* This is a recap of DBMS 1+2 with more and consistent set of examples.
* No new concept will be covered,.
* We might have a little longer session today.
* Transactions + Indexing. will be covered after MySQL.
* If you think you have understood DBMS 1+2.

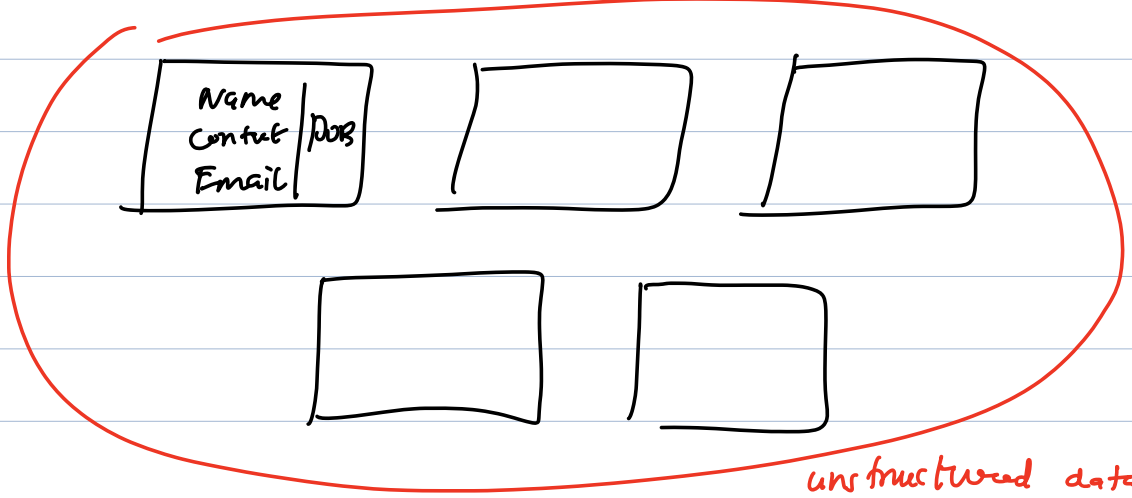
- ERD
- Cardinality — 1-1, 1-N, M-N Participation
- 1NF, 2NF, 3NF } Normalization
- FD.

* Own a restaurant

perks.

invite to a holiday party, who have ordered atleast 10 meals

50% off on their favorite dishes



unstructured data.

Entity

Customers

name	email	contact	DOB
Sahil	—	90- -61	17-11-99
Nishit	—	—	—

attributes/ cols/ fields.

Structured Data.

name	email	contact	DOB
Sahil	—	—	—
Nishit			
Neha			

Customers

order
Burger
Pie
Burger

Relational DBMS

Table

orders

Name
Burger
Pie
Cake

relationship

id

id	name	contact	dob	email
1	Sahil	91- -		def —
2				
3	Sahil	90- - -		abc —

Unique values appear only once in a column.

Unique identifier = Key.

Composite key => more than one cols to identify a record.

id	name	phone	dob
1	Sahie	—	17-11-1995
2	–	–	–
3	–	–	–

Customers

1 customer orders ~~1 item~~ ~~1 order~~ for now.

id	item_id	cust_id	name	timestamp
1	1	1	Burger	✓
2	1	2	Burger	✓
3	1	1	Burger	✓

Same item ordered >1 times.

Subjective

id	name	phone	dob
1	Sahie	—	17-11-1995
2	–	–	–
3	–	–	–

SK = { id, name, phone, dob ; id, name ; id, phone ; phone ; id, dob ; id }

Composite key = key that contains >= 2 attributes

CK ↓ PK

min. no. of cols. id, phone

1 out of CKs. id.

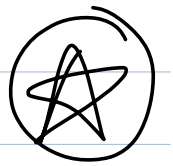
id

Alternate Key = what else could have been a PK.

= all the other CK.

phone

Relationships



* 1 cust can have 1 fav dish
* many cust can have same fav dish.

id	name	contact	email	dish_id
1	Sahil	90--61	- --	1
2	-	- -	-	1
3	- -	- -	- -	2

FK

Foreign Key.

on the many side

Customers

Referential Integrity

id	name	desc.	price
1	Burger	- -	100
2	Pie	—	₹00
3	-	-	₹00

Dishes

1 - many

1 - N relationship

1 dish — N customers.

N customers — 1 dish.

---

1 cust → N orders

1 order → 1 customer.

1 cust can have many orders.
1 order can have many dishes.
1 dish can be part of many orders.

1 order – N dish
1 dish – N orders

id	name	contact	email

Customers

Dishes

id	name	desc.	price
1	Dosa	—	100
2	Pie	—	200
3	Pizza	—	300

Order Id
1
1, 2
2

Order 1 has Dishes 1 & 2

FK.

id	cust_id	date	Dish
1	1	—	1, 2
2	2		2, 3

Orders.

Order 2 has dishes 2 & 3

Order Dishes

Order_id	dish_id	qty
1	1	2
1	2	1
2	2	2
2	3	1

In M-N relationships, we need a mapping table.

id	name	contact	email
1			
2			
3			

Customers

id	name	desc.	price
1			
2			
3			

Dishes

Cust-id	dish_id
1	1
2	2
3	1

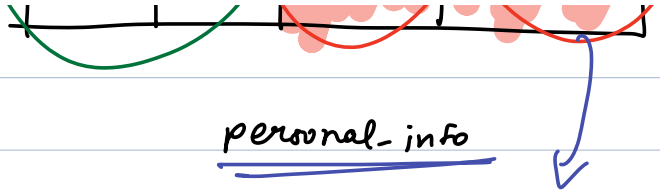
1-N

1-1 relationship

security purposes

id	name	contact	email
1	Sahil		
2			

Customers



personal_info

id	contact	email
1	90...61	- - -
2		

1—1

DBMS

↓

Access control

Table specific access

Entity
Attributes
Relationships

⇒ ER Diagram

⇒ ER Diagrams

↳ ⇒ Normalization

↳ FD

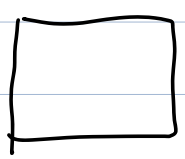
Cascading

Practically, we will do it in next class.

① Modeling & Planning the DB

(1) What does your DB need to store?

Entities



Entity.

Rectangle

→ customers

→ dishes

→ orders

→ ~~reservations~~

→ events

1 cust — n events

1 event — n customers

many - many.

M : N

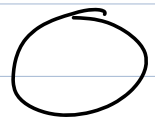
Q2) What info is stored in the entities?

Customers
- → firstName
- → lastName
- → email
- → phone
- → birthday
- → address
- → city
- → state.

Dishes
- → name
- → desc.
- → price

Events
- → name
- → date
- → location
- → desc.

Attribute



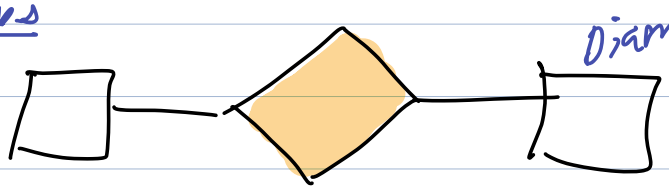
Oval

Orders.
- → date.

many dishes done by a customer.

Q3) How these entities are connected?

Relationships



Diamond.

Cardinality

1 : 1

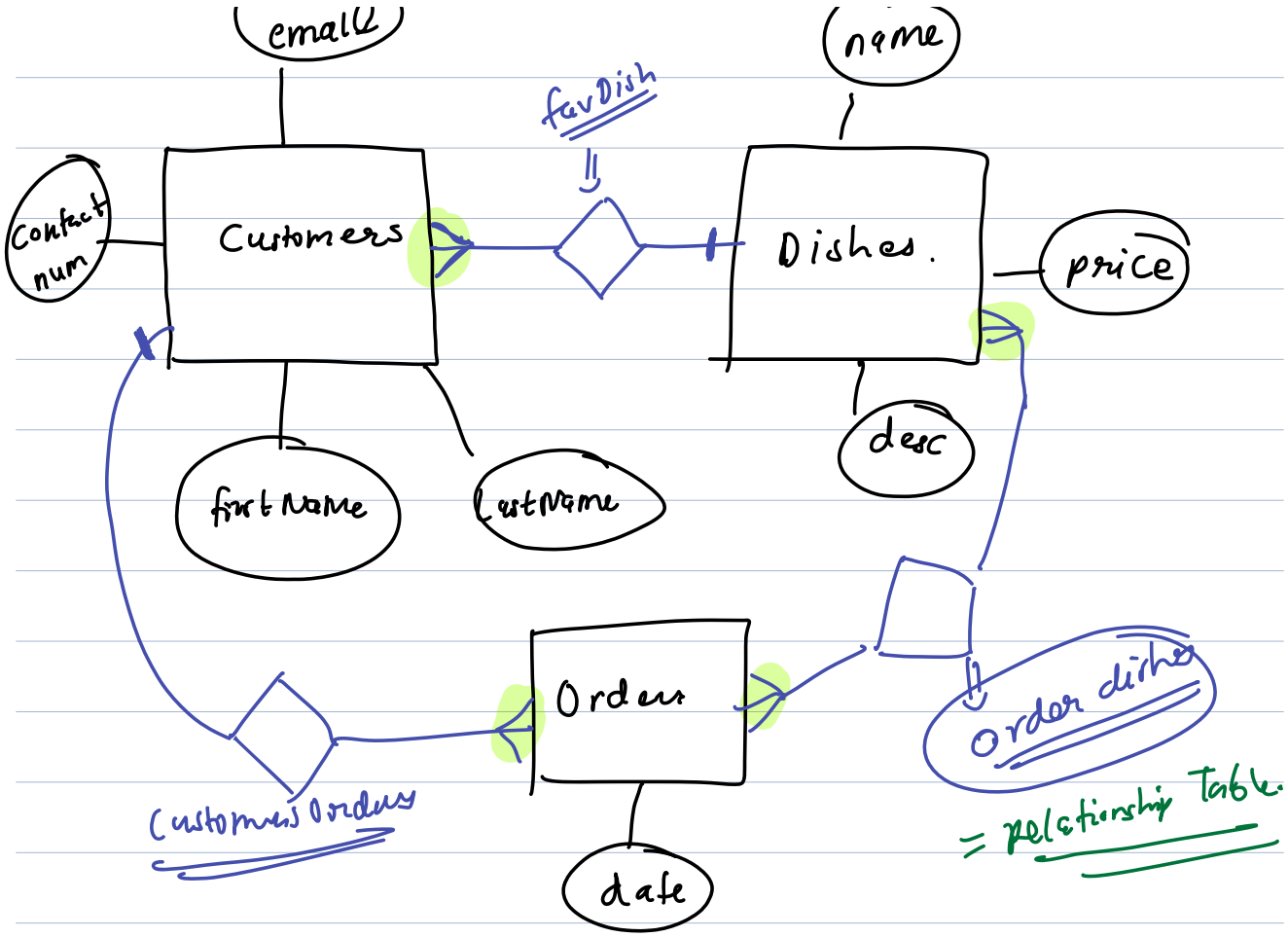
1 : N

M : N

fav.

1 dish — N cust

1 cust — N orders.



⇒ Normalization

- optimizing our DB.
- low redundancy
- high integrity.

✓ 1NF — 1 cust can have multiple fav dishes.

M:N ⇒ 1:N on both sides.

1 dish can be fav. for many customers

id	firstname	lastName	favdish 1	favDish 2	favDish 3
1	Sahil	Bansal	10	20	130

Not in 1NF

Values in each cell should be atomic and tables should not have repeating groups.

id	firstname	lastName	favdish
1	Sahil	Bansal	10,20,130

Not in 1NF

Cust_dishes

Cust_id	dish_id
1	10
1	20
1	130
2	20
3	20

Cust Events

m : n

2NF

Events

id	name	date	location
1	holiday party	29-04-22	Scalar, BLR
2	"	29-05-22	"
3	"	10-05-22	"
4	Winter Party	—	Coimbatore
5	"	—	"
6	Summer Party	28-05-22	Scalar, BLR

t_1

t_2

Two events with same name will happen at the same location.

Business Req.

$t_1[name] = t_2[name]$

$\Rightarrow t_1[location] = t_2[location]$

$\Rightarrow$ name → location (FD)

Key : {name, date}

Location is dependent on the name.

2NF: No value in a table should depend on (FD.) part of a Key that can be uniquely identify a row

$\Rightarrow$ There should be no partial dependency. (PK, CK.)

Events

name	date
Holiday	29-04-22
Holiday	30-04-22
Winter	31-01-22

2NF ✓

EventLocations

name	location
Holiday	SeaView, BLR
Winter	CBR

name → location

part or full key,

✓ Full dependency.

3NF

All the dishes at half the price in the lunch.

Dishes

1NF ✓

2NF ✓

id	name	desc.	price
1	Burger	—	100
2	Pie	—	200
3	Pizza	—	300

~~Lunch price~~
~~50~~
~~100~~
~~150~~

id → price → Lunch price.

Dish Lunch price

disc %
50 %

Transitive functional dependency.

3NF => Value should not be stored if they can be calculated from another non-key field.

=> No transitive dependencies.

Denormalization

* The process of intentionally duplicating some data, in violation of normalization rules.
* It is done to a previously normalized DB.

Orders

id	cust_id	price
1	1	100
2	2	300
3	2	200

Dishes

id	...	price
1		100
2		200

OrdersDishes

order_id	dish_id
1	1
2	1
2	2
3	2

price

Order 1 => 100
order 2 => 300
Order 3 => 200

JOINS

Revenue came from Cust 2 => 500

For every order, total price

* Slow server
* Huge DB
* Huge no. of request.

Speed needs to be optimized.