LLD : Schema Design

Agenda

→ What is Schema Design

→ How to find entities → tables

→ How to represent relationships in Schema

→ Code in Spring Boot ← We will never write SQL Queries

m : m

↓

Mapping table

PostgreSQL DB

⇒ Create entities

⇒ let Spring Boot create

tables for this

→ Never persisting the data

→ Need to persist data in

→ Database

Class : Blueprint of entities

Schema : Blueprint of how entities will be stored in a database

1 Do the class diag

↓

do the Schema design

Assume we are talking about SQL DB

⇒ How are you going to store rel^n

→ How are you going to store data

ont relationships

Good Schema

1. Less Redundancy
2. Fast Retrieval

Our Schema design should keep in mind the Access pattern

Case Study

→ Design schema for Scaler Academy

VI

Use Case

→ Students should be able to login

→ Students should have a profile with their

Name, email and college

ID, username, password

attributes of Student

Anything which has a set of info attached to it (attribute / data)

Approach

① Find diff entities in the requirement

② Draw a class diagram

VI

CD

Students

class Student {

long

String username;

String name;

String email;

String password;

String college;
}

ID: Unique identifier that helps us when persuing data.

Every class should map: 1 to a table in database

V2

Use cases

→ All previous use cases are still there

→ Scaler has multiple courses

→ Every student can enroll for 0/1/2 courses

→ Every course only has a name.

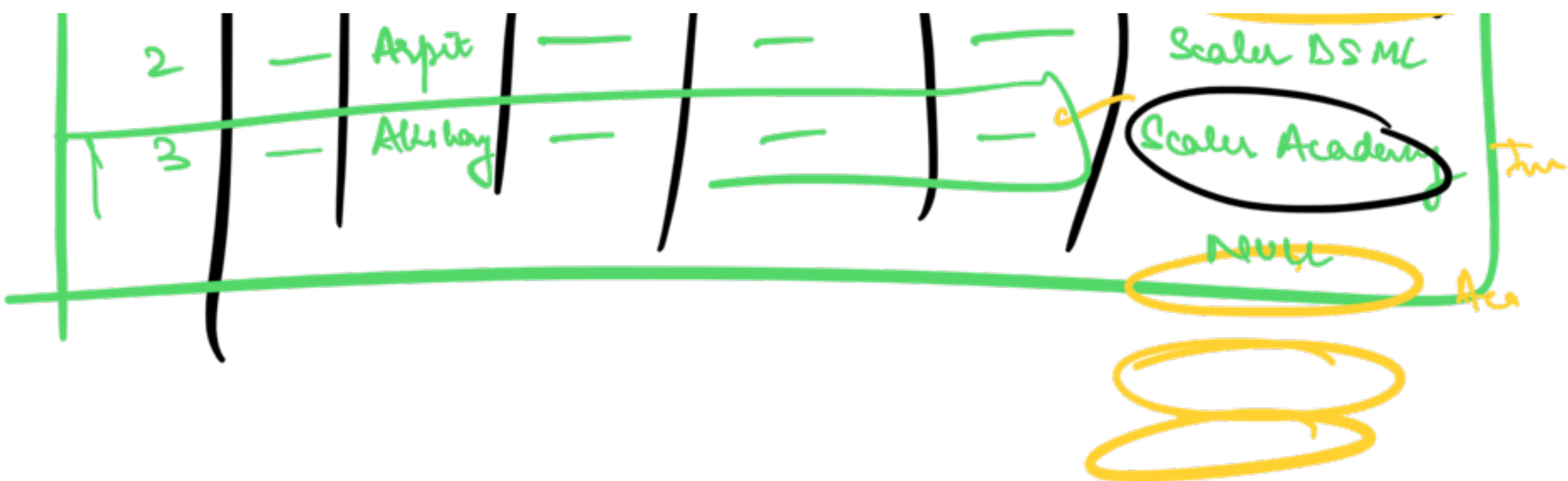
Sol^n 1

→ Don't add any more table

Students

String

id	UN	Name	Email	password	college	course_name
1	—	Name	—	—	—	Scaler Academy, IV



① Using extra storage due to redundancy

② Storage errors

③ If update Name, have to go through multiple rows

④ What if more attributes related to course

⑤ Not able to store a course till there is a student enrolled into it.

Solution

- id
- Name
- Username
- email
- password
- college
- Course

- id
- name

class Student {

long id;

String name;

String username;

String email

String password

String college

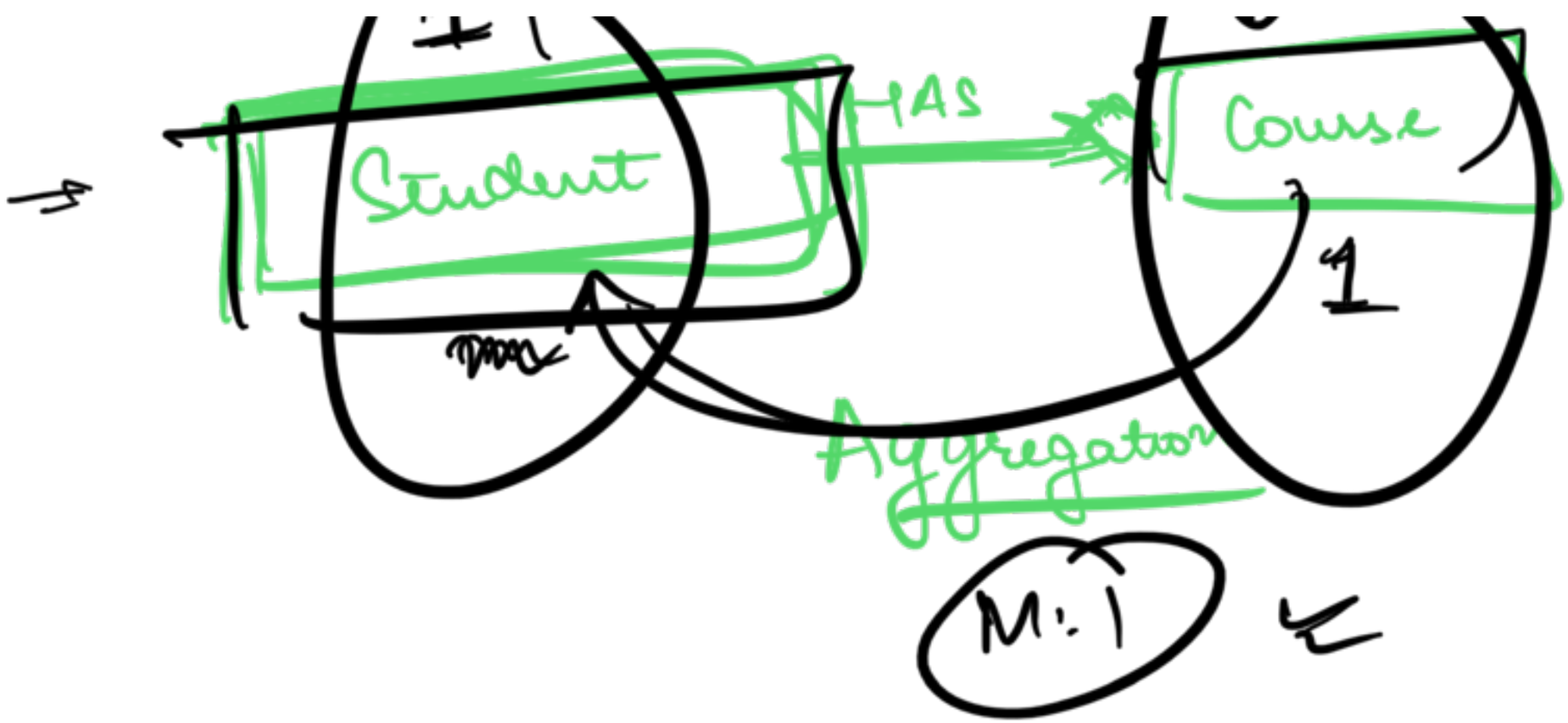
Course course;

}

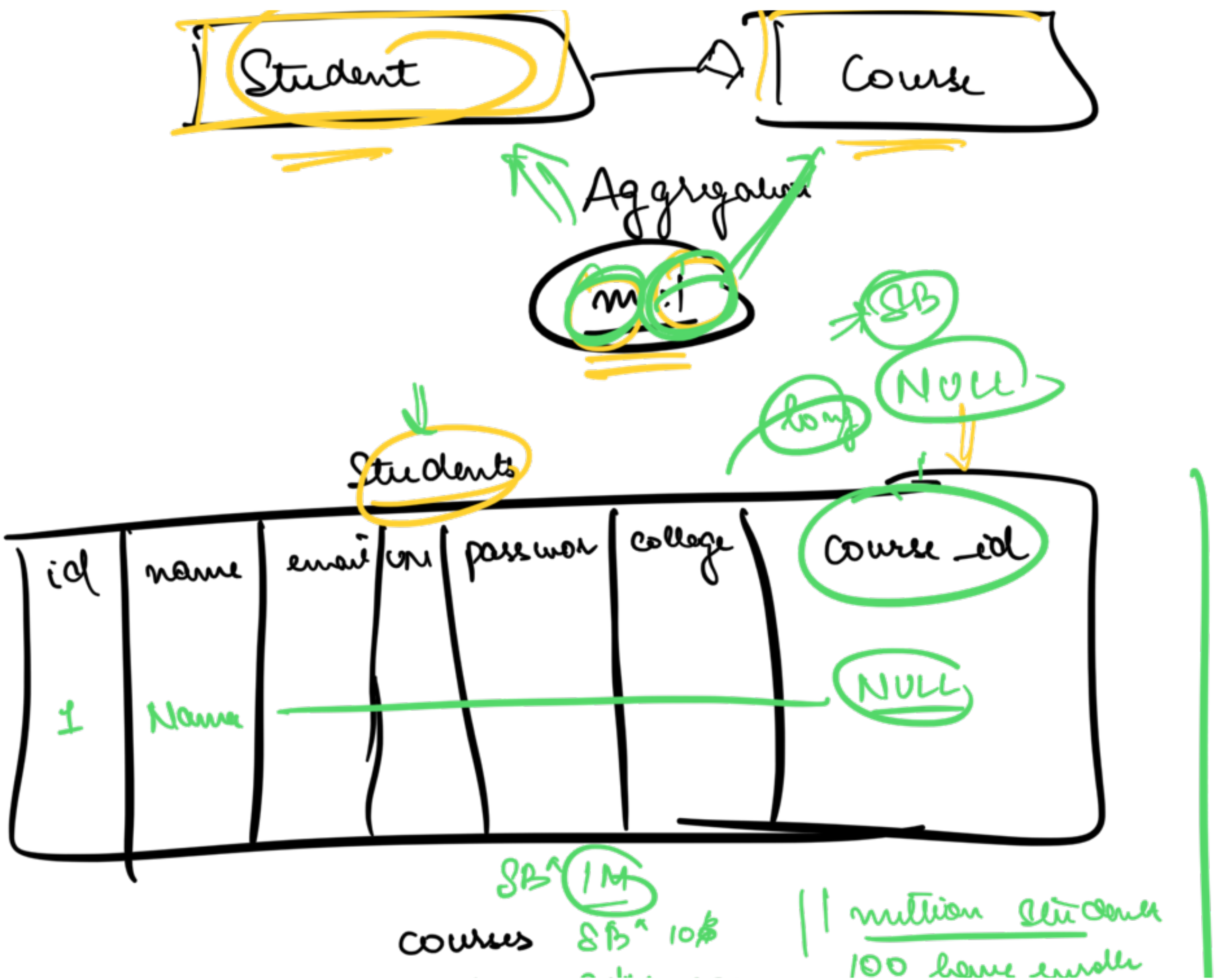
class Course {

long id;

String name


Student
HAS
Course
1
Aggregation
M:1


1:1
1:m
m:1
m:m


Student
Course
Aggregation
BB
NULL
Students
id
name
email
password
college
course_id
1
Name
NULL
SB
IM
courses
SB 10$
million students
100 have enrolled

1   8 KB = 1000 + 8 MB

id	name

lot of nulls

Waste of Space

1 : M

OR

M : 1

Put the id of 1 side in m side

Sparse Relationship

→ separate mapping

Students

id	name	email	dn	password	college

Courses

id	Name

Student_course_Mapping

Student id	Course id
1	2

1 : M

OR

M : 1

⇒

Put the c'd of 1 in m side

BUT

If rel^n is sparse

⇒ Separate Mapping Table

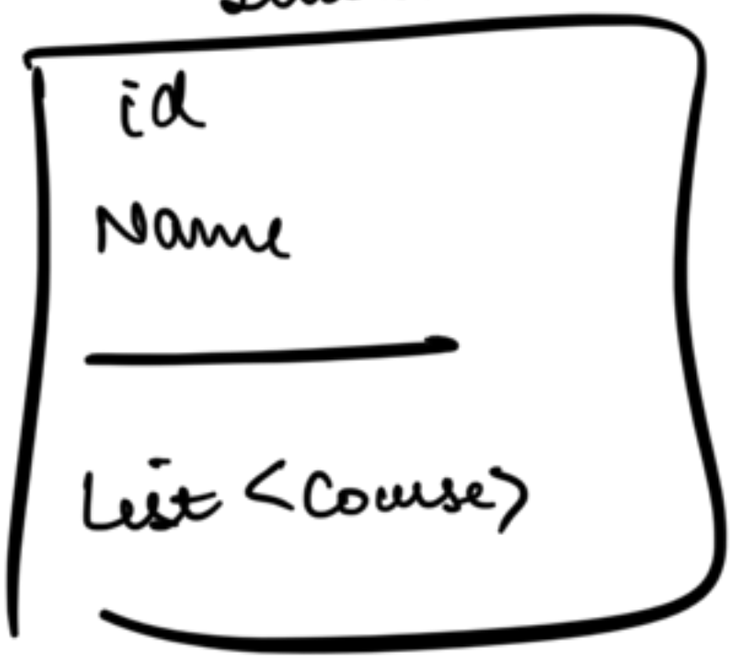
V3

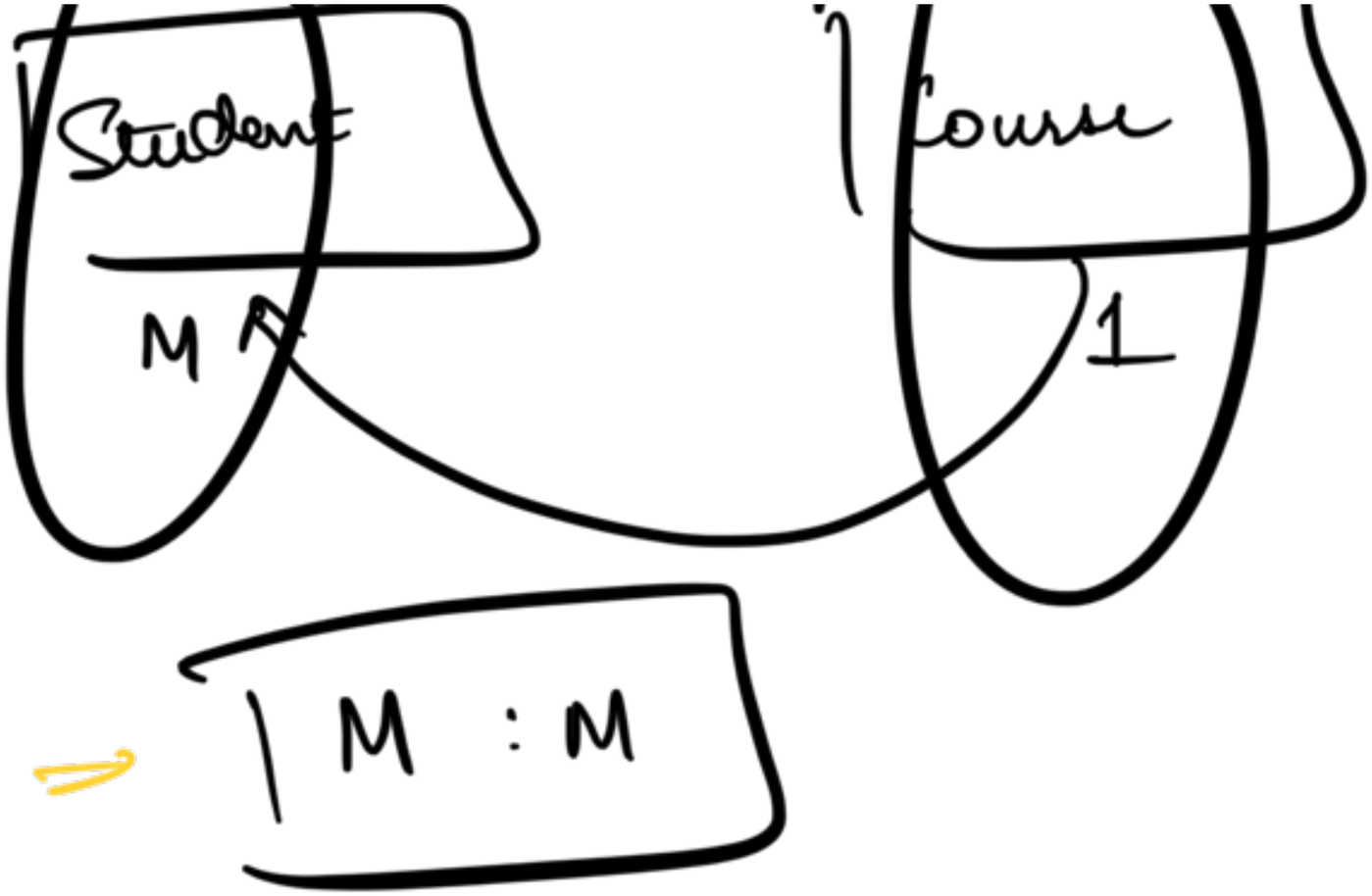
→ Every student can opt for EXACTLY 4 courses

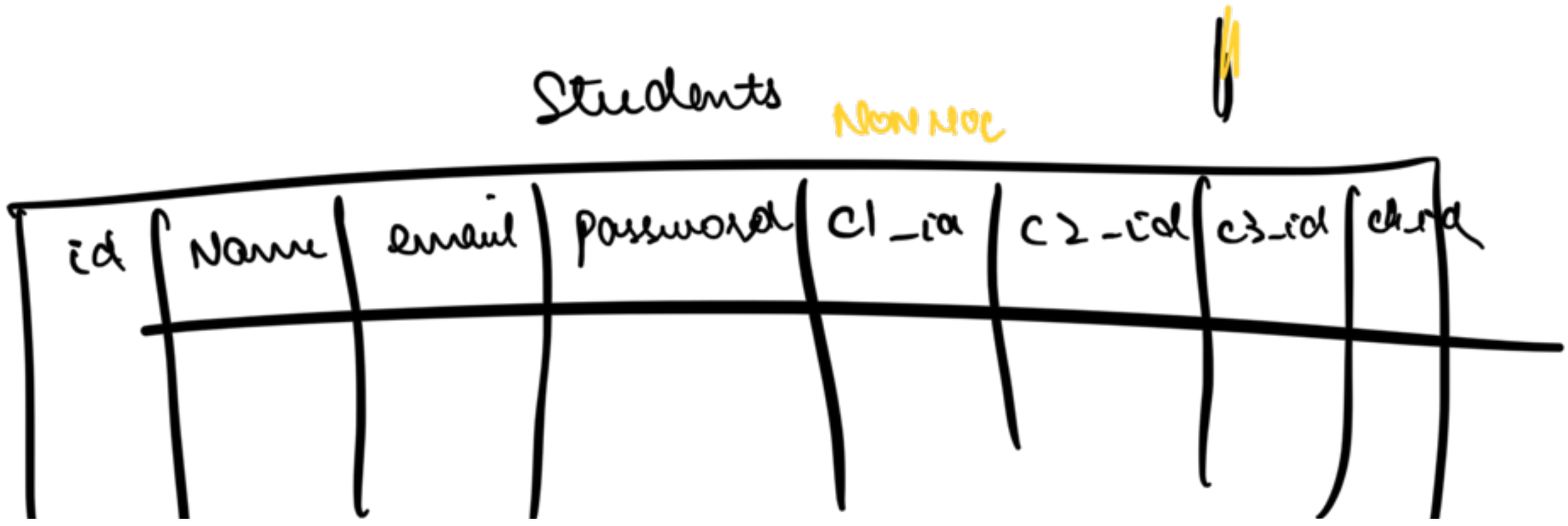
Student

⇒

Student




Student
Course
M
1
M : M


Students
NON NOC
id
Name
email
password
C1_id
C2_id
C3_id
C4_id

P1 : Check if user id is enrolled in course c1

2, 3, 45

Select * from users

where c1_id == c1

or c2_id == c1

or c3_id == c1

or c4_id == c

AN user_id = U1

⇒ find total # of students enrolled in a course

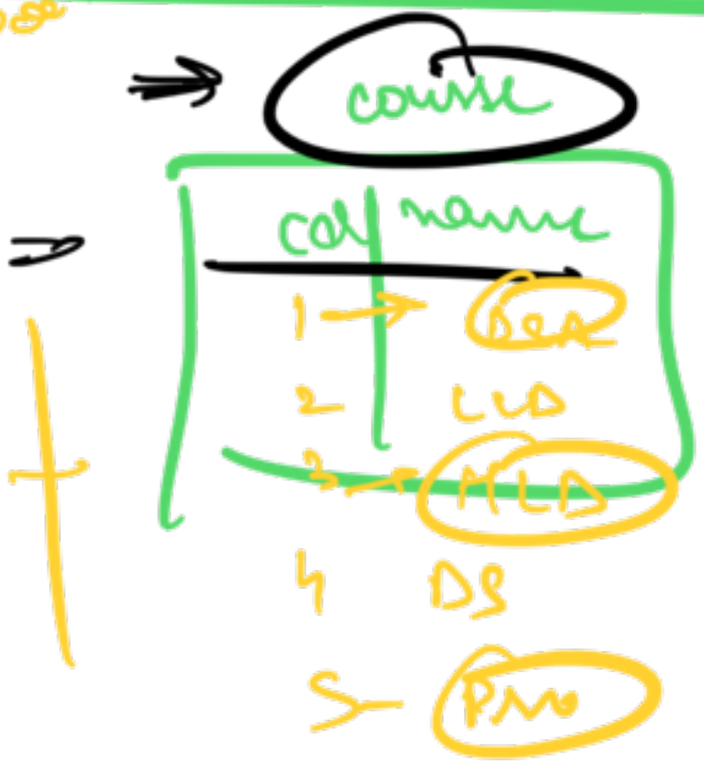
⇓

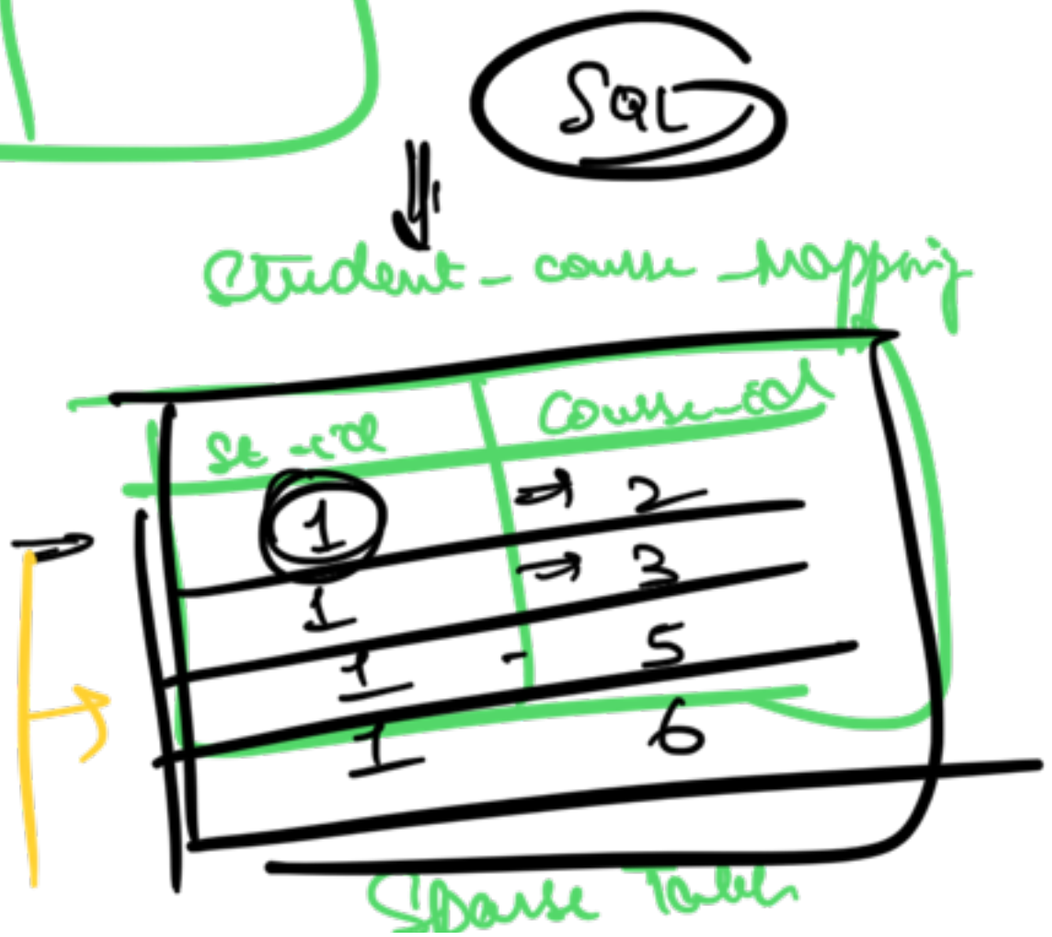
Con → Difficult Query ←

Pro → No joins

⇒ Easy to follow constraints

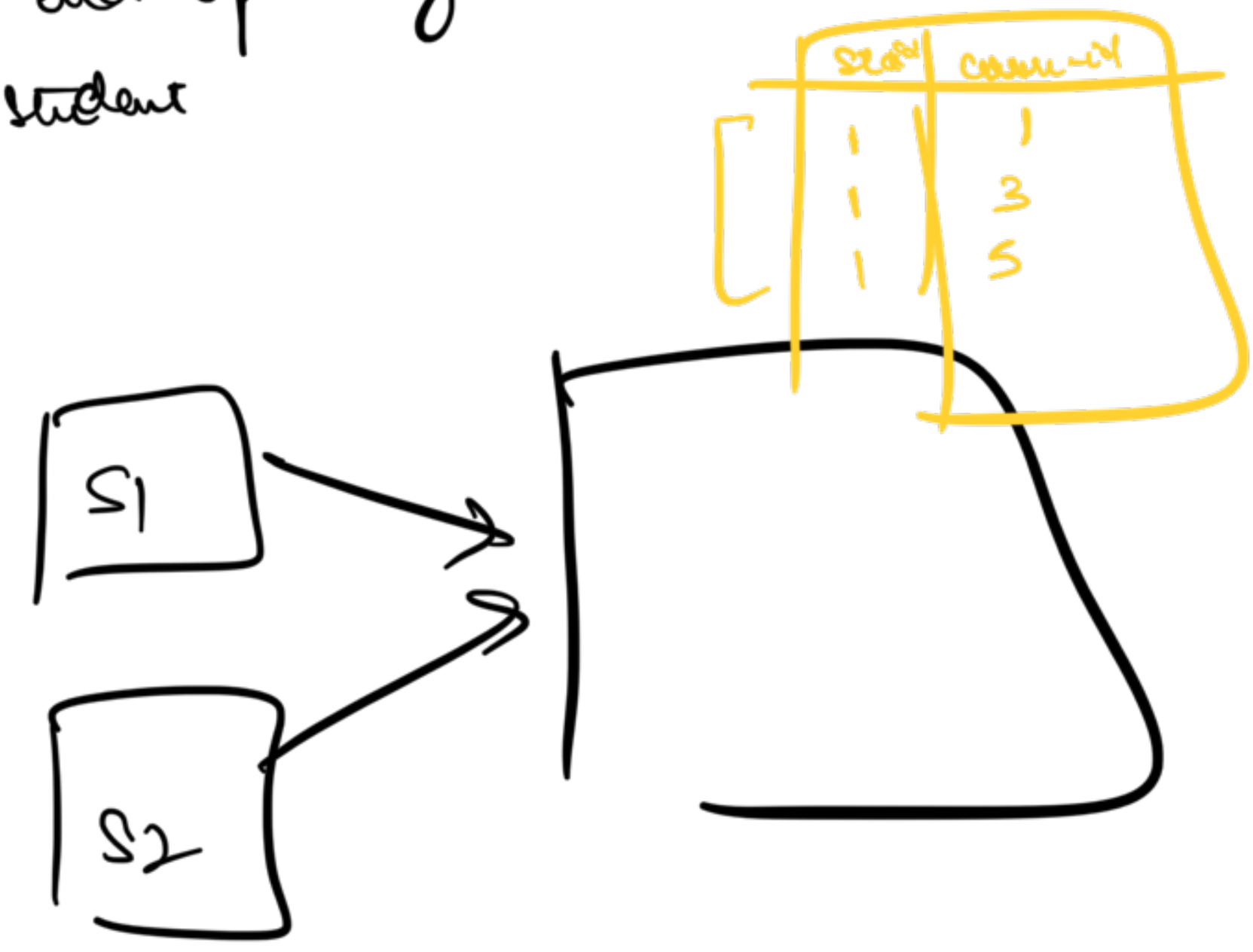
M:M ⇒ represent as a separate mapping table





CONS

① Need to ensure at app^{n} level that we dont end up adding more than 4 courses of a student



Bank ⇒ 4 account per aadhar

1:M
OR
M:1

⇒ FK of 1 side in M side

But

If Sparse → Sep Mapping Table

↑ App^y code will have to manage consistency

M:M ⇒ Separate Mapping Table

1:1 : Id of any side or other side

1:1 → Any on any → Mapping Table →

1:M
OR → id of 1 on M side
M:1

→ Mapping Table → Spark

M:M → Mapping Table

Break a wall

Don

JCB

Batch

V4

1. Student
2. Courses
3. Any student can enroll for any # of cours
4. Every course has course end exam
5. Every exam has a duration → attr of exam
6. One exam can be a part of multiple course Cr

Academy

7 exams

90

Exam

Aptitude

Header

3 out

DS

90

Out

(7) ~~Specify~~ Every exam has a specific date for every course

Acad → API → SNA
→ 1:45 PM
→ DS A
DS → SMS
LL → 6th

(8) If a student is enrolled in multiple courses with same exam → give exam twice

(9) 1 student can attempt exam only once per course

(10) We have to store marks ~~of every~~ for every attempt

→ 10:45 PM

11:30

→ Create Design

→ Code in Spring Boot

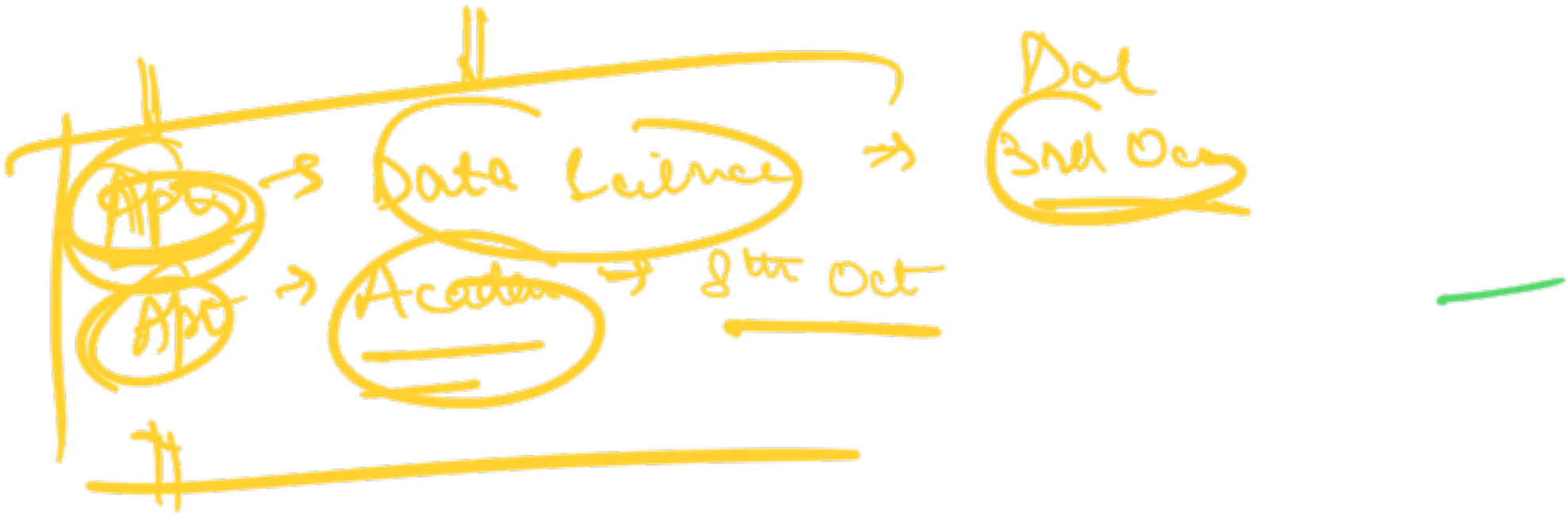
---

① Class Diagram (Entities) → some info/ attributes

→ 

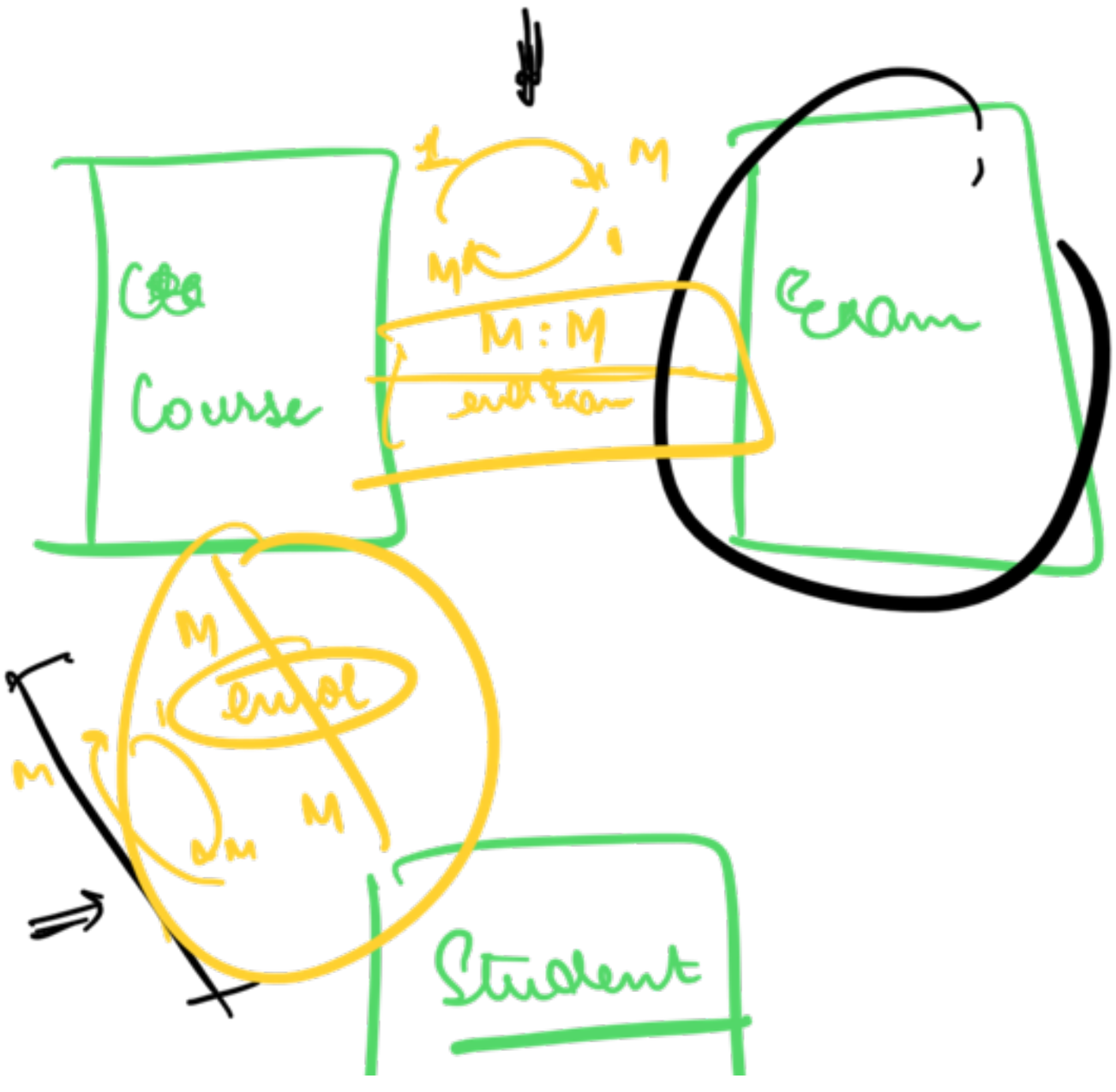
attr of entities

(1) Even relationships can have attributes

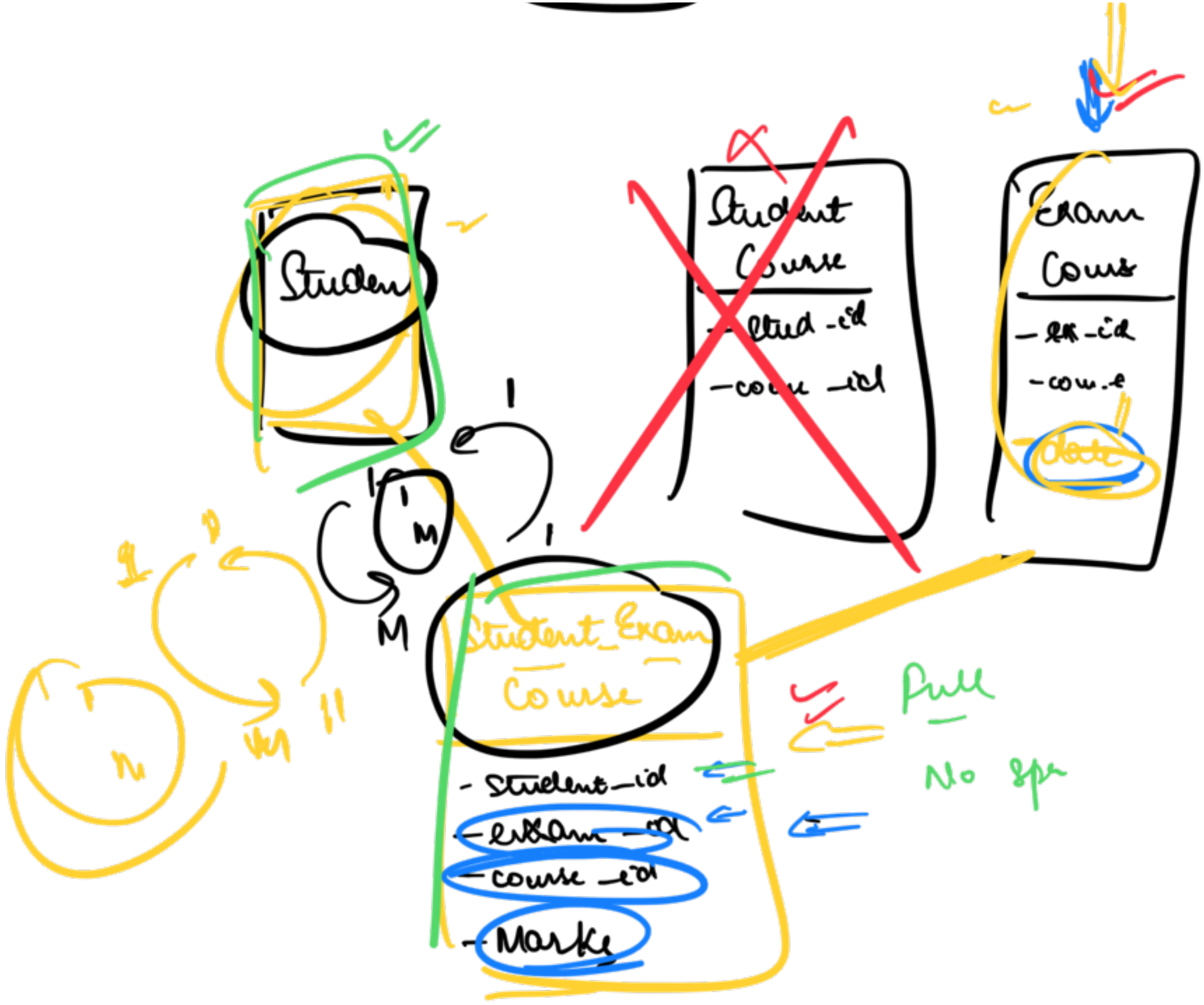


Exam	Course	Date
Aptitude	DS	3rd Oct
Aptitude	Academy	1st Oct

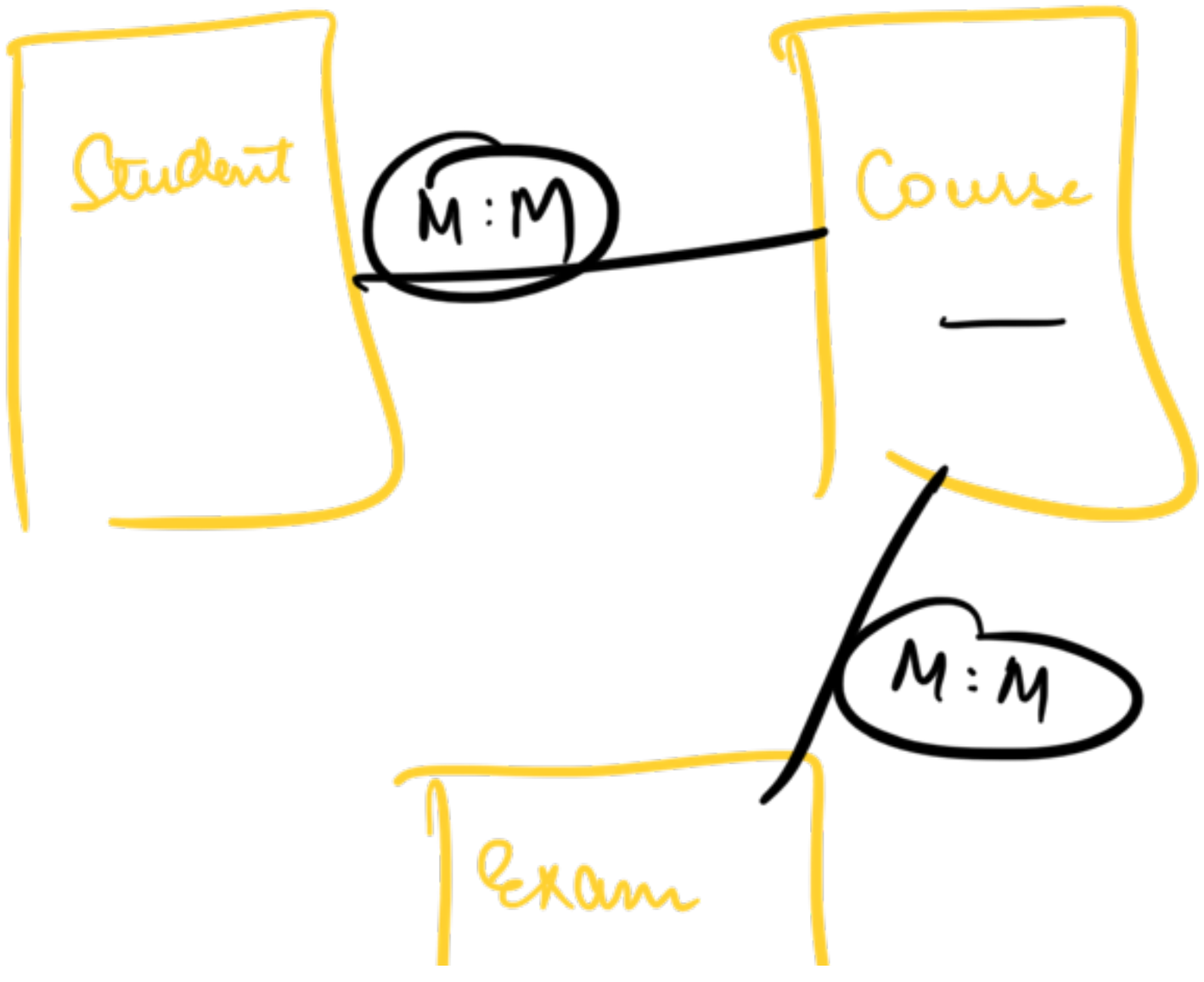
Even rel^s can have attributes



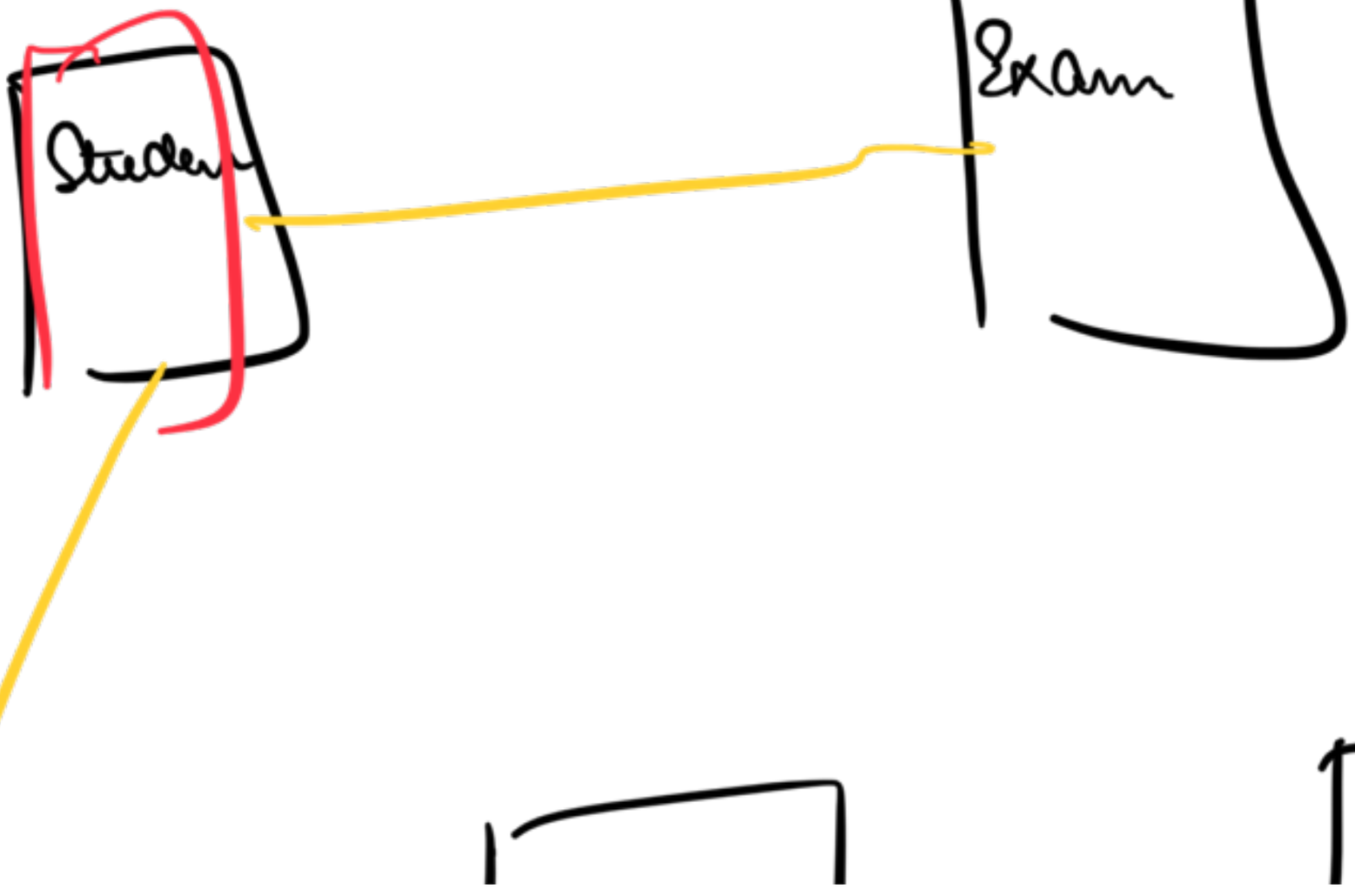
After listing all entities, start finding out the relationship between entities


Student
Student Course
- Stud_id
- cour_id
Exam Cours
- Ex_id
- cour_id
- Date
M
M
N
M
1
1
1
Student_Exam Course
- Student_id
- Exam_id
- course_id
- Marks
Pull
No Sp

Step 1 Identify the entities



Step2 Include every relⁿ as entity



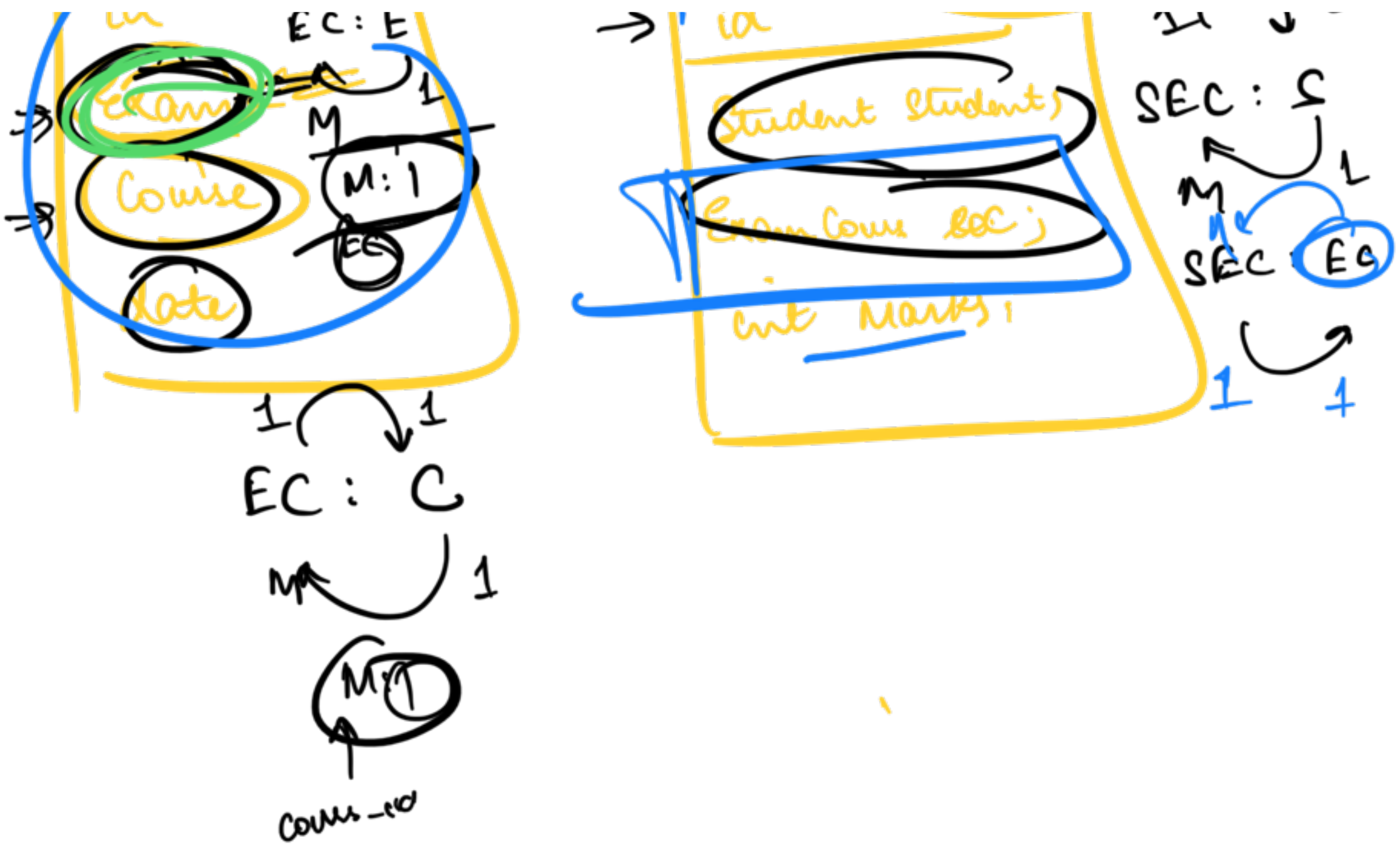
When do we store a relⁿ as a separate class

① If 1:M
Or
M:1

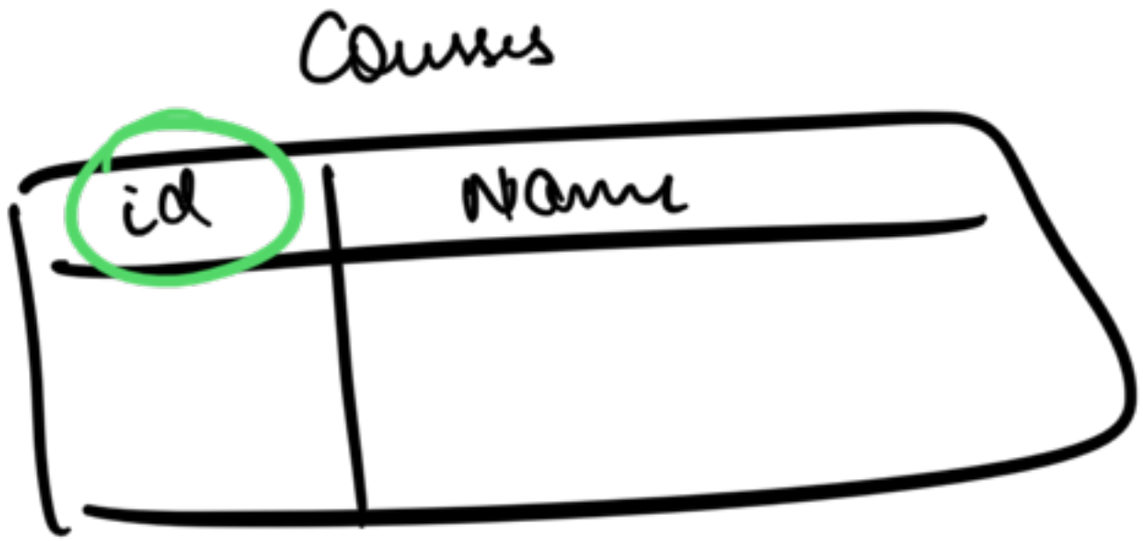
but we don't want to add
column of 1 on another
(Spouse)

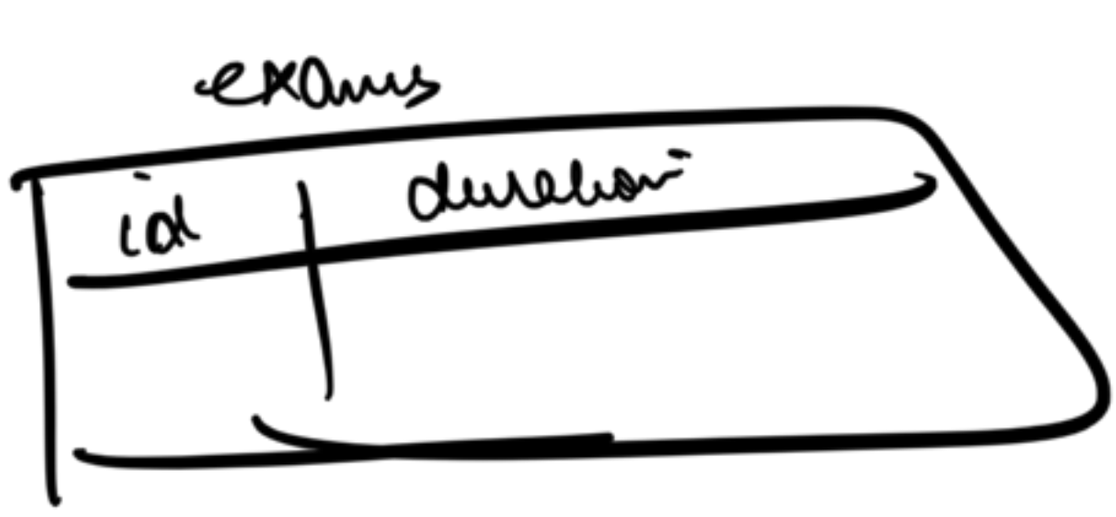
② If M:M && AND the rel² has attributes

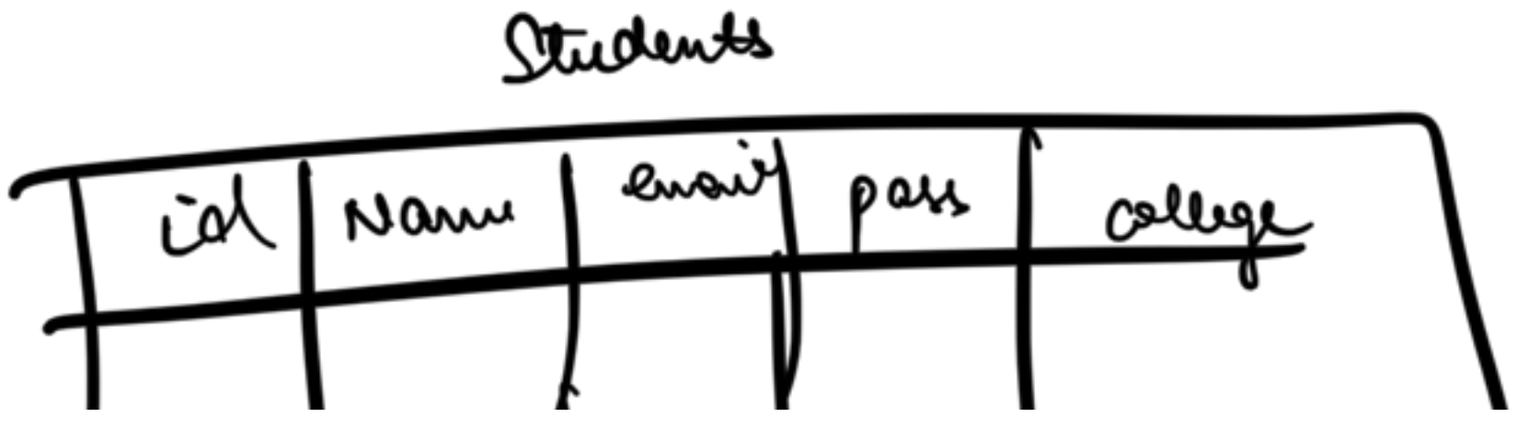
Final Class Diagram


EC : E
1
M
M : 1
EG
Course
Exam
Date
1
1
EC : C
M
1
Student Student)
Exam Course EC)
int Marks;
SEC : S
M
1
SEC : EG
1
1

Schema Design





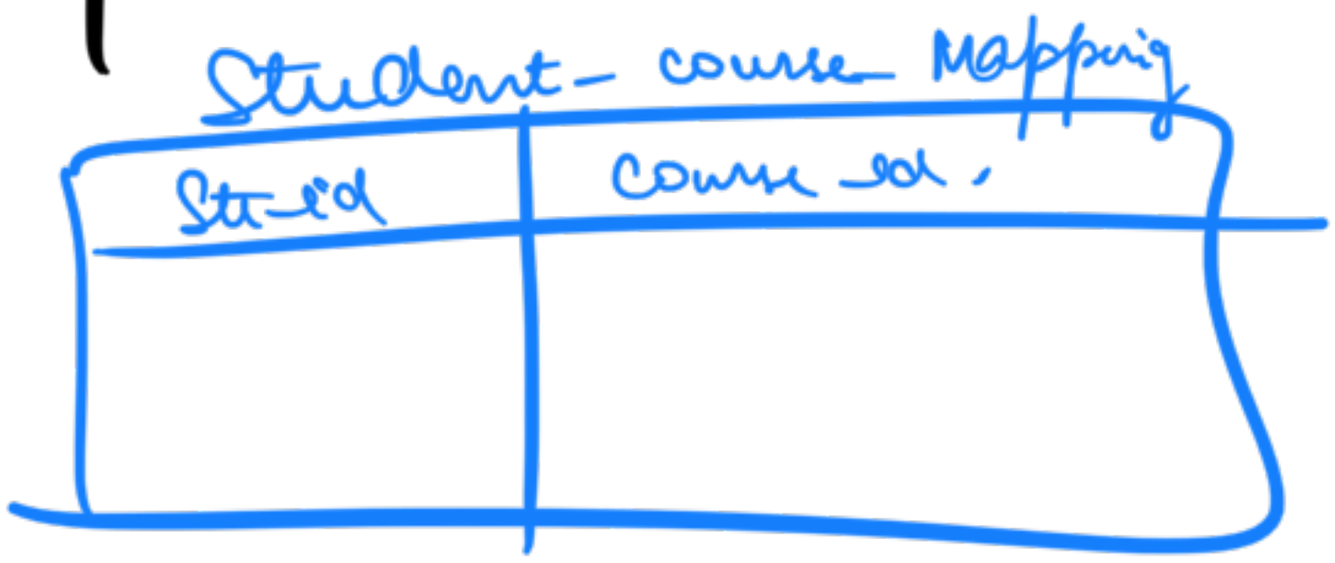


exam_course

id	exam_id	course_id	date

Student_exam_course

id	Student id	exam_course_id	Marks



How to approach a Schema design

① Find all the entities in the use cases

② Rep each entity as a class

③ For every pair of class, see if there is a relation between them:

a) 1:1 Put an attribute of any class in any

b) 1:M or M:1 Put a list<Other Class> on other

c) M:M ⇒ create a separate class Course Student

④ Go through all use cases and find other rel^s

⑤ Remove Mapping classes with no attr

⑥ for the final classes → do Schema class

Use Cases

Use Case Diag

Class Diag

Schema Desig

Data base to use

45 min

HW

1. Rewatch the class

& Kup copy in hand

and draw

2. Write use cases in a paper

→ come with schema design

→ ① Code

→ ② # More case studies

③ LLD interview