID, CID, Grade)
VALUES (1, 'DB101', NULL);

-- Check what we have
SELECT * FROM Student;
SELECT * FROM Enrolment;
```

Lecture Summary

Key Takeaways

- In **SSMS**, the loop is: connect →write SQL →run →inspect objects/errors.
- DDL builds the structure: `CREATE DATABASE`, `CREATE SCHEMA`, `CREATE TABLE`.
- Correctness comes from **constraints** (PK/FK/UNIQUE/NOT NULL/CHECK/DEFAULT).
- Relationships are enforced with **foreign keys** (and a junction table for many-to-many).
- `ALTER` evolves safely; `DROP`/`TRUNCATE` are powerful and dangerous.

Next lecture: SQL DML — `SELECT`, `WHERE`, `ORDER BY`.