=> Agenda:

i) What is LLD?

ii) Why learn LLD?
- work/development
- Interview

iii) types of LLD interviews
- theoretical
- design
- Machine coding

iv) How to approach LLD problems

v) Design LLD of a system like Paytm

vi) Class diagram/ Schema design

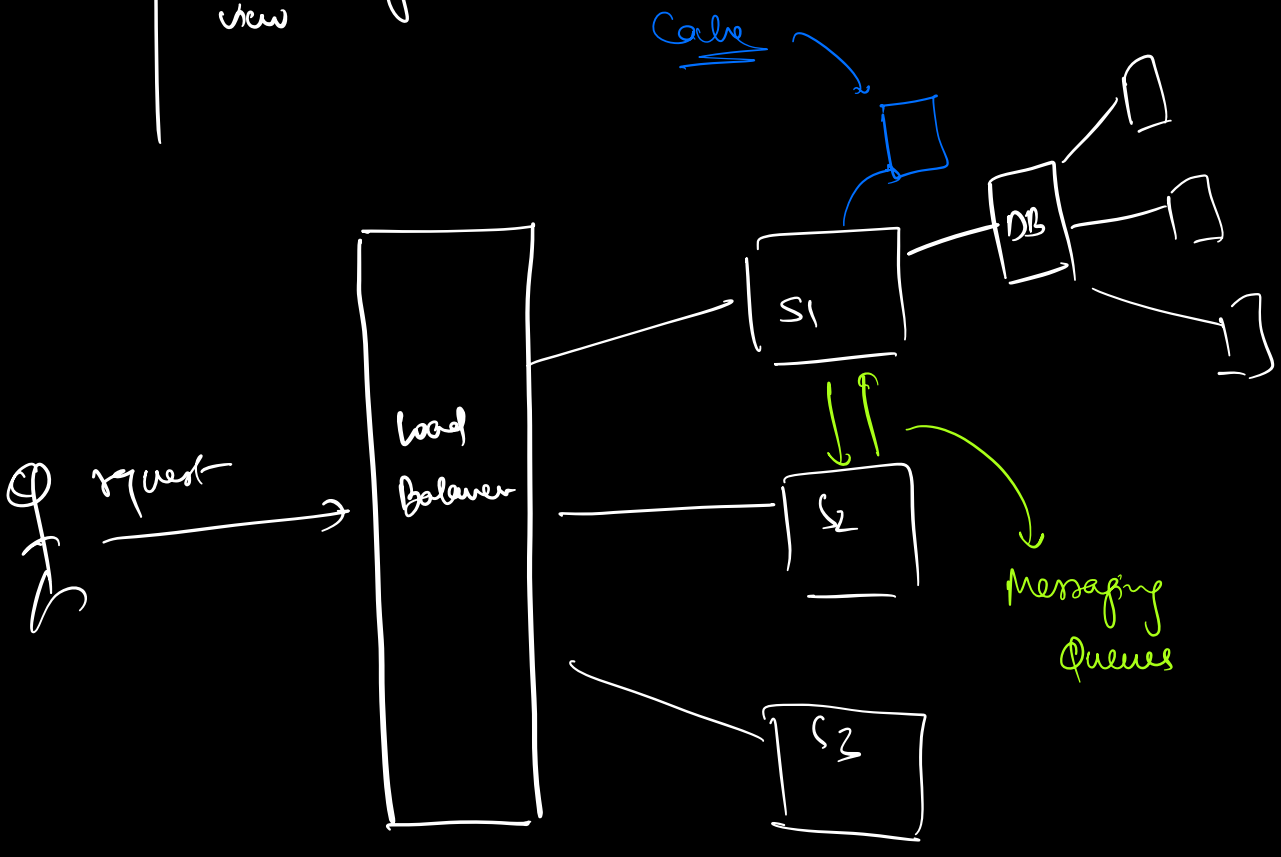
=> What is LLD?

- System Design
  - HLD [High level]
  - LLD [Low level]

HLD

- overview
- abstract
- bird's eye view

LLD



LLD => go into details

↳ implementation of the s/w system

→ code of system

- classes
- methods
- attributes
- how diff classes will talk to each other
- DB design / schema

9hrs = 100%

12% ⇒ engineer spends only 12% of time to actually code

30LPA ⇒ 12% = 3.6LPA → code

Things engineers do:-

1) Meetings → gather req.
2) Debugging → read & refactor code
3) Documentation → read code and create documents for easier code understanding
4) Design → structure
5) Analyzing → reading code
6) Code review → reading code
7) coffee
8) Gossip

most of the time is spent on

reading / understanding code

vs.

writing code

⇒ Why we need to learn LLD:-

LLD
- SOLID
- Design Patterns
- Case Studies

I) Understandable
II) Reusable
III) Extensible ⇒ adding new features
IV) Maintainable

↳ keep running as it
→ debug
→ update of libraries

⇒ LLD interviews:

Theoretical	LLD Design [45-60 mins]	Machine Coding [2-2.5 hrs]
→ service based, mid level	→ Product companies	
→ OOPs	→ Design the system	
→ 45-60 mins	→ gather requirements and clarify	→ X
→ No design	→ Use case diagram	→ X (mostly not required)
	→ Class diagram	
	→ Schema design	Working code is needed
	→ Code (X)	

↓

Code only the classes

PM → [Clarify
Suggest
Gather more requirement]
↳ no design

⇒ How to approach solving a LLD problem:

1) Gather requirements:

→ ask questions
→ suggest ideas
→ Create visualization

2) Clarify requirement:

→ Current scope
→ Future scope
→ Behaviour → ✱ details of each feature and functionality

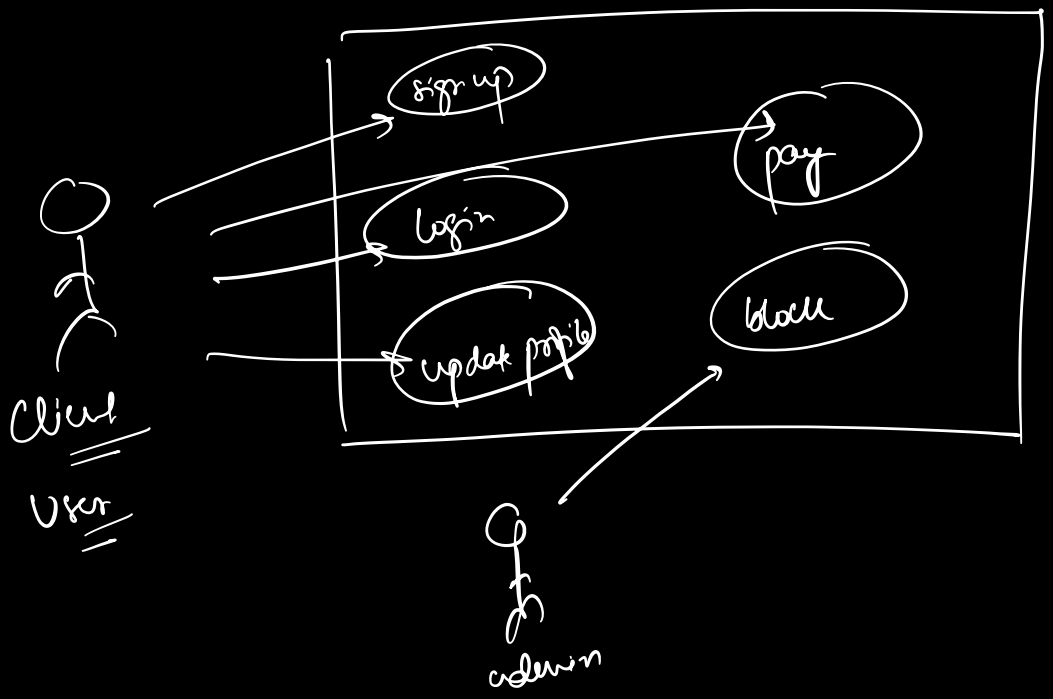
PRD
↓
Product Requirement Docs

3) Use case diagram

↓

Type of a UML diagram

Unified Modelling Language



⇒ Class diagram:

→ Dont have to write every class in system

→ Write classes for different entities

- User
- Transaction
- Payment

How

i) Think of all the nouns in the requirement

ii) Visualize the user journey

BMS ⇒ City, Movies, Theatres, Shows, Seats, User, Payment, Ticket

Class diagram : A diagram of a system depicting all the classes and their relationships

⇒ Schema design :

i) What tables will be present in your DB ?

ii) What is the relationship b/w those tables ?

iii) How to represent the relationship ?

diff schemas,

DB normalization (reduce redundancy) — 1NF, 2NF, 3NF, BCNF

Class diagram → Cardinality → Schema design

Cardinality → relationship

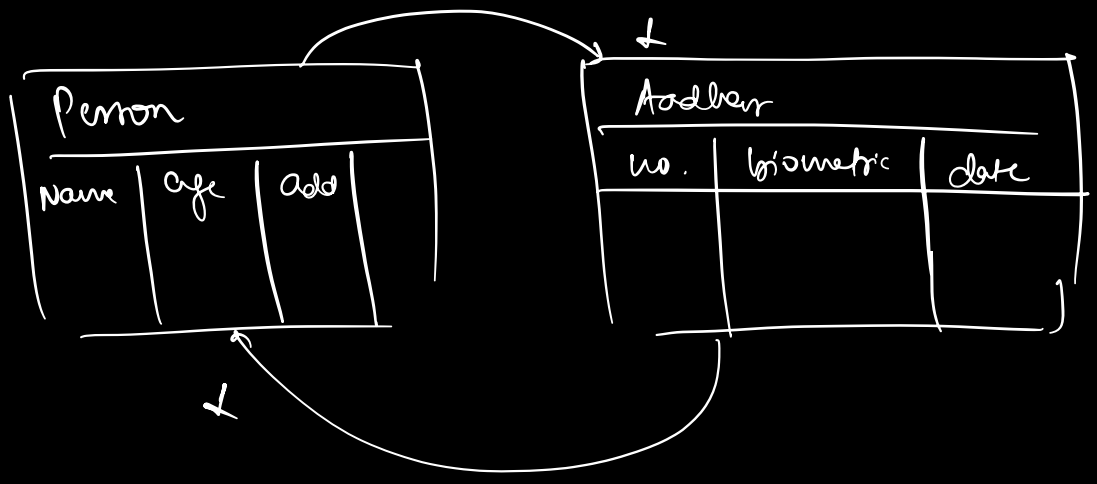
1 : 1

1 : M

M : 1

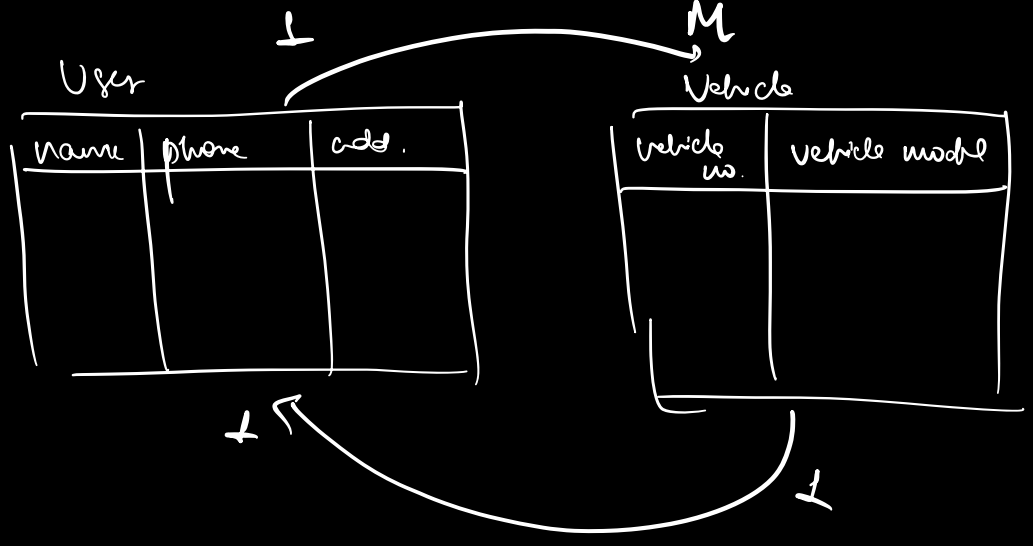
M : M

1 : 1 =>



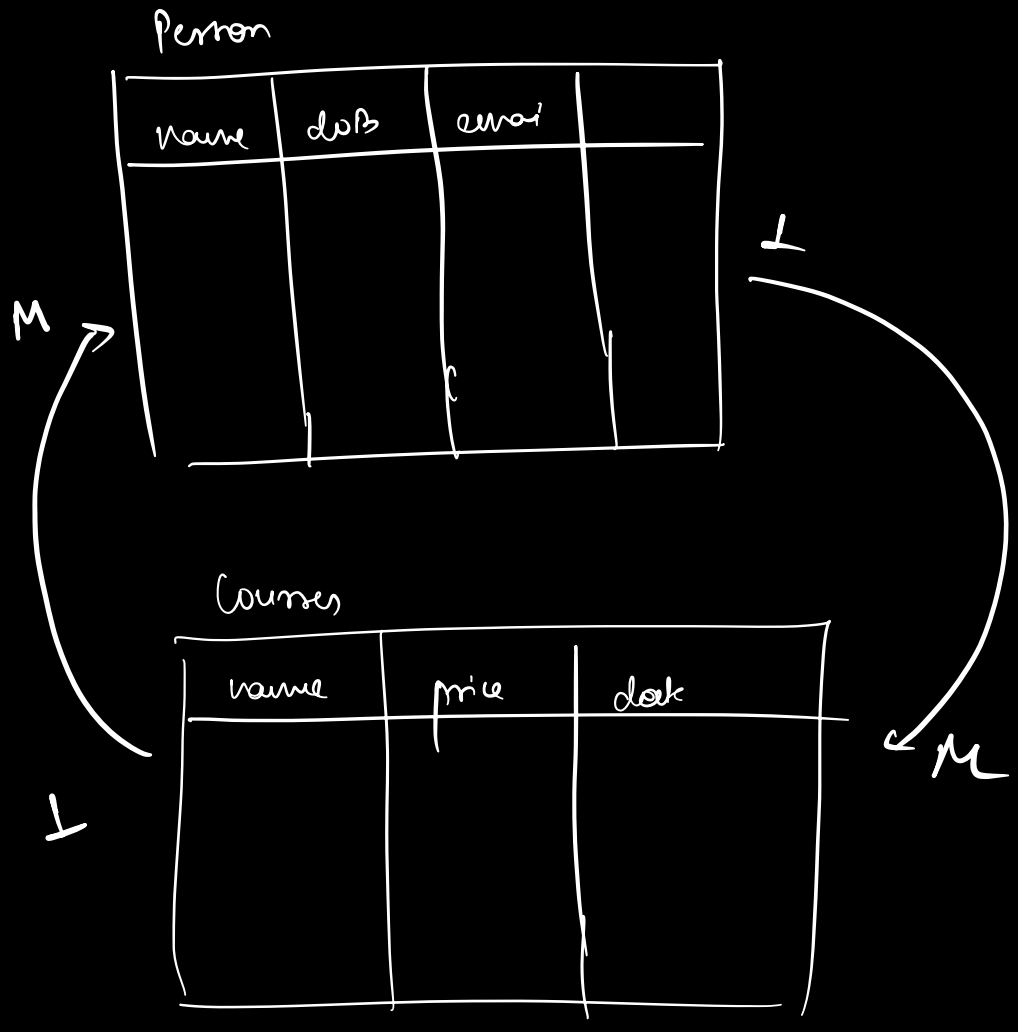
1 : M =>

M : 1

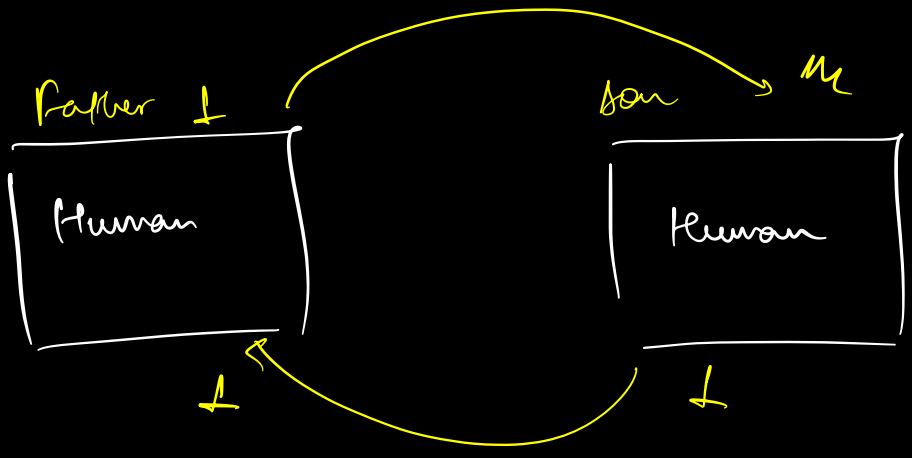


1 : M

M:M

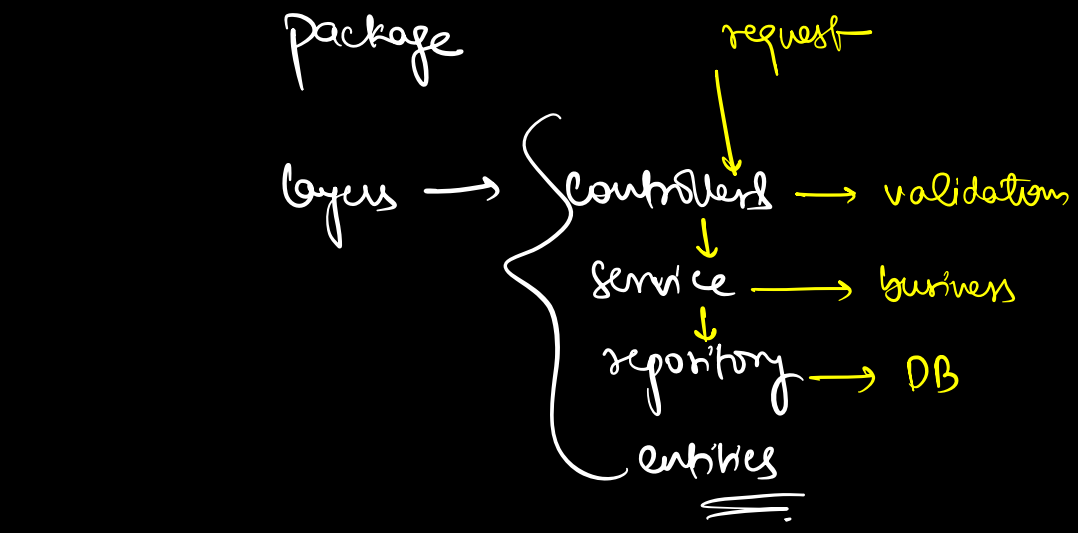


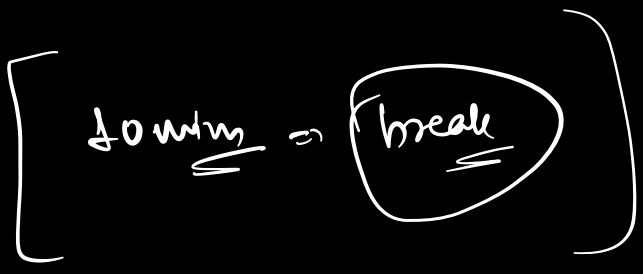
M:1



ASSUME ⇒ ASS + U + ME

⇒ Code structure :





LLD

1) Scaler
2) Head First Design Pattern
3) Refactoring Guru
4) Effective Java
5) Clean code.

23 DESIGN PAYTM

1) Gathering requirements :

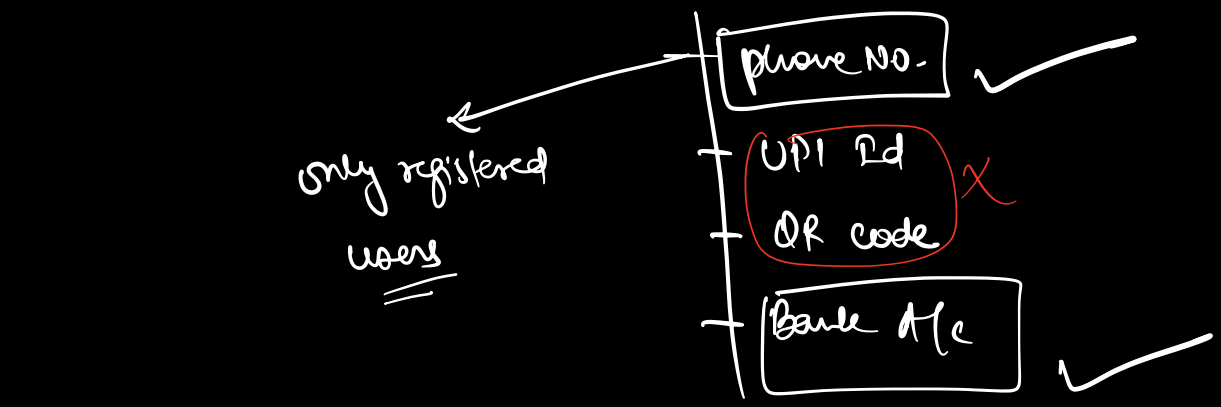
→ Signup → email, mobile no.
↳ Send OTP

→ login → email, mobile no.
↓ ↓
password OTP

→ Setup user details → KYC details
↓
verify kyc

→ Send money to other users

MVP
↓
minimum viable product



→ Source for payment

→ debit card, credit card, UPI

→ See history

⇒ Pagination

⇒ 10 transaction / page

⇒ most recent first

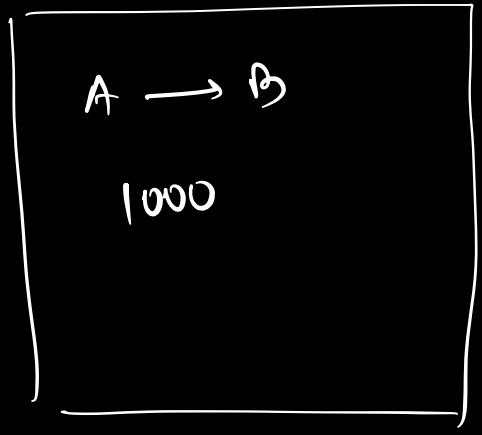
2) How do you handle transactions:

i) we want to allow retry

ii) we should allow double payment ~~null~~

iii) If the sender/reciever has any issue, we will cancel the transaction and refund any money.

⇒ User Journey



Payment {

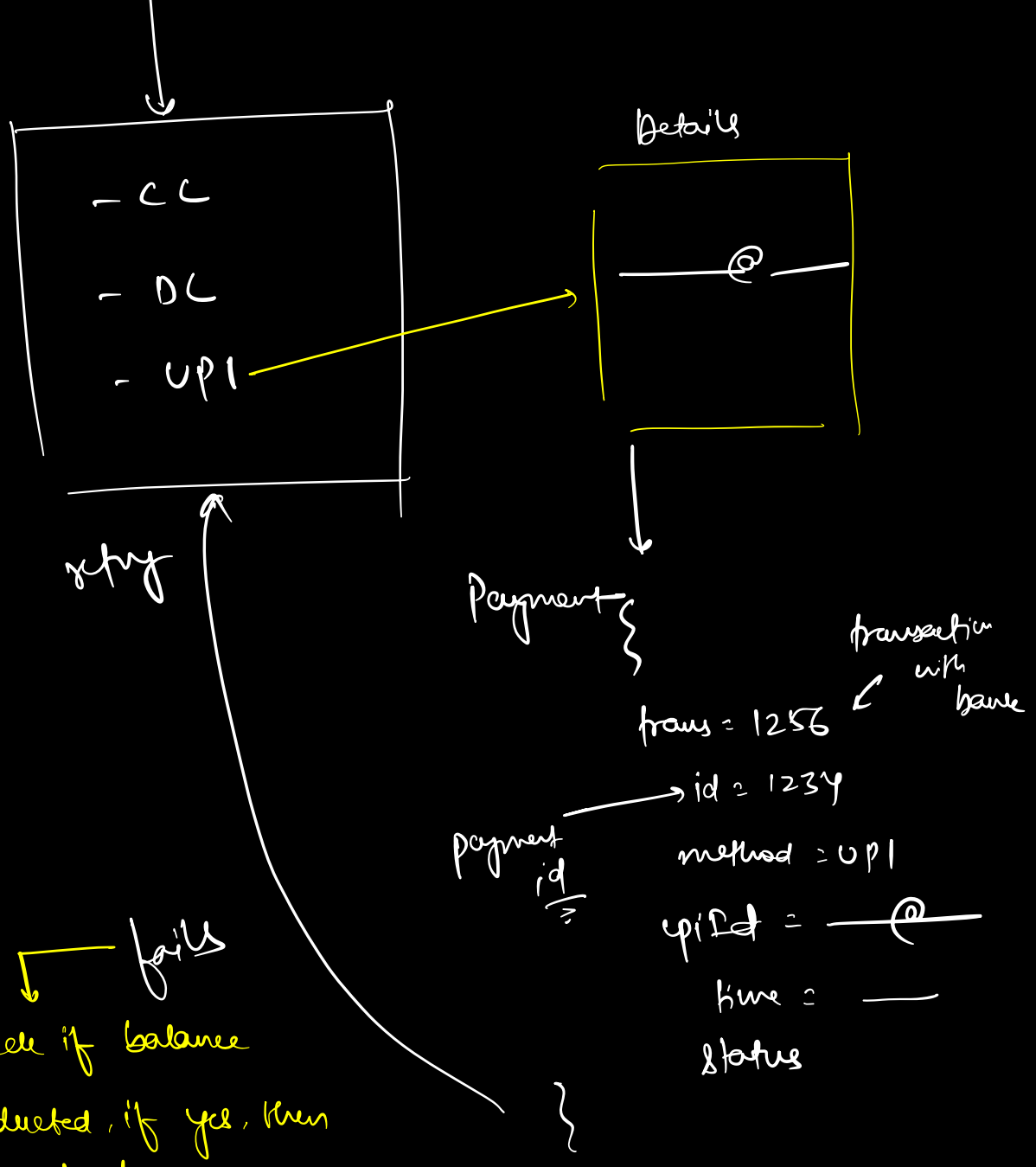
id = 1234

src = A

dest = B

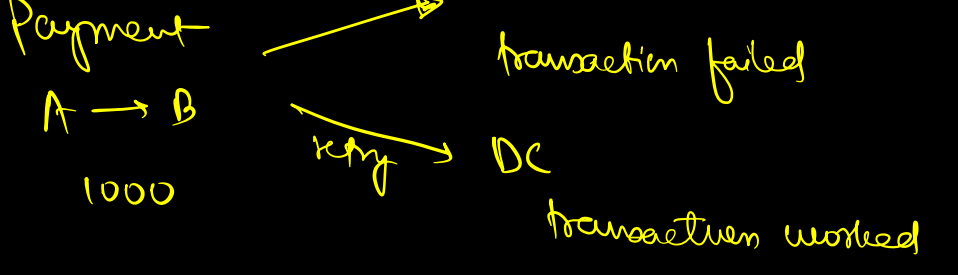
amount = 1000

}

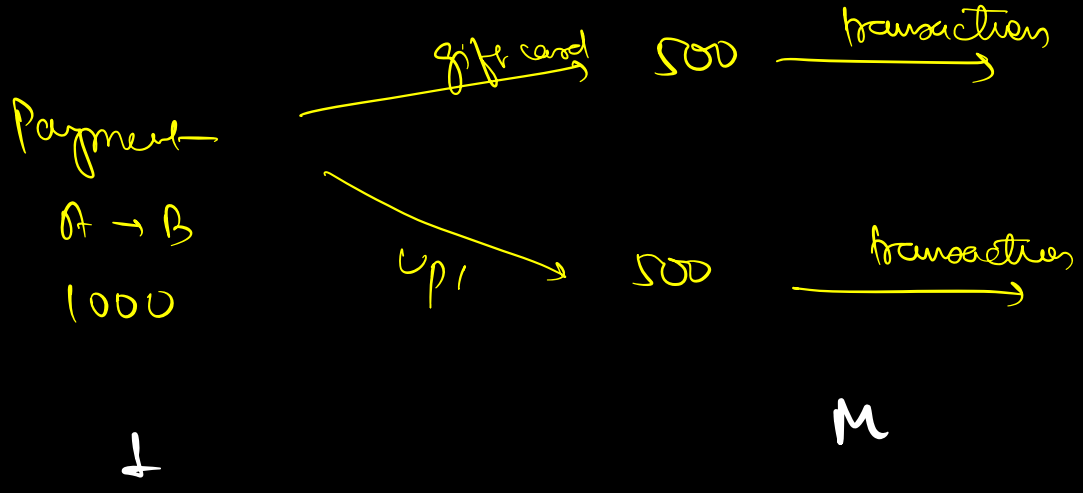


1 payment can have many transactions

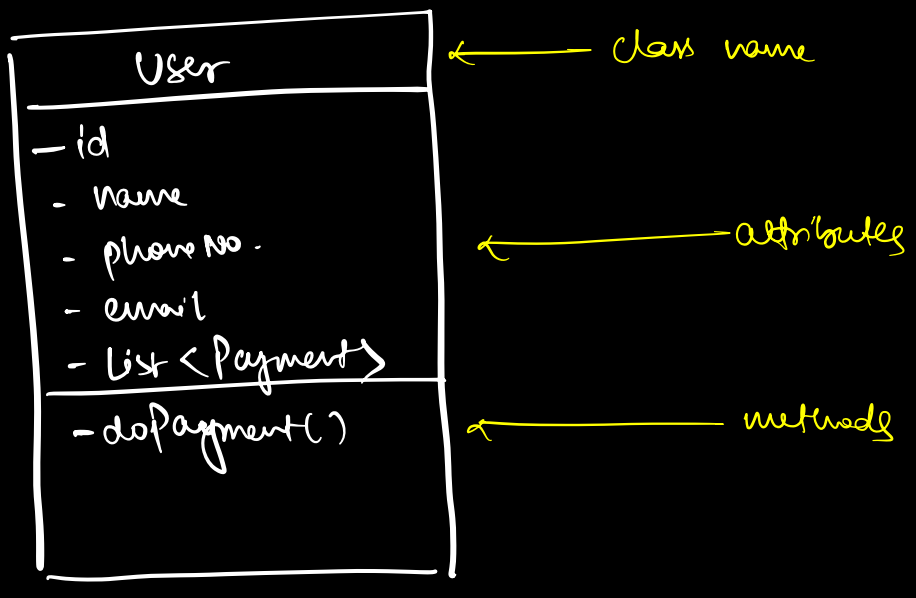
1) Failed transactions :

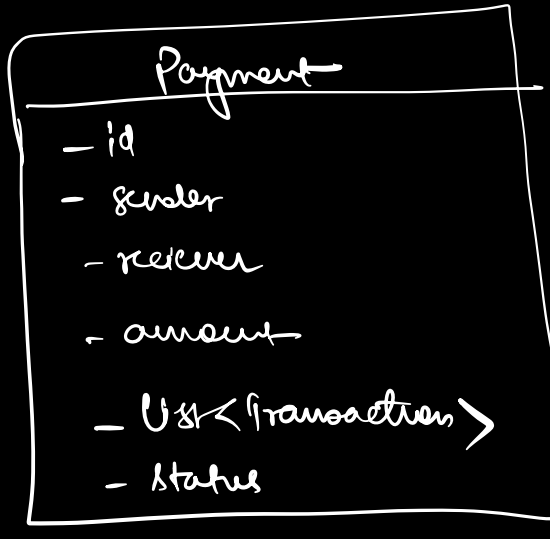


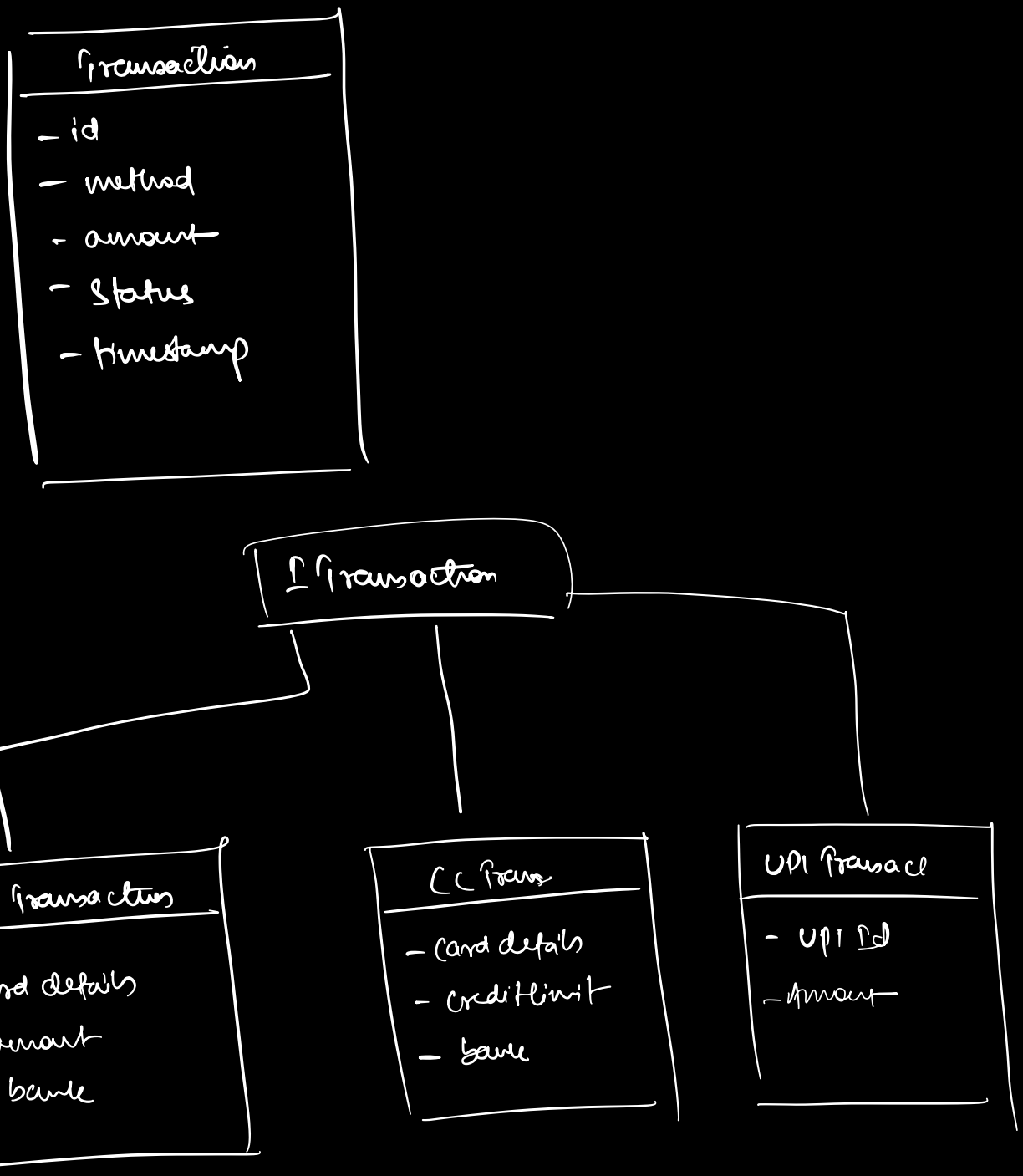
4) Multi-part payment



⇒ Class diagram:





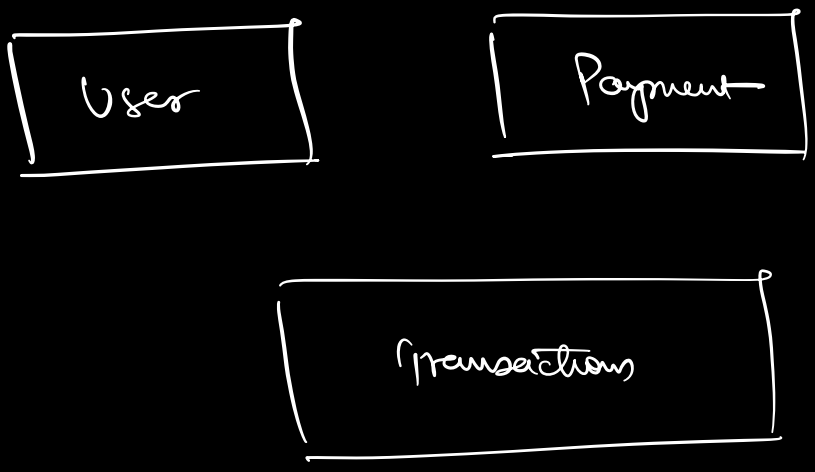


Strategy Design Pattern

↓

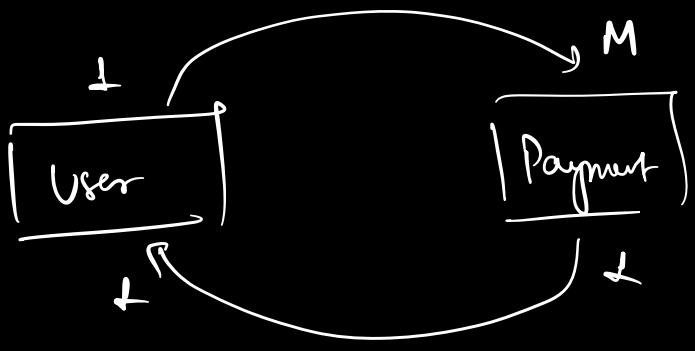
Creak a payment factory for creating multiple payment ops.

Factory Design Pattern

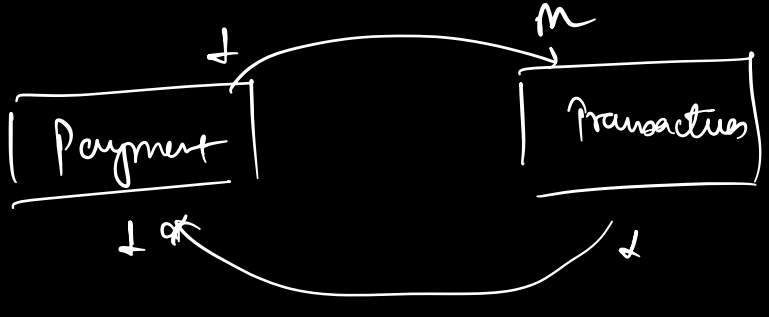


⇒ Cardinality:

1. User & Payment ——— 1 : M



2. Payment & Transaction ——→ 1 : M

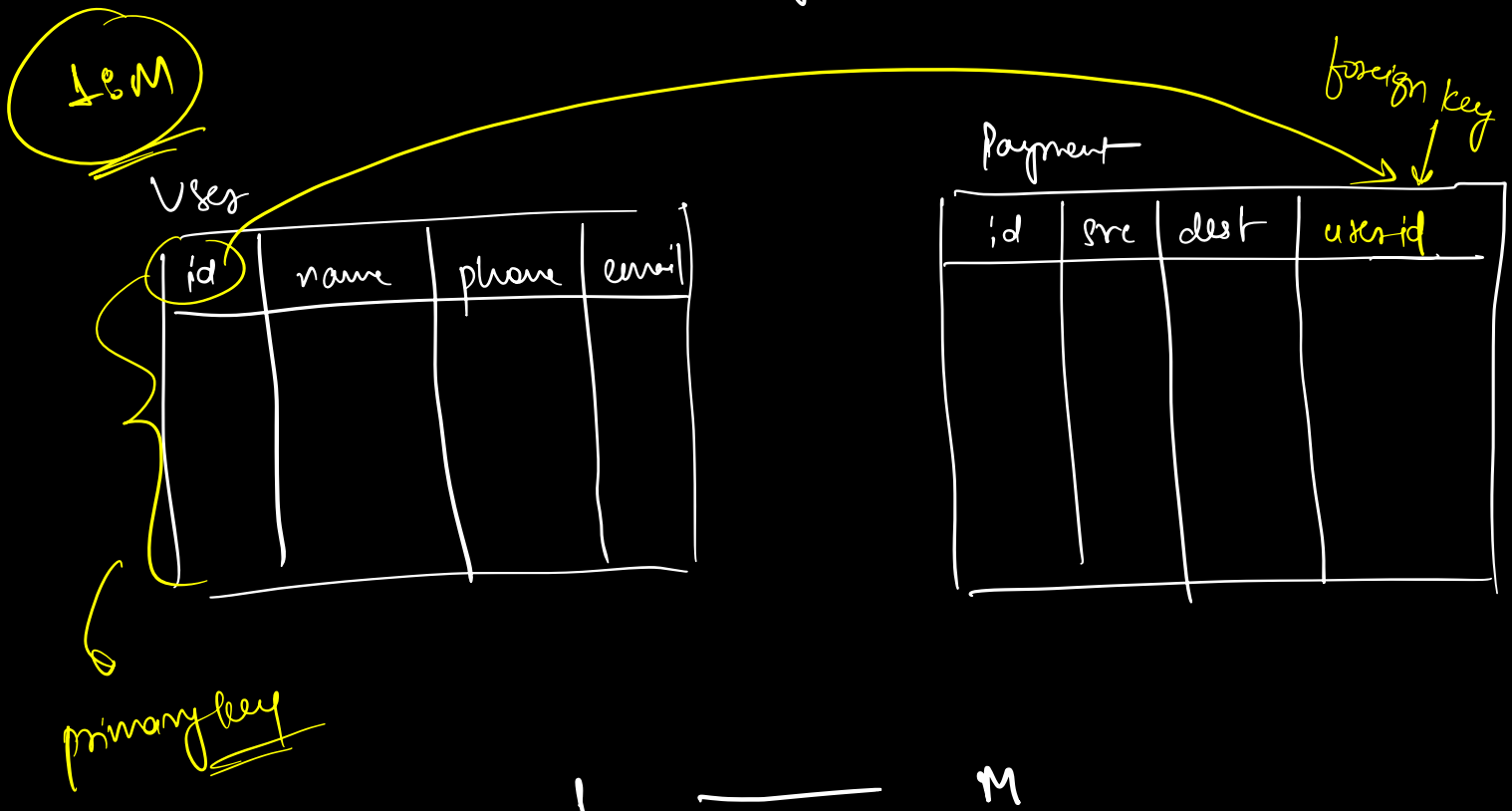


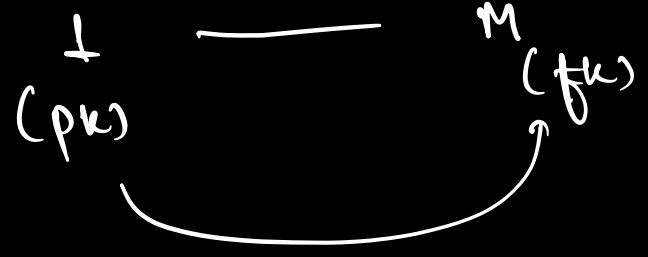
⇒ Schema design:

→ Found all entities for system ✓

→ found cardinality of entities ✓

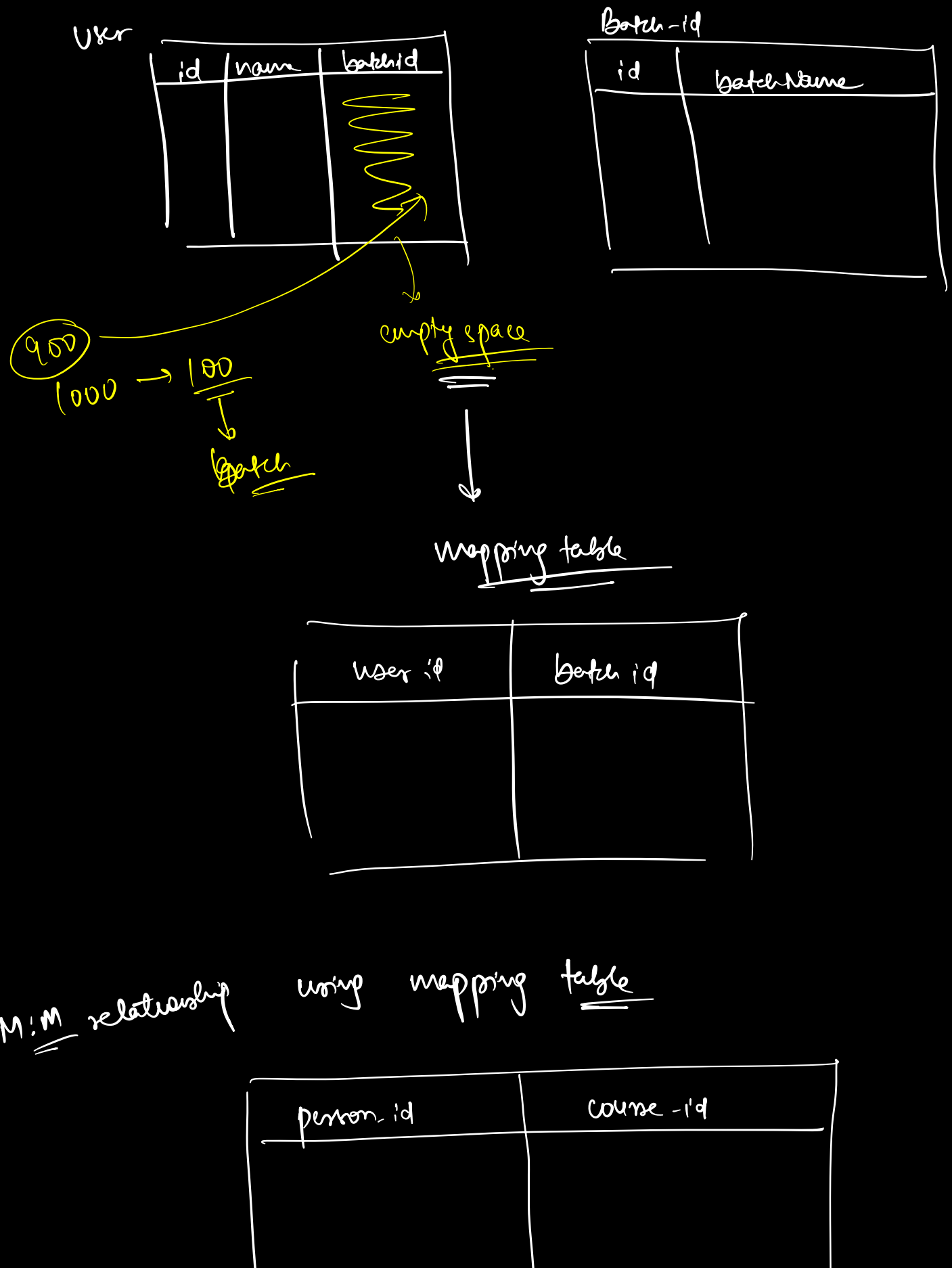
→ How do you represent cardinality





⇒ mapping table

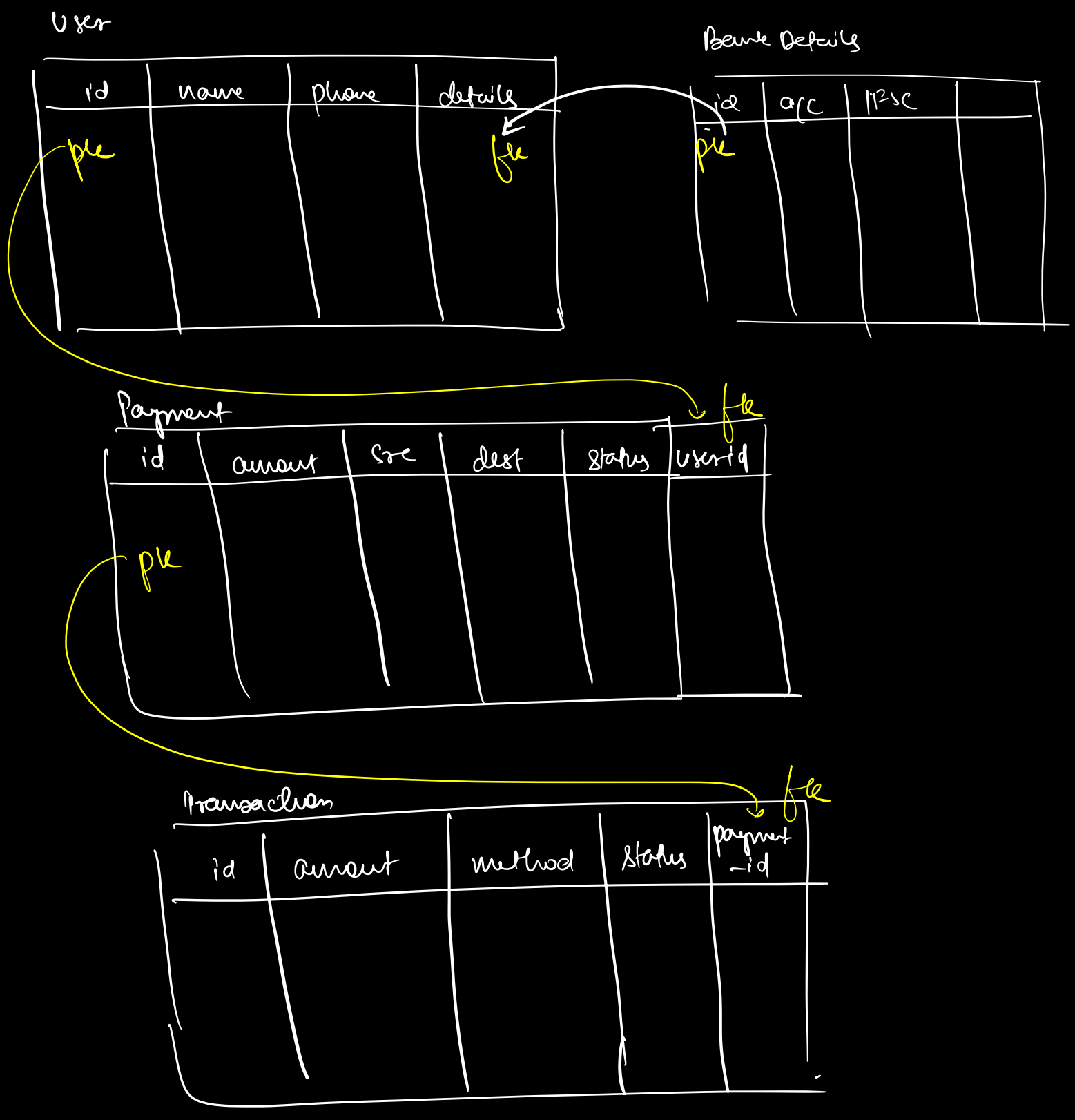
user-id	payment-id
1	1
1	2
1	3
2	10
2	15



M:M relationship using mapping table

person_id	course_id

person courses


User
id
name
phone
details
pk
fk
Bank Details
id
acc
IFSC
pk
Payment
id
amount
src
dest
status
userid
fk
pk
Transaction
id
amount
method
status
payment_id
fk