Unit – IV

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).

PURPOSE OF NORMALIZATION OR SCHEMA REFINEMENT

Database Normalization is a technique of organizing the data in the database. Normalization is a systematic approach of decomposing tables to eliminate data redundancy and undesirable characteristics like Insertion, Update and Deletion Anomalies. It is a multi-step process that puts data into tabular form by removing duplicated data from the relation tables. If a database design is not perfect, it may contain anomalies, which are like a bad dream for any database administrator. Managing a database with anomalies is next to impossible.

Normalization is used for mainly two purpose,

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

Problem Without Normalization

Without Normalization, it becomes difficult to handle and update the database, without facing data loss. Insertion, Updation and Deletion Anamolies are very frequent if Database is not Normalized. To understand these anomalies let us take an example of **Student** table.

S_id	S_name	S_Address	Subject_opted
401	Kesava	Noida	JAVA
402	Rama	Panipat	DBMS
403	Krishna	Jammu	DBMS
404	Kesava	Noida	Data Mining

- **Updation Anamoly :** If data items are scattered and are not linked to each other properly, then it could lead to strange situations. For example, when we try to update one data item having its copies scattered over several places, a few instances get updated properly while a few others are left with old values. Such instances leave the database in an inconsistent state.

  Example: To update address of a student who occurs twice or more than twice in a table, we will have to update **S_Address** column in all the rows, else data will become inconsistent.

- **Insertion Anamoly :** We tried to insert data in a record that does not exist at all.

  Example: Suppose for a new admission, we have a Student id(S_id), name and address of a student but if student has not opted for any subjects yet then we have to insert **NULL** there, leading to Insertion Anamoly.

- **Deletion Anamoly :** We tried to delete a record, but parts of it was left undeleted because of unawareness, the data is also saved somewhere else.

  Example: If (S_id) 402 has only one subject and temporarily he drops it, when we delete that row, entire student record will be deleted along with it.

CONCEPT OF FUNCTIONAL DEPENDENCY

Functional dependency (FD) is a set of constraints between two attributes in a relation. Functional dependency says that if two tuples have same values for attributes A1, A2,..., An, then those two tuples must have to have same values for attributes B1, B2, ..., Bn.

Functional dependency is represented by an arrow sign (→) that is, X→Y, where X functionally determines Y. The left-hand side attributes determine the values of attributes on the right-hand side.

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 alpha is a set of attributes and beta is_subset_of alpha, then alpha holds beta.
- **Augmentation rule** − If a → b holds and y is attribute set, then ay → by also holds. That is adding attributes in dependencies, does not change the basic dependencies.
- **Transitivity rule** − Same as transitive rule in algebra, if a → b holds and b → c holds, then a → c also holds. a → b is called as a functionally that determines b.

TRIVIAL FUNCTIONAL DEPENDENCY

- **Trivial** − If a functional dependency (FD) X → Y holds, where Y is a subset of X, then it is called a trivial FD. Trivial FDs always hold.
- **Non-trivial** − If an FD X → Y holds, where Y is not a subset of X, then it is called a non-trivial FD.
- **Completely non-trivial** − If an FD X → Y holds, where x intersect Y = Φ, it is said to be a completely non-trivial FD.

PROPERTIES OF FUNCTIONAL DEPENDENCIES:

1. Reflexive: If $Y \subseteq X$ then X → Y is a Reflexive Functional Dependency.

Ex: AB→A , $A \subseteq AB$ holds. Therefore AB→A is a Reflexive Functional Dependency.

2. Augmentation: If X→Y is a functional dependency then by augmentation, XZ→YZ is also a functional dependency.

3.Transitivity: IF X→Y and Y→Z are two functional dependencies then by transitivity, X→Z is also a functional dependency.

4. Union: If X→Y and X→Z are two functional dependencies then, X→YZ is also a functional dependency.

5. Decomposition: If X→YZ is a functional dependency then X→Y and X→Z are also functional dependencies.

CLOSURE SET OF A FUNCTIONAL DEPENDENCY (F^+)

It is a set of all functional dependencies that can be determined using the given set of dependencies. It is denoted by F^+.

Attribute Closure (X^+): It is a set of all the attributes that can be determined using X. It is denoted by X^+, where X is any set of attributes.

Example:

R(A,B,C) F:{A→B , B→C}

A^+={A,B,C} B^+={B,C} C^+={C}

AB^+={A,B,C} AC^+={A,C,B} BC^+={B,C} ABC^+={A,B,C}

Identifying keys in the given relation based on Functional Dependencies associated with it

X^+ is a set of attributes that can be determined using the given set X of attributes.

- If X^+ contains all the attributes of a relation, then X is called "**Super key**" of that relation.
- If X^+ is minimal set, then X is called "**Candidate Key**" of that relation.

If no closure contains all the elements then in such a case we can find independent attributes of that relation i.e., the attributes that which are not in the R.H.S. of any dependency. If the closure of the Independent attributes contains all the elements then it can be treated as a candidate key.

If the closure of independent attributes also doesn't contain all the elements then we try to find the key by adding dependent attributes one by one. If we couldn't find key then we can add groups of dependent attributes till we find a key to that relation.

FIRST NORMAL FORM (1NF)

1NF is designed to disallow multi valued attributes, composite attributes and their combinations. Means 1NF allows only atomic values i.e., the attribute of any tuple must be either 1 or NULL value.

A relation having multi valued and composite attributes is known as Un Normalized Relation. Removal of These multi valued and composite attributes will turn the UN Normalized Relation to 1NF Relation.

Example:

Professor		
ID	**Name**	**Salary**
1	Rama	{40000,10000,15000,10000}

Un Normalized Relation Since **salary** is a **Multi valued** attribute

We can eliminate this multi valued attribute by splitting the salary column to more specific columns like Basic, TA, DA, HRA. The Above relation in 1NF is as follows.

Professor					
ID	**Name**	**Basic**	**TA**	**DA**	**HRA**
1	Rama	40000	10000	15000	10000

1NF

Every Relation in the Relation Database must be in 1NF.

SECOND NORMAL FORM (2NF):

It depends on the concept of Full Functional Dependences(FFD) and disallows Partial Functional Dependencies(PFD) i.e., for a relation that has concatenated primary key, each attribute in the table that is not part of the primary key must depend upon the entire concatenated key for its existence. If any attribute depends only on one part of the concatenated key, then the relation fails **Second normal form**.

A relation with Partial Functional dependencies can be made as in 2NF by removing them.

PFD: part of key → Non Key

FFD: Key → Non key

Example:

Medication			
Patient No.	**Drug**	**No. of units**	**Pname**
P1	D1	10	Kiran
P1	D2	20	Kiran
P1	D3	15	Kiran
P2	D4	15	Raj

Dependencies

Patient No. → Pname

Patient No.,Drug → No. of units

The above relation is in 1NF but not in 2NF. Key for the relation is **Patient No.** and **Drug.** After eliminating the Partial functional dependencies, the decomposed relations are:

Patient	
Patient No.	**Pname**
P1	Kiran
P2	Raj

Parent Relation

Key: Patient No.

Dependencies

Patient No. → Pname

Medication		
Patient No.	**Drug**	**No. of units**
P1	D1	10
P1	D2	20
P1	D3	15
P2	D4	15

Child Relation

Key: Patient No. and Drug

Patient No. is **Foreign key** referring Patient No. in the Parent Relation (Patient)

Dependencies

Patient No.,Drug → No. of units

The above 2 relations satisfy 2NF. They don't have partial functional dependencies.

Note: If key is only one attribute then the relation is always in 2NF.

THIRD NORMAL FORM (3NF):

Third Normal form applies that every non-prime attribute of a relation must be dependent on primary key, or we can say that, there should not be the case that a non-prime attribute is determined by another non-prime attribute. So this *transitive functional dependency* should be removed from the relation and also the relation must be in **Second Normal form.** Simply, It is based on the concept of transitive dependencies. 3NF disallows the transitive dependencies.

Dependency: Key → Non Key

Transitive Dependency: Non Key → Non Key

A relation satisfying 2NF and with Transitive Functional dependencies can be made as in 3NF by removing the transitive functional dependencies.

Example:

Contains			
Patient No.	**Pname**	**Ward No.**	**Ward Name**
P1	Kumar	W1	ICU
P2	Kiran	W1	ICU
P3	Kamal	W1	ICU
P4	Sharath	W2	General

Key

Patient No. and Pname

Dependencies

Patient No.,Pname → Ward No.

Ward No. → Ward Name (Transitive Dependency)

Ward No. → Ward Name is transitive because Ward No. is not a key. To make this relation satisfy 3NF transitive dependency must be removed from it. It can be done by decomposing the relation. The decomposed relations are as follows:

Ward	
Ward No.	**Ward Name**
W1	ICU
W2	General

Key

Ward No.

Dependencies

Ward No. → Ward Name

Contains		
Patient No.	**Pname**	**Ward No.**
P1	Kumar	W1
P2	Kiran	W1
P3	Kamal	W1
P4	Sharath	W2

Primary Key

Patient No. and Pname

Foreign Key

Word No. refers Ward No. in relation Ward

Dependencies

Patient No.,Pname → Ward No.

The above two relations satisfy 3NF. They don't have transitive dependencies.

Note 1: A relation R is said to be in 3NF if whenever a non trivial functional dependency of the form X → A holds then either X is a super key or A is a prime attribute.

Note 2: If all attributes are prime attributes then the relation is in 3NFbecause with such attributes no partial functional dependencies and transitive dependencies exists.

BOYCE - CODD NORMAL FORM(BCNF):

BCNF is a higher version of the Third Normal form. This form deals with certain type of anamoly that is not handled by 3NF. A 3NF relation which does not have multiple overlapping candidate keys is said to be in BCNF. For a relation to be in BCNF, following conditions must be satisfied:

- R must be in 3rd Normal Form
- and, for each functional dependency (X -> Y), X should be a super Key.

(OR)

A relationship is said to be in BCNF if it is already in 3NF and the left hand side of every dependency is a candidate key. A relation which is in 3NF is almost always in BCNF. These could be same situation when a 3NF relation may not be in BCNF the following conditions are found true.

1. The candidate keys are composite.
2. There are more than one candidate keys in the relation.
3. There are some common attributes in the relation.

Professor Code	Department	Head of Dept.	Percent Time
P1	Physics	Ghosh	50
P1	Mathematics	Krishnan	50
P2	Chemistry	Rao	25
P2	Physics	Ghosh	75
P3	Mathematics	Krishnan	100

Consider, as an example, the above relation. It is assumed that:

1. A professor can work in more than one department
2. The percentage of the time he spends in each department is given.
3. Each department has only one Head of Department.

Dependencies of the above relation are:

Department,Professor Code → Head of the Depatrment

Department,Professor Code → Percent time

Department → Head of the Depatrment

Head of the Department,Professor Code → Depatrment

Head of the Department,Professor Code → Percent time

The given relation is in 3NF. Observe, however, that the names of Dept. and Head of Dept. are duplicated. Further, if Professor P2 resigns, rows 3 and 4 are deleted. We lose the information that Rao is the Head of Department of Chemistry.

The normalization of the relation is done by creating a new relation for Dept. and Head of Dept. and deleting Head of Dept. form the given relation. The normalized relations are shown in the following.

Professor_work		
Professor Code	**Department**	**Percent Time**
P1	Physics	50
P1	Mathematics	50
P2	Chemistry	25
P2	Physics	75
P3	Mathematics	100

Dependencies

Department,Professor Code → Percent time

Department is foreign key referring **Department** in the **Deartment_Details** relation

Depatrment_Details	
Department	**Head of Dept.**
Physics	Ghosh
Mathematics	Krishnan
Chemistry	Rao

Dependencies

Department → Head of the Department

FOURTH NORMAL FORM (4NF)

When attributes in a relation have multi-valued dependency, further Normalization to 4NF and 5NF are required. A **multi-valued dependency** is a typical kind of dependency in which each and every attribute within a relation depends upon the other, yet none of them is a unique primary key. Consider a vendor supplying many items to many projects in an organization. The following are the assumptions:

1. A vendor is capable of supplying many items.
2. A project uses many items.
3. A vendor supplies to many projects.
4. An item may be supplied by many vendors.

A multi valued dependency exists here because all the attributes depend upon the other and yet none of them is a primary key having unique value.

Vendor Code	Item Code	Project No.
V1	I1	P1
V1	I2	P1
V1	I1	P3
V1	I2	P3
V2	I2	P1
V2	I3	P1
V3	I1	P2
V3	I1	P3

The given relation has a number of problems. For example:

1. If vendor V1 has to supply to project P2, but the item is not yet decided, then a row with a blank for item code has to be introduced.

2. The information about item I1 is stored twice for vendor V3.

Observe that the relation given is in 3NF and also in BCNF. It still has the problem mentioned above. The problem is reduced by expressing this relation as two relations in the Fourth Normal Form (4NF). A relation is in 4NF if it has no more than one independent multi valued dependency or one independent multi valued dependency with a functional dependency.

The table can be expressed as the two 4NF relations given as following. The fact that vendors are capable of supplying certain items and that they are assigned to supply for some projects in independently specified in the 4NF relation.

Vendor_Supply	
Vendor Code	**Item Code**
V1	I1
V1	I2
V2	I2
V2	I3
V3	I1

.

Vendor_Project	
Vendor Code	**Project No.**
V1	P1
V1	P3
V2	P1
V3	P2

SUMMARY

Input Relation	Transformation	Output Relation
All Relations	Eliminate variable length record. Remove multi-attribute lines in table.	1NF
1NF	Remove dependency of non-key attributes on part of a multi-attribute key.	2NF
2NF	Remove dependency of non-key attributes on other non-key attributes.	3NF
3NF	Remove dependency of an attribute of a multi attribute key on an attribute of another (overlapping) multi-attribute key.	BCNF
BCNF	Remove more than one independent multi-valued dependency from relation by splitting relation.	4NF
4NF	Add one relation relating attributes with multi-valued dependency.	5NF

PROPERTIES OF DECOMPOSITION:

Every Decomposition must satisfy 2 properties.

1. Lossless join
2. Dependency Preserving

1. Lossless join:

If we decompose a relation R into relations R1 and R2,

- Decomposition is lossy if $R1 \bowtie R2 \supset R$
- Decomposition is lossless if $R1 \bowtie R2 = R$

To check for lossless join decomposition using FD set, following conditions must hold:

1. Union of Attributes of R1 and R2 must be equal to attribute of R. Each attribute of R must be either in R1 or in R2. Att(R1) U Att(R2)=Att(R)

2. Intersection of Attributes of R1 and R2 must not be NULL. $Att(R1) \cap Att(R2) \neq \Phi$
3. Common attribute must be a key for at least one relation (R1 or R2) $Att(R1) \cap Att(R2) \rightarrow Att(R1)$ or $Att(R1) \cap Att(R2) \rightarrow Att(R2)$

For Example, A relation R (A, B, C, D) with FD set {A->BC} is decomposed into R1(ABC) and R2(AD) which is a lossless join decomposition as:

1. First condition holds true as Att(R1) U Att(R2) = (ABC) U (AD) = (ABCD) = Att(R).
2. Second condition holds true as $Att(R1) \cap Att(R2) = (ABC) \cap (AD) \neq \Phi$
3. Third condition holds true as $Att(R1) \cap Att(R2) = A$ is a key of R1(ABC) because A->BC is given.

Dependency Preserving Decomposition

If we decompose a relation R into relations R1 and R2, All dependencies of R either must be a part of R1 or R2 or must be derivable from combination of FD's of R1 and R2.

For Example, A relation R (A, B, C, D) with FD set{A->BC} is decomposed into R1(ABC) and R2(AD) which is dependency preserving because FD A->BC is a part of R1(ABC).

Decomposition of a relation is done when a relation in relational model is not in appropriate normal form. Relation R is decomposed into two or more relations if decomposition is lossless join as well as dependency preserving.

CONCEPT OF SURROGATE KEY:

In database design, it is a good practice to have a primary key for each table. There are two ways to specify a primary key: The first is to use part of the data as the primary key. For example, a table that includes information on employees may use Social Security Number as the primary key. This type of key is called a natural key. The second is to use a new field with artificially-generated values whose sole purpose is to be used as a primary key. This is called a surrogate key.

A surrogate key has the following characteristics:

1) It is typically an integer.

2) It has no meaning. You will not be able to know the meaning of that row of data based on the surrogate key value.

3) It is not visible to end users. End users should not see a surrogate key in a report.

Surrogate keys can be generated in a variety of ways, and most databases offer ways to generate surrogate keys. For ex, Oracle uses SEQUENCE , MySQL uses AUTO_INCREMENT , and SQL Server uses IDENTITY.

Surrogate keys are often used in data warehousing systems, as the high data volume in a data warehouse means that optimizing query speed becomes important. Using a surrogate key is advantageous because it is quicker to join on a numeric field rather than a non-numeric field. This does come at a price — when you insert data into a table, whether via an ETL Process or via an "IINSERT INTO" statement, the system needs to take more resources to generate the surrogate key.

There are no hard rules on when to employ a surrogate key as opposed to using the natural key. Often the data architect would need to look at the nature of the data being modeled and stored and consider any possible performance implications.