Agenda :

i) How important is LLD?

ii) LLD has different types of interviews

iii) How to start a design problem

iv) Steps to design something

v) Design LLD for payment apps :

1) class diagram

ii) Schema design

iii) Tips to code

⇒ 3 different types :

Technical Theory	Design Round	Machine Coding Round
→ TCS / Infosys / Wipro / Cgpu / PwC / Deloitte. - - -	→ Microsoft, Google, Amazon, Atlassian, Meta, Myntra	→ Swiggy, Cred, Uber, Flipkart, Scaler. Phonepe
→ Theory based or syntax based	→ Single live problem Statement	→ Problem statement is given end to end
→ minimal to no coding	→ Requirement gathering	→ Design it
→ minimal to no design	→ Design the system end to end	→ Write end2end functional code
→ 45 mins - 50 mins	→ Class diagram / Schema Design / Code Structure	→ Write test
		→ 2 hrs [120 mins]

→ How to start with a design problem?

→ LLD ⇒ Low level Design

↓

tells us how to write the code

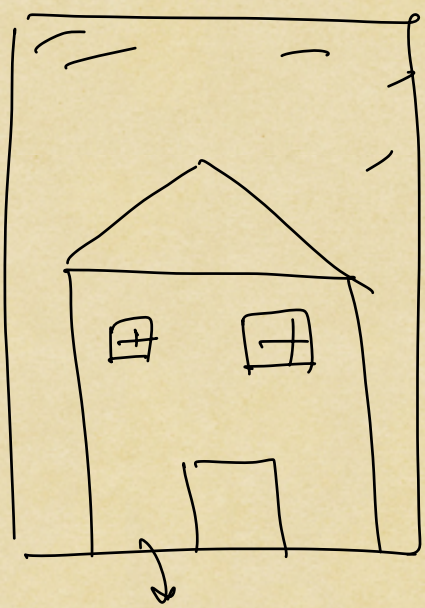
↓

how to write the code such that :

i) easy to read

ii) easy to maintain

iii) easy to extend



rooms → size

width → walls

washroom → size

taps / shower / pipelines / electric lines

maintain ⇒ running the code as it is

⇒ bug fixes, version upgrades, patch fixes

extend ⇒ adding new things

→ adding new features

→ adding new capabilities

⇒ Spend more time in discussing what/how to code than actually coding

→ Requirement gathering

→ Start the design
- use case diagrams
- class diagrams
- Schema design
- code structure

⇒ STEPS

∴ Step ⇒ 0 → Get an overview:

Align yourself with the thought process of the interviewer/PM.

Overview

you already know about the system

* align your understanding with the expectations of the interviewer.

you dont know about the system

* ask the other person to give you full understanding of the product

* What do you want me to design?

→ Entities

→ Entities & their relationships

→ Appn [end to end]

* Do you want me to persist data?

↓

Store in DB

↓

do you also want the schema design?

design of DB, how to store the data

* How will we interact with the system?

→ REST APIs

→ Command line interface

→ Queues [messaging queues]

→ Hard coded

Step - 1 → Requirement gathering

Gather and clarify requirements

→ Suggest ideas with rationale and ask we should support them

→ 5-8 core features

→ try to visualize the system

→ Jot down the points.

⇒ [→ for every feature, try to think about edge cases and future requirements.

⇒ Requirements for payment app:

1st step for User

* Signup → email Id → password or phoneNumber → OTP
* Login → email Id → password or phoneNumber → OTP

* Add profile, bank details and do kyc:

→ KYC
: name
: phoneNo.
: PAN
: Aadhaar

→ Bank Details
: Bank A/c No.

: IFSC code

: Branch Name

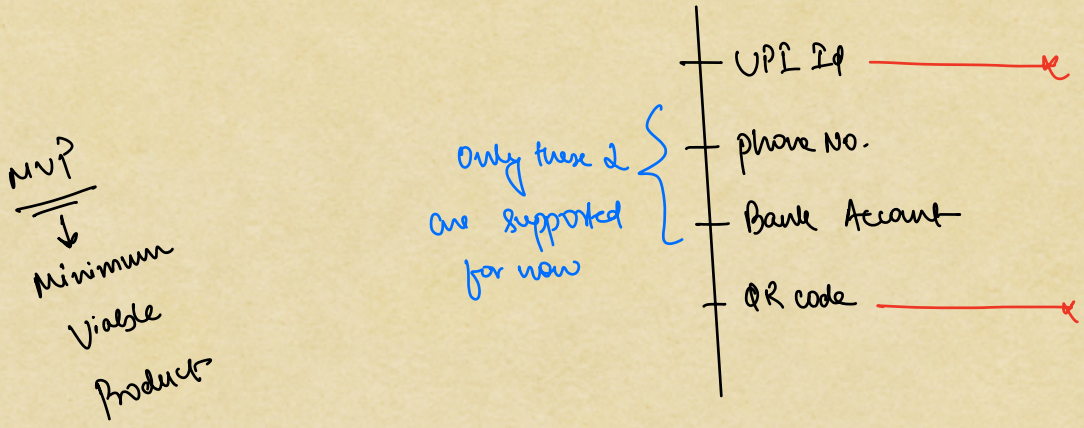
: A/C Holder Name

: Card Details

* make payment

[send money to someone else]

via :



MVP?

↓

Minimum Viable Product

* source of payment :

→ Debit card

→ Credit card

→ UPI

wallet →

* see history of payments

→ most recent first

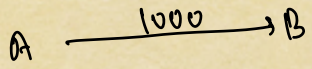
→ pagination

→ 10 transactions / page

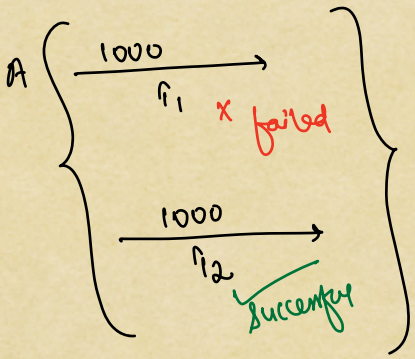
How do you handle transactions?

i) We want to allow retries

ii) We should allow multi-transaction payments

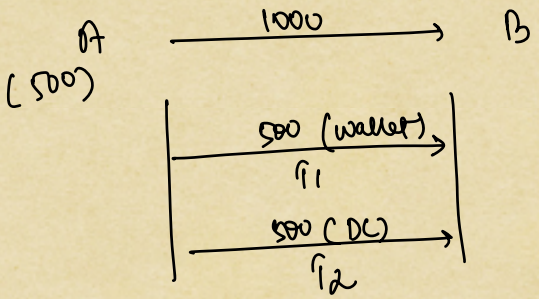


a)



⇒ 2 transactions
(1 failed, 1 successful)
for 1 payment

b)

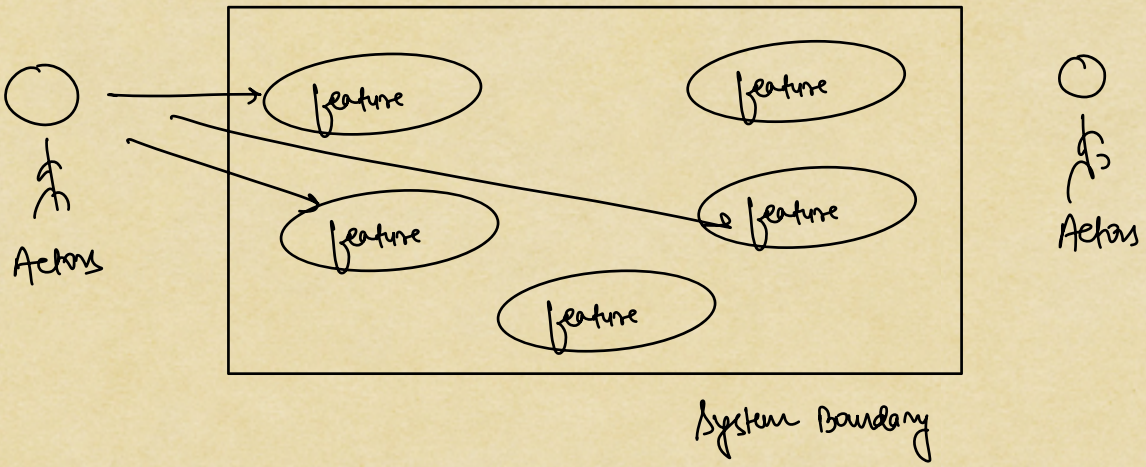


⇒ 2 transactions
(2 successful)
for 1 payment

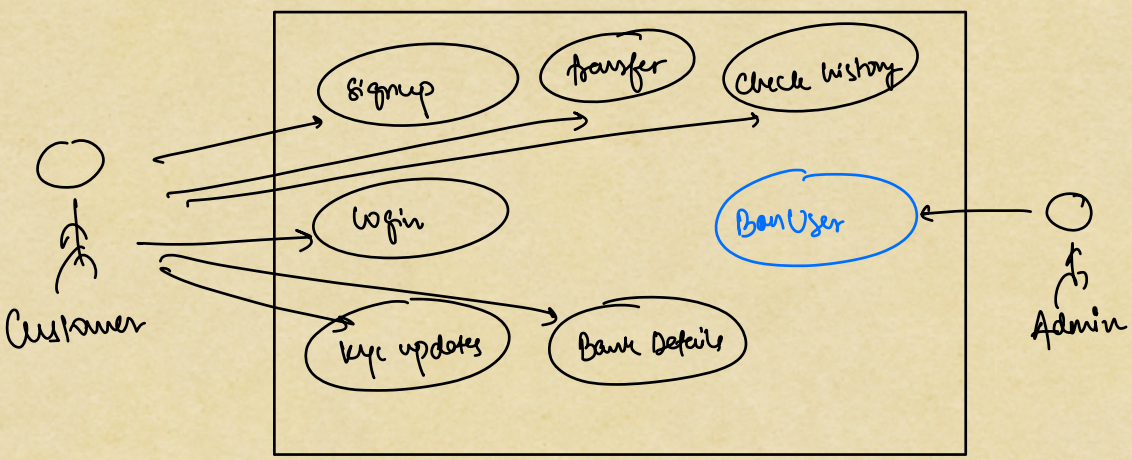
Atomicity
ACID

iii) If sender/reciever has any problem and any transaction fails, we will cancel the whole thing and rollback.

Step2 → Use Case Diagram



→ use case diagram for payment app :-



↓

Use case diagram should contain the features discussed in requirement gathering.

Step 3 ⇒ CLASS DIAGRAM.

- entities
- relationship b/w entities
- enums, interfaces, abstract classes etc.
- design patterns that we will use

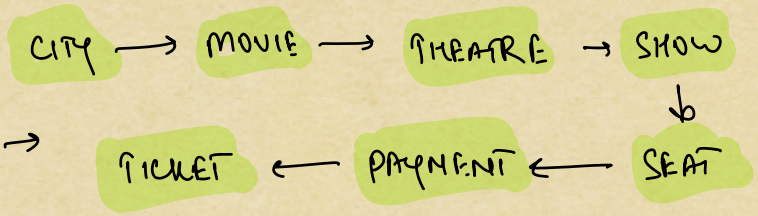
↓

while following SOLID principles

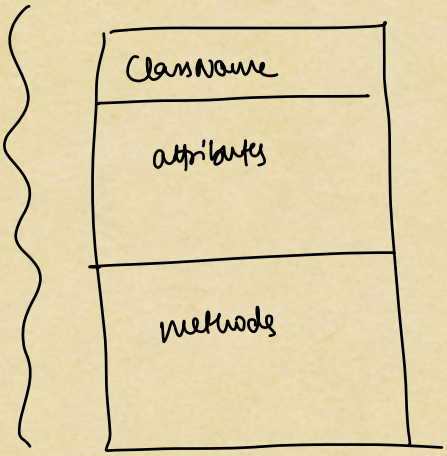
1) NOUNS IN YOUR REQUIREMENT

2) VISUALISE THE USER JOURNEY

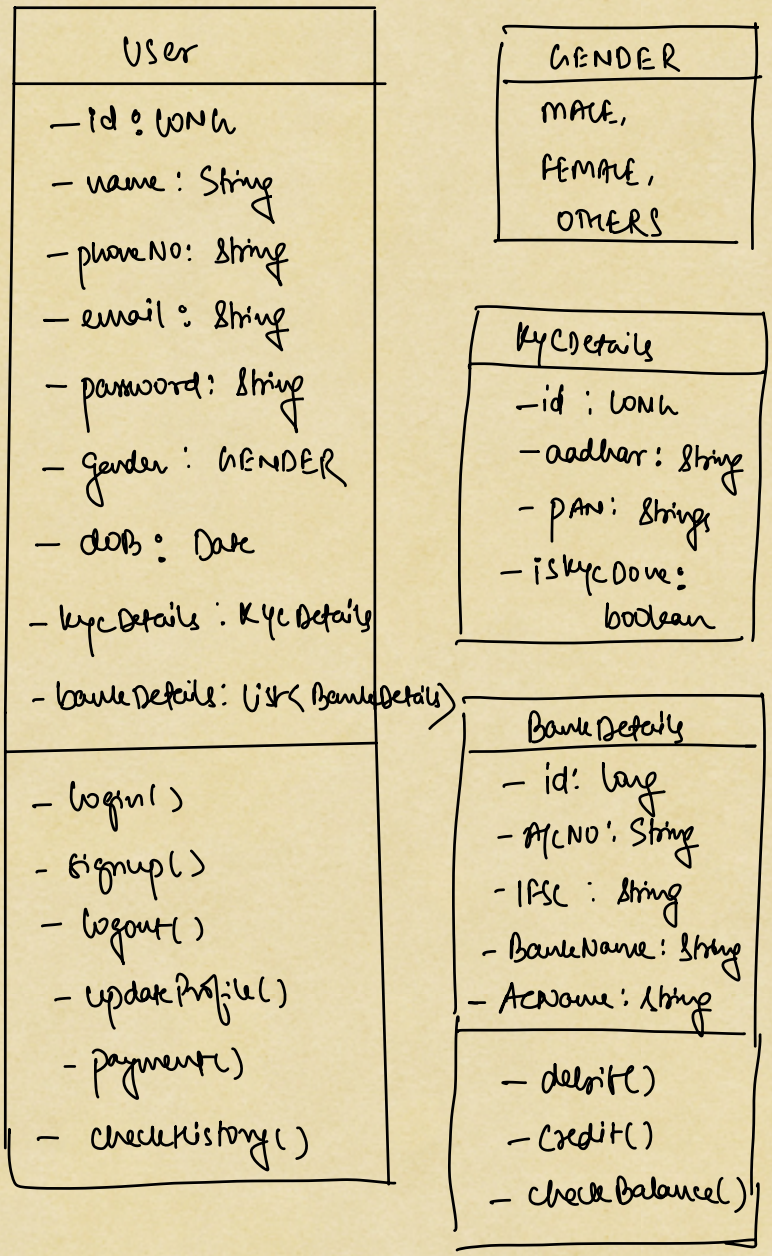
example → BookMyShow



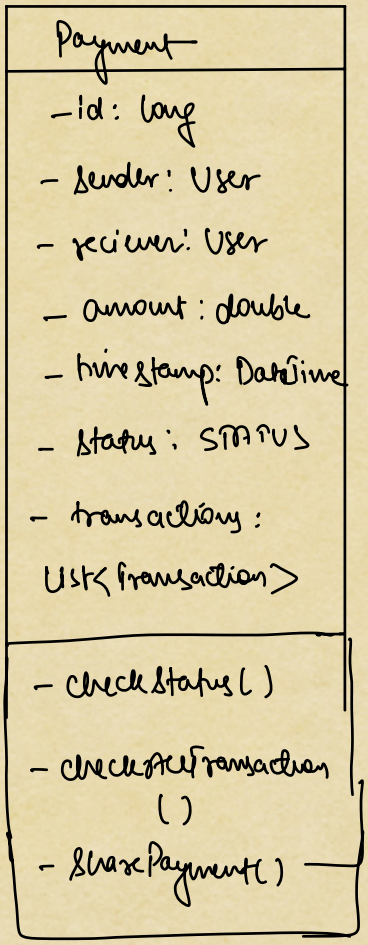
→ Entities for payment app:



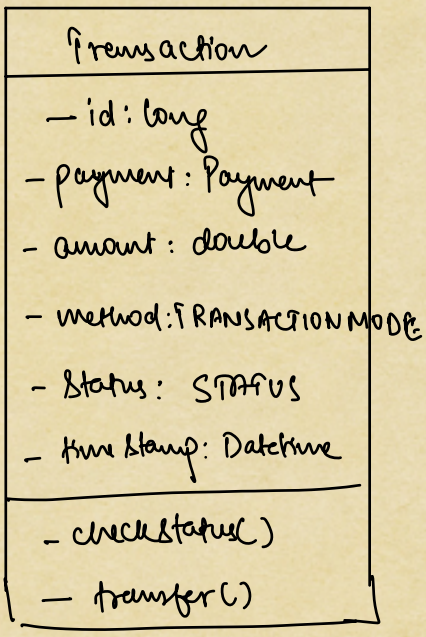
1) User

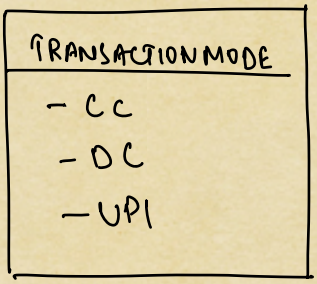

User
- id : LONG
- name : String
- phoneNO : String
- email : String
- password : String
- Gender : GENDER
- DOB : Date
- kycDetails : KycDetails
- bankDetails : List<BankDetails>
- login()
- signup()
- logout()
- updateProfile()
- payment()
- checkHistory()
GENDER
MALE,
FEMALE,
OTHERS
KycDetails
- id : LONG
- aadhar : String
- PAN : String
- isKycDone : boolean
BankDetails
- id : long
- ACNO : String
- IFSC : String
- BankName : String
- ACName : String
- debit()
- credit()
- checkBalance()

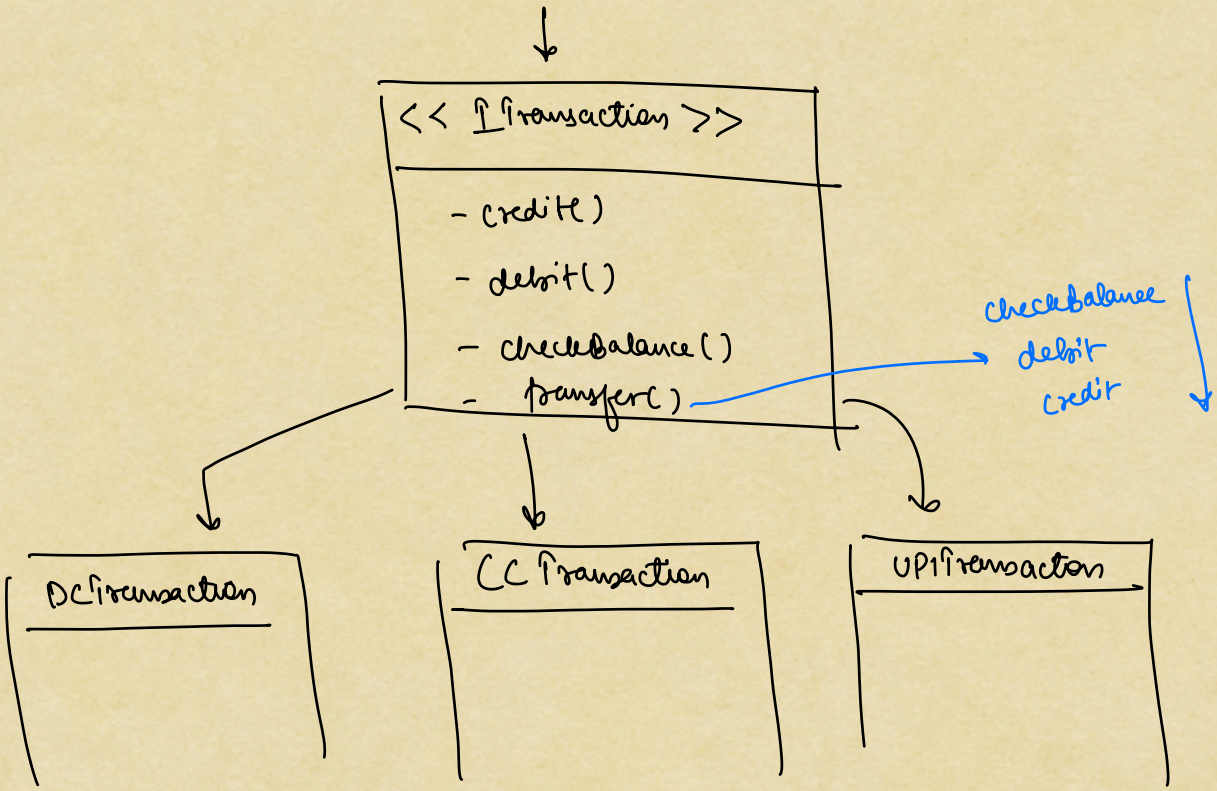
2) Payment



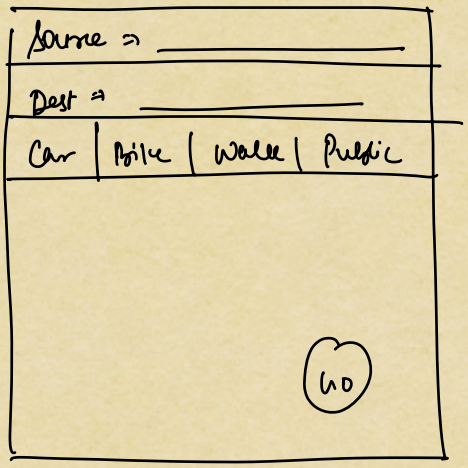
3) TRANSACTION







Strategy Design Pattern



task is achieved in any mode of transportation

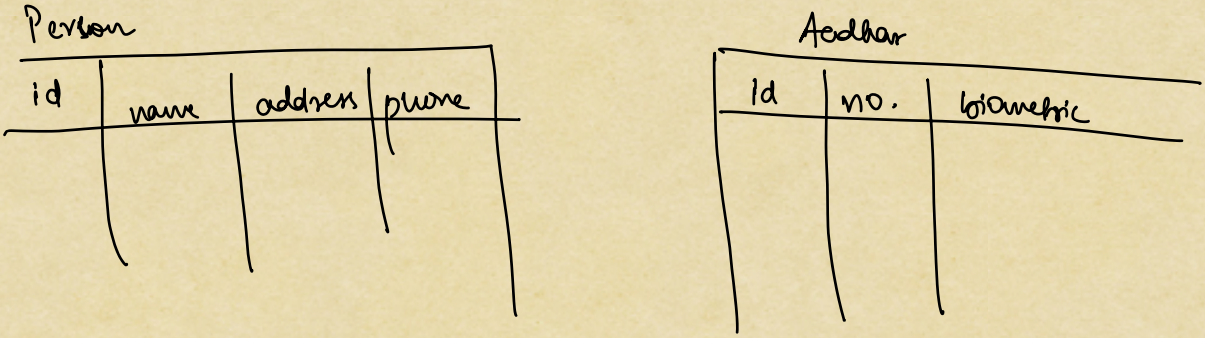
* Whenever we have multiple ways of achieving our goal and we want to keep the flexibility of choosing either one of the ways. we generally use **Strategy Design Pattern**

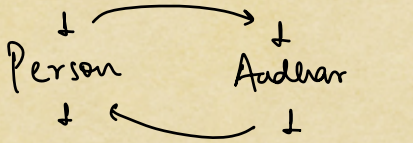
* We will have to create list of payment objects during execution.

⇒ Factory Design Pattern

CARDINALITY:

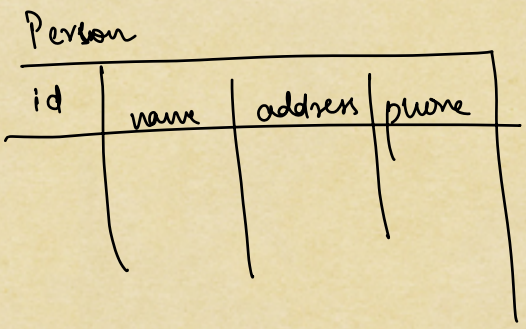
1:1 | 1:M | M:M

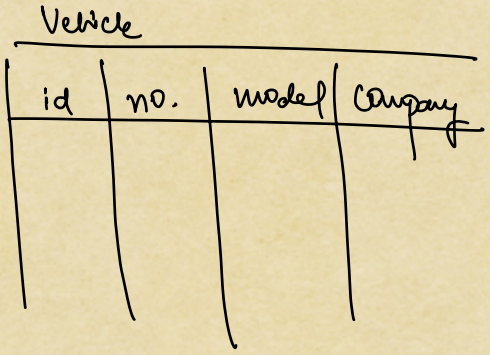


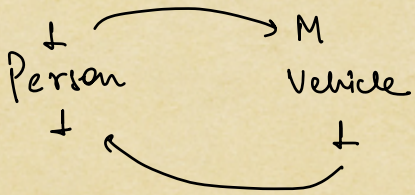


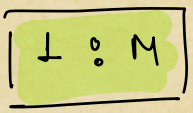
1 : 1

field LR RL
max max



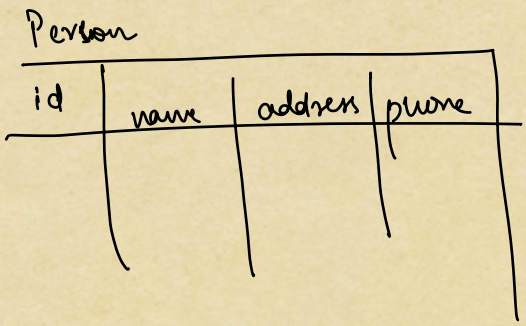


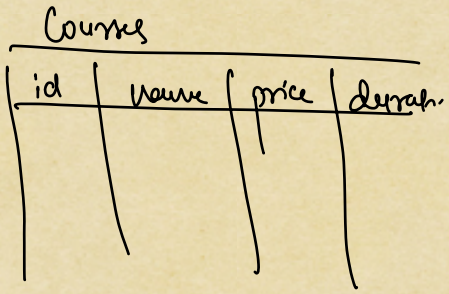


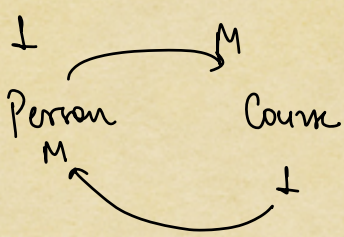


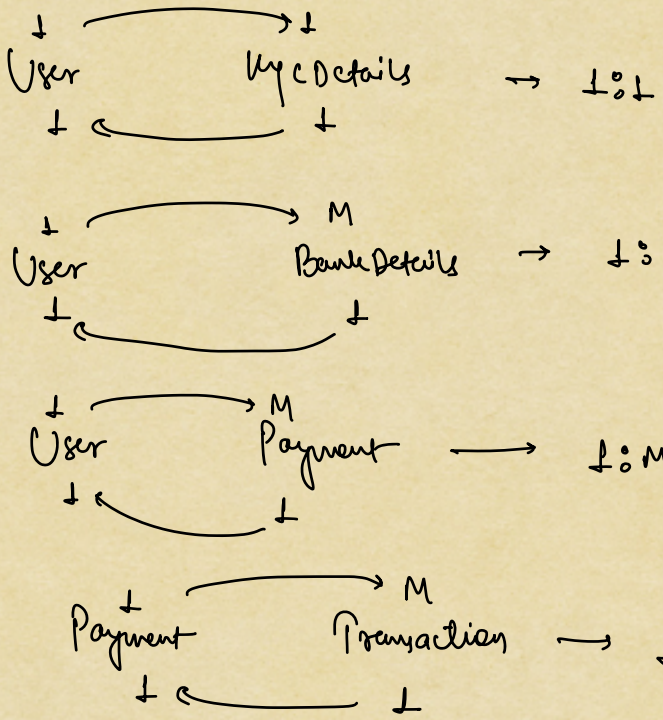
Person - Vehicle ⇒ 1 : M

Vehicle - Person ⇒ M : 1









⇒ Schema Design

i) Found all entities ✓

ii) Found all cardinalities among entities ✓

iii) How do you represent cardinalities.

User

id	name	email	phone	.	--

(PK)

KYC Details

id	aadhaar	PAN	kycDonestatus	UserId (fk)

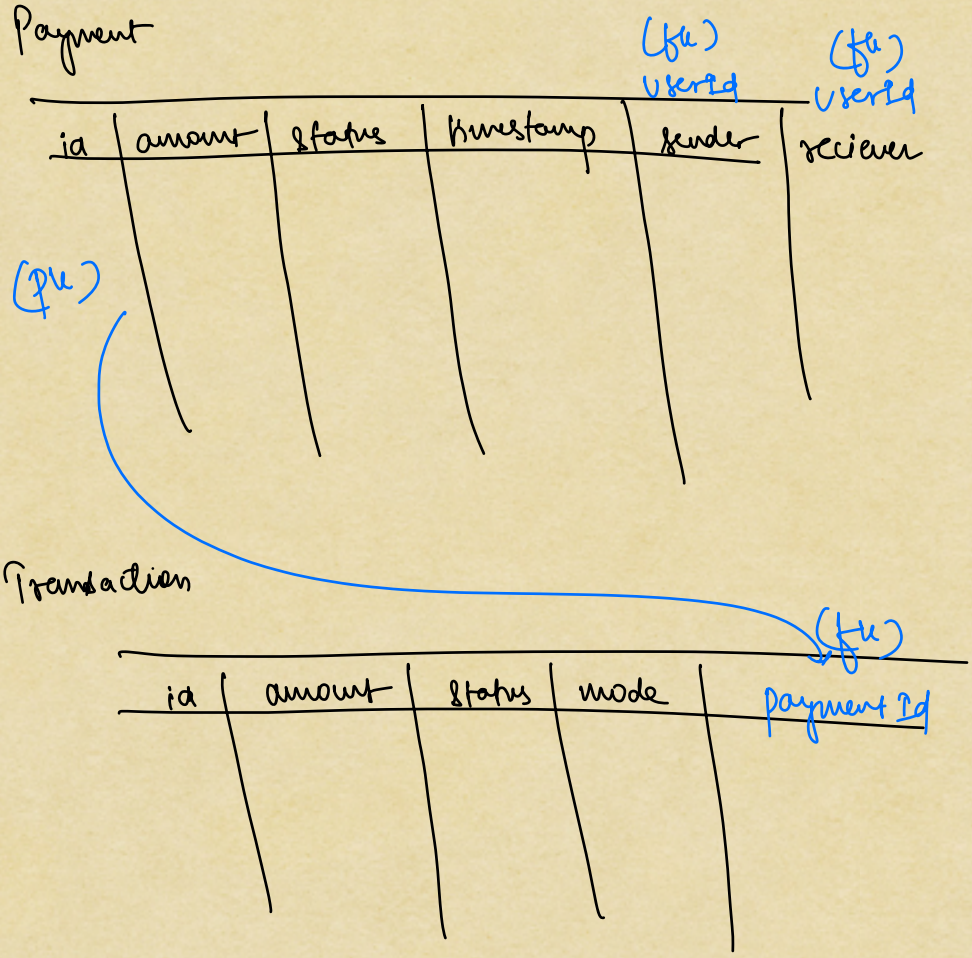
1 : 1 ⇒ pk of 1 side ⇒ fk to other side

Bank Details

id	ACNO.	IFSC	Bank ---	UserId (fk)

1 : M ⇒ take the pk of 1 side
put it in as fk in M side

User → 1 : M → Bank Details



eg → M:M

Person

id	name	address	phone
1	Raghi		
2	Desisai		
3	Richa		
4	Vinay		

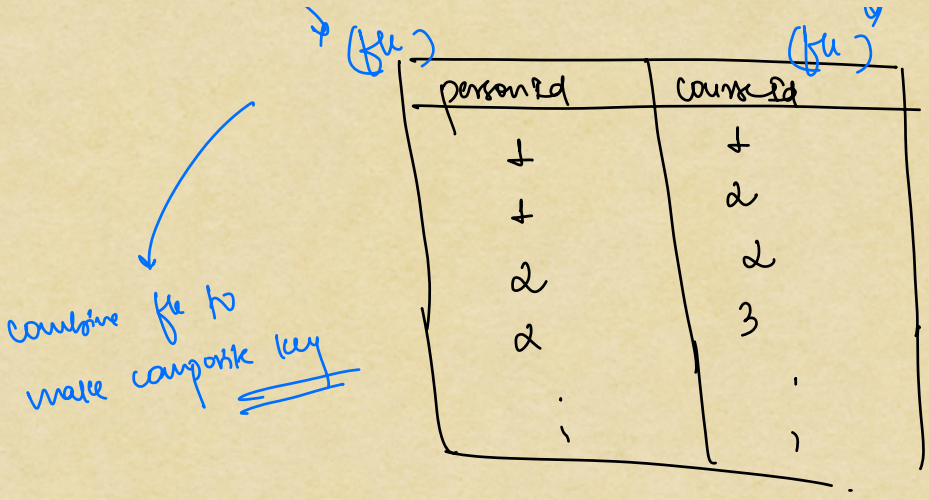
Courses

id	name	price	desp
1	Scaler		
2	Gym		
3	Swim		
4	Music		

M:M

↓

mapping table



→ Scaler.com/topics

→ Head first Design Pattern