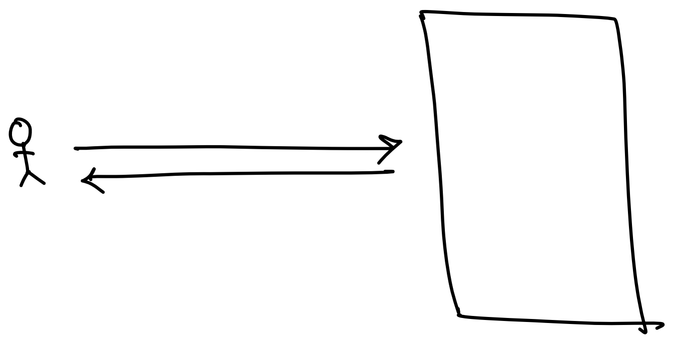
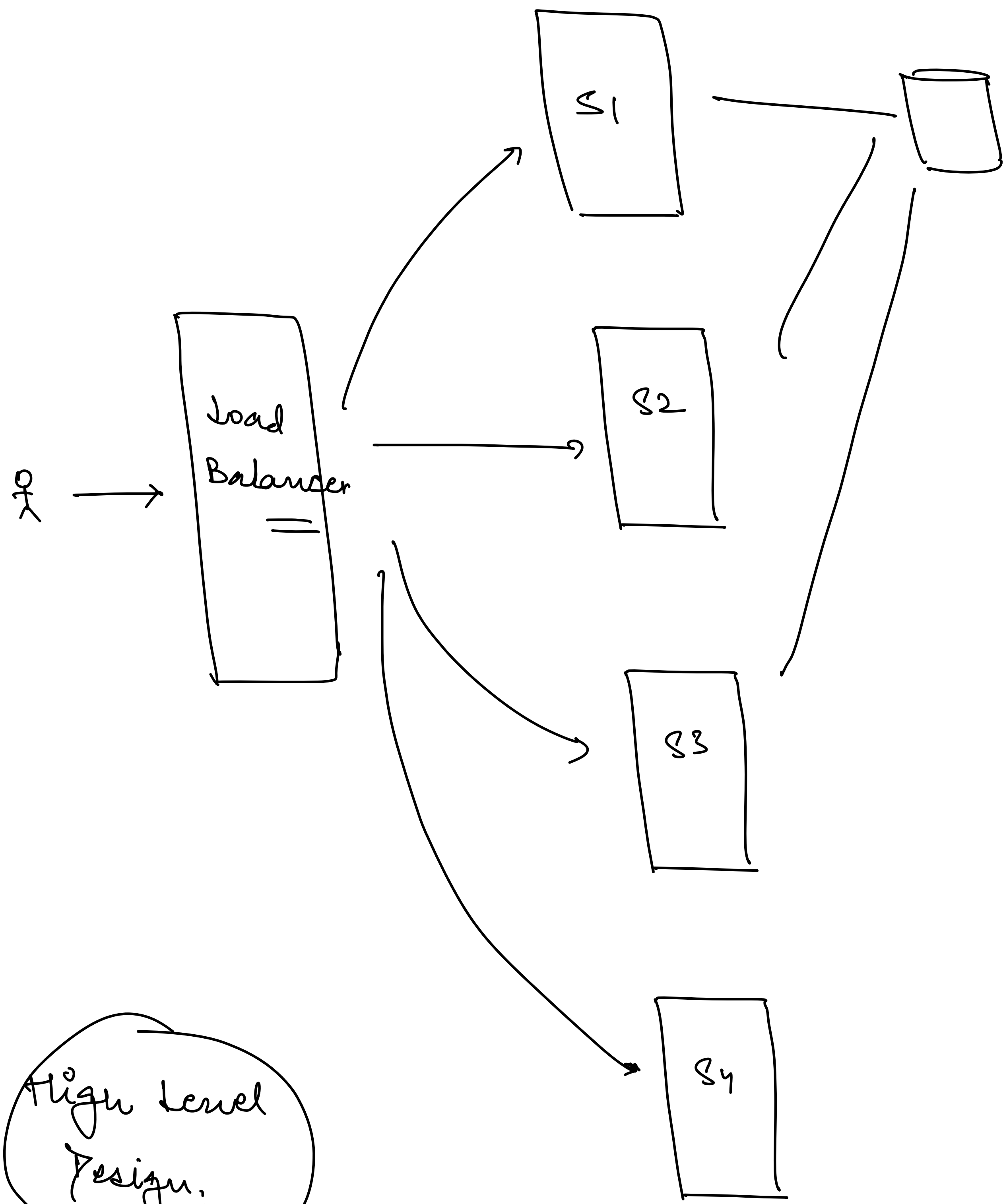
Agenda.

1. What is LLD ?
2. Why LLD is important ?
3. How to approach any LLD problem ?
   - Requirements Gathering
   - Class Diagram
   - Schema Design
4. LLD of Payments Apps.
5. Doubts.

LLD : Low Level Design. ⇒ High Level Design. } Overview
Not going into much details.



→ Overloaded
→ Single Point of failure



High Level Design.

⇒ Overview of Different layers interacting with each other to serve the request.

⇒ LLD: Study of software running inside the m/c.

How to write good software.

⇒ CODE.

⇒ How to write good code.

LLD.
- → Understandable & Readable.
- → Extensible
- → Maintainable.

→ Which classes | Abstract Class | . . .

② Why LLD is important.

⇒ To write good code in our work.

⇒ To get Hired.

⇒ Debugging
Designing
Testing
Requirements
Code Review

} Interacts.

⇒ At least one $\boxed{\text{LLD round.}}$

→ Problem statement.

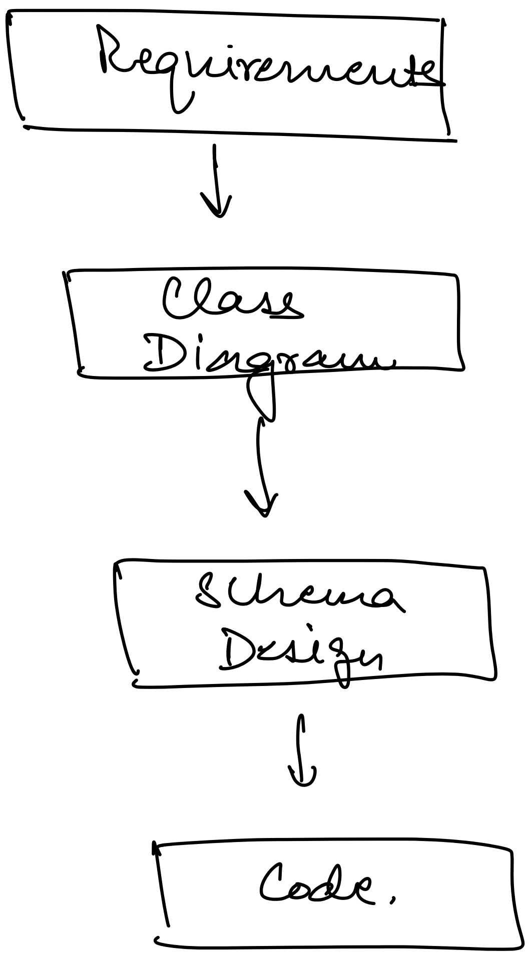
(2-3 hours)

Machine Coding } → Flipkart | Cred | Swiggy | Zomato | Phonepe | Paytm | . . . . .

→ Requirement Gathering (5-8 core features)
→ Class Diagram.
  ↳ What all the classes / interfaces will be there in the system.
  → Design Patterns
  → Design Principles (SOLID)
→ Schema Design.
(Database)

- → Tables
- → Columns in a table
- → Relations b/w the ~~tables~~. (Cardinalities)

→ CODE.



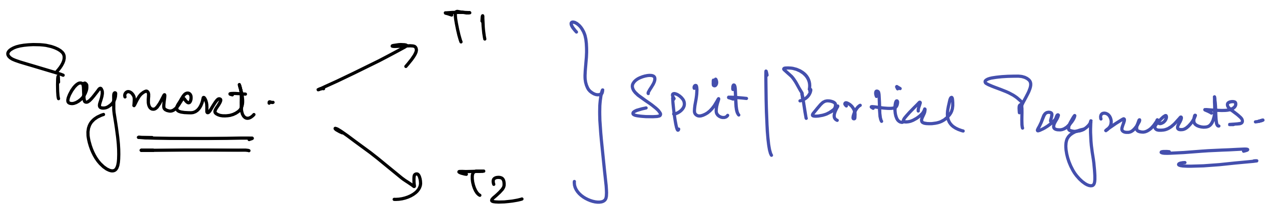
⇒ Interview Problem.

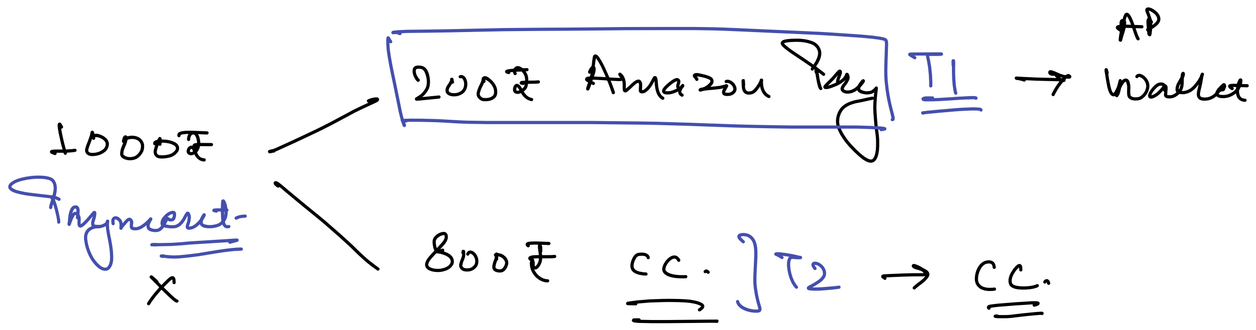
⇒ Design a Payment Apps.

1. Requirement Gathering ✓
2. Clarify requirements. → Edge Cases.
3. Class Diagram.
   ⇒ Entities | Classes...

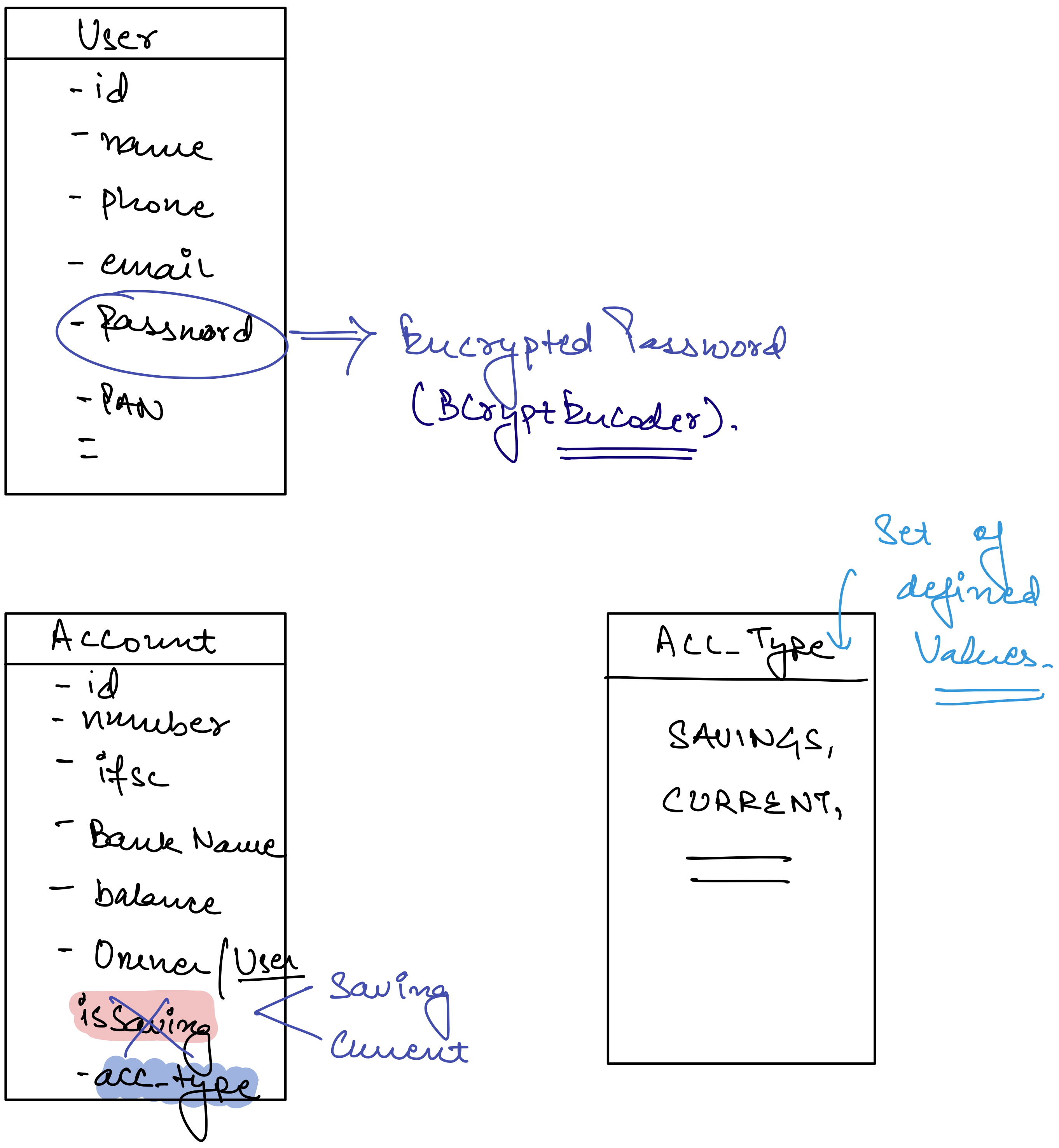
⇒ Go through all the requirements and find the Nouns (Entities) for which we want to store the data in our systems.

⇒ Class.





⇒ Class Diagram.



Payment

source (user)
destination
amount
id
time
~~isSuccess~~
Status
List<Transactions>

Transaction

- id
- amount
- MODE — UPI, Wallet, CC, DC
- time
- Status
- Source
- dest

Status

SUCCESS,
FAILURE,
IN_PROG,
REFUNDED,

SCHEMA DESIGN.

→ Assume it to be a Relational DB.

→ Tables.

→ What all the columns.

→ Relationships b/w the table.

⇒ For every class that we have come up in the class diagram, create a table for that.

Users

id	name	Phone	email	Password . .	.

accounts

id	acc-no	ifsc	branch	

transactions

id	payment-id

Payments

id	~~[- - - - ·]~~

Status

id	Value

acc_type

id	Value

Primitive

1) Store simple attributes as it is in the DB tables.
(String | Bool | int | ...)

2) for non primitive attributes, find the Cardinality & apply the respective rules.

Payments

—	—	— —	transactions
1	1000	—	~~[T₁, T₂, T₃ ...]~~

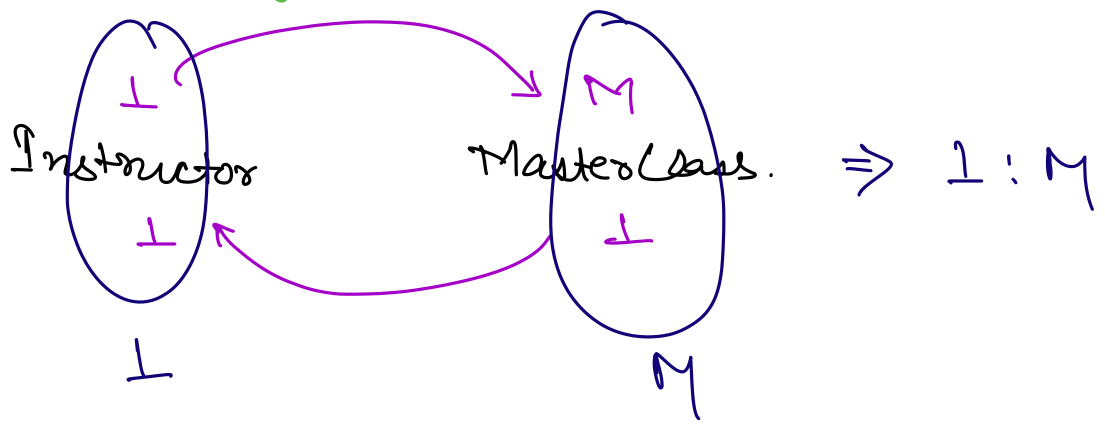
⇒ We can't store lists

Cardinality : How many A's are connected with how many B's.
A — B

1 : 1] ⇒ Id of one side on other side.

1 : M }
M : 1 } Id of ① side on Ⓜ side

M : M] ⇒ Mapping Table.



instructors

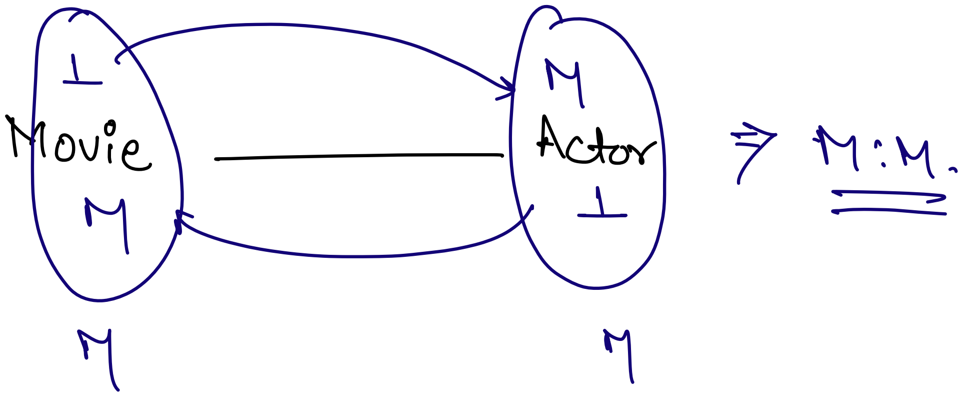
id	name	~~master_classes~~
1	Deepak	~~[1, 2, 3, ...]~~

✗

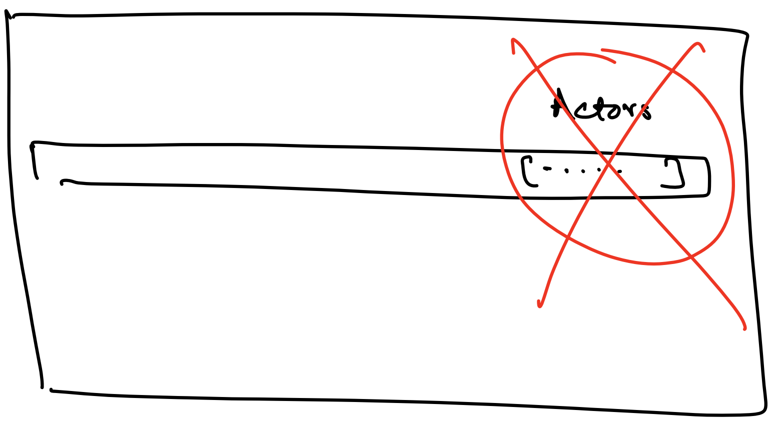
master_class.

id	title	instructor_id
1	LLD	123

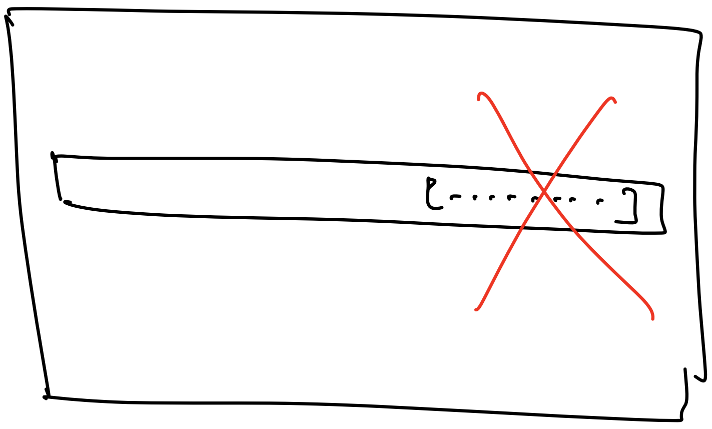
✓



movies



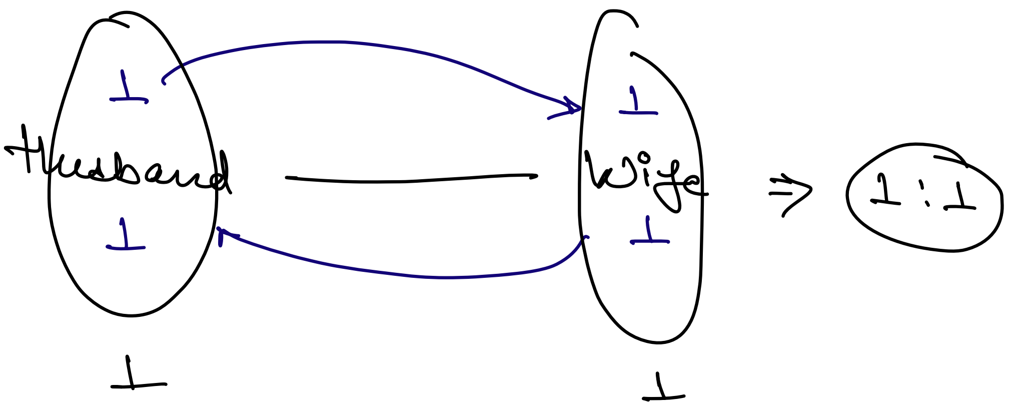
actors

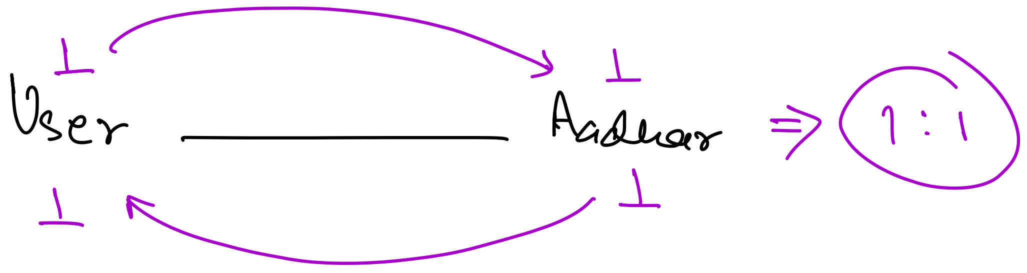


⇒ New table ≡ <u>Mapping</u> <u>Table.</u>

movie_actors

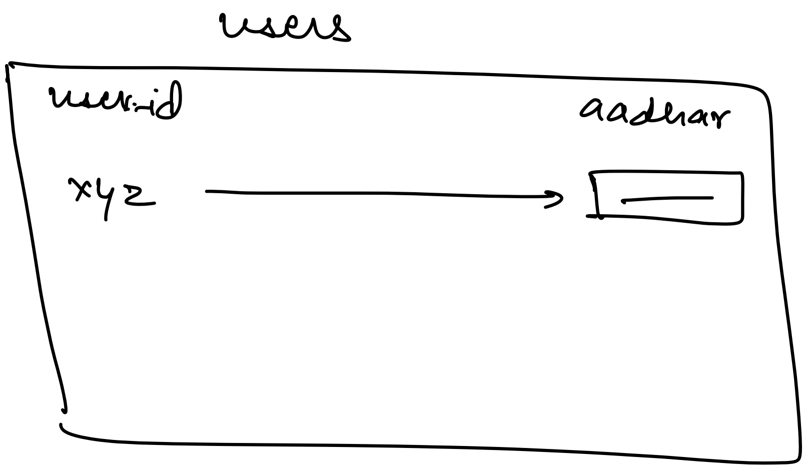
movie_id	actor_id
123	1
123	2
122	3
⋮	⋮

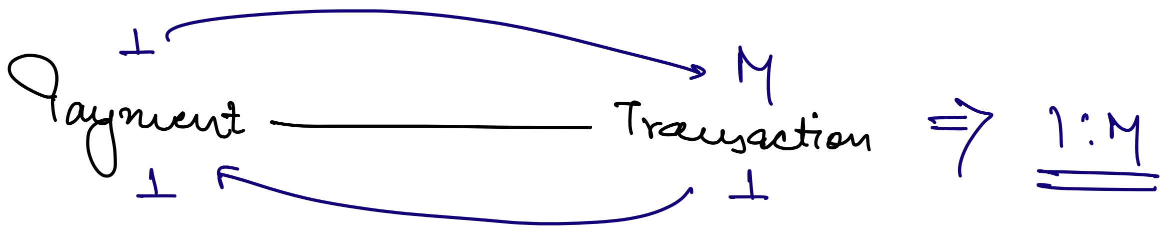




husbands

	wife_id
100	123


users
user_id
aadhar
xyz


1
M
Payment
Transaction
1
1
⇒ 1:M

*