* Introduction to Normalisation

→ Database schema :- blueprint

It is a blueprint of the actual database, that we create. How we will store data, structure of the table etc. are defined in it. It also describes the relationship and constraints between the data elements, including tables, fields and relationships between tables.

→ Database instance :-

It is the actual database that we have prepared in the DBMS at any point of time.

→ Functional Dependency :-

In RDBMS (Relational database management system) we have to always remember 10 things :-

we refers
- ↳ Columns as attributes
- ↳ rows as tuples

- functional dependency defines relationships between two attributes (actually refering to 2 columns). We represent functional dependency with a notation like this :- $X \longrightarrow Y$

determinent $X \longrightarrow Y$ dependent

This actually means Y is functionally dependent on X. "Y" depends on "X", means that, for every valid value of "X" we can uniquely identify "Y"

For example :—

e_id	e_name	e_salary
1.	abc	1000
2.	def	2000
3.	abc	3000

→ Employee Table

So, let's assume you have a Employee table in your database. Now, here we can say that, their exists a bunch of functional dependencies between different columns, for example:

e_id ⟶ e_name (employee name depends on employee id)

e_id ⟶ e_salary (employee salary depends on employee id)

Why? Because employee id (e_id) is going to be unique for every employee. So for unique employee id we can find the name of that employee and same for the salary.

But, just think about one thing, can we write like this –

e_name → e_salary ✗ { This is not a functional dependency

Can we say that e_salary can depend on e_name. Is this a functional dependency, the answer is **No**

Because, two employees can have same name. and now we can not identify salary uniquely by the name of the employee. because there are multiple employees with the same name.

- Functional dependencies actually helps us to reach a better database design because using functional dependency, we can identify keys of a table and we can identify a bunch of anomilies in the database that we have created which leads us to **Normalisation**.

* Some rules of Functional dependency / Properties

1. **Reflexivity** :–

If A is a set of attributes and B is a subset of A, then the functional dependency A → B holds true.

For example :–

{roll_no, name} → name is valid

2. Augmentation :- (Partial Dependency)

The rule of augementation states that if $X \rightarrow Y$ is a functional dependency, then for any Z that is a subset of X, we can also infer the functional dependency $Z \rightarrow Y$. This means that if we know that the value of Z determines the value of Y, then we can also infer that any subset of Z also determines the value of Y.

For example :-

Project id

e_id	e_name	e_age	p_id	e_address
1	Abc	21	11	a-1
2	def	21	12	a-2
3	ghi	21	13	a-3
2	def	21	14	a-2
3	ghi	21	12	a-3

There is a problem with this database design. Let's see can we say,

e_id $\rightarrow$ e_name { e_name is functionally dependent on e_id

Can we say that if we want to identify name uniquely, we can identify it using e_id and p_id both together.

(e_id, p_id) $\rightarrow$ e_name

The problem is if we want to change the e_address of e_id with '2'. Now we have to go every row and change the address (see arrows in table).

Because we have stored redundant data. So there is existing partial dependency

→ How to identify partial dependency ???

(e_id, p_id) ⟶ e_name

Left hand side ← (e_id, p_id) ⟶ e_name → Right Hand side

The Right Hand side could have been identify just by single attribute but here we have another attribute also than can be also used to uniquely identify e_name. but this creates a problem like data repetition.

Note:

> RHS should be completely dependent on LHS, not just part of it.

3. Transitivity :-

If $X \to Y$ and $Y \to Z$ then $X \to Z$

Example :-

e_id → e_zipcode
e_zipcode → e_state
⇒ e_id → e_state

* Database keys :

Keys are set of attributes that helps us to uniquely identify a record in different situation.

There are different kinds of keys : —

→ Super key

→ composite key

→ candidate key

→ primary key

→ foreign key

→ alternate key

Super key : —

→ A set of attributes within a table than can uniquely identify a record.

For example : —

e_id	e_name	e_phone

{ (e_id)
(e_id, e_name)
(e_phone)
(e_phone, ename)
⋮
}

- This set of all the attributes is called super keys.
- Multiple super keys can exist

Candidate key :—

Minimum set of attributes that can uniquely identify a record.

For example :-

e_id
e_phone
} These are 2 different ~~keys~~ candidate keys for the table.

Composite keys :—

A key that consists of 2 or more than 2 attributes, that together uniquely identify a record.

→ There should be atleast two attributes

→ The attributes that for composite key are not ~~only~~ any key independently.

For example :—

student table

s_id	course_id	marks
1	1	80
1	2	40
2	1	30
2	3	30

We can't take just s_id, because it can not identify alone uniquely because there can be multiple record with same s_id. Same for course_id.

But sid along with course_id together can identify record uniquely.

(s_id, course_id)

↪ Composite key can be a candidate key but a candidate key can be with just one attribute or more than one attribute.

→ Composite key will always have atleast 2 attributes

Primary key :—

There can be more than one candidate key. We can choose any one non-null candidate key to become primary key.

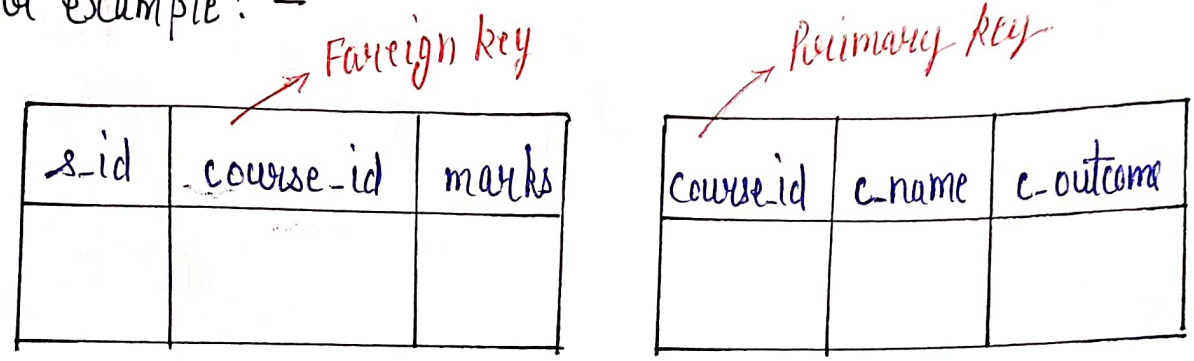
Alternate key :—

All candidate keys apart from primary key are alternate keys.

Foreign key :-

It is an attributes which is primary key in some other table.

For example :-



Primary key is also present in other table, so that one another key is foreign key. because this course_id is a primary key of some another table,

So by defining relationship b/w two tables, or multiple tables, if you find a attribute which is Primary key of some another table, then that attribute in your table will called as Foreign key.

* Functional dependency closure :-

f^* → it contains all the rules implied by a functional dependency.

→ $f \rightarrow \{ A \rightarrow B,\ C \rightarrow D,\ B \rightarrow C \}$

$f^+ \rightarrow \{ A \rightarrow A, A \rightarrow B, B \rightarrow C, A \rightarrow C, C \rightarrow D, A \rightarrow D,$
$B \rightarrow D \ldots\ldots\ldots \}$

→ Attribute closure : —

It defines all the attributes that can be determine using an attribute.

For example : —

$A^* \longrightarrow B\ C\ D$

$B^* \longrightarrow B\ D$