Unit-3: DATA BASE DESIGN

Functional dependency is a relationship that exists when one attribute uniquely determines another attribute. The functional dependency is a relationship that exists between two attributes. It typically exists between the primary key and non-key attribute within a table

- If R is a relation with attributes X and Y, a functional dependency between the attributes is represented as X->Y, which specifies Y is functionally dependent on X.
- Here X is a determinant set and Y is a dependent attribute
- Functional dependency in a database serves as a constraint between two sets of attributes.

Example: Assume we have an employee table with attributes: Emp_Id, Emp_Name, Emp_Address.

Here Emp_Id attribute can uniquely identify the Emp_Name attribute of employee table because if we know the Emp_Id, we can tell that employee name associated with it.

Functional dependency can be written as:

- Emp_Id → Emp_Name

Advantages of Functional Dependency

- Functional Dependency avoids data redundancy. Therefore same data do not repeat at multiple locations in that database
- It helps you to maintain the quality of data in the database
- It helps you to defined meanings and constraints of databases
- It helps you to identify bad designs
- It helps you to find the facts regarding the database design

Decomposition:

- Decomposition is the process of breaking down in parts or elements.
- It replaces a relation with a collection of smaller relations.
- It breaks the table into multiple tables in a database.
- It should always be lossless, because it confirms that the information in the original relation can be accurately reconstructed based on the decomposed relations.
- If the relation has no proper decomposition, then it may lead to problems like loss of

information.

- Decomposition is used to eliminate some of the problems of bad design like anomalies, inconsistencies, and redundancy.

Following are the properties of Decomposition, or types of decomposition

1. Lossless Decomposition
2. Dependency Preservation
3. Lack of Data Redundancy

Lossless Decomposition

- Decomposition must be lossless. It means that the information should not get lost from the relation that is decomposed.
- The lossless decomposition guarantees that the join of relations will result in the same relation as it was decomposed.

Example:

EMPLOYEE_DEPARTMENT table:

EMP_ID	EMP_NAME	EMP_AGE	EMP_CITY	DEPT_ID	DEPT_NAME
22	Denim	28	Mumbai	827	Sales
33	Alina	25	Delhi	438	Marketing
46	Stephan	30	Bangalore	869	Finance
52	Katherine	36	Mumbai	575	Production
60	Jack	40	Noida	678	Testing

The above relation is decomposed into two relations EMPLOYEE and DEPARTMENT

EMPLOYEE table:

EMP_ID	EMP_NAME	EMP_AGE	EMP_CITY
22	Denim	28	Mumbai
33	Alina	25	Delhi
46	Stephan	30	Bangalore
52	Katherine	36	Mumbai
60	Jack	40	Noida

DEPARTMENT table

DEPT_ID	EMP_ID	DEPT_NAME
827	22	Sales
438	33	Marketing
869	46	Finance
575	52	Production
678	60	Testing

Now, when these two relations are joined on the common column "EMP_ID", then the resultant

relation will look like:

Employee ⋈ Department

EMP_ID	EMP_NAME	EMP_AGE	EMP_CITY	DEPT_ID	DEPT_NAME
22	Denim	28	Mumbai	827	Sales
33	Alina	25	Delhi	438	Marketing
46	Stephan	30	Bangalore	869	Finance
52	Katherine	36	Mumbai	575	Production
60	Jack	40	Noida	678	Testing

2. Dependency Preservation Dependency is an important constraint on the database.

Every dependency must be satisfied by at least one decomposed table.

- This decomposition property can only be done by maintaining the functional dependency.
- In this property, it allows to check the updates without computing the natural join of the database structure.
- If a relation R is decomposed into relation R1 and R2, then the dependencies of R either must be a part of R1 or R2 or must be derivable from the combination of functional dependencies of R1 and R2.

For example, suppose there is a relation R (A, B, C, D) with functional dependency set

- (A->BC). The relational R is decomposed into R1 (ABC) and R2 (AD) which is dependency preserving because FD A->BC is a part of relation R1(ABC).

3. **Lack of Data Redundancy** Lack of Data Redundancy is also known as **a Repetition of Information.**

- The proper decomposition should not suffer from any data redundancy.
- The careless decomposition may cause a problem with the data.
- The lack of data redundancy property may be achieved by Normalization process.

Normalization: Normalization is a database design technique which organizes tables in a manner that reduces redundancy, maintains consistency and dependency of data.

It divides larger tables to smaller tables and links them using relationships.

- Normalization of data can be considered a process of analyzing the given relation schemas based on their FDs and primary keys to achieve the desirable properties of

(1) minimizing redundancy

(2) minimizing the insertion, deletion, and update anomalies

Anomalies in DBMS

What is an Anomaly: A Problem that can occur in poorly unplanned, un normalized data base were all the data is stored in a one table(a flat file database)

- There are three types of anomalies that occur when the database is not normalized. These are –1.Insert 2. update 3. Delete

1. **An insert anomaly** occurs when certain attributes cannot be inserted into the database without the presences of other attributes

2. An delete anomaly occurs when certain attributes are lost due to deletion of other attributes

3. An update anomaly exists when one or more instances of duplicated data is updated, but not all

Example: Suppose a manufacturing company stores the employee details in a table named employee that has four attributes: emp_id for storing employee's id, emp_name for storing employee's name, emp_address for storing employee's address and emp_dept for storing the department details in which the employee works. At some point of time the table looks like this:

emp_id	emp_name	emp_address	emp_dept
101	Rick	Delhi	D001
102	Rick	Delhi	D002
102	Maggie	Agra	D890

166	Glenn	Chennai	D900
167	Glenn	Chennai	D900

Update anomaly: In the above table we have two rows for employee Rick as he belongs to two departments of the company. If we want to update the address of Rick then we have to update the same in two rows or the data will become inconsistent. If somehow, the correct address gets updated in one department but not in other then as per the database, Rick would be having two different addresses, which is not correct and would lead to inconsistent data.

Insert anomaly: Suppose a new employee joins the company, who is under training and currently not assigned to any department then we would not be able to insert the data into the table if emp_dept field doesn't allow nulls.

Delete anomaly: Suppose, if at a point of time the company closes the department D890 then deleting the rows that are having emp_dept as D890 would also delete the information of employee Maggie since she is assigned only to this department.

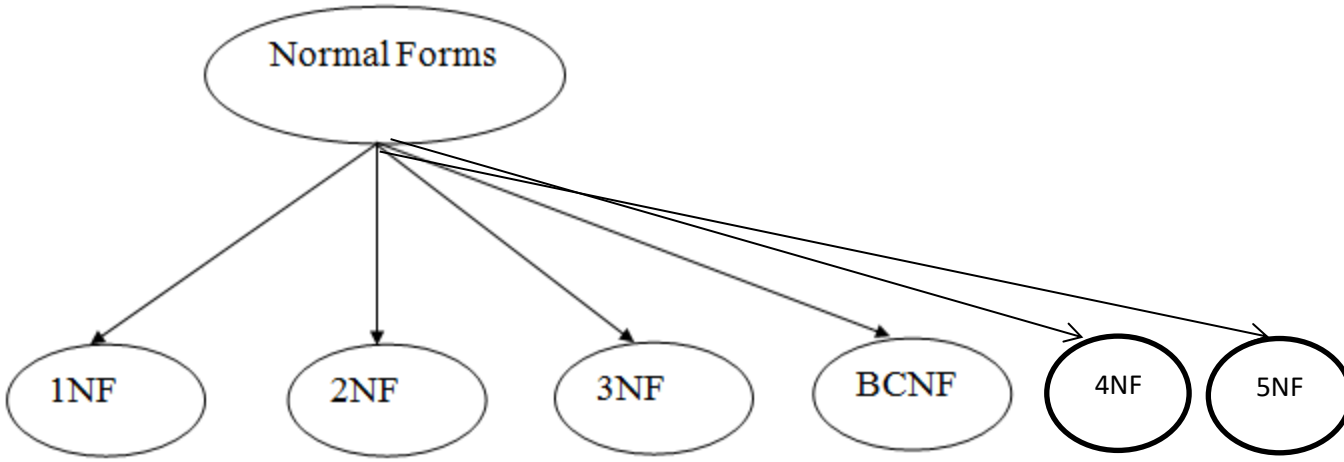
Update, deletion, and insertion anomalies are very undesirable in any database. Anomalies are avoided by the process of normalization.

Normalization

- Normalization is the process of organizing the data in the database.
- 1Normalization is used to minimize the redundancy from a relation or set of relations. It is also used to eliminate the undesirable characteristics like Insertion, Update and Deletion Anomalies.
- Normalization divides the larger table into the smaller table and links them using relationship.
- The normal form is used to reduce redundancy from the database table.

Types of Normal Forms

1.First Normal Form
2. Second Normal Form
3.Third Normal Form
4. Boyce Codd Normal Form
5. Fourth Normal form
5.Fifth Normal Form



Normal Form	Description
1NF	A relation is in 1NF if it contains an atomic value.
2NF	A relation will be in 2NF if it is in 1NF and all non-key attributes are fully functional dependent on the primary key.
3NF	A relation will be in 3NF if it is in 2NF and no transition dependency exists.
4NF	A relation will be in 4NF if it is in Boyce Codd normal form and has no multi-valued dependency.
5NF	A relation is in 5NF if it is in 4NF and not contains any join dependency and joining should be lossless.

First Normal Form (1NF)

A relation will be 1NF if it contains an atomic value.

- It states that an attribute of a table cannot hold multiple values. It must hold only single-valued attribute.
- First normal form disallows the multi-valued attribute, composite attribute, and their combinations.

Example: Relation EMPLOYEE is not in 1NF because of multi-valued attribute EMP_PHONE.

EMPLOYEE table:

EMP_ID	EMP_NAME	EMP_PHONE	EMP_STATE
14	John	7272826385, 9064738238	UP
20	Harry	8574783832	Bihar
12	Sam	7390372389, 8589830302	Punjab

The decomposition of the EMPLOYEE table into 1NF has been shown below:

EMP_ID	EMP_NAME	EMP_PHONE	EMP_STATE
14	John	7272826385	UP
14	John	9064738238	UP
20	Harry	8574783832	Bihar
12	Sam	7390372389	Punjab
12	Sam	8589830302	Punjab

This way, if we want to edit some information related to 7272826385, we do not have to touch the data corresponding to 9064738238. Also, observe that each row stores unique information. There is no repetition. This is the First Normal Form.

Second Normal Form (2NF)

Before we learn about the second normal form, we need to understand the following –

- Prime attribute – An attribute, which is a part of the candidate-key (primary key), is known as a prime attribute.
- Non-prime attribute – An attribute, which is not a part of the prime-key, is said to be a non-prime attribute.

The second normal form says that it should be in 1NF and every non-prime attribute should be fully functionally dependent on prime key attribute

Example: Suppose a school wants to store the data of teachers and the subjects they teach. They create a table that looks like this: Since a teacher can teach more than one subjects, the table can have multiple rows for a same teacher.

Tacher Id	Subject	Teacher Age
111	Maths	38
111	Physics	38
222	Biology	38
333	Physics	40
333	Chemistry	40

- **Candidate Keys**: {teacher_id, subject}

  Non prime attribute: teacher_age

- The table is in 1 NF because each attribute has atomic values. However, it is not in 2NF because non -key attribute teacher_age is dependent on teacher_id alone

which is a proper subset of candidate key. This violates the rule for 2NF as the rule says "non-key attributes are fully functional dependent on the primary key.".

- To make the table complies with 2NF we can break it in two tables like this:

teacher_details table:

teacher_id	teacher_age
111	38
222	38
333	40

teacher_subject table:

teacher_id	subject
111	Maths
111	Physics

222	Biology
333	Physics
333	Chemistry

Now the tables comply with Second normal form (2NF).

What are Keys?

- A DBMS key is an attribute or set of an attribute which helps you to identify a row(tuple) in a relation(table). They allow you to find the relation between two tables. Keys help you uniquely identify a row in a table by a combination of one or more columns in that table.

Why do we need keys?

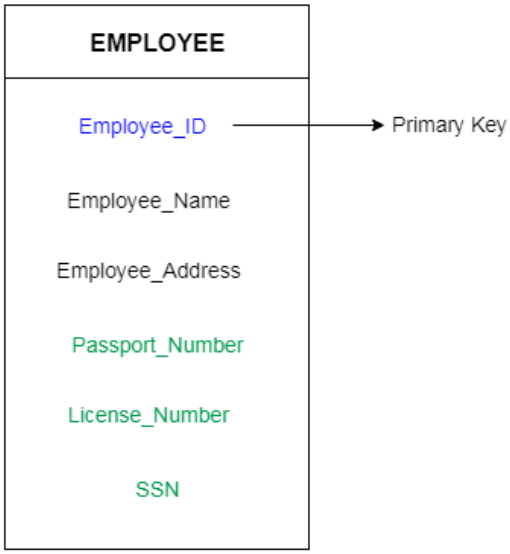
- Keys help you to identify any row of data in a table. In a real-world application, a table could contain thousands of records. Moreover, the records could be duplicated. Keys ensure that you can uniquely identify a table record despite these challenges.

- Allows you to establish a relationship between and identify the relation between tables
- Help you to enforce identity and integrity in the relationship.
- Various Keys in Database Management System

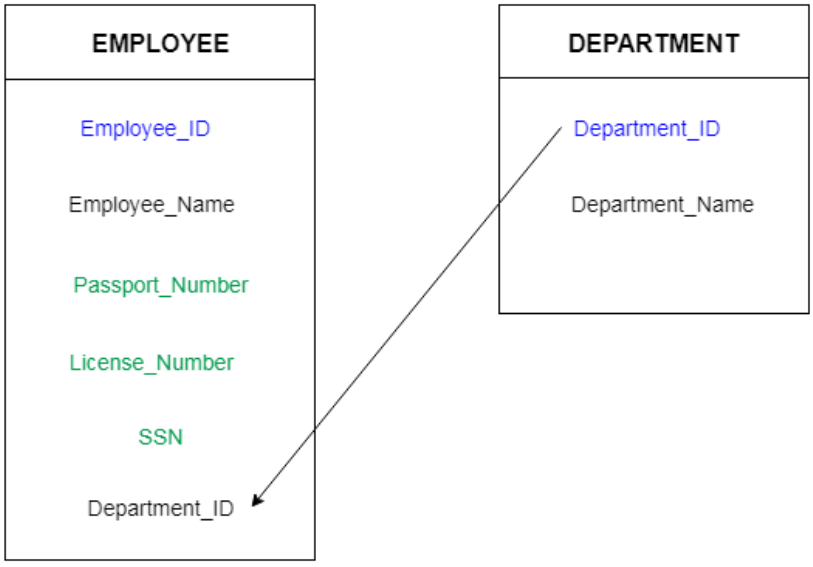
DBMS has following seven types of Keys each have their different functionality:

1. **Primary Key**
2. **Foreign Key**
3. **Super Key**
4. **Candidate Key**

1. **Primary Key:** A primary key is a field in a table which uniquely identifies each row/record in a database table. Primary keys must contain unique values. A primary key column cannot have NULL values.



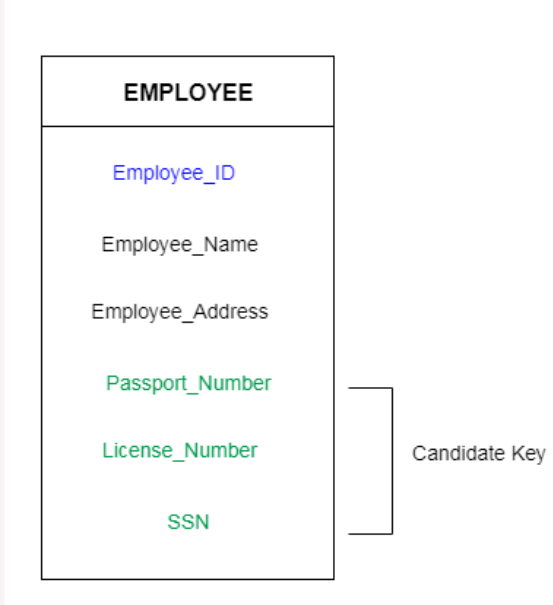
2. **A foreign key** is a key used to link two tables together. This is sometimes also called as a referencing key.
    - A Foreign Key is a column or a combination of columns whose values match a Primary Key in a different table.
    - The relationship between 2 tables matches the Primary Key in one of the tables with a Foreign Key in the second table.



3. Candidate key

- A candidate key is an attribute or set of an attribute which can uniquely identify a tuple.
- The remaining attributes except for primary key are considered as a candidate key. The candidate keys are as strong as the primary key.

For example: In the EMPLOYEE table, id is best suited for the primary key. Rest of the attributes like SSN, Passport_Number, and License_Number, etc. are considered as a candidate key.



4. **Super Key:** A super key is a group of single or multiple keys which identifies rows in a table. A Super key may have additional attributes that are not needed for unique identification.

   For example: In the above EMPLOYEE table, for(EMPLOEE_ID, EMPLOYEE_NAME) the name of two employees can be the same, but their EMPLYEE_ID can't be the same. Hence, this combination can also be a key.

   The super key would be EMPLOYEE-ID, (EMPLOYEE_ID, EMPLOYEE-NAME), etc.

Third Normal form (3NF)

A table design is said to be in 3NF if both the following conditions hold:

- Table must be in 2NF
- Transitive functional dependency of non-prime attribute on any super key should be removed.

An attribute that is not part of any candidate key is known as non-prime attribute.

An attribute that is a part of one of the candidate keys is known as prime attribute.

Example: Suppose a company wants to store the complete address of each employee, they create a table named employee_details that looks like this:

emp_id	emp_name	emp_zip	emp_state	emp_city	emp_district
1001	John	282005	UP	Agra	Dayal Bagh
1002	Ajeet	222008	TN	Chennai	M-City
1006	Lora	282007	TN	Chennai	Urrapakkam
1101	Lilly	292008	UK	Pauri	Bhagwan
1201	Steve	222999	MP	Gwalior	Ratan

Super keys: {emp_id}, {emp_id, emp_name}, {emp_id, emp_name, emp_zip}…so on

Candidate Keys: {emp_id}
Non-prime attributes: all attributes except emp_id are non-prime as they are not part of any candidate keys.

Here, emp_state, emp_city & emp_district dependent on emp_zip. And, emp_zip is dependent on emp_id that makes non-prime attributes (emp_state, emp_city & emp_district) transitively dependent on super key (emp_id). This violates the rule of 3NF.

To make this table complies with 3NF we have to break the table into two tables to remove the transitive dependency:

employee table:

emp_id	emp_name	emp_zip
1001	John	282005
1002	Ajeet	222008
1006	Lora	282007
1101	Lilly	292008
1201	Steve	222999

employee_zip table:

emp_zip	emp_state	emp_city	emp_district
282005	UP	Agra	Dayal Bagh

222008	TN	Chennai	M-City
282007	TN	Chennai	Urrapakkam
292008	UK	Pauri	Bhagwan
222999	MP	Gwalior	Ratan

Boyce Codd normal form (BCNF)

It is an advance version of 3NF that's why it is also referred as 3.5NF. BCNF is stricter than 3NF. A table complies with BCNF if it is in 3NF and for every functional dependency X->Y, X should be the super key of the table.

Example: Suppose there is a company where in employees work in **more than one department**. They store the data like this:

emp_id	emp_nationality	emp_dept	dept_type	dept_no_of_emp
1001	Austrian	Production and planning	D001	200
1001	Austrian	stores	D001	250
1002	American	design and technical	D134	100

		support		
1002	American	Purchasing department	D134	600

Functional dependencies in the table above:
emp_id -> emp_nationality
emp_dept -> {dept_type, dept_no_of_emp}

Candidate key: {emp_id, emp_dept}

The table is not in BCNF as neither emp_id nor emp_dept alone are keys.

To make the table comply with BCNF we can break the table in three tables like this:
emp_nationality table:

emp_id	emp_nationality
1001	Austrian
1002	American

emp_dept table:

emp_dept	dept_type	dept_no_of_emp
Production and planning	D001	200
stores	D001	250
design and technical support	D134	100
Purchasing department	D134	600

emp_dept_mapping table:

emp_id	emp_dept
1001	Production and planning
1001	stores

1002	design and technical support
1002	Purchasing department

Functional dependencies:
emp_id -> emp_nationality
emp_dept -> {dept_type, dept_no_of_emp}

Candidate keys:
For first table: emp_id
For second table: emp_dept
For third table: {emp_id, emp_dept}

This is now in BCNF as in both the functional dependencies left side part is a key

4th Normal Form

For a table to satisfy the Fourth Normal Form, it should satisfy the following two conditions:

1. It should be in the **Boyce-Codd Normal Form**.
2. And, the table should not have any **Multi-valued Dependency**.ie 2 independent multivalued attributes should not be dependent on same key attribute.

What is Multi-valued Dependency?

A table is said to have multi-valued dependency, if the following conditions are true,

1. For a dependency A → B, if for a single value of A, multiple value of B exists, then the table may have multi-valued dependency.

Example

Below we have a college enrolment table with columns s_id, course and hobby.

s_id	course	hobby
1	Science	Cricket
1	Maths	Hockey
2	C#	Cricket
2	Php	Hockey

As you can see in the table above, student with s_id **1** has opted for two courses, **Science** and **Maths**, and has two hobbies, **Cricket** and **Hockey**.

Well the two records for student with s_id **1**, will give rise to two more records, as shown below, because for one student, two hobbies exists, hence along with both the courses, these hobbies should be specified.

s_id	course	hobby
1	Science	Cricket
1	Maths	Hockey
1	Science	Hockey
1	Maths	Cricket

And, in the table above, there is no relationship between the columns course and hobby. They are independent of each other.

So there is multi-value dependency, which leads to un-necessary repetition of data and other anomalies as well.

How to satisfy 4th Normal Form?

To make the above relation satisfy the 4th normal form, we can decompose the table into 2 tables.

Course Opted Table

s_id	**course**
1	Science
1	Maths
2	C#
2	Php

And, **Hobbies Table**,

s_id	**hobby**
1	Cricket
1	Hockey
2	Cricket
2	Hockey

Now this relation satisfies the fourth normal form.

5th Normal Form

- A relation is in 5NF if it is in 4NF and not contains any **join dependency** and joining should be lossless (which ensures that no tuples are **generated** or **lost**, when relations are reunited through a natural join).

- 5NF is satisfied when all the tables are broken into as many tables as possible in order to avoid redundancy.
- 5NF is also known as Project-join normal form (PJ/NF).

Example

SUBJECT	LECTURER	SEMESTER
Computer	Anshika	Semester 1
Computer	John	Semester 1
Math	John	Semester 1
Math	Akash	Semester 2
Chemistry	Praveen	Semester 1

In the above table, John takes both Computer and Math class for Semester 1 but he doesn't take Math class for Semester 2. In this case, combination of all these fields required to identify a valid data.

Suppose we add a new Semester as Semester 3 but do not know about the subject and who will be taking that subject so we leave Lecturer and Subject as NULL. But all three columns together acts as a primary key, so we can't leave other two columns blank.

So to make the above table into 5NF, we can decompose it into three relations P1, P2 & P3:

P1

SEMESTER	SUBJECT
Semester 1	Computer
Semester 1	Math
Semester 1	Chemistry

Semester 2	Math

P2

SUBJECT	LECTURER
Computer	Anshika
Computer	John
Math	John
Math	Akash
Chemistry	Praveen

P3

SEMSTER	LECTURER
Semester 1	Anshika
Semester 1	John
Semester 1	John
Semester 2	Akash
Semester 1	Praveen

Example 2: Consider the above schema, with a case as "if a company makes a product and an agent is an agent for that company, then he always sells that product for the company". Under these circumstances, the ACP table is shown as:

Table – ACP

AGENT	COMPANY	PRODUCT
A1	PQR	Nut
A1	PQR	Bolt
A1	XYZ	Nut
A1	XYZ	Bolt
A2	PQR	Nut

The relation ACP is again decompose into 3 relations. Now, the natural Join of all the three relations will be shown as:

Table – R1

AGENT	COMPANY
A1	PQR
A1	XYZ
A2	PQR

Table – R2

AGENT	PRODUCT
A1	Nut
A1	Bolt
A2	Nut

Table – R3

COMPANY	PRODUCT
PQR	Nut
PQR	Bolt
XYZ	Nut
XYZ	Bolt

****