Chapter 5

Database Constraints and Relational Database Design

Overview:

Integrity Constraints, Referential Integrity, Assertion & Triggers, Closure set of attributes & FDs, Database Anomaly, Functional Dependency, Normalization (1NF, 2NF, 3NF, BCNF, 4NF, 5NF, 6NF) and Join dependency.

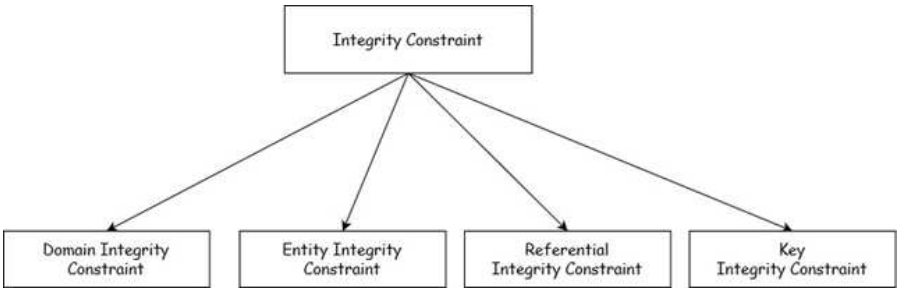
INTEGRITY CONSTRAINTS

- Constraints restricts the values the table can store.
- Integrity Constraints allows us to define certain data requirements that the data in the database need to meet.
- If the user tries to insert the data that do not meet the requirements integrity constraints will not allow this.
- Integrity constraints ensure that the data insertion, updating, and other processes have to be performed in such a way that data integrity is not affected.
- It ensures accuracy & consistency in relational databases.

- ### Integrity Constraints in Create Table

    1. **Not Null**: It enforces a column to NOT accept NULL values.
    2. **Unique Key**: It ensures that all values in a column are different.
    3. **Check**: To limit the value range that can be placed in a column.
    4. **Primary Key**: A primary key is a column or a set of columns in a table whose values uniquely identify a row in the table.
    5. **Foreign Key** : Columns defined as foreign keys refer to the primary key of other tables.
    6. **Default**: The Default constraints is used to insert a default value into the column.

Types of Integrity Constraints



1. Domain Integrity Constraint
2. Entity Integrity Constraint
3. Referential Integrity Constraint
4. Key Integrity Constraint

1. Domain Integrity Constraint

- Enforcing domain integrity simply means that a column accepts only valid values, as dictated by the application's business rules.
- Domain Integrity means the definition of a valid set of values for an attribute. We define
    - ➔ Data type
    - ➔ Length or size
    - ➔ Is null value allowed
    - ➔ Is the value unique or not for an attribute
- Example: cannot set an integer variable to ‘abc’.

2. Entity Integrity Constraint

- Its rule applies to Primary Keys.
- The entity integrity constraint states that primary keys can’t be null.

3. Referential Integrity Constraint

- The referential integrity constraints apply to Foreign Keys.
- The referential integrity constraint is specified between two tables & it is used to maintain the consistency among rows between the two tables.
- The rules are:

1. You can't delete a record from the primary table if matching records exist in the related other table.
2. You can't change the primary key value in the primary table if that record has related records.
3. You can't enter a value in the foreign field of the related table that doesn't exist in the primary key of the primary table.
4. However, you can enter a NULL value in the foreign key, specifying that the records are unrelated.

Table I. Student_name

Roll_no	Name
1	John
2	Von
3	Neumann

Table II. Student_address

Roll_no	Address
1	Patan
2	Satdobato
3	Kalanki

Roll_no in **Table II** is foreign key in **Table I**

- **Cascading** : When referential integrity is violated during modification, instead of just rejecting the modification, you can cascade:
  - ➔ Delete Cascade : In a delete cascade, anything that has references to the deleted item is also deleted.
  - ➔ Update Cascade: In an update cascade, when the updated item results in a violation of referential integrity, the system will update accordingly to fix the problem.

4. Key Integrity Constraint

- Keys are the entity set that is used to identify an entity within its entity set uniquely.
- An entity set can have multiple keys, but out of which one key will be the primary key. Primary key should be unique.

ASSERTIONS

- An assertion is a predicate expressing a condition that we wish the database always to satisfy.
- An assertion in SQL takes the form

  create assertion <assertion_name> check <predicate condition>
- In case of assertion the system tests it for validity & tests it again on every update that may violate the assertion.
- This testing may introduce a significant amount of overhead; hence assertions should be used with great care.
- DBMS checks the assertion after any change that may violate the expression.
- Assertions are like column & table constraints except that are specified separately from the table definitions.
- **Example**: The sum of all loan amounts for each branch must be less than the sum of all account balances at that branch.

  create assertion sum_constraint check (not exists
  (select * from branch where (select sum(amount) from loan
  where loan.branch_name= branch.branch_name) >=
  (select sum(account) from account where
  amount.branch_name = branch.branch_name))
- **Example:** In sales (rest, soda, price) no rest may charge an average more than $3.

  create assertion example check (not exists
  (select rest from sales group by rest having avg(price)>3));

TRIGGERS

- A trigger is a statement that is executed automatically by the system as a side effect of a modification to the database. Or,
- It is a stored procedure which automatically invokes whenever a certain event in the database occurs.
- Most DBMS just include triggers not assertions.

- To design a trigger mechanism we must:
  - ➔ Specify the conditions under which the trigger is to be executed.
  - ➔ Specify the actions to be taken when the trigger executes.
- Trigger components: Three components
  1. Event: When the event happens, the trigger is activated.
  2. Condition (Optional): If the condition is true, the trigger executes, otherwise skipped.
  3. Action: The actions performed by the trigger.

Trigger: Events

Events in Trigger is defined as

Syntax: create trigger ***trigger_name***
before | after insert | update | delete
on ***table_name***
for each row
Trigger_body

trigger_name : name of the trigger

before | after : Specifies when trigger executed is executed.

insert | update | delete : Operations to be performed

table_name : relation named

for each row : specifies trigger for each row which is executed.

trigger_body: operation to be performed as trigger is fired

- **Example:** Create Trigger XYZ
  After Update on Students
  For each row
  set Students.grade= grade +1;

Compare & contrast between Assertion & Trigger

- ➔ Assertions do not modify the data, they check certain conditions.
- ➔ Triggers are more powerful because they can check conditions & also modify the data.

- ➔ Assertions are not linked to the specific tables in the database & not linked to specific events.
- ➔ Triggers are linked to specific tables & specific events.
- ➔ All assertions can be implemented as triggers (one or more).
- ➔ Not all triggers can be implemented as assertions.
- ➔ Oracle does not have assertions.

NORMALIZATION

- Database Normalization is the process of removing redundant data from your tables in order to improve storage efficiency, data integrity & scalability.
- In other words, it is the process of restructuring the logical data model of database which helps
  I. To eliminate redundancy & reduce repeating group
  II. To organize data efficiently
  III. To reduce the potential for anomalies during data operations
- The formal classifications used for describing a relational database level of normalization are called normal forms.
- Normal forms are abbreviated as NF.
- **What happens without normalization ?**
  - ➔ A non normalized database suffers from data anomalies
  - ➔ A non normalized database may store data representing a particular referent in multiple locations.
- **Advantages of Normalization**
  1. A smaller database can be maintained as normalization ***eliminates the duplicate data***. Overall size of the database is reduced as a result.
  2. Better performance is ensured which can be linked to the above point. As databases become lesser in size, the passes through the data becomes faster and shorter thereby ***improving response time and speed***.
  3. Narrower tables are possible as normalized tables will be fine-tuned and will have lesser columns which allows for more data records per page.

4. Have the options of ***joining (Joins)*** only when the table is needed.

- **Disadvantages of Normalization**
    1. More tables to join as by spreading out data into more tables, the need to ***join table's increases and the task becomes more tedious***. The database becomes harder to realize as well.
    2. ***Data model becomes extremely difficult*** to query against as the data model is optimized for applications, not for ad hoc querying. (Ad hoc query is a query that cannot be determined before the issuance of the query. It consists of an SQL that is constructed dynamically and is usually constructed by desktop friendly query tools.). Hence it is hard to model the database without knowing what the customer desires.
    3. Proper knowledge is required on the various normal forms to execute the normalization process efficiently. Careless use may lead to terrible design filled with ***major anomalies and data inconsistency***.

DATABASE ANOMALY

- Anomalies are problems that can occur in poorly planned un normalized databases where all the data gets stored in one table
- Anomalies in DBMS
    1. Update Anomaly
    2. Insert Anomaly
    3. Delete Anomaly

Emp_id	Emp_name	Emp_address	Emp_dept
101	John	Ktm	D001
101	John	Ktm	D002
123	Von	Bkt	D890
166	Neumann	Llt	D900
166	Neumann	Llt	D004

Table name: Employee

1. **Update Anomaly**
    - ➔ An update to some data in some but not all of those locations results in an update anomaly, yielding inconsistent data.
    - ➔ *Example*: If we want to update the address of John from table Employee, we have to update the address of two rows or the data will become inconsistent.

2. **Insert Anomaly**
    - ➔ An Insert anomaly occurs when certain attributes cannot be inserted into the database without the presence of other attributes.
    - ➔ *Example: We cannot add a new employee if he/she is in the training phase & doesn't have an Emp_dept yet. It will not allow inserting the new data if Emp_dept field doesn't allow nulls.*

3. **Delete Anomaly**
    - ➔ A delete anomaly exists when certain attributes are lost because of the deletion of other attributes.
    - ➔ *Example: 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 Von since he is assigned only to this department.*

FUNCTIONAL DEPENDENCY

- Dependency: It describes the relationship between two or more attributes
- **Functional Dependency**
    - ➔ It is a constraint that determines the relation of one attribute to another attribute in a DBMS.
    - ➔ It helps to maintain the quality of data in the database.
    - ➔ A functional dependency is denoted by an arrow "→".
    - ➔ The functional dependency of A on B is represented by A → B. It should be read as 'A determines B' or ' B is functionally dependent on A'.
- *Example*: Suppose we have a student table with attributes

Student_Rollno	Student_Name	Student_faculty
112	John	Computer
113	Von	IT
114	Neumann	Civil

- Here, Student_Rollno uniquely identifies the Student_Name attribute of the student table because if we know the student roll no we can tell the student name associated with it.
- Functional dependency is represented by **Student_Rollno → Student_Name.**
- **Types of Functional Dependency**
    1. Trivial Dependency
    2. Non trivial Dependency
    3. Full functional Dependency
    4. Partial Functional Dependency
    5. Multivalued Dependency
    6. Transitive Dependency

1. Trivial Dependency

- The dependency of an attribute on a set of attributes is known as trivial functional dependency if the set of attributes includes that attribute. In this type of dependency RHS is the subset of LHS.
- Symbolically: **A → B is trivial functional dependency if B is the subset of A.**
- *Example*: Consider a table with two columns Student_Rollno & Student_Name. (Student_Rollno, Student_Name) → Student_Rollno is a trivial functional dependency as Student_Rollno is a subset of {Student_Rollno, Student_Name}.

2. Non Trivial Dependency

- If a functional dependency X → Y holds true where Y is not a subset of X then the dependency is called a non-trivial functional dependency.
- Symbolically: **A → B is trivial functional dependency if B is the subset of A.**

- *Example: An employee table with three attributes : emp_id, emp_name, emp_address.*
  Then emp_id → emp_name is non trivial.
  Here emp_name is not a subset of emp_id.

3. **Full Functional Dependency**

- It is related to composite determinants.
- If more than one attribute is necessary to determine another attribute in an entity then such a determinant is termed as composite determinant.
- In full functional dependency, it is necessary to use all the attributes of the composite determinant to identify objects uniquely.
- From the Order relation below, Full Functional Dependencies are
  (Order, Line) → Qty
  (Order, Line) → Price

Table name: Order

Order	Line	Qty	Price
A001	001	10	200
A002	002	20	400
A001	002	20	800
A004	001	15	300

4. **Partial Dependency**

- This is the situation that exists if it is necessary to only use a subset of the attributes of the composite determinant to identify its object uniquely.
- From the Student table below,
  Full Functional Dependencies: **(Student, Unit) → Grade**
  Partial Functional Dependencies: **Unit → Room**

Table name: Student

Student	Unit	Room	Grade
9900100	A01	TH224	2
9900010	A01	TH224	14
9901011	A02	JS075	3
9900100	A02	JS075	7
9900001	A01	TH224	16

5. Multivalued Dependency

- A multivalued attribute of an entity is an attribute that can have more than one value associated with the key of the entity. For example, a large company could have many divisions, some of them possibly in different cities.
- Multivalued dependency occurs when there are more than one independent multivalued attributes in a table.
- It is a full constraint between two sets of attributes in a relation.
- Multivalued dependency occurs when two attributes in a table are independent of each other but both depend on a third attribute.
- *Example:* Consider a bike manufacturing company which produces two colors (Black and White) in each model every year.

Bike_model	Manuf_year	Color
M1001	2007	Black
M1001	2007	Red
M2012	2008	Black
M2012	2008	Red
M2222	2009	Black
M2222	2009	Red

- Here, columns Manuf_year and Color are independent of each other and dependent on Bike_model. In this case these two columns are said to be multivalued dependent on Bike_model.

- These dependencies can be represented like this:

  Bike_model → Manuf_year
  Bike_model → Color

6. Transitive Dependency

- A functional dependency is said to be transitive if it is indirectly formed by two functional dependencies.
- **X → Z** is a transitive dependency if the following three functional dependencies holds true:

  X → Y
  Y does not → X
  Y → Z
- A transitive dependency can only occur in a relation of three or more attributes. This dependency helps us in normalizing the database in 3NF (Third normal Form).
- Example:

Book	Author	Author_age
Game of thrones	**George R.R Martin**	**66**
Harry Potter	**JK Rowling**	**49**
Dying in the Light	**George R.R Martin**	**66**

- The following cases can be visualized from the above table:

  Book → Author (If we know the book, we know the author name)
  Author does not → Book
  Author → Author_age
- As per the transitive dependency,

Book → **Author_age** should hold that makes sense because if we know the book name we can know the author's age.

STAGES OF NORMALIZATION

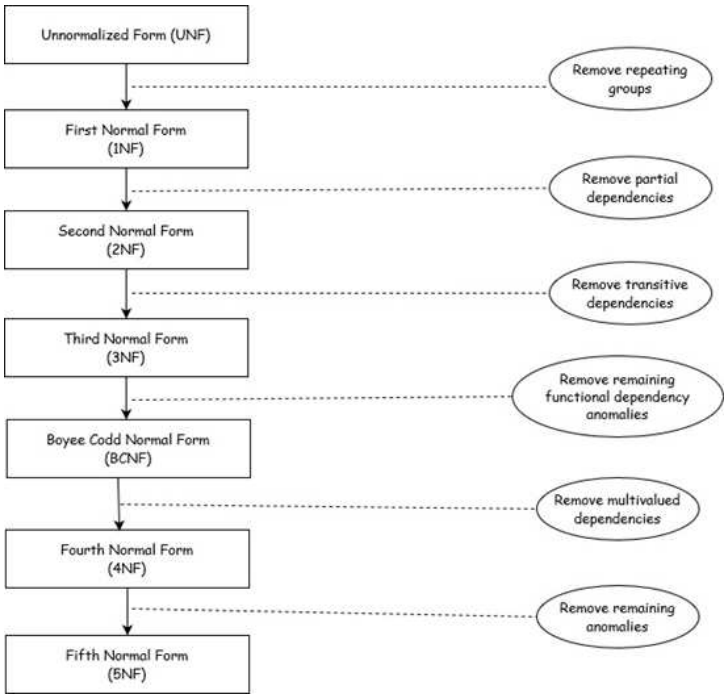


Figure I Stages of normalization

1. **Unnormalized Normal Form (UNF)**

- No normalization rules are applied to it.
- It suffers from various anomalies.
- A relation has a repeating group, so it has more than one value for the given key.
- Each tuple may contain multiple sets of values for some columns.
- It is not '**atomic**' in nature.

Example: Relation: Employee

emp_id	emp_name	emp_address	emp_mobile
901	Messi	Ktm	981114526
902	Ronaldo	Brt	986659844, 985565896
903	Pogba	Lalitpur	985651223, 981166215

The emp_mobile values for emp_id violate the **atomic** condition.

2. **First Normal Form (1NF)**

- As per the rule of 1NF, an attribute(column) of a table cannot hold multiple values. It holds only **atomic** values.
- Values stored in a column should be of the same domain.
- All the columns in a table should have unique names.

Example:

emp_id	**emp_name**	**emp_address**	**emp_mobile**
901	Messi	Ktm	981114526
902	Ronaldo	Brt	986659844
903	Pogba	Lalitpur	985651223

The above table is now in 1NF.

3. **Second Normal Form (2NF)**

- A table is said to be in 2NF if both the following conditions holds:
    1. Table should be in 1NF (First Normal Form)
    2. No ***non prime attributes*** is dependent on the proper subset of any candidate key of table i.e. to remove partial dependency.

Non prime attribute: An attribute that is not the part of any ***candidate key*** is known as non prime attribute.

Example: Suppose a school wants to store the data of teachers and the subjects that they teach.They create a table that looks like:

teacher_id	**subject**	**teacher_age**
111	DSA	27
111	DBMS	27
222	Applied	29
333	SOM	27
333	Steel	27

Since a teacher can teach more than one subject the table can have multiple rows for the same teacher.

Candidate key = {teacher_id, subject}
Non prime attribute = teacher_age

- However, it is not in 2NF because non prime attribute teacher_age is dependent on teacher_id alone which is a proper subset of a candidate key.
- To make the table complies with 2NF we can break it into two tables like this:

Relation name: teacher_details

teacher_id	**teacher_age**
111	27
222	29
333	27

Relation name: subject_table

teacher_id	**subject**
111	DSA
111	DBMS
222	Applied
333	SOM
333	Steel

4. **Third Normal Form (3NF)**

- The table is said to be in 3NF if both the conditions holds:
    1. Table must be in 2NF
    2. **Transitive functional dependency** of non prime attribute on any super key must be removed.

In other words, A table is in 3NF if it is in 2NF and for each functional dependency X-->Y One of the following conditions holds:

1. X is a super key of table
2. Y is a prime attribute of the table.

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

emp_id	emp_name	emp_zip	state	emp_city	emp_district
1001	ronaldo	021	S1	BRT	Morang
1002	messi	051	S2	BRJ	Parsa
1003	pogba	014	S3	Ktm	Kathmandu
1004	hazard	061	S4	Pok	Kaski
1005	bruno	081	S5	Npl	Banke

Super keys : { emp_id}, {emp_id, emp_name}, {emp_id, emp_name, emp_zip} ..... so on.
Candidate key: {emp_id}

- We can clearly see that all attributes except emp_id are non-prime as they are not part of the candidate key.
- **Note:** X → Y
  (non prime attribute) (non prime attribute)
  It should not be the case for the table to be in 3NF.

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

To make the table complies with 3NF we need to break the table into two tables to remove transitive dependency.

Employee table

emp_id	emp_name	emp_zip
1001	ronaldo	021
1002	messi	051
1003	pogba	014
1004	hazard	061
1005	bruno	081

Employee_zip table

emp_zip	**state**	**emp_city**	**emp_district**
021	S1	BRT	Morang
051	S2	BRJ	Parsa
014	S3	Ktm	Kathmandu
061	S4	Pok	Kaski
081	S5	Npl	Banke

So, the table is in 3NF.

5. **Boyce Codd Normal Form (BCNF)**
    - Improved form of 3NF
    - Also referred as 3.5NF
    - A table complies with BCNF if
        1. It should be in 3NF
        2. For every functional dependency X → Y, X should be the super key of the table.

           Note: X → Y

           (non prime attribute) (prime attribute)

           It should not be the case in BCNF.

Example:

College_faculty relation

Student_id	**Subject**	**Faculty**
101	Java	Teacher1
101	C++	Teacher2
102	Java	Teacher1
103	C#	Teacher3
104	Java	Teacher1

➔ It satisfies 1NF (atomic)

➔ **Student_id + Subject** (Together forms a primary key)

➔ Dependencies:

- (Student_id, Subject) → Faculty

- **Faculty → Subject**
- No partial dependencies
- No transitive dependencies

As of case: **Faculty → Subject**
(non prime attribute) (prime attribute)

This shows that the table is not in BCNF. To make it compile to BCNF we need to break the table into two tables with the introduction of attribute **faculty_id.**

Student Table

Student_id	Faculty_id
101	1

Faculty Table

Faculty_id	Faculty	Subject
1	Teacher1	Java

So, now the table is in BCNF.

Comparison Between 3NF and BCNF

- **Relations in BCNF and 3NF**
  - ➔ BCNF : redundancy is comparatively low
  - ➔ 3NF: redundancy can be observed
- **Dependencies and Decomposition in BCNF & 3NF**
  - ➔ BCNF: Lossless decomposition is hard to achieve; preservation of functional dependencies may or may not be achieved
  - ➔ 3NF: Lossless decomposition can be achieved; there is preservation of all functional dependencies.

6. **Fourth Normal Form (4NF)**

- A table is said to be in 4NF only if
  - ➔ It should be in BCNF
  - ➔ It contains no **multivalued dependencies**
- The redundancy caused by MVDs (Multivalued dependencies) can't be removed by transforming the database schema to BCNF.

Example:
Let us consider the following relation:

Subject	Author	Book
DSA	ABC	DBACE Book
DSA	CDF	William Book
DSA	CDF	DBACE Book
DSA	ABC	William Book

Decomposing the above table into 4NF, we get the following two tables:

Author_relation

Subject	Author
DSA	ABC
DSA	CDF

Book_relation

Subject	Book
DSA	DBACE Book
DSA	William Book

So, it is in 4NF.

7. **Fifth Normal Form (5NF)**

- A table is said to be in 5NF if
  - ➔ It should be in 4NF
  - ➔ It doesn't contain any join dependency & joining should be lossless.

- **Join dependency in DBMS**:
  - ➔ If a table can be recreated by joining multiple tables & each of this table have a subset of attributes of the main table then the table is in join dependency.
  - ➔ It is a generalization of multivalued dependency.

- 5NF is satisfied when all the tables are broken into as many tables as possible.
- 5NF is also known as **Project join normal form (PJNF)**.

Example: **R** relation

Subject	**Lecturer**	**Semester**
Computer	ABC	S1
Computer	EFG	S1
Math	EFG	S1
Math	HIJ	S2
DBMS	KLM	S1

➔ EFG takes both Computer & Math class for S1 but he doesn't take Math class for S2.
➔ Suppose we add a new *Semester* as **S3** but do not know about the subject & who will be taking the subject so we leave *Lecturer* and *Subject* as NULL.
➔ But all three columns together act as a primary key, so we can't leave the other two columns blank.

So to make the above table into 5NF we can decompose the table into three relations R1, R2 and R3.

R1

Semester	**Subject**
S1	Computer
S1	Math
S1	DBMS
S2	Math

R2

Subject	**Lecturer**
Computer	ABC
Computer	EFG

Math	EFG
Math	HIJ
DBMS	KLM

R3

Semester	**Lecturer**
S1	ABC
S1	EFG
S1	EFG
S2	HIJ
S1	KLM

So, now the table is in 5NF.

DECOMPOSITION

- The process of decomposition of a relation R into a set relations R1, R2….Rn is based on identifying different components & using that as a basis of decomposition.
- Properties of decomposition are:
    1. Lossless Join Decomposition
    2. Attribute Preservation
    3. Redundancy
    4. Dependency Preservation

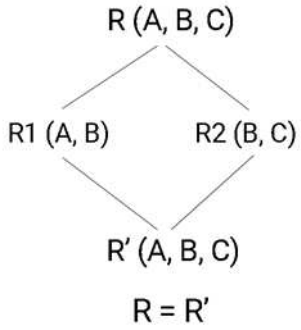
1. Lossless Join Decomposition

- A decomposition of a relation R into relations R1, R2, R3 ….. Rn is said to be lossless if the relation R is always the natural join of the relation R1, R2, ….. Rn.
- A decomposition is lossless if we can recover:

  R(A, B, C) →Decompose →R1(A, B) R2(B, C) →Recover -> R'(A, B, C)
  Thus, R' = R

2. Attribute Preservation

- The attributes in R will appear in at least one relation schema Ri in the decomposition, i.e. no attribute is lost.



- From the figure of decomposition, we can see relation R1(A , B) and R2(B , C) have the attributes A , B and C (all attributes preserved from original relation(R)).

3. Redundancy

- The data may be in different forms that may be completely unnecessary.
- Decomposition helps to eliminate problems like redundancy.
- In some cases, if the relation has no proper decomposition, then it may lead to problems like loss of information.
- Loss of information must be avoided as much as possible.

4. Dependency Preservation

- A decomposition D= { R1, R2, R3…..Rn} of R is dependency preserving with respect to F if the union of the projections of F on each Ri in D is equivalent to F i.e. (F1 U F2 U……..Fn) = F
- If a decomposition is not dependency preserving some dependency is lost in decomposition.
- To check this condition, take the join of two or more relations in the decomposition.

Sixth Normal Form (6NF)

- It is also referred as DKNF (Domain Key Normal Form)
- A relation will be in DKNF if every content of the relation is a logical consequence or domain constraints or key constraints i.e. each and every data in the database must be logical consequences of key or domain constraints.
- Key constraints: Each attribute participating in construction of key must be NOT NULL
- Domain Constraints: It can be defined as the definition of a valid set of values for an attribute
- In order for a table to be in 6NF, it has to comply with the 5NF first.
- Let's take a simple example with a table already in 5NF:

Username	**Department**	**Status**

➔ Here, in the users table, every attribute is non null and the primary key is the username.

➔ This table is in 5NF because each join dependency is implied by the unique candidate key of the table (Username). More specifically, the only possible join dependencies are: {username, status}, {username, department}.

➔ The 6NF would be like

Users relation

Username	**Status**

Users_dept relation

Username	**Department**

So, now it is in 6NF

CLOSURE SET OF ATTRIBUTES DEPENDENCIES

- It is the set of all attributes that can be computed using given attributes.Closure of attribute set {X} is denoted as $(X)^+$.
- How to find attribute closure of an attribute set?
  - ➔ Add elements of the attribute set to the result set.
  - ➔ Recursively add elements to the result set which can be functionally determined from the elements of the result set.
- An **algorithm to compute α+**, the closure of α under F is as follows :

  result = α
  while (changes to result) do
  for each functional dependency β -> γ in F do
  begin
  if β ⊆ result then result = result ∪ γ
  end

- Example: Let us consider R(A, B, C) and two functional dependencies :

  A -> B
  B -> C

  Now,
  A^+= {A, B, C} is the closure set of A.
  B^+ = {B, C} is the closure set of B.
  C^+ = {C} is the closure set of C.

- Example : Consider an EMPLOYEE Table

E-ID	E-NAME	E-CITY	E-STATE
E001	John	Kathmandu	Kathmandu
E002	Mary	Kathmandu	Kathmandu
E003	John	Janakpur	Pokhara

Given : R(E-ID, E-NAME, E-CITY, E-STATE)

FDs = { E-ID->E-NAME, E-ID->E-CITY, E-ID->E-STATE, E-CITY->E-STATE }

The attribute closure of E-ID can be calculated as:

- ➔ Add E-ID to the set {E-ID}
- ➔ Add Attributes which can be derived from any attribute of set. In this case, E-NAME and E-CITY, E-STATE can be derived from E-ID. So these are also a part of closure.

As there is one other attribute remaining in relation to be derived from E-ID. So result is:

$(E\text{-}ID)^+$ = {E-ID, E-NAME, E-CITY, E-STATE } Similarly,
$(E\text{-}NAME)^+$ = {E-NAME}
$(E\text{-}CITY)^+$ = {E-CITY, E_STATE}

CLOSURE SET OF FUNCTIONAL DEPENDENCIES

- If F is a set of functional dependencies then the closure of F, denoted by F^+, is the set of all functional dependencies implied by F.
- Armstrong axioms are a set of rules used to find F+.

Rule 1 - Reflexivity rule

If $\beta \subseteq \alpha$ then $\alpha \rightarrow \beta$

Rule 2 - Augmentation rule

It states that addition of attributes in dependencies does not change the basic dependencies. That is, if a -> b holds and y is attribute set, then ay -> by also holds true.

Rule 3 - Transitivity

If a -> b holds and b -> c holds then a -> c also holds

Additional Rules

Union Rule : If $\alpha \rightarrow \beta$ and $\alpha \rightarrow \gamma$, then $\alpha \rightarrow \beta\gamma$.

Decomposition Rule : If α -> βγ holds then α -> β and α -> γ.

Pseudo-transitivity Rule : If α -> β and βγ - δ holds then αγ -> δ holds.

- Example : Consider a STUDENT table with functional dependencies :
  Stu_Id → Stu_Name
  Stu_Id → STU_province

Stu_Id	Stu_name	Stu_phone	Stu_province	Stu_country	Stu_age

- **How to find functional dependencies for a relation?**
  ➔ Functional Dependencies in a relation are dependent on the domain of the relation. Consider the STUDENT relation given in the above table.

We know that Stu_Id is unique for each student.

So Stu_Id → Stu_Name, Stu_Id→ Stu_phone, Stu_Id→ Stu_province, Stu_Id→ Stu_country and Stu_Id → Stu_age all will be true.

Similarly, Stu_province → Stu_country will be true as if two records have the same Stu_province, they will have the same Stu_country as well.

- **How to find Candidate Keys and Super Keys using Attribute Closure?**
  ➔ If attribute closure of an attribute set contains all attributes of relation, the attribute set will be super key of the relation.
  ➔ If no subset of this attribute set can functionally determine all attributes of the relation, the set will be candidate key as well.

(Stu_Id, Stu_name, Stu_phone)$^+$ = {Stu_Id, Stu_name, Stu_phone, Stu_province, Stu_country, Stu_age }

(Stu_Id, Stu_name)$^+$ = {Stu_Id, Stu_name, Stu_phone, Stu_province, Stu_country, Stu_age }

(Stu_Id)$^+$ = {Stu_Id, Stu_Name, Stu_phone, Stu_province, Stu_country, Stu_age }

So, we can see that **(Stu_Id, Stu_name)** will be super key but not candidate key because its subset (Stu_Id)$^+$ is equal to all attributes of the relation.

Thus **Stu_Id** is the candidate key.

Example : For given FD, find all the possible F+

R= (A, B, C, G, H, I)

F= { A → B, A → C, CG → H, CG → I, B → H}

Solution:

A → H by transitivity from A → B and B → H

AG → I by augmenting A → C with G, to get AG → CG and then transitivity CG → I

Question: Consider the following relation R(A, B, C, D, E) having functional dependencies

A → BC, C → B, D → E, E → D.

1. List all the closure of the attributes.
2. Find the candidate key for the relation.

Q.1. Explain 1NF, 2NF, 3NF and 3.5NF in brief with its appropriate examples.

Q.2. What does A → B mean ? Explain how BCNF differentiate itself from 3NF with appropriate examples.

Q.3. What do you mean by join dependency? Explain 4NF & 5NF in brief.

Q.4. Define non prime attributes. Explain BCNF, 4NF & 5NF with appropriate examples.

Q.5. Define normalization. Explain different stages of normalization.

Q.6. Define integrity constraints. Write about constraints while creating a table.

Q.7. Explain different types of functional dependencies in brief.

Q.8. What is a database anomaly ? Explain its types with examples.

Q.9. Write short notes on the

I. Closure set of functional dependencies.

II. DKNF

III. 3.5 NF

IV. Assertions Vs Triggers