[How to approach LLD Problems] → Type of LLD Interview
→ How to approach

And

[Design a Pen → Amazon] → 1 Type of LLD Problems
→ Check our learning of "How to approach LLD" problem

Types of LLD Interviews

3 Types

THEORY ROUND

- LLD Round
- Theoretical Round
- Coding Round

Design Round

Machine Coding Round

→ SWE
→ SWE+

Coding Round

Amazon, Microsoft Apple

→ Some mid tier

est companies Not exactly tech

What is Singleton DP?

What is Factory DP

What are SOLID

How do constructors work in inheritance

Dup

→ Very solid knowledge of OOPs, SOLID, DP

(45-60 mins)

→ Non startup tech companies

→ Don't care about pure tech stack

→ Abstract Problem Statement (Single Line)

"Design a Pen"

Coding

"Design a Parking Lot"

Expectations

⇒ (1) Gather Requirements

NEVER

ASS + U + ME

↑

Sorted

Given a set of #, give the Min & Max

(2) Clarify Requirements

↳ ask about edge cases

↳ ask about working of

behaviour

③ Use Case Diagram

④ Class Diagram (for entities)

90%

⑤ Schema Design

99%

Just write ~~code~~ code for a particular class/ set of classes in a Google Doc or paper

⇒ NO WORKING CODE

follow up Question

— How will you handle conc

— What will be cardinality of

$M:N$ b/w X : Y

Machine Coding (2 - 2.5 hrs)

→ Mostly in startups ⇦

→ Detailed a problem statement

⇒ all the rq are already listed for you.

⇒ Clarify Requirements = 30 Mins

⇒ Use Case Diagram ⇒ AARE

⇒ Class Diagram ⇐ Mandatory

⇒ Schema Design

→ Code → Working Code is expected

① Handles the given set of req

② A good code practices.

1 hr

1.15 hr

90%

Write Test Cases

Discuss the sol^n and follow up q^n → 15-20m

MC

20-30 Min : Discussion and Design

1hr - 1hr 20min : Code

20 - 30 Min : Discussing the code & follow ups

SFN

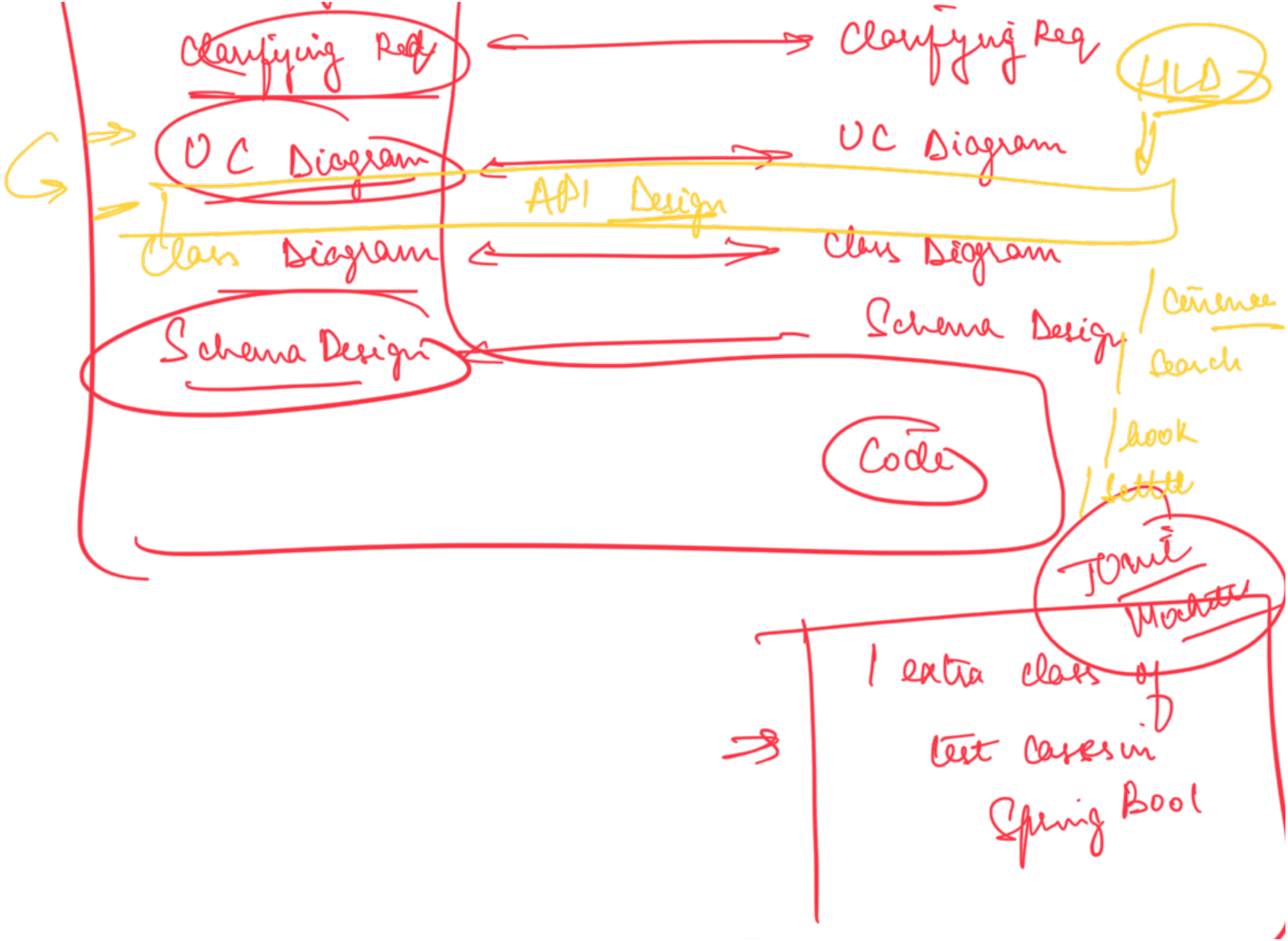
Mouresh

Kumar

Design

MC

Gathering Req <= Sharing a doc



Clarifying Req
UC Diagram
Class Diagram
Schema Design
Clarifying Req
UC Diagram
API Design
Class Diagram
Schema Design
HLD
Search
Book
Code
JUnit
Mockito
1 extra class of TDD
test cases in Spring Boot

(1) Gathering Requirements (5-8 min)

"Design a parking lot"

→ To get a lot of ideas

Steps

1. Ask the interviewer for an overview

- You already know a bit
- you don't know

→ Given an overview to interviewer

→ Ask specific q's

① Overview of the env of use | Type of app^n

② ask the scope

→ Entity

③ how will someone access
Desktop Games → Games → SDL, TTF

→ Management System
Real System
⇒ Persist Data
→ MVC

App ⇒ No need to persist data

→ Engineering problems

→ Distributed Cache

① Entity

② Singer Desktop App

③ Web Based App

Env

Steps

① Overview from interviewer

② Suggest Ideas

with why

- → undo
- → comp player
- → diff levels of AI
- → leaderboard
- → player prof

③ Visualize the system

ask q's from top to bottom

(Think of what is currently there that might change)

Parking lot

- → Multiple floor
- → Diff. types of veh

Solid for

→ After While getting the req: try to note down the req and summary to

in terview before going to design

11

Clarify the Requirements

working of
→ about the behaviour

→ Edge cases

→ starting and ending

98 + 14

Ask q^n that are around

(1) Current Scope

(2) Future Scope

(3) Working of a Behaviour

Ask q^v

that impact

design

→ class diagra

→ Schema

→ add new

entity/

interface

(3) Use Case Diagram

→ Verify of interview needs it

4 Class Diagram

→ No rules to follow like OML Spec

→ But

① Mention the cardinality of rel^n

② Diff b/w inheritance & association

Y

→ No need to draw all classes that will be there in code

→ Only draw models and classes that are linked to model

→ No need to mention Services, Repos, Helpers

→ How to identify classes

1. Noun → For Real Systems & Eng Problems
2. Visualization → For games and entities

Schema Design

→ No need to be exact on how tables will be drawn

→ What all table did you draw and how well

Student

mentor

Code

→ OK if you use any framework

→ OK to not write test cases.

→ OK to not cater to all the req

↳ NOT OK if none of the req work completely

⇒ when writing the code :

1) Code all the entities

2) Go req by req

↳ No need to connect to a real data base

↳ Replicate in-memory DB

- Users should be able to register on the platform

APIs

1. /users/register
2. /users/forgetPass

Database {

list < Students ——

list < Bat

⇒ Still design is imp

→ DMS

Student Repository {

Map <ID, Student>

}

Table → Map <ID, Data>

Index =>

Hint:

→ it will also involve a Map

→ DB is always indexed on ID

→ Pen

→ TTT

→ Class Student Database {
→ private Map<ID, Student>

→ Same ( ) {

findBy ~~Id~~ (long id) {

findBy Name (String name) {

}

}

Gathering Req (5-8 mins)

Clarifying Req (5-8 mins)] (2 Mins)

UCD V/S APIs] 5 Mins

Class Diagram ⇒ Only entities & classes around entity

→ interfaces

→ association

⇒ Now

Games ← ⇒ Visualization

and

entities → Calculator

→ Bird

→ Pen

→ Vending Machine

→ Elevator

→ Cardinality b/w classes

Schema Design ⇒ format doesn't matter

⇒ list of tables and col^s within the respective tables

⇒ Models

code

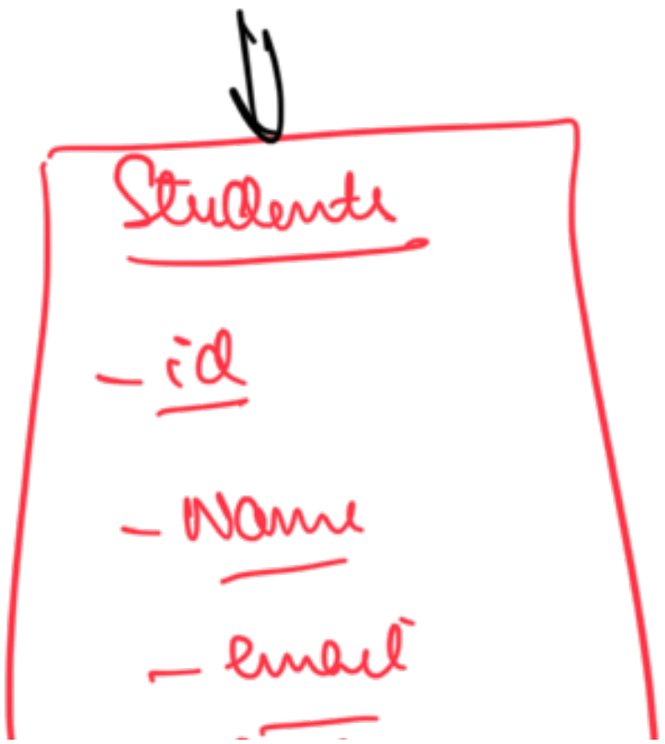
→ Code Reg I by 1

→ Mock the database

→ Good quality

→ Good package structure

→ Design pattern



1 ← gen

30 min

Most people prefer using a framework

- Java: Spring Boot
Python: Django

→ Design and Code a Pen

→

Component

Any req that we are given ⇒

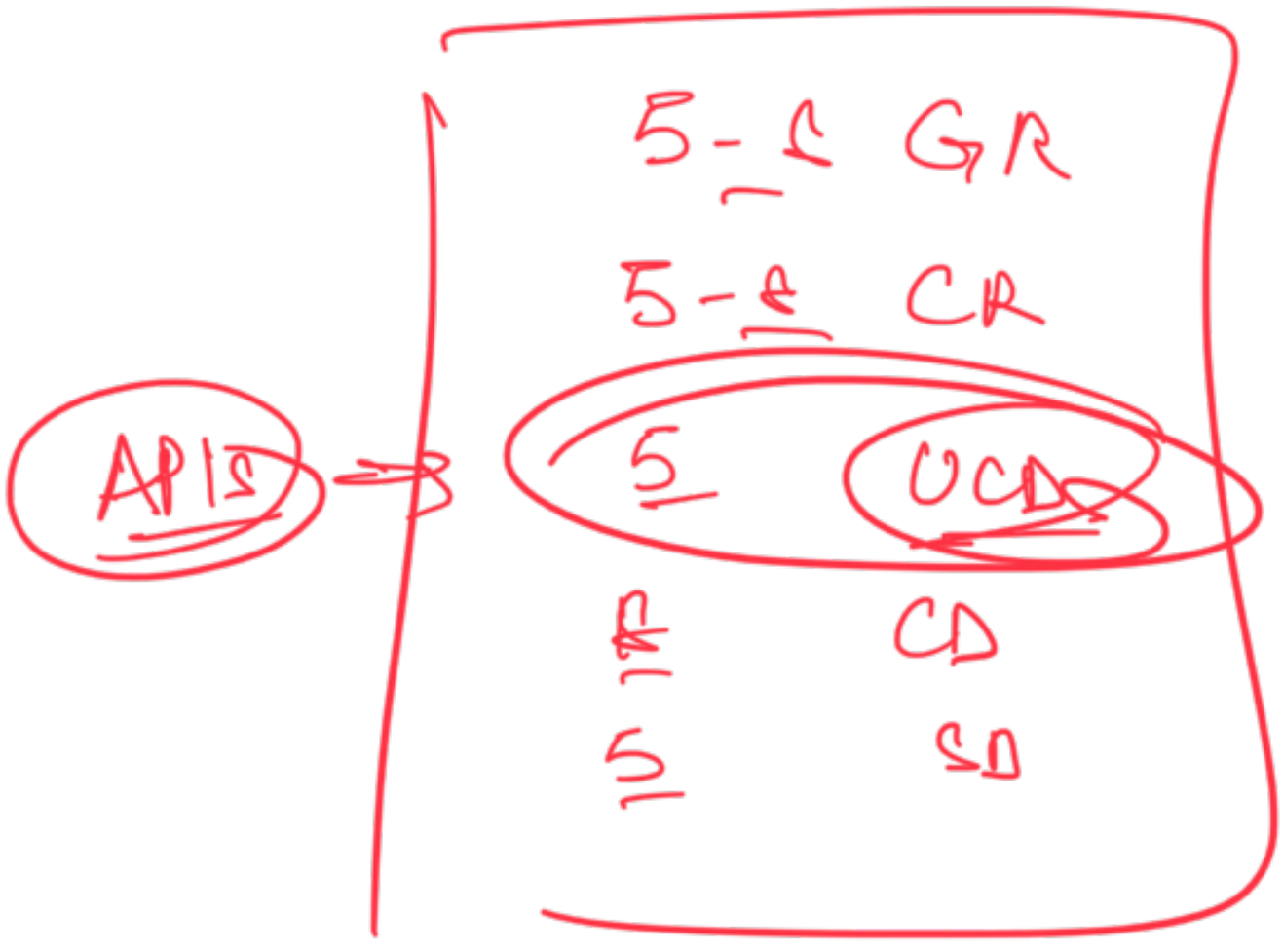
if it is a web based app

⇒ API

People should be able to reach other people at a location

MC round

Search & location = 0.5



Student

Course

Frontend : UI Components

Backend : Class, Schema Design

Architecture : Scalability, Data Layer

HLD

→ API

→ Schema Design

SWET

System Design Round

→ GR

→ CR

→ APIs

→ Schema Design

→ Architecture

LLD

LLD

=

HLD

=

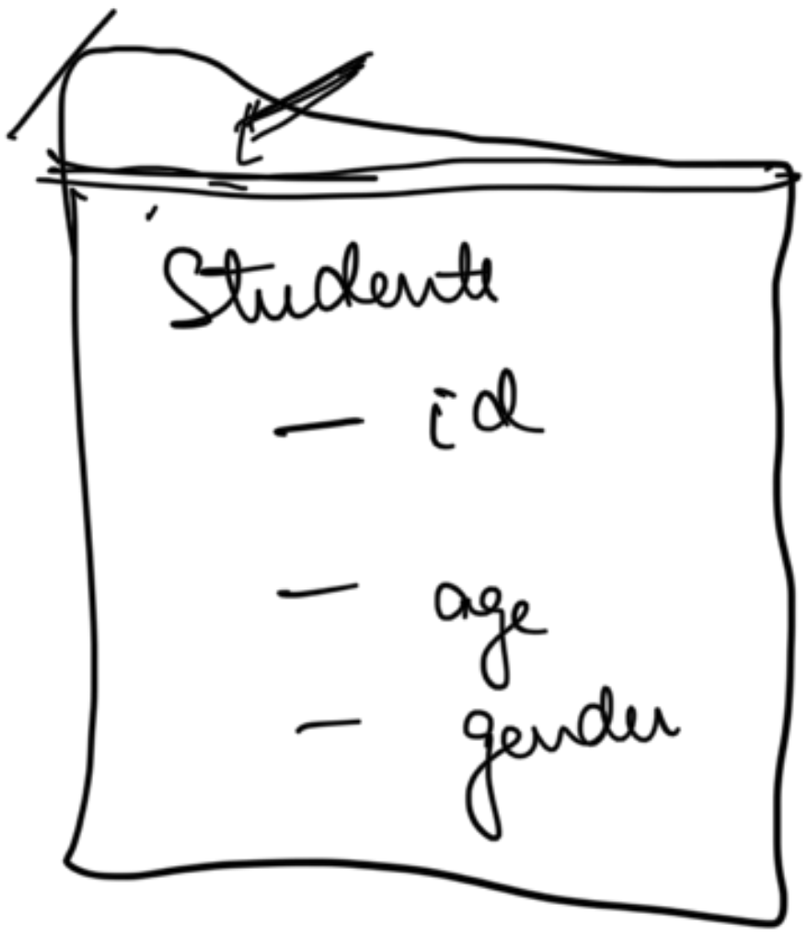
LLD

=

HLD

Docker Containe

Load Balan



⇒ draw.io

* Whiteboard

→ Google Doc

→ Google Drawing

10 users

name
ID
address

getAttrValue for Attr (String attrName) User user)

getAttrValue(attrName)

Case Name:

return user.Name

3 String

1 →

2 →

3 → Name

Module Exam ⟷ Student Module Exam

1 — M

ME {
M → 1
E → 2
}

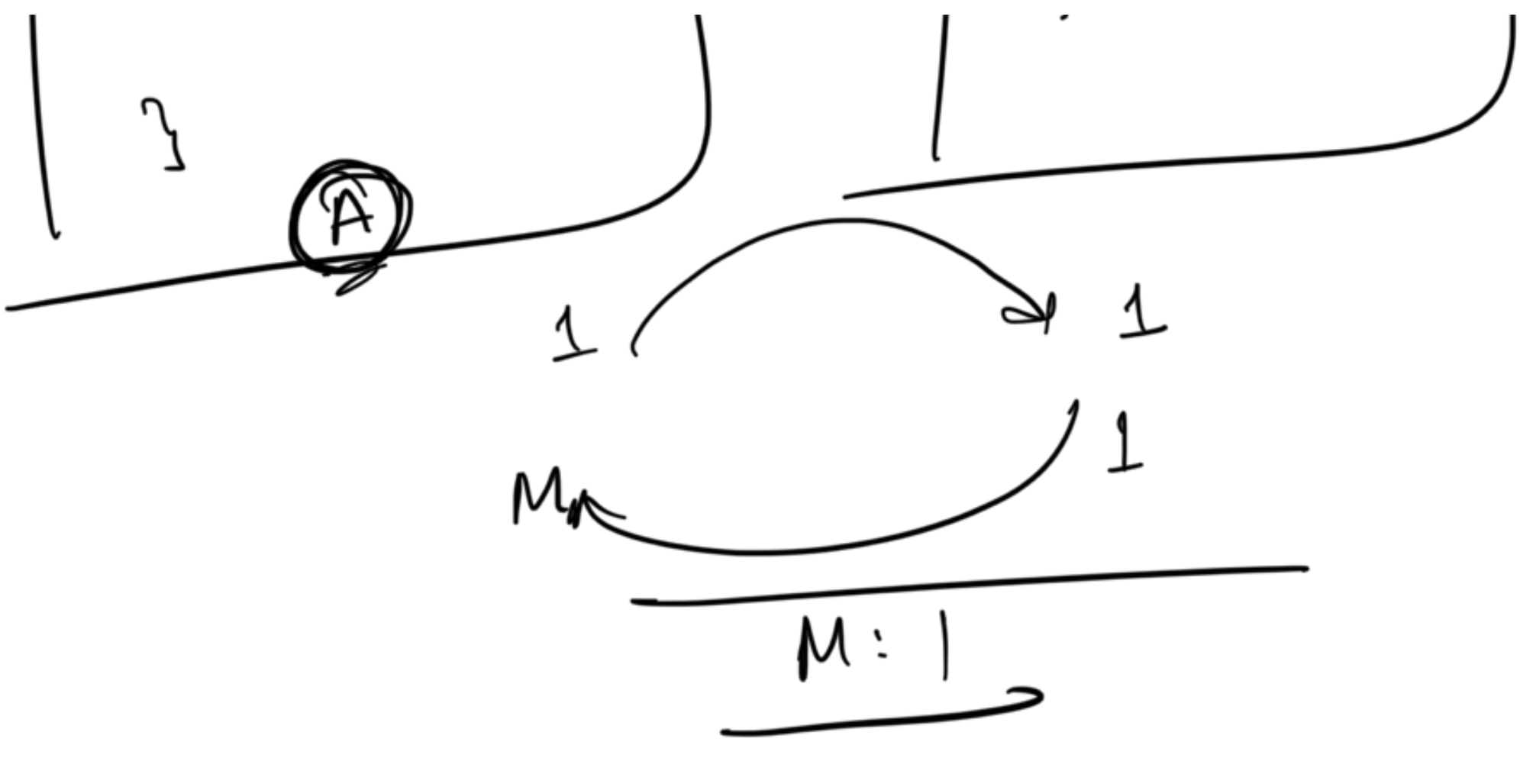
SME {
S (1) (2) (3) (4) (5)
ME (1, 2)
}

1 — 1

1 — 1 : M

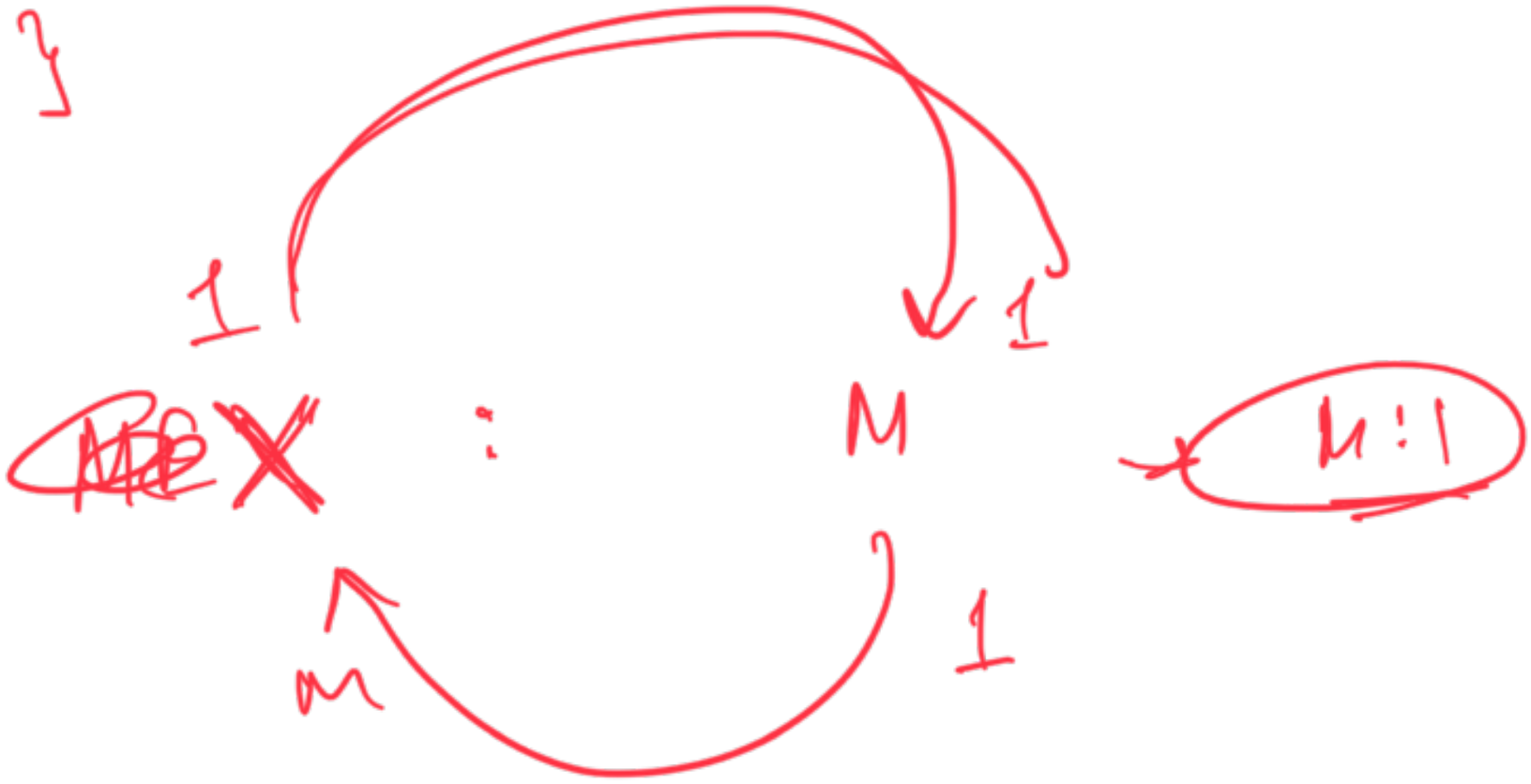
Student {
Batch
}

Batch {
}



Module

Exam



Module_id = 1

ME → (1, 2) (1, 5) (1, 7)

Video_actor

Video

id

Video_id

actor_id

actor

pk = (v_id, a_id)

Admin

SQL

Views

Transactions

Procedures

SQL

- Joins

→ Subqu