Introduction to DBMS and Relational Model

Contents

Code and Notes are @ https://github.com/nishithjain/Design_Pen

1. Introduction Database Management System

- What is a database?
    - Let's split the word Database into two. Data & Base.
    - What is a base?
        - The part on which it stands. So, database means, where data resides.
    - It is not random data, but it is organized collection of interrelated data around a particular use case.
        - For example: An educational institute might store data about students, classes, batches. These data are interrelated.

- What is Database Management System?
    - We need to manage the data, like, update the existing data, delete the data.
    - We need a system to do it efficiently.

- Why do we need DBMS?
    - If we were to store the data in files.
        - **Data Redundancy and Consistency**: Data is often duplicated in different files, leading to inconsistency when one file is updated, but others are not.
        - **Data Integrity**: Maintaining data integrity (e.g., ensuring data types or constraints) is difficult and must be manually managed.
        - **Data Security**: Files offer limited or no mechanisms to restrict access, making it harder to implement granular security policies.
        - **Concurrency Control**: Handling simultaneous access by multiple users can lead to conflicts (e.g., data corruption, overwriting issues).
        - **Data Backup and Recovery**: You must manually manage backups, and recovery in case of failures is challenging and error-prone.
        - **Efficient Query Processing**: Retrieving specific data from large files requires scanning the entire file, which can be slow and inefficient.
        - **Data Independence**: Changing the structure of files often requires changes to the application code.
        - **Data Relationships**: Managing relationships (e.g., between customers and orders) is complex and prone to errors.
        - **Performance Optimization**: Performance tuning (e.g., indexing, caching) requires custom implementation, which can be inefficient.

Types of Database Management Systems

- Types are characterized by 2 things.
    - How they store the data.
    - What requirements do they fulfill.

- Relational DBMS (SQL database)
    - A database system that follows the **relational model** to store data.
    - Example: MySQL, Postgres, Oracle SQL

- Non-Relational DBMS (No SQL database)
    - Example:
        - Key-Value: Redis
        - Document Model: MongoDB
        - Columnar: Casandra
        - Graph: Neo4J

- What is Relational Model?
    - Relational Model is a type of data model.
    - Data model is collection of concepts that we use to describe data in database. Concepts are nothing but tables, schema, key, etc...

- Mathematical Definition of Relational Model.
    - The database is represented as a collection of multiple relations. Each relation will have a set of tuples and these may be related to a tuple in another relation.
        - Here relation is nothing but tables. There for, the database is represented as a collection of multiple tables.
    - Let's see properties of a relation (table)
        - The definition says, each relation will have set of tuples. Here tuple means row.
    - The definition also says 'set of tuples'. Set has 2 properties:
        - No duplicates are allowed in set.
        - Order doesn't matter in set. Order of columns also doesn’t matter.
    - Value in each cell is atomic.
        - No multivalued cell such as list, JSON, array.

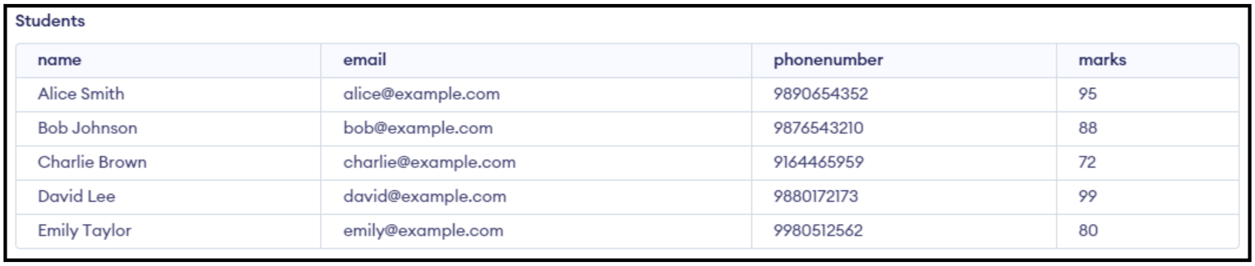
2. Key

- Some set of columns whose values will be present in exactly in 1 row.
- Set of columns that help to uniquely identify a row.
    - Example: A Table `Student` has `name`, `email`, `phonenumber` as columns, we can consider `email` as key. Because, `email` is unique.
- A key must present in every row.

Types of keys

Super Key:

- A set of columns whose values can be used to uniquely identify a single row.
    - Example: A Table `Student` has `name`, `email`, `phonenumber`, `marks` as columns as shown below…



Students

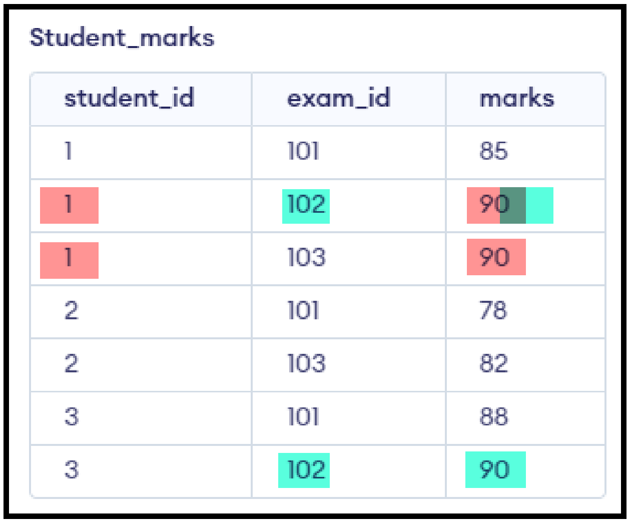
name	email	phonenumber	marks
Alice Smith	alice@example.com	9890654352	95
Bob Johnson	bob@example.com	9876543210	88
Charlie Brown	charlie@example.com	9164465959	72
David Lee	david@example.com	9880172173	99
Emily Taylor	emily@example.com	9980512562	80

- Candidates for super key are `email`, `phonenumber`, {`name`, `email`}, {`name`, `phonenumber`}, {`email`, `phonenumber`}.

Column Name	Is Super Key
`name`	✗
`email`	✓
`phonenumber`	✓
`marks`	✗
`{name, marks}`	✗
`{name, email}`	✓
`{name, phonenumber}`	✓
`{email, phonenumber}`	✓

Candidate Key:

- A Super key of minimum size such that if we remove any column from it, the remaining columns is no longer a super key. In other words, a candidate key is a super key that does not have any unnecessary columns.
- A table can have more than one candidate key. When this happens, one of them is usually chosen as the primary key.
    - Example: We have a table `student_marks`, which has the data for every student, for every exam they gave, we are storing their marks.



Student_marks

student_id	exam_id	marks
1	101	85
1	102	90
1	103	90
2	101	78
2	103	82
3	101	88
3	102	90

Column Name	Is Super Key	Comments	Is Candidate Key
`student_id`	✗		Should be Super Key
`exam_id`	✗		Should be Super Key
`marks`	✗		Should be Super Key
`{student_id, marks}`	✗	Same student can have same marks in different exam.	Should be Super Key
`{exam_id, marks}`	✗	2 students can have same marks in same exam.	Should be Super Key
`{student_id, exam_id}`	✓	1 student can take a particular exam only once.	✓
`{student_id, exam_id, marks}`	✓	1 student, 1 exam, marks.	✗

- Every super key is a candidate key? No
- Every candidate key is a super key? Yes

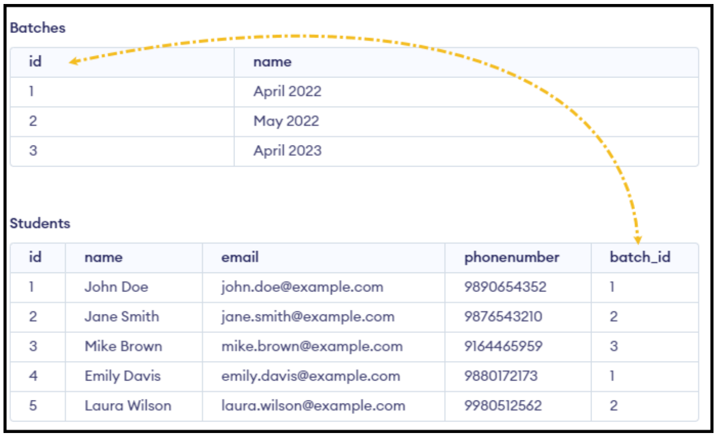
Primary Key:

- We don't want useless columns in Primary key. Hence, primary key is any one of the candidate keys that is specified as a key when creating a new table.
    - Database enforces no 2 rows have duplicate values in those set of columns (Primary key columns).
    - Database will not allow empty values in those columns (Primary key columns).

- If we create a table Student with only Name as column. Since we should have a primary key column. We must create another column as ID column as primary key.

Foreign Key:

- Allows us to uniquely identify a row of another table.
- We have use cases where we want to store relation between multiple tables.



Batches

id	name
1	April 2022
2	May 2022
3	April 2023

Students

id	name	email	phonenumber	batch_id
1	John Doe	john.doe@example.com	9890654352	1
2	Jane Smith	jane.smith@example.com	9876543210	2
3	Mike Brown	mike.brown@example.com	9164465959	3
4	Emily Davis	emily.davis@example.com	9880172173	1
5	Laura Wilson	laura.wilson@example.com	9980512562	2

SQL Used

```
CREATE TABLE students (
    name VARCHAR(100) NOT NULL,
    email VARCHAR(100) NOT NULL UNIQUE,
    phonenumber VARCHAR(15) NOT NULL,
    marks INT CHECK (marks >= 0 AND marks <= 100)
);

INSERT INTO students (name, email, phonenumber, marks)
VALUES ('Alice Smith', 'alice@example.com', '9890654352', 95);
INSERT INTO students (name, email, phonenumber, marks)
VALUES ('Bob Johnson', 'bob@example.com', '9876543210', 88);
INSERT INTO students (name, email, phonenumber, marks)
VALUES ('Charlie Brown', 'charlie@example.com', '9164465959', 72);
INSERT INTO students (name, email, phonenumber, marks)
VALUES ('David Lee', 'david@example.com', '9880172173', 99);
INSERT INTO students (name, email, phonenumber, marks)
VALUES ('Emily Taylor', 'emily@example.com', '9980512562', 80);

CREATE TABLE IF NOT EXISTS student_marks (
    student_id INT NOT NULL,
    exam_id INT NOT NULL,
    marks INT CHECK (marks >= 0 AND marks <= 100)
    );
```

```sql
INSERT INTO student_marks (student_id, exam_id, marks) VALUES
(1, 101, 85),
(1, 102, 90),
(1, 103, 90),
(2, 101, 78),
(2, 103, 82),
(3, 101, 88),
(3, 102, 90);

drop table students;
drop table student_marks;

CREATE TABLE IF NOT EXISTS batches (
    id INT PRIMARY KEY,
    name VARCHAR(50) NOT NULL
);

-- Insert some rows into the batches table with manual IDs
INSERT INTO batches (id, name) VALUES
(1, 'April 2022'),
(2, 'May 2022'),
(3, 'April 2023');

CREATE TABLE IF NOT EXISTS students (
    id INT PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    email VARCHAR(100) NOT NULL UNIQUE,
    phonenumber VARCHAR(15) NOT NULL,
    batch_id INT,
    FOREIGN KEY (batch_id) REFERENCES batches(id)
);

-- Insert some rows into the students table with manual IDs and batch associations
INSERT INTO students (id, name, email, phonenumber, batch_id) VALUES
(1, 'John Doe', 'john.doe@example.com', '9890654352', 1),
(2, 'Jane Smith', 'jane.smith@example.com', '9876543210', 2),
(3, 'Mike Brown', 'mike.brown@example.com', '9164465959', 3),
(4, 'Emily Davis', 'emily.davis@example.com', '9880172173', 1),
(5, 'Laura Wilson', 'laura.wilson@example.com', '9980512562', 2);
```

3. Schema Design

- As a software developer, we do prepare design document.
  - Class Diagram - How will you implement application. (LLD)
  - Architectural Diagram - What infrastructure layers will be there. (HLD)
  - Schema Design - What tables will be there in our Database.

- Database Schema:
  - How will you store the data to be able to handle the given set of requirements.
  - Schema is blueprint of a real database. A pictorial representation of how database is going to be structured.
- Why do we need Schema design?
  - Should be able to handle all the requirements.
  - Handle requirements efficiently.
  - Avoid anomalies (issues such as redundancy which can cause inconsistency and more storage).

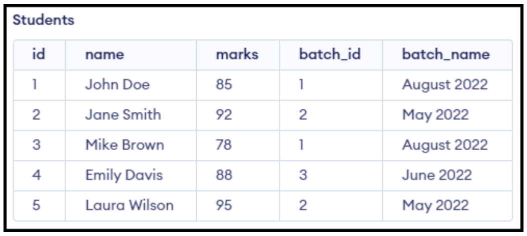
Anomalies

- Issue due to redundancy.
- There are 3 types of anomalies...
  - Insertion anomaly
  - Deletion anomaly
  - Update anomaly
- When we have redundancy in database these anomalies happen.

Insertion Anomaly

- Suppose there is a table Students with `id`, `name`, `marks`, `batch_id`, `batch_name` as columns as shown below...

```
CREATE TABLE IF NOT EXISTS students (
    id INT PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    marks INT CHECK (marks >= 0 AND marks <= 100),
    batch_id INT NOT NULL,
    batch_name VARCHAR(50) NOT NULL
);
```



Students

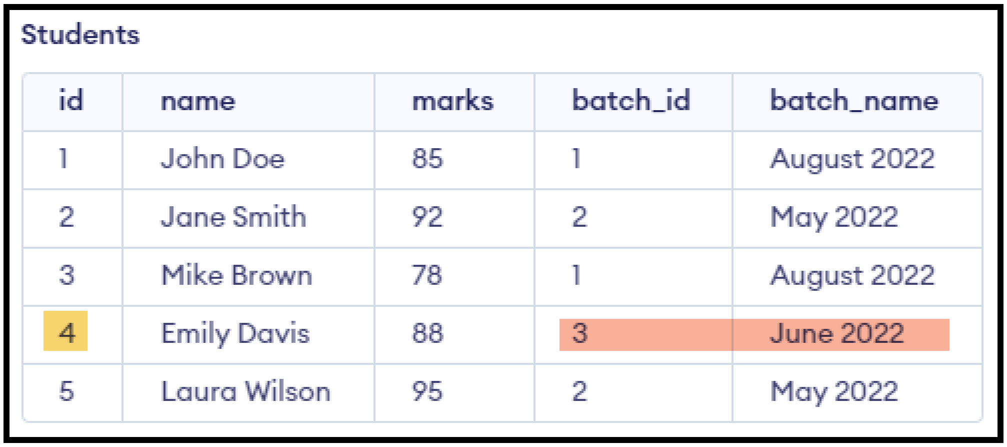
id	name	marks	batch_id	batch_name
1	John Doe	85	1	August 2022
2	Jane Smith	92	2	May 2022
3	Mike Brown	78	1	August 2022
4	Emily Davis	88	3	June 2022
5	Laura Wilson	95	2	May 2022

- New requirement has come: Create a new batch named `December 2022`. No students in that `December 2022` batch yet.
- Since, no students are present in that batch, it is not possible to create a new row (`id` is primary key, name cannot be null.)

- Is this table `Students` is correctly created? Answer is no. There is lot of redundancy in the table.
- The `batch_name` column has redundancy. Note that `'August 2022'` is repeated many times. Because of this redundancy we have insertion anomaly.
- Insertion anomaly is *inability to store data about a particular entity till the time we have data about something else*.

Deletion Anomaly

- New request has come. We need to delete the student with `id` 4 (Imagine that this student has registered for `June 2022` batch first and is the only student in that batch)
- If we delete that student, the batch information is also deleted. Note that the student with `id` 4 is in `June 2022` and there is only 1 student in that batch.



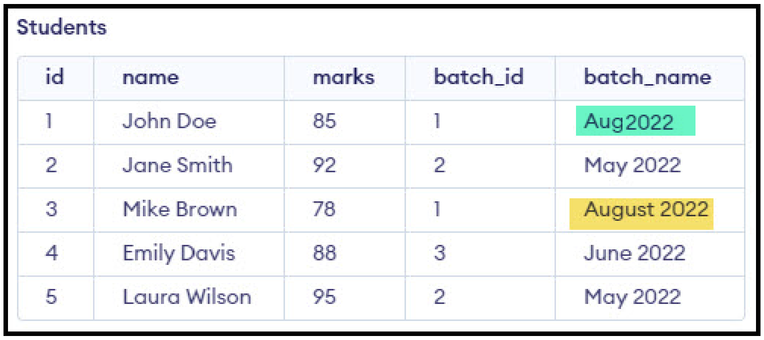
Students

id	name	marks	batch_id	batch_name
1	John Doe	85	1	August 2022
2	Jane Smith	92	2	May 2022
3	Mike Brown	78	1	August 2022
4	Emily Davis	88	3	June 2022
5	Laura Wilson	95	2	May 2022

- Deletion anomaly is, at the time of deleting something, we might end up deleting something else.

Update Anomaly

- A new requirement has come where we need to change the batch_name `'August 2022'` to `'Aug2022'`.
- Imagine, we have written a query to update from `'August 2022'` to `'Aug2022'`. After updating couple of rows, machine has gone down… The state of table will look as show below…



Students

id	name	marks	batch_id	batch_name
1	John Doe	85	1	Aug2022
2	Jane Smith	92	2	May 2022
3	Mike Brown	78	1	August 2022
4	Emily Davis	88	3	June 2022
5	Laura Wilson	95	2	May 2022

- Update anomaly is, at the time of updating something, we might end up in inconsistencies.

Database Normalization

- It is the techniques that we use to handle redundancy.
- There is something called normal forms. Normal forms are guidelines used in database design to reduce redundancy and improve data integrity by organizing data into tables.
    - 1NF
    - 2NF
    - 3NF
    - 4NF
    - 5NF
    - 6NF
    - BCNF (Boyce Codd Normal Form)
- We don't use these NF in practice. We never use normal forms in reality. There is more practical way to approach database design.

How to do Schema Design in Practice

- Schema design of Scaler.
- Let's list out the requirements.
    - Scaler has multiple students.
    - Scaler has multiple batches. Each student belong to exactly one batch at a time. one batch can have multiple students.
    - Every batch has a current instructor.
    - Every batch has multiple classes.
    - 1 class may involve students from multiple batchs
    - Every student has a student buddy.
    - Every student has a mentor.
    - For every mentor we store their company and number of session the mentor has taken.
    - We have to store for every batch a student belong to, date of joining that batch.
    - For every student for every class we have to store attendance.

Steps

1. Find all the nouns that are there in the requirements.
    - Out of all the nouns find the nouns that we want to store information about.
    - Create 1 table for each such noun.
    - Good Practices:
        - Name of each table should ideally plural.
        - Represent table name in snake case. (Example: mentor_sessions)
    - Nouns identified:
        - students
        - batches
        - instructors
        - classes
        - mentors
        - companies

2. For each of these nouns, find what all we need to store.
    - Create an id column
    - If no realation with another noun, create a column for that (primitive attributes). For example, Students will have name and name has no relation with other nouns. Hence it is a primitive attribute.

```sql
CREATE TABLE students (
    id INT PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    email VARCHAR(100) UNIQUE NOT NULL,
    phone_number VARCHAR(15),
    graduation_year INT
);
```

```sql
CREATE TABLE batches (
    id INT PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    start_date DATE,
    number_of_students INT
);
```

```sql
CREATE TABLE instructors (
    id INT PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    email VARCHAR(100) UNIQUE NOT NULL,
    average_rating DECIMAL(3, 2),
    years_of_experience INT
);
```

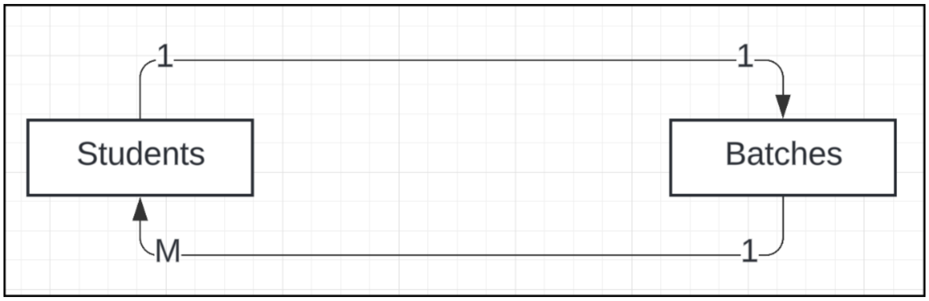
```sql
CREATE TABLE classes (
    id INT PRIMARY KEY,
    start_time TIME,
    title VARCHAR(100),
    endtime TIME
);
```

```sql
CREATE TABLE mentors (
    id INT PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    email VARCHAR(100) UNIQUE NOT NULL,
    number_of_mentees INT,
    average_rating DECIMAL(3, 2)
);
```

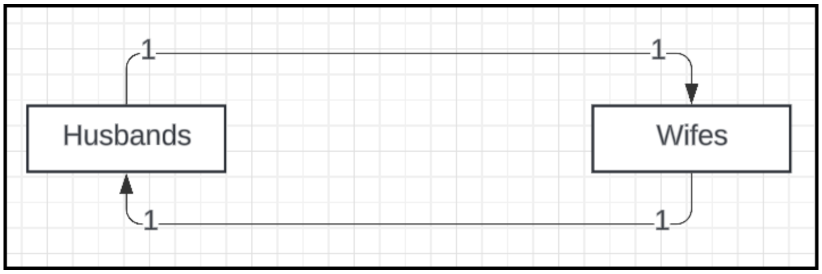
```sql
CREATE TABLE companies (
    id INT PRIMARY KEY,
    name VARCHAR(100) NOT NULL
);
```

3. Now we have to represent relations.
    - But how to represent relation?
    - For relation,
        - Which two tables are related.
        - What relation between entities (Find cardinality).

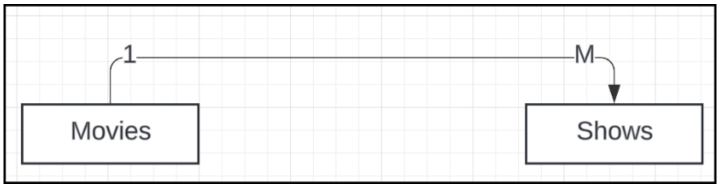
- What is Cardinality?
    - *Cardinality* refers to the relationship between two entities. It defines how entities in one table relate to entities in another.
    - Let's say we have Students and Batches
        - 1 Student is allotted to 1 Batches.
        - 1 Batches can have multiple Students.



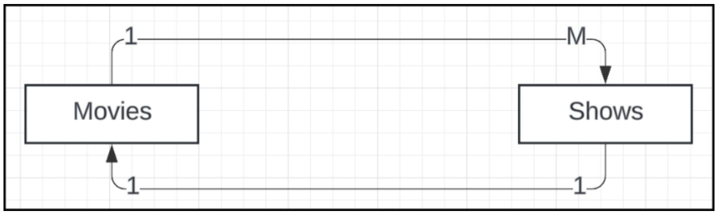
- Let’s say we have Husbands and Wifes table
    - 1 husband has 1 wife.
    - 1 wife has 1 husband.



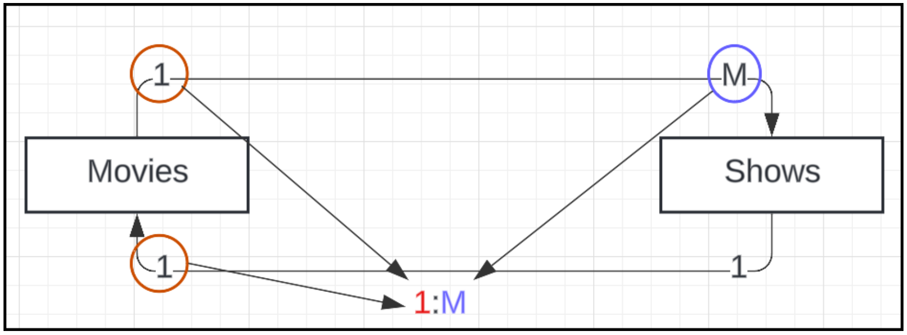
- There are 4 types cardinality :
    1. 1:1
    2. 1:m
    3. m:1
    4. m:m
- Steps to find the cardinality…
    - Let’s say we have Movies and Shows table…
        - Go from Left to right and ask the question 1 movie how many shows?



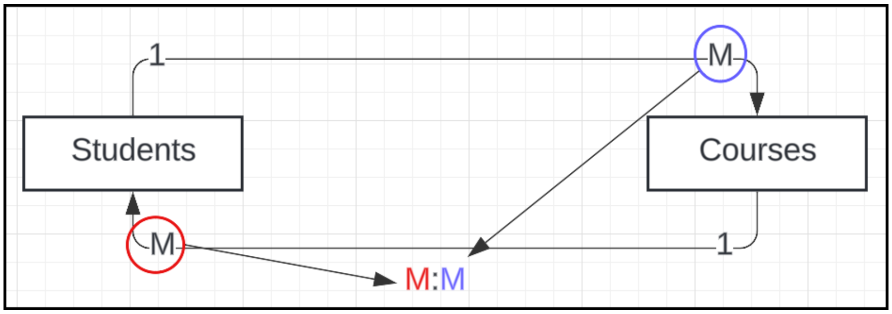
        - Go from Right to left and ask the question 1 show can have how many movies?



- If there no m, put 1 else put M.



- For relation in column, if cardinality is
  - 1:1, id of any 1 side on other side.
  - M:1, id of 1 side on m side. (Because we can have a list on 1 side and we cannot store list)
  - 1:M, id of 1 side on m side. (Because we can have a list on 1 side and we cannot store list)
  - M:M, Mapping table. (We have to create a new table, because there will be list on both sides)



- We create a new table, Student_Course with student_id, courese_id as columns.