Syllabus: Schema Refinement (Normalization): Purpose of Normalization or schema refinement, concept of functional dependency, normal forms based on functional dependency(1NF, 2NF and 3 NF), concept of surrogate key, Boyce-codd normal form(BCNF), Lossless join and dependency preserving decomposition, Fourth normal form(4NF), Fifth Normal Form (5NF).

Normalization

Database Normalization is a technique of organizing the data in the database. Normalization is a systematic approach of decomposing tables to eliminate data redundancy (repetition) and undesirable characteristics like Insertion, Update and Deletion Anomalies. It is a multi-step process that puts data into tabular form, removing duplicated data from the relation tables.

Normalization is used for mainly two purposes,

- Eliminating redundant (useless) data.
- Ensuring data dependencies make sense i.e. data is logically stored.

Purpose of Normalization

Normalization is the process of structuring and handling the relationship between data to minimize redundancy in the relational table and avoid the unnecessary anomalies properties from the database like insertion, update and delete. It helps to divide large database tables into smaller tables and make a relationship between them. It can remove the redundant data and ease to add, manipulate or delete table fields.

A normalization defines rules for the relational table as to whether it satisfies the normal form. A **normal form** is a process that evaluates each relation against defined criteria and removes the multivalued, joins, functional and trivial dependency from a relation. If any data is updated, deleted or inserted, it does not cause any problem for database tables and help to improve the relational table integrity and efficiency.

Objective of Normalization

- It is used to remove the duplicate data and database anomalies from the relational table.

- Normalization helps to reduce redundancy and complexity by examining new data types used in the table.
- It is helpful to divide the large database table into smaller tables and link them using relationship.
- It avoids duplicate data or no repeating groups into a table.
- It reduces the chances for anomalies to occur in a database.

Types of Anomalies in DBMS

There are three types of anomalies that occur when the database is not normalized. These are – Insertion, update and deletion anomaly. Let's take an example to understand this.

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	Venkat	Delhi	D001
101	Venkat	Delhi	D002
123	Bhanu	Agra	D890
166	Srinu	Chennai	D900
166	Srinu	Chennai	D004

The above table is not normalized. We will see the problems that we face when a table is not normalized.

Update anomaly: In the above table we have two rows for employee **Venkat** as he belongs to two departments of the company. If we want to update the address of **Venkat** then we have to update the same in two rows or the data will become inconsistent. If the correct address gets updated in one department but not in other then as per the database, **Venkat** 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 **Bhanu** since he is assigned only to this department.

So, we need to avoid these types of anomalies from the tables and maintain the integrity, accuracy of the database table. Therefore, we use the normalization concept in the database management system.

Concept of Functional Dependency

Functional Dependency (FD) is a constraint that determines the relation of one attribute to another attribute in a Database Management System (DBMS). Functional Dependency helps to maintain the quality of data in the database. It plays a vital role to find the difference between good and bad database design.

A functional dependency is denoted by an arrow " → ". The functional dependency of **X** on **Y** is represented by **X → Y**. Let's understand Functional Dependency in DBMS with example.

Example:

Employee number	Employee Name	Salary	City
1	Dana	50000	San Francisco
2	Francis	38000	London
3	Andrew	25000	Tokyo

In this example, if we know the value of **Employee number**, we can obtain Employee Name, city, salary, etc. By this, we can say that the city, Employee Name, and salary are functionally depended on Employee number.

Rules of Functional Dependencies (Armstrong's Axioms):

If **F** is a set of functional dependencies then the closure of **F**, denoted as **F⁺**, is the set of all functional dependencies logically implied by **F**. Armstrong's Axioms are a set of rules, that when applied repeatedly, generates a closure of functional dependencies.

- **Reflexive rule:** If **X** is a set of attributes and **Y** is_subset_of **X**, then **X** holds a value of **Y**.
- **Augmentation rule:** When **x → y** holds, and **c** is attribute set, then **ac → bc** also holds. That is adding attributes which do not change the basic dependencies.
- **Transitivity rule:** This rule is very much similar to the transitive rule in algebra if **x → y** holds and **y → z** holds, then **x → z** also holds. **X → y** is called as functionally that determines **y**.

Types of Functional Dependencies in DBMS

There are mainly four types of Functional Dependency in DBMS. Following are the types of Functional Dependencies in DBMS:

- **Multivalued Dependency**
- **Trivial Functional Dependency**
- **Non-Trivial Functional Dependency**
- **Transitive Dependency**

Multivalued Dependency in DBMS

Multivalued dependency occurs in the situation where there are multiple independent multivalued attributes in a single table. A multivalued dependency is a complete constraint between two sets of attributes in a relation. It requires that certain tuples be present in a relation. Consider the following Multivalued Dependency Example to understand.

Example:

Car_model	Maf_year	Color
H001	2017	Metallic
H001	2017	Green
H005	2018	Metallic
H005	2018	Blue
H010	2015	Metallic
H033	2012	Gray

In this example, **maf_year** and **color** are independent of each other but dependent on **car_model**. In this example, these two columns are said to be multivalue dependent on **car_model**.

This dependence can be represented like this:

car_model → maf_year

car_model → colour

Trivial Functional Dependency in DBMS

The Trivial dependency is a set of attributes which are called a **trivial** if the set of attributes are included in that attribute.

So, **X → Y** is a trivial functional dependency if **Y** is a subset of **X**. Let's understand with a Trivial Functional Dependency Example.

For example:

Emp_id	Emp_name
AS555	Harry
AS811	George
AS999	Kevin

Consider this table of with two columns **Emp_id** and **Emp_name**.

{Emp_id, Emp_name} → Emp_id is a trivial functional dependency as **Emp_id** is a subset of {Emp_id,Emp_name}.

Non Trivial Functional Dependency in DBMS

Functional dependency which also known as a nontrivial dependency occurs when **A → B** holds **true** where **B** is not a subset of **A**. In a relationship, if attribute **B** is not a subset of attribute **A**, then it is considered as a non-trivial dependency.

Example:

Company	CEO	Age
Microsoft	Satya Nadella	51
Google	Sundar Pichai	46
Apple	Tim Cook	57

{Company} → {CEO} (If we know the Company, we knows the CEO name)

But CEO is not a subset of Company, and hence it's non-trivial functional dependency.

Transitive Dependency in DBMS

A Transitive Dependency is a type of functional dependency which happens when **t** is indirectly formed by two functional dependencies. Let's understand with the following Transitive Dependency Example.

Example:

Company	CEO	Age
Microsoft	Satya Nadella	51
Google	Sundar Pichai	46
Alibaba	Jack Ma	54

{Company} → {CEO} (If we know the company, we know its CEO's name)

{CEO} → {Age} If we know the CEO, we know the Age

Therefore according to the rule of transitive dependency:

{Company} → {Age} should hold, that makes sense because if we know the company name, we can know his age.

Note: You need to remember that transitive dependency can only occur in a relation of three or more attributes.

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

Normal Forms

Normal forms are used to eliminate or reduce redundancy in database tables. Here are the most commonly used normal forms:

- **First normal form(1NF)**
- **Second normal form(2NF)**
- **Third normal form(3NF)**
- **Boyce & Codd normal form (BCNF)**

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

Second normal form (2NF)

- In the **2NF**, relational must be in **1NF**.
- In the second normal form, all non-key attributes are fully functional dependent on the primary key.

Example: Let's assume, a school can store the data of teachers and the subjects they teach. In a school, a teacher can teach more than one subject.

TEACHER table

TEACHER_ID	SUBJECT	TEACHER_AGE
25	Chemistry	30
25	Biology	30
47	English	35
83	Math	38
83	Computer	38

In the given table, non-prime attribute **TEACHER_AGE** is dependent on **TEACHER_ID** which is a proper subset of a candidate key. That's why it violates the rule for 2NF.

To convert the given table into **2NF**, we decompose it into two tables:

TEACHER_DETAIL table:

TEACHER_ID	TEACHER_AGE
25	30
47	35
83	38

TEACHER_SUBJECT table:

TEACHER_ID	SUBJECT
25	Chemistry
25	Biology
47	English
83	Math
83	Computer

Third Normal Form (3NF)

- A relation will be in **3NF** if it is in **2NF** and not contain any transitive partial dependency.
- **3NF** is used to reduce the data duplication. It is also used to achieve the data integrity.
- If there is no transitive dependency for non-prime attributes, then the relation must be in third normal form.

A relation is in third normal form if it holds atleast one of the following conditions for every non-trivial function dependency **X →Y**.

1. **X** is a super key.
2. **Y** is a prime attribute, i.e., each element of **Y** is part of some candidate key.

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}…soon
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 wherein 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 support	D134	100
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.

Surrogate key

A surrogate key in DBMS is the key or can say a unique identifier that uniquely identifies an object or an entity in their respective fields. A surrogate key is used for representing existence for data analysis. It is the unique identifier in a database. It represents an outside entity as a database object but is not visible to the user and application. A surrogate key is also known by various other names, which are pseudo key, technical key, synthetic key, arbitrary unique identifier, entity identifier and database sequence number.

Implementing Surrogate Key

Let's implement an example to understand the working and role of a Surrogate key in DBMS:

Consider an example of **Tracking System**, where we have the following attributes:

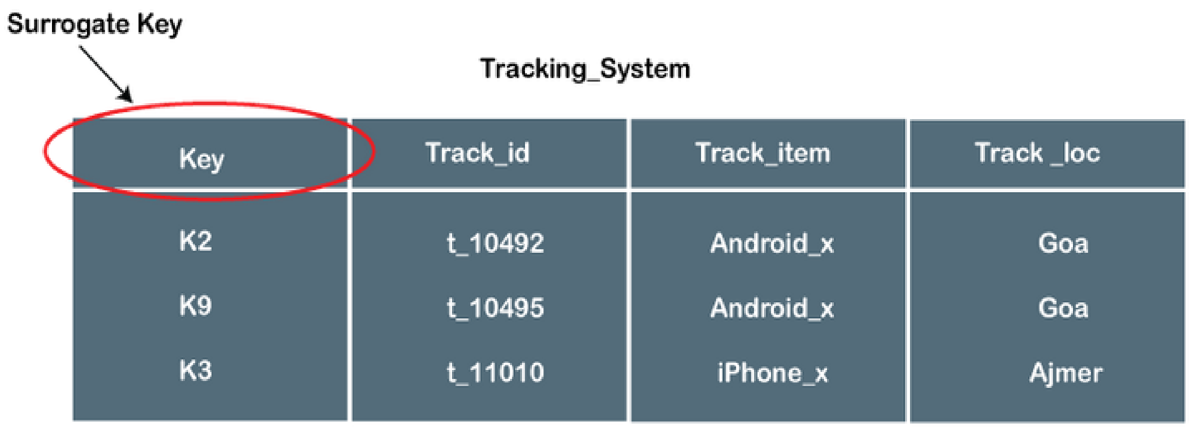
Key: An attribute holding the key for each tracking id.

Track_id: An attribute holding the tracking id of the item.

Track_item: An attribute holding the name of the item that is being tracked.

Track_loc: An attribute holding the location of the tracking item.

The below diagram represents the above-described **Tracking_system** table:


Surrogate Key
Tracking_System

Key	Track_id	Track_item	Track _loc
K2	t_10492	Android_x	Goa
K9	t_10495	Android_x	Goa
K3	t_11010	iPhone_x	Ajmer

From the above table, we can see that the Key attribute of the **Tracking_System** table is the Surrogate key because the value of the Key column is different for different locations and id of the item.

Fourth normal form(4NF)

- A relation will be in 4NF if it is in Boyce Codd normal form and has no multi-valued dependency.
- For a dependency **A → B**, if for a single value of **A**, multiple values of **B** exist, then the relation will be a multi-valued dependency.

What is Multi-valued Dependency?

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

- 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.
- Also, a table should have at-least **3** columns for it to have a multi-valued dependency.

- And, for a relation **R(A,B,C),** if there is a multi-valued dependency between, **A** and **B**, then **B** and **C** should be independent of each other.

If all these conditions are true for any relation(table), it is said to have multi-valued dependency.

Example

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

STU_ID	COURSE	HOBBY
21	Computer	Dancing
21	Math	Singing
34	Chemistry	Dancing
74	Biology	Cricket
59	Physics	Hockey

The given STUDENT table is in 3NF, but the COURSE and HOBBY are two independent entities. Hence, there is no relationship between COURSE and HOBBY.

In the STUDENT relation, a student with STU_ID, **21** contains two courses, **Computer** and **Math** and two hobbies, **Dancing** and **Singing**. So, there is a multi-valued dependency on STU_ID, which leads to unnecessary repetition of data.

So, to make the above table into 4NF, we can decompose it into **two** tables:

STUDENT_COURSE

STU_ID	COURSE
21	Computer
21	Math
34	Chemistry
74	Biology
59	Physics

STUDENT_HOBBY

STU_ID	HOBBY
21	Dancing
21	Singing
34	Dancing
74	Cricket
59	Hockey

Fifth Normal Form (5NF)

- A relation is in 5NF if it is in 4NF and not contains any join dependency and joining should be lossless.
- 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 act 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