Data Manipulation Language (DML)

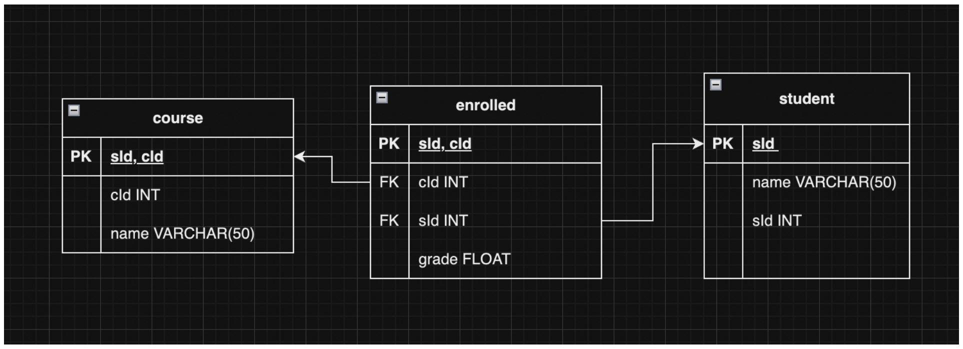
Data Manipulation Language (DML) is the part of SQL that deals with managing the data inside tables. While Data Definition Language (DDL) changes the structure (the blueprint), DML is how we actually add, change, and remove information.
The most common DML commands are:

- **INSERT INTO**: Add new rows to a table.
- **UPDATE**: Change data in existing rows.
- **DELETE FROM**: Remove rows from a table.

DML is how we work with the real data—adding new students, updating grades, deleting dropped courses, and so on.

Schema Setup

Let's assume we already have this schema (defined earlier in the DDL section):



```
CREATE TABLE student (
    sId INT PRIMARY KEY,
    name VARCHAR(255) NOT NULL
);

CREATE TABLE course (
    cId INT PRIMARY KEY,
    name VARCHAR(255) NOT NULL
);

CREATE TABLE enrolled (
    sId INT,
    cId INT,
    grade FLOAT,
    PRIMARY KEY (sId, cId),
    FOREIGN KEY (sId) REFERENCES student(sId),
    FOREIGN KEY (cId) REFERENCES course(cId)
);
```

Inserting Data

We use `INSERT INTO` to add new rows to our tables.

Adding Students

```
INSERT INTO student (sId, name) VALUES (1, 'Alice');
INSERT INTO student (sId, name) VALUES (2, 'Bob');
INSERT INTO student (sId, name) VALUES (3, 'Sara');
```

Adding Courses

```
INSERT INTO course (cId, name) VALUES (480, 'Database Systems');
INSERT INTO course (cId, name) VALUES (394, 'Data Engineering');
```

Enrolling Students in Courses

We can also insert into the `enrolled` table, giving initial grades:

```
INSERT INTO enrolled (sId, cId, grade) VALUES (1, 480, 0);
INSERT INTO enrolled (sId, cId, grade) VALUES (2, 394, 0);
INSERT INTO enrolled (sId, cId, grade) VALUES (1, 394, 0);
```

Notes:

- If you omit a column in the `INSERT`, it must either have a default value or allow `NULL`.
- If you try to insert a row with a duplicate primary key (e.g., another student with `sId = 1`), the database will reject it.
- Foreign key constraints ensure only valid students and valid courses can be added into `enrolled`.

Updating Data

We use `UPDATE` to modify existing rows in our tables.

```
UPDATE enrolled
SET grade = 4.0
WHERE sId = 2 AND cId = 394;
```

This sets Bob's grade in Data Engineering to 4.0.

Deleting Data

We use `DELETE FROM` to remove rows from our tables.

```
DELETE FROM student
WHERE sId = 2;
```

This removes Bob from the student table.
Caution:

- If Bob is still enrolled in any courses, the foreign key constraint may prevent deletion.
- To handle this, databases can be set to `ON DELETE CASCADE`, which automatically removes related enrollments, or you must manually delete from `enrolled` first.
- Without a `WHERE` clause, `DELETE FROM student;` would remove *all students*.

Risks of UPDATE or DELETE Without a WHERE Clause

Important: Always use a WHERE clause! Without it, the database would update *every row* in the table. For example:

```
UPDATE enrolled SET grade = 4.0;
```

would give every student a grade of 4.0. Similarly:

```
DELETE FROM enrolled;
```

would remove *all* enrollments from the table.

Security Risks and Protection

Accidentally running an UPDATE or DELETE without a WHERE clause is one of the most common and dangerous mistakes in SQL. From a security and data protection perspective:

- **Data loss:** Without a WHERE clause, an entire table of data can be corrupted or deleted in a single command.
- **Privilege management:** Production databases often restrict UPDATE and DELETE permissions to administrators or specific roles to reduce risk.
- **Safe mode in tools:** Many SQL management tools (e.g., MySQL Workbench, pgAdmin) include settings to warn or block queries that affect all rows.
- **Transactions:** Wrapping updates and deletes inside a transaction allows you to `ROLLBACK` if the command affects more rows than expected. (We are not covering transactions in this course.)
- **Row count checks:** Good practice is to check how many rows will be updated/deleted using a SELECT first:

  ```
      SELECT COUNT(*) FROM enrolled WHERE grade < 2.0;
  ```

  and then apply the DELETE or UPDATE with the same WHERE clause.
- **Audit logging:** Some databases log every change to critical tables, allowing administrators to detect or even revert unintended mass updates/deletes.

Best Practice for Engineers

Always:

1. Write the UPDATE/DELETE with a WHERE clause.
2. Run a SELECT with the same condition to preview the affected rows.
3. Use transactions so mistakes can be rolled back safely.