Chapter 5

Database constraints and Relational model

Relational database design

- ✓ A relational database is a database based on the relational model of data, as proposed by E. F. Codd in 1970.
- ✓ It has since become the dominant database model for commercial applications.
- ✓ Today, there are many commercial Relational Database Management System (RDBMS), such as Oracle, IBM DB2 and Microsoft SQL Server.
- ✓ There are also many free and open-source RDBMS, such as MySQL.
- ✓ A relational database organizes data in tables (or relations). A table is made up of rows and columns. A row is also called a record (or tuple). A column is also called a field (or attribute).
- ✓ The relationships that can be created among the tables enable a relational database to efficiently store huge amount of data, and effectively retrieve selected data.
- ✓ A language called SQL (Structured Query Language) was developed to work with relational databases.

Database Design Objective

A well-designed database should fulfill following objectives.

- ✓ **Eliminate Data Redundancy:** the same piece of data shall not be stored in more than one place. This is because duplicate data not only waste storage spaces but also easily lead to inconsistencies.
- ✓ **Scalability:** The design should accommodate future growth and evolving data requirements
- ✓ **Accuracy and Integrity:** the data stored in the database is correct, complete, and consistent. This is achieved by enforcing data validation rules, such as data type, length, and format.
- ✓ **Performance:** database can process data efficiently and quickly, even under high load conditions
- ✓ **Privacy and Security:** The objective of privacy and security in a database is to protect the confidentiality, integrity, and availability of the data stored in the database. This is achieved by implementing access controls, such as user authentication and authorization.

Integrity constraints

- ✓ In Database Management Systems, integrity constraints are pre-defined set of rules that are applied on the table fields(columns) or relations to ensure that the overall validity, integrity, and consistency of the data present in the database table is maintained.
- ✓ Evaluation of all the conditions or rules mentioned in the integrity constraint is done every time a table insert, update, delete, or alter operation is performed.
- ✓ The data can be inserted, updated, deleted, or altered only if the result of the constraint comes out to be True. Thus, integrity constraints are useful in preventing any accidental damage to the database by an authorized user.

Types of Integrity constraints in DBMS

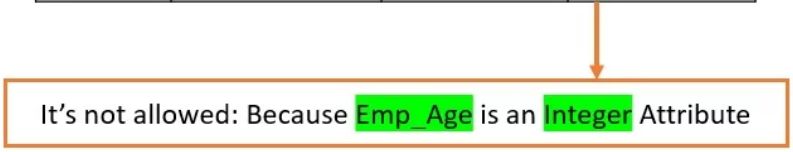
1. Domain Integrity Constraint
2. Entity Integrity Constraint
3. Referential Integrity Constraint
4. Key Constraints

1. Domain integrity constraints

- ✓ Domain integrity constraints in SQL can be defined as the definition of a valid set of values for an attribute.
- ✓ The data type of domain includes integer, character, string, time, date etc. Furthermore, the value of the attribute must be available in the corresponding domain.
- ✓ A domain constraint is a set of rules that restricts the kind of attributes or values a column or relation can hold in the database table. i.e. we can specify if a specific column can hold null values or not, furthermore, if the values have to be unique or not, the size of values or the data type that can be entered in the column, the default values for the column, etc.

Let us consider the following table

Emp_ID	Emp_Name	Emp_City	Emp_Age
101	Arun	Kathmandu	32
102	Sita	Pokhara	36
103	Rita	Chitwan	41
104	Gopal	Butwal	Z



2. Entity Integrity Constraints

- ✓ Entity Integrity Constraint is used to ensure the uniqueness of each record or row in the data table.
- ✓ In other words, the entity integrity constraint states that primary key value can't be null. This is because the primary key value is used to identify individual rows in relation and if the primary key has a null value, then we can't identify those rows.
- ✓ The unique also key helps in uniquely identifying a record in the data table. It can be considered somewhat similar to the Primary key as both of them guarantee the uniqueness of a record. But unlike the primary key, a unique key can accept NULL values and it can be used on more than one column of the data table.

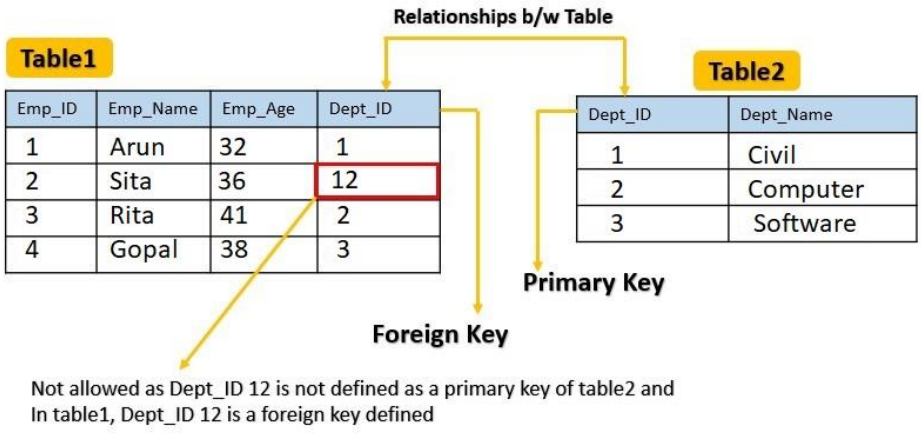
Emp_ID	Emp_Name	Emp_City	Emp_Age
101	Arun	Kathmandu	32
102	Sita	Pokhara	36
103	Rita	Chitwan	41
	Gopal	Butwal	38

NOT ALLOWED AS PRIMARY KEY CAN'T CONATIN A NULL VALUE

3. Referential integrity constraints

- ✓ Referential integrity constraint is specified between two tables.
- ✓ Referential Integrity Constraint is defined as it ensures that there always exists a valid relationship between two tables.
- ✓ Furthermore, this makes confirms that if an foreign key exists in a table relationship then it should always reference a corresponding value in the 2nd table or it should be null.

Let us consider the following example



4. Key Constraints

There are a number of key constraints in SQL that ensure that an entity or record is uniquely or differently identified in the database. There can be more than one key in the table but it can have only one primary key.

SQL's key constraints include the following:

- ✓ Primary Key
- ✓ Foreign Key
- ✓ Unique Key

Emp_ID	Emp_Name	Emp_Age
101	Arun	32
102	Pokhara	36
103	Rita	41
102	Gopal	38

Not allowed because all row must be unique

Assertions

- ✓ An assertion is a statement in SQL that ensures a certain condition will always exist in the database.
- ✓ Same as domain or other constraints, assertions differ in the way that they are defined separately from table definitions.
- ✓ An assertion in SQL takes the form

  create assertion <assertion-name> check <predicate condition>
- ✓ When an assertion is made, the system tests it for validity, and 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.
- ✓ The assertion is said to be violated if the query result is not empty and hence the user is denied from modifying the database.

Example :

Let us consider the following relation

sailors(sid,sname,rating,age)
Boats(bid,bname,color)
Reserves(sid,bid,day)

Create an Assertion to For number of boats plus number of sailors is < 100

```
CREATE ASSERTION check_number
 CHECK  ((SELECT  COUNT (sid) FROM SAILORS )  +
(SELECT COUNT (bid)  FROM BOATS  )<100);
```

Triggers

A trigger is a statement that the system executes automatically as a side effect of modification of database.

To design a trigger mechanism, we must meet two requirements.

- ✓ Specify when a trigger is to be executed. This is broken up into an event that causes the trigger to be checked and a condition that must be satisfied for trigger execution to proceed.
- ✓ Specify the actions to be taken when the trigger executes

Once we enter a trigger into a database, the database system takes on the responsibility of executing it whenever the specified event occurs and the corresponding condition satisfied.

Syntax:

```
CREATE [OR REPLACE] TRIGGER trigger_name
{BEFORE | AFTER } {INSERT | UPDATE | DELETE}
ON table_name
[FOR EACH ROW]
[WHEN (condition)]
BEGIN
   -- Trigger body: SQL statements or code
END;
```

CREATE [OR REPLACE] TRIGGER: Specifies the creation of a new trigger or replaces an existing trigger with the same name.

BEFORE | AFTER: Specifies when the trigger should be executed relative to the triggering event. A "BEFORE" trigger fires before the event, and an "AFTER" trigger fires after the event.

INSERT | UPDATE | DELETE: Indicates the event that triggers the execution of the trigger. You can specify multiple events using commas.

ON table_name: Specifies the name of the table on which the trigger is defined.

FOR EACH ROW: Indicates that the trigger should be executed once for each affected row (used for row-level triggers).

BEGIN and END: Enclose the trigger body, which contains the SQL statements or code to be executed when the trigger is fired.

Need of triggers

Triggers can be used to implement certain integrity constraints that cannot be specified using the constraint mechanism of SQL. Triggers are useful mechanisms for alerting humans or for starting certain tasks automatically when certain conditions are met.

Suppose a warehouse wishes to maintain the a minimum inventory of each item. When inventory level of an items falls below the minimum level, an order can by placed automatically. On update of the inventory level of each item, the trigger compares the current inventory level with minimum inventory level for the item, and if the level is at below new order is created.

Implementation

First, we need to create a table called "Items" that will store the item details, including the inventory level and the minimum level. The table schema might look like this:

Items (ItemID ,ItemName ,InventoryLevel , MinimumLevel)

We need a table to store the generated orders when the inventory level falls below the minimum level. Create a table called "Orders" with the necessary columns to store the order details.

Orders (OrderID , ItemID ,Quantity ,OrderDate)

Now, we'll create a trigger that will automatically place an order when the inventory level falls below the minimum level for any item. The trigger will be executed after an update operation on the "Items" table.

```
DELIMITER //
CREATE TRIGGER check_inventory_level
AFTER UPDATE ON Items
FOR EACH ROW
BEGIN
   DECLARE current_inventory INT;
   DECLARE minimum_level INT;
   SET current_inventory = NEW.InventoryLevel;
   SET minimum_level = NEW.MinimumLevel;

   IF current_inventory < minimum_level THEN
      INSERT INTO Orders (ItemID, Quantity, OrderDate)
      VALUES (NEW.ItemID, (minimum_level - current_inventory), CURDATE());
   END IF;
END //
DELIMITER ;
```

Here in orders table we have assumed that it we defined the "OrderID" column to be an auto-increment primary key.

Note that auto-increment allows a unique number to be generated automatically when a new record is inserted into a table. Often primary key field that we would like to be created automatically every time a new record is inserted.

In this trigger:

- The trigger is defined as an "AFTER UPDATE" trigger on the "Items" table.
- For each updated row, the trigger checks the current inventory level (NEW.InventoryLevel) and the minimum inventory level (NEW.MinimumLevel).
- If the current inventory level is less than minimum level, a new order is inserted into the "Orders" table. The order includes the ItemID, the quantity needed to reach the minimum level (minimum_level - current_inventory), and the current date (CURDATE()).

Functional dependency

- ✓ A functional dependency is a relationship between two attributes, typically between the Primary key and other non-key attributes within a table.
- ✓ For a given relation R with attribute X and Y, Y is said to be functionally dependent on X, if given value for each X uniquely determines the value of the attribute in Y.
- ✓ X is called determinant and Y is the object of the determinant and denoted by X→Y.

Example:

emp_id	emp_name	address	salary
101	ram	kathmandu	25000
102	shyam	pokhara	47000
103	hari	chitwan	65000

Assume we have a employee table with attributes emp_id,emp_name,address,salary.

Here emp_id attribute can uniquely identify the emp_name attribute of employee table because if we know the emp_id we can tell the employee's name associated with it.

Functional dependency can be written as:

emp_id→emp_name

we can say that emp_name is functionally dependent on emp_id.

Similarly,

emp_id→address
emp_id→salary

compound determinant

- ✓ If more than one attribute is necessary to determine another attribute, then such determinant is called as compound Determinant

Let us consider the Student relation

Student(student_id,student_name,salutation,address,course_id,course_name,marks)

{student_id,course_id} → marks

- ✓ In this example, the compound determinant is (student_id, course_id).
- ✓ It indicates that the attributes marks is determined by the combination of student_id and course_id.

Types of functional dependency

- ❖ **Full functional dependency**

- ✓ A full functional dependency is dependency in which a non-key attributes is dependent on all the attributes of composite key.
- ✓ This is the situation in which all the attributes of the composite key is used to uniquely identify its object.
- ✓ A Functional dependency X→Y is said to be full functional dependency if removal of any attribute A from X removes the dependency, which means set of attributes on the left side cannot be reduced further.

- ❖ **Partial functional dependency**

- ✓ A partial dependency is dependency in which a non-key attribute is dependent on only a part of composite key.
- ✓ This is the situation in which only a subset of the attributes of the composite key is used to uniquely identify its object.
- ✓ A Functional dependency X→Y is partial dependency if some attributes A ∈ X can be removed from X and dependency still holds.

Example:

student_id	student_name	salutation	course_id	course_name	marks
1	Ram	Mr.	1	DBMS	92
2	Shyam	Mr.	2	C++	85
3	Gita	Ms.	1	DBMS	78
4	Hari	Mr.	3	Java	75
5	Ritu	Ms.	2	C++	68

Here, {student_id,course_id}→marks

is an example of **full functional dependency**. Removal of any attributes breaks the dependency.

Here,student_id or course_id alone cannot determine marks.

Again,

{student_id,course_id}→{course_name}

is an example of **partial dependency.**

{course_id}→{course_name} still holds true. Here, removal of student_id do not have any effect.

- ❖ **Trival functional dependency**

X →Y is trivial functional dependency if Y is a subset of X.

The following dependencies are also trivial: X→X and Y→Y

Example

In above student relation

{student_id,student_name}→student_name

{course_id,course_name}→course_name

student_id→student_id

are examples of trival functional dependency.

- ❖ **Non-trivial functional dependency**

- ✓ X → Y has a non-trivial functional dependency if Y is not a subset of X.
- ✓ When X intersection Y is NULL, then X → Y is called as complete non-trivial.

Example:

In above student relation

{student_id,course_id} →marks

Student_id → student_name

Course_id → course_name

- ❖ **Transitive dependency**

- ✓ A Transitive Dependency is a type of functional dependency which happens when it is indirectly formed by two functional dependencies.
- ✓ If X→ Y and Y→Z then X→Z is a transitive dependency.

Example:In above student relation, we have the following functional dependencies:

student_id →student_name(Each student_id uniquely determines the student_name)

student_name →salutation (Each student_name uniquely determines the salutation)

Therefore, by transitive dependency, we can conclude that student_id indirectly determines the salutation. i.e. student_id→salutation

❖ **Multivalued dependency**

- ✓ A multivalued dependency between two attributes X and Y in relation R is represented as X→→Y which means for each value of attribute X there can be multiple corresponding value of Y.
- ✓ multivalued dependency occurs when there are more than one independent multivalued attributes in a relation.
- ✓ A multivalued dependency consist of at least two attributes that are dependent on third attribute that's why it requires at least three attributes.

Example:

course	teacher	book
DBMS	Ramesh	Database concept
DBMS	Nikita	Fundamental on DBMS
DBMS	Ramesh	Fundamentals on DBMS
DBMS	Nikita	Database concept

Here attributes teacher and book are dependent on course and independent to each other.

As well course is associated with set of teachers and set of books .The representation of these dependencies is shown below.

course→→teacher

course→→ book

Closure set of functional dependencies

- ✓ For a given set of functional dependencies F, there are certain other functional dependencies that are logically implies by F.
- ✓ For example if A→B and B→C, then we can infer that A→C.
- ✓ The set of all functional dependencies logically implied by F is the closure of F.
- ✓ Closure of F is denoted by F^+.
- ✓ F^+ is superset of F.

We can use the following three rules to find logically implied functional dependencies. By applying this rules repeatedly ,we can find all of F^+ for given F. This collection of rules is called Armstrong's axioms in honor of the person who purposed it.

Reflexivity rule: If α is a set of attributes and β ⊆ α,then α → β holds.

Augmentation rule: If α → β holds and γ is a set of attributes ,then γ α →γ β holds.

Transitivity rule: if α → β holds and β →γ holds then α →γ holds.

We can further simplify the the computation of F^+ by using the following addition rule

union rule: if α → β holds and α → γ holds, then α → β γ holds.

Decomposition rule: if α → β γ holds then, α → β holds and α → γ holds

Pseudotransitivity rule: if α → β holds and γ β →δ holds then α γ →δ holds
Let us apply our rules to the example of schema R = (A, B, C, G, H, I) and the set F of functional dependencies {A → B, A → C, CG → H, CG → I, B → H}.

We list several members of F^+ here:

A→BC

- Since A→B and A→C , the union rule implies that A→BC

A→H

- Since *A → B* and *B → H* hold, we apply the transitivity rule.

CG → HI

- Since *CG → H* and *CG → I* , the union rule implies that *CG →HI* .

AG → I

- Since A → C and CG → I, the pseudo transitivity rule implies that AG → I holds

Another way of finding that AG → I holds is as follows: We use the augmentation rule on A → C to infer AG → CG. Applying the transitivity rule to this dependency and CG → I, we infer AG → I.

F^+={A → B, A → C, CG → H, CG → I, B → H, A→BC, A→H , *CG → HI,* AG → I}

Closure of Attribute Sets

Closure of attribute set is the set of attributes which are functionally dependent on the attribute set.Let α be a set of attributes. We call the set of all attributes functionally determined by α under a set *F* of functional dependencies the **closure** of under *F*; we denote it by α^+.

Algorithm

```
result := α;
repeat
for each functional dependency  β → γ   in F do
begin
if β  ⊆ result then result := result ∪ γ  ;
end
until (result does not change)
```

Here, an algorithm, written in pseudocode to compute α^+. The input is a set *F* of functional dependencies and the set of attributes. The output is stored in the variable *result.*

Let R = (A, B, C, G, H, I) and the
F= {A → B, A → C, CG → H, CG → I, B → H}

Now compute $(AG)^+$

Let result= {AG}

1st Iteration

For A → B

causes us to include B in result. To see this fact, we observe that
A → B is in F, A ⊆ result (which is AG), so result := result ∪ B.

∴result={ABG}

For A → C

A ⊆ {ABG} is true

∴result={ABCG}

For CG → H

CG ⊆ {ABCG} is true

∴result={ABCGH}

For CG → I

CG ⊆ {ABCGH} is true

∴result={ABCGHI}

For B → H

B ⊆ {ABCGHI} is true

∴result={ABCGHI}

2nd Iteration

For A → B

A ⊆ {ABCGHI} is true

∴result={ABCGHI}

For A → C

A ⊆ {ABCGHI} is true

∴result={ABCGHI}

For CG → H

CG ⊆ {ABCGHI} is true

∴result={ABCGHI}

For CG → I

CG ⊆ {ABCGHI} is true

∴result={ABCGHI}

For B → H

B ⊆ {ABCGHI} is true

∴result={ABCGHI}

We get the final result ={ABCGHI}

no new attributes are added to *result*, and the algorithm terminates.

$\therefore (AG)^+ = \{ABCGHI\}$

Assignment:

Let R={A,B,C,D,E,F}

F= {A → BC, E → CF, B→ E, CD → EF}

Now compute $(AB)^+$

Anomaly in DBMS

- ✓ Anomaly refer to unexpected or undesirable behaviors that can occur when manipulating or accessing data within a database.
- ✓ These anomaly are typically the result of data inconsistencies, redundancies, or dependencies that can lead to incorrect or illogical results.
- ✓ Anomalies can occur due to various reasons, such as poor database design, incomplete or inconsistent data entry, or improper data manipulation operations.

These anomalies can be categorized into three types:

1. Insertion anomaly
2. Deletion anomaly
3. Updation anomaly

Example

EmployeeID	Name	Address	Department	HOD
1	Ramesh	Kathmandu	Civil	Shyam
2	Anish	Butwal	Computer	Hari
3	Ritu	Pokhara	Civil	Shyam
4	Sita	Chitwan	Computer	Hari
5	Nabin	Pokhara	Computer	Hari

1. **Insertion anomaly**

- ✓ Insertion anomalies occur when it is not possible to add data to a database without including additional, unrelated information.
- ✓ In above example, suppose we you want to add a newly hired employee who hasn't assigned a department yet, we may not be able to insert the employee information into the table due to the missing department information or we will have to set department information NULL.
- ✓ If we insert the 100 employee of same department then the department information will be repeated for all those 100 employees.

2. Updation anomaly

- ✓ Updation anomaly occur when modifying data within a database leads to inconsistent information. This happens when data is duplicated across multiple records, and updates are made to some records but not others, resulting in inconsistent values.
- ✓ In above example, if Hari is no longer HOD of computer department then all the employee records have to be updated and if by mistake we miss any records it will lead to data inconsistency.

3. Deletion anomaly

- ✓ Deletion anomalies occur when removing data from a database results in unintentional loss of other related data.
- ✓ Suppose if employee named Ramesh leaves company and if we want to delete Ramesh information then department information, i.e deparment name civil and HOD Shyam also removed.

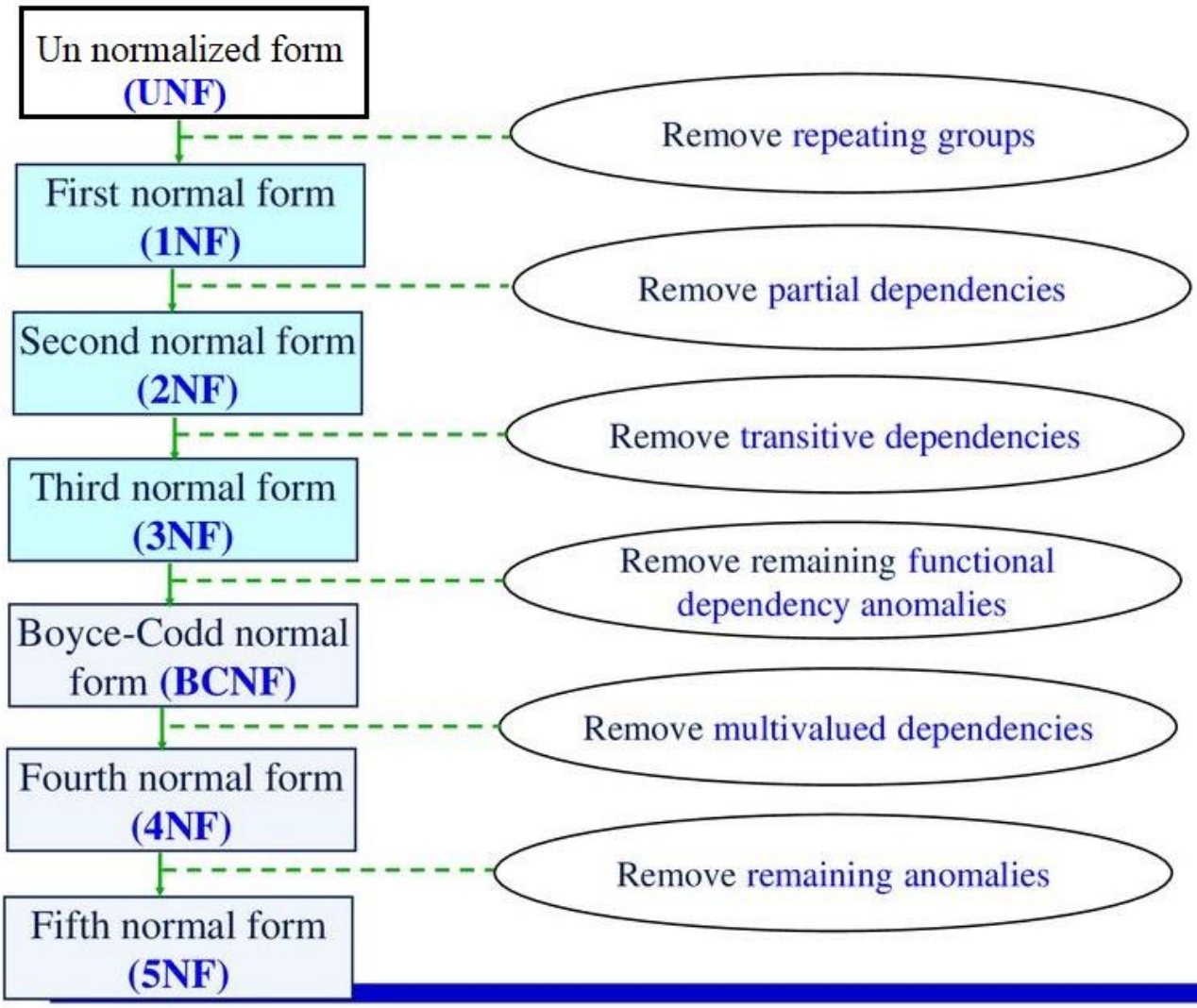
Normalization

- ✓ Database Normalization is a technique of organizing the data in the database.
- ✓ It 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, removing duplicated data from the relation tables.

Advantages of Normalization

- ✓ Normalization ensures data integrity by minimizing data redundancies and inconsistencies in a database.
- ✓ It optimizes data storage, reducing disk space usage and storage costs.
- ✓ Normalized databases tend to have improved performance, especially when executing complex queries.
- ✓ Updates and maintenance become simpler and less prone to errors with a normalized database structure.
- ✓ Normalization provides scalability and flexibility, allowing for easier expansion and modification of the database as it evolves.

Stages of Normalization



Unnormalized form (UNF)

- ✓ A relation is unnormalized when it has not any normalization rules applied to it, and it suffers from various anomalies.
- ✓ It has repeating group.(i.e. it has more than one value for a given cell.

Example:

Emp_id	Emp_name	Project
1	Ram	A,B
2	sita	C
3	Hari	D,E

First Normal Form (1NF)

A relation is said to be in 1NF if and only if

- ✓ Each cell is single valued i.e no repeating groups
- ✓ Values stored in a column should be of the same domain
- ✓ No two rows are identical

Now transforming above relation into 1NF,we get

Emp_id	Emp_name	Project
1	Ram	A
1	Ram	B
2	sita	C
3	Hari	D
3	Hari	E

Second Normal Form (2NF)

A relation is in 2NF if and only if,

- ✓ It is in 1NF and
- ✓ Every non-prime attribute is fully dependent on Primary key.(has no partial dependency)

In other words we can say that in 2NF no non-prime attribute is dependent on the proper subset of any candidate key of the table.

Converting from 1NF to 2NF

1. Identify the primary key for the 1NF relation.
2. Identify functional dependencies in the relation.
3. If partial dependencies exits on the primary key remove them by placing them in a new relation along with copy of their determinant.

Example:

Let us consider the following relation named **Studentcourse**

sid	student_name	course_id	course_name	Grade
1	Ram	1	Java	A
2	Ritu	1	Java	B
1	Ram	2	MERN Stack	B
2	Ritu	3	C ++	C

it has a composite key (Here, the composite key means primary key on two attributes “sid” and “course_id” and non-prime attributes are not fully functionally dependent on the primary key attribute.

For example,

- ✓ The “student_name” depends on “sid” and not depends on “course_id”. (referred to a partial dependency)
- ✓ The “course” depends on “course_id” and not depends on “student_id”. (referred to a partial dependency)

Now decompose table as follows

student

student_id	student_name
1	Ram
2	Ritu

course

course_id	course_name
1	Java
2	MERN stack
3	C++

Score

student_id	course_id	Grade
1	1	A
2	1	B
1	2	B
2	3	C

Third Normal Form (3NF)

A relation is in 3NF if

- ✓ It is in 2NF.
- ✓ Every non-prime attribute is non-transitively dependent on the primary key. Which means there should not be case that non-prime attribute is functionally dependent on another non-prime attribute.

converting from 2NF to 3NF

1. identify the primary key in the 2NF relation
2. Identify functional dependencies in the relation
3. If transitive dependency exits on the primary key remove them by placing them in a new relation along with copy of their dominant.

Example:

Let us consider the following relation

employee_info

emp_id	emp_name	zip_code	city
1	Ram	501	Kathmandu
2	Hari	502	Pokhara
3	Ritu	501	Kathmandu
4	Gopal	503	Butwal
5	Nisha	502	Pokhara

In above relation emp_id is a primary key. All other attributes are dependent on emp_id, so it is in 2NF but city ,non key attribute is dependent on zip_code, another non-key attributes .So it is not in 3NF.Now decompose above table as below.

Employee table

emp_id	emp_name	zip_code
1	Ram	501
2	Hari	502
3	Ritu	501
4	Gopal	503
5	Nisha	502

Employee_zip table

zip_code	city
501	Kathmandu
502	Pokhara
503	Butwal

Boyce Codd Normal Form(BCNF)

BCNF (Boyce Codd Normal Form) is the advanced version of 3NF which means For BCNF relation already should be in 3NF.

The relation schema *R* is in BCNF with respect to a set *F* of functional dependencies if for all functional dependencies in F^+ of the form

$$\alpha \rightarrow \beta$$

where $\alpha \subseteq R$ and $\beta \subseteq R$, at least one of the following holds:

- $\alpha \rightarrow \beta$ is trivial (i.e., $\beta \subseteq \alpha$)
- α is a superkey for *R*

In other words, we can also say that a relation is in BCNF if every determinant is a candidate key.

If a relation has only one candidate key then 3NF and BCNF are equivalent.

Example:

Let us assume

- ✓ For each subject ,each student is taught by one teacher
- ✓ Each teacher teaches only one subject
- ✓ Each subject is taught by several teachers

student	course	teacher
Ram	DBMS	Hari
Ram	Microprocessor	Ramesh
Anita	Math	Gita
Bipana	DBMS	Rojan
Laxman	DBMS	Hari

In above relation candidate key can be considered as {student, course}

and following functional dependency exits in above relation

1) {student,course}→teacher

2) teacher→course

In 2) Teacher is not a candidate key but determines course so in above figure is not in BCNF. so we need to decompose it.

Now decompose it as follows

table1

student	teacher
Ram	Hari
Ram	Ramesh
Anita	Gita
Bipana	Rojan
Laxman	Hari

table 2

teacher	course
Hari	DBMS
Ramesh	Microprocessor
Gita	Math
Rojan	DBMS

Fourth Normal Form 4NF

A relation R is in 4NF if

- ✓ it is in BCNF
- ✓ It contains no multivalued dependencies.

Example:

course	teacher	book
DBMS	Ramesh	Database concept
DBMS	Nikita	Fundamental on DBMS
DBMS	Ramesh	Fundamentals on DBMS
DBMS	Nikita	Database concept

Here, course is associated with set of teachers and set of books. which can be represented as

course→→teacher

course→→ book

Now, above relation can be decomposed as

table1

course	teacher
DBMS	Ramesh
DBMS	Nikita

table2

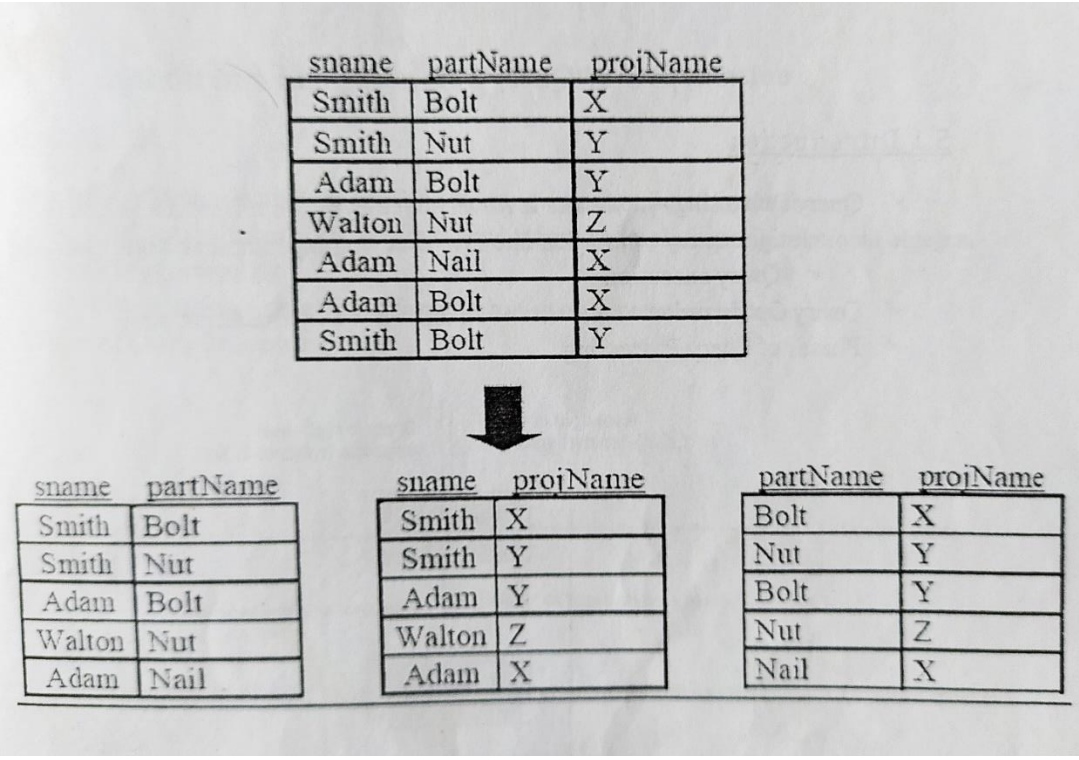
course	book
DBMS	Database concepts
DBMS	Fundamentals on DBMS

Join dependency and 5NF

- ✓ A relation R is subject to a join dependency (JD) if R can always can be recreated by joining multiple tables each having subset of attributes R.
- ✓ Let R be the relation schema and $R_1, R_2, R_3 \ldots R_N$ are projection of R ,
  a legal relation r of R satisfies the joined dependency (JD) if join of the projection of relation on R_i [i=1,2,3,.....N] is equal to
  $R = \prod_{R1}(r) \bowtie \prod_{R2}(r) \bowtie \prod_{R2}(r) \ldots\ldots\ldots\ldots \bowtie \prod_{RN}(r)$
- ✓ A Fifth normal form (5NF) is based on join dependencies.
- ✓ A relation R is in 5NF if and only if it satisfies the following conditions:
  1. R should be already in 4NF.
  2. It cannot be further non loss decomposed (join dependency).
- ✓ A join dependency describes a situation where a relation can be decomposed into two or more smaller relations, and the original relation can be reconstructed by performing a join operation on these smaller relations.

  Example: consider the relation supply (sname,partName,projName)

  Now it can be decomposed as



sname	partName	projName
Smith	Bolt	X
Smith	Nut	Y
Adam	Bolt	Y
Walton	Nut	Z
Adam	Nail	X
Adam	Bolt	X
Smith	Bolt	Y

sname	partName
Smith	Bolt
Smith	Nut
Adam	Bolt
Walton	Nut
Adam	Nail

sname	projName
Smith	X
Smith	Y
Adam	Y
Walton	Z
Adam	X

partName	projName
Bolt	X
Nut	Y
Bolt	Y
Nut	Z
Nail	X

Decomposition

- ✓ When a relation in a relational model is not appropriate normal form then the decomposition of a relation is required.
- ✓ In a database, it breaks a table into multiple tables.
- ✓ 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.

Desirable properties of decomposition are:

- ➢ Attribute preservation
- ➢ Lossless join decomposition
- ➢ Dependency preservation
- ➢ Lack of redundancy

❖ Attribute preservation

- ✓ Attribute preservation refers to the property of maintaining all the attributes of the original relation during the process of breaking it down into smaller relations.
- ✓ The goal is to ensure that no attribute is lost or eliminated during the decomposition.
- ✓ Each attribute should be represented in at least one of the decomposed relations to ensure that the information is retained.

❖ Lossless join decomposition

- ✓ A decomposition of a relation R into relations R1, R2, R3. . . . . RN is called lossless decomposition if the relation R is always the natural join of relations R1, R2, R3. . . . . RN.
- ✓ It should be noted that natural join is the way to recover the relation form the decomposed relations.
- ✓ i.e if $R=\prod_{R1}(R) \bowtie \prod_{R2}(R) \bowtie \prod_{R3}(R)\ldots\ldots\ldots \prod_{Rn}(R)$ is lossless decomposition otherwise lossy decomposition.
- ✓ Decomposition of relation R =(A,B,C)

A	B	C
1	a	x
2	b	y
3	c	z

R1=(A,B)

A	B
1	a
2	b
3	c

R2=(B,C)

B	C
a	x
b	y
c	z

A	B	C
1	a	x
2	b	y
3	c	z

$\prod_{A,B,C}(R1 \bowtie R2)$

The decomposition of R into R1 and R2 is lossless because $R=\prod_{R1}(R) \bowtie \prod_{R2}(R)$

Also we can check dependency is lossless or not by using the concept of functional dependency. Using it we can say the two relation R1 and R2 decomposed from relation R is lossless if at least one of the following condition is satisfied.

R1 ∩ R2→R1 or R1 ∩ R2→R2

Example:

Given R(A,B,C) F={A→B,B→C} can be decomposed in two different ways.

R1=(A,B),R2=(B,C)

- Lossless join decomposition: **R1 ∩ R2={ B } and B→BC**

R1=(A,B),R2=(A,C)

- Lossless join decomposition: **R1 ∩ R2={ A } and A→AB**

- **Lack of redundancy:** Redundancy is a problem of repetition of data in database. such problem should be avoided as far as possible.

- **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. $(F1 \cup F2 \cup \ldots\ldots FN)^+ = F^+$

Example:

Given R(A, B, C) and F={A → B, B → C}

The above relation R can be decomposed in two different ways.

1. R1 ={A,B} and R2 = {B, C} Here F1 ={A → B } and F2 ={B → C}

Here $(F1 \cup F2)^+ = F^+$, Therefore the decomposition is dependency preserving.

2 . R1 ={A,B} and R2 = {A, C} Here F1 ={A → B } and F2 ={A → C}

Here $(F1 \cup F2)^+ \neq F^+$, Therefore the decomposition is not dependency preserving.

Denormalization in DBMS

- ✓ Denormalization is a database optimization technique where we add redundant data in the database to get rid of the complex join operations.
- ✓ This is done to speed up database access speed.
- ✓ Denormalization is done after normalization for improving the performance of the database.
- ✓ The data from one table is included in another table to reduce the number of joins in the query and hence helps in speeding up the performance.

Example: Suppose after normalization we have two tables first, Employee table and second Department table. The Employee has the attributes as employee_id,employee_name,address and department_id.

Employee table

employee_id	employee_name	address	department_id
1	Ramesh	Kathmandu	1
2	Krishna	Pokhara	2
3	Ritu	Butwal	1
4	Roshan	Chitwan	3

The Department table is related to the Student table with department_id as the foreign key in the Student table.

Department table

department_id	department_name	HOD
1	Computer	Shiva
2	Civil	Gaurav
3	IT	Shankar

If we want the name of employees along with the department_name then we need to perform a join operation. The problem here is that if the table is large we need a lot of time to perform the join operations. So, we can add the data of deparment_name from department table to the Employee table and this will help in reducing the time that would have been used in join operation and thus optimize the database.

Employee table

employee_id	employee_name	address	department_id	department_name
1	Ramesh	Kathmandu	1	Computer
2	Krishna	Pokhara	2	Civil
3	Ritu	Butwal	1	Computer
4	Roshan	Chitwan	3	IT

when we use Denormalization?

- ✓ When the redundant data doesn't require to be updated frequently or doesn't update at all. In our example above, the redundant data is department_name and doesn't change frequently.
- ✓ When there is a need to join multiple tables frequently in order to get meaningful data.

Advantages of Denormalization

- ✓ Read Operations are faster as table joins are not required for most of the queries.
- ✓ Write query is easy to write to perform read, write, update operations on database.

Disadvantages of Denormalization

- ✓ Requires more storage as redundant data needs to be written in the tables.
- ✓ Data write operations are slower due to redundant data.
- ✓ Data inconsistencies are present due to redundant data.
- ✓ It requires extra effort to update the database. This is because when redundant data is present, it is important to update the data in all the places else data inconsistencies may arise.

Lossy and Lossless Decomposition

Lossless decomposition

- ✓ lossless decomposition of a relation ensures that no information is lost during decomposition.
- ✓ In lossless decomposition, the original information can be fully recovered from the decomposed relations using appropriate join operations.
- ✓ Lossless decomposition is a database design technique where the decomposition of a relation preserves all the information and dependencies present in the original relation.
- ✓ We could achieve this by decomposing the table into multiple tables while maintaining the necessary dependencies.

Let us consider the following relation

studentcourse table

student_id	student_name	address	course_id	course_name
1	Ram	Butwal	1	DBMS
2	Sita	Kathmandu	2	C++
3	Hari	Pokhara	2	C++
4	Gopal	Dharan	1	DBMS

In the given example, if we want to perform a lossless decomposition, we would need to ensure that the relationship between students and courses is preserved.

This table has redundant data as the course_Id and course_name are common for several students. Let's decompose this relation into two relations.

student table

student_id	student_name	address	course_id
1	Ram	Butwal	1
2	Sita	Kathmandu	2
3	Hari	Pokhara	2
4	Gopal	Dharan	1

course table

course_id	course_name
1	DBMS
2	C++

Now if we join student and course table then original information can be retrieved.

As we know,

Dependencies in original relation:

student_Id →student_name

course_Id →course_name

student_id→course_id

These dependencies are still present in the decomposed relations. Thus we can say that this decomposition is dependency preserving.

Lossy decomposition

- ✓ Lossy decomposition is a database design technique where the decomposition of a relation (table) results in losing some information or dependencies.
- ✓ In other words, the original information cannot be fully recovered from the decomposed relations.

Now decomposing original relation studentcourse into two tables.

student table

student_id	student_name	address
1	Ram	Butwal
2	Sita	Kathmandu
3	Hari	Pokhara
4	Gopal	Dharan

course table

course_id	course_name
1	DBMS
2	C++

In this decomposition, the relation of Student and Course is lost, there is no way to form the original relation from these two relations as the information that suggests who is attending which course is lost during decomposition.

Here, we have lost the association between the students and the courses they are enrolled in.

The relationship between students and courses cannot be inferred from these decomposed tables alone.

Here dependency in original relation is lost.

student_id→course_id

1. **Convert the following 2NF relation into 3NF(Name as Primary key) [PU:2012 fall]**

Name	Address	Phone	Salary	Post
Gill	Ktm	456789	20000	Engineer
Van	Bkt	654321	20000	Engineer
Robert	Ktm	456789	20000	Engineer
Brown	Bkt	654322	10000	Overseer
Albert	Ktm	454545	10000	Officer

solution:

Here, table is in 2NF because it is in 1NF and all attributes fully functionally dependent on the primary key Name.

Now, converting the table to 3NF,a table is in 3NF if and only if

- ✓ It is in 2NF
- ✓ There is no transitive functional dependency. i.e There should not be the case that non-prime attribute is determined by another non-prime attribute.

Here name is a primary key, and here showing that attributes post and salary has transitive functional dependency.

Name→Post

Post→Salary

Here salary is determined by the post even they both are non-prime attribute. So the table is not in 3NF. Now breaking the table.

Table 1

Name	Address	Phone	Post
Gill	Ktm	456789	Engineer
Van	Bkt	654321	Engineer
Robert	Ktm	456789	Engineer
Brown	Bkt	654322	Overseer
Albert	Ktm	454545	Officer

Table 2

Post	Salary
Engineer	20000
Overseer	10000
Officer	10000

Here ,

Table1(Name,Address,Phone,Post)
Table2(Post,Salary)
Both tables are in 3NF because it satisfies the above 3NF criteria.

2. How will you make a given table std_master with attributes: st_id, st_name, instructor_id, inst_name, course_id1, course_name1, course_id2, course_name2, course_id3, course_name3 in 1^{st} ,2^{nd} and 3^{rd} Normal forms, Write the steps. **[PU:2011 fall]**

Solution:

UNF

If the relation is in unnormalized form i.e it contains one or more repeating groups, or each row may contain multiple set of values for some columns.

1NF

To take the table into 1NF following criteria must be satisfied.

- ✓ If there are no repeating values in a column (All entries must be atomic)
- ✓ It must not have repeating columns.

Here, course_id1,course_id2,course_id3 are repeating columns, changing it to only course_id and similarly ,course_name1,course_name2,course_name3 into course_name only then,

the 1NF table relation schema will be

stdmaster(st_id,st_name,instructor_id,inst_name,course_id,course_name)

Now this is in 1NF.

2NF

A relation will be in 2NF if

- ✓ It is in 1NF and
- ✓ All non-prime attributes are fully functional dependent on the key attribute or primary key (There must not be any partial dependency)

But here partial dependency exists. All attributes are not fully functional dependent on primary key.

For example:

s_id →st_name

instructor_id →inst_name

course_id→course_name

Now, to remove these partial dependencies we can decompose the table as follows:

student(st_id,st_name)
instructor(instructor_id,inst_name)
course(course_id,course_name)
teaching_info (st_id, course_id, instructor_id)

3NF

For any relation to be in 3NF it must satisfied following properties.

- ✓ It is in 2NF.
- ✓ Every non-prime attribute is non-transitively dependent on the primary key. Which means there should not be case that non-prime attribute is functionally dependent on another non-prime attribute.

Here, all the above tables are already in 3NF.

3. Normalize the given relation upto 3NF

EMPLOYEE_ID	NAME	JOB_CODE	JOB	STATE_CODE	STATE_NAME
E001	Anish	J01	Sales	3	Bagmati
E001	Anish	J02	Marketing	3	Bagmati
E002	Bikash	J02	Marketing	5	Lumbini
E002	Bikash	J03	Analyst	5	Lumbini
E003	Ankit	J01	Sales	5	Lumbini

For 1 NF

- ✓ A relation is in 1NF if all the contained values should be atomic values.
- ✓ In above relation all entries are atomic, so it is already in 1NF.

For 2 NF

A relation will be in 2NF if

- ✓ It is in 1NF and
- ✓ All non-prime attributes are fully functional dependent on the key attribute or primary key (There must not be any partial dependency)

Here we considered {(EMPLOYEE_ID+ JOB_CODE)} as composite primary key . Due to these combination we can uniquely identify all the records of the table.

But, here partial dependency exists. All attributes are nor fully functional dependent on primary key.

For example:

EMPLOYEE_ID →NAME
EMPLOYEE_ID →STATE_CODE
JOB_CODE →JOB
Now, we can remove this partial dependency as follows.

Employee table

EMPLOYEE_ID	NAME	STATE_CODE	STATE_NAME
E001	Anish	3	Bagmati
E002	Bikash	5	Lumbini
E003	Ankit	5	Lumbini

Jobs table

JOB_CODE	JOB
J01	Sales
J02	Marketing
J03	Analyst

Employee_roles table

EMPLOYEE_ID	JOB_CODE
E001	J01
E001	J02
E002	J02
E002	J03
E003	J01

For 3NF

For any relation to be in 3NF it must satisfied following properties.
✓ It is in 2NF.
✓ Every non-prime attribute is non-transitively dependent on the primary key. Which means there should not be case that non-prime attribute is functionally dependent on another non-prime attribute.

Finding all functional dependencies in **Employee table**
EMPLOYEE_ID→STATE_CODE
STATE_CODE→STATE_NAME
Here we find one transitive functional dependency.

EMPLOYEE_ID→STATE_CODE →STATE_NAME

So, we have to remove this transitive dependency. Here, STATE_NAME which is non-prime attribute depends on another non-prime attribute STATE_CODE.

Employee table

EMPLOYEE_ID	NAME	STATE_CODE
E001	Anish	3
E002	Bikash	5
E003	Ankit	5

Jobs table

JOB_CODE	JOB
J01	Sales
J02	Marketing
J03	Analyst

Employee_roles table

EMPLOYEE_ID	JOB_CODE
E001	J01
E001	J02
E002	J02
E002	J03
E003	J01

States table

STATE_CODE	STATE_NAME
3	Bagmati
5	Lumbini

Assignment:

Consider the following bank database
Branch_schema=(branch_name,branch_city,assets)
Loan_schema=(loan_number,branch_name,amount)

Write an assertions for bank database to ensure that assets value for koteshwor branch is equal to the sum of all the amounts lent by the koteshwor branch.