Unit- 4: Database Normalization

Pitfalls in Relational Database Design

- The main goal of relational database is to create / find good collection of relational schemas. Such that database should allow us to store data / information without unnecessary redundancy and should also allow to retrieve information easily. Some goals are:
    - To avoid redundant data from database.
    - To ensure that relationships among attributes are represented.
    - To facilitate the checking of updates for the violation of database integrity constraints.
- A bad relational database design may lead to:

    o Repetition of information.

    – That is, it leads data redundancy in database, so obviously it requires much storage space .

    - Inability to represent certain information.

Example 1: Consider the relational schema

Branch_loan = (beranch_name, branch_city, assets, customer_name, loan_no, amount)

branch_name	Branch_city	assets	Customer_ name	Loan_no	amount
kathmandu	baneshwor	25000	rohan	L - 15	3000
Lalitpur	patan	19000	mohan	L - 17	5000
Kathmandu	baneshwor	25000	raju	L - 19	10000
Pokhara	Prithibi nagar	17000	manoj	L - 10	7000
Lalitpur	patan	19000	swikar	L - 30	9000

Redundancy:

• Data for branch_name, branch_city, assets are repeated for each loan that provides by bank.

• Storing information several times leads waste of storage space / time.

• Data redundancy leads problem in Insertion, deletion and update.

Insertion problem

- We cannot store information about a branch without loan information, we can use null values for loan information, but they are difficult to handle.

Deletion problem

- In this example, if we delete the information of Manoj (i.e. DELETE FROM branch_loan WHERE customer_name = 'Manoj';), we cannot obtain the information of pokhara branch (i.e. we don't know branch city of pokhara, total asset of pokhara branch etc.

Update problem

- Since data are duplicated so multiple copies of same fact need to update while updating one. It increases the possibility of data inconsistency. When we made update in one copy there is possibility that only some of the multiple copies are update but not all, which lead data/database in inconsistent state.

Introduction to Anomalies and Types

- Database anomalies refer to errors and inconsistencies in the database.
- It creates problems while performing the operations like insertion, deletion, and modification.
- An anomaly is simply an error or inconsistency in the database. An anomaly may occur in the database if insertion, deletion, modification etc are not done properly. It creates inconsistency and unreliability in the database.
- It is necessary to remove anomalies for error free processing in relations.
- The anomalies can be eliminated by redefining a relation into two or more relations.

Types of anomalies

- There are three types of anomalies that may occur in relational database
  1. **Insertion Anomalies:** occur during the insertion of a record. It is the inability to add data to the database due to absence of other data.

- The insertion anomaly occurs when new record is inserted in relation. In this anomaly user cannot insert a fact about an entity until he has an additional fact about another entity

2. **Deletion Anomalies:** occur during the deletion of record. It is unintended loss of data due to deletion of other data
    - The deletion anomaly occurs when a record is deleted from the relation. In this anomaly, the deletion of facts about an entity automatically deletes the fact of another entity
3. **Updation or Modification Anomalies:** occur during the updating of a record. It is unintended modification of related data due to updating of a particular data.
    - The modification anomaly occurs when the record is updated in the relation. In this anomaly, the modification in the value of specific attribute requires modification in all records in which that value occurs.

Example

Student_Table

StudentNum	**CourseNum**	**Student Name**	**Address**	**Course**
S21	9201	Jones	Edinburgh	Accounts
S21	9267	Jones	Edinburgh	Accounts
S24	9267	Smith	Glasgow	physics
S30	9201	Richards	Manchester	Computing
S30	9322	Richards	Manchester	Maths

Insertion Anomalies - An Insertion Anomaly occurs when certain attributes cannot be inserted into the database without the presence of other attributes. For example - we can't add a new course unless we have at least one student enrolled on the course

Deletion Anomalies - A Deletion Anomaly exists when certain attributes are lost because of the deletion of other attributes. For example, consider what happens if Student S30 is the last student to leave the course - All information about the course is lost

Modification/ Updation Anomalies: An Updation Anomaly exists when one or more instances of duplicated data is updated, but not all. For example, consider Jones moving address - you need to update all instances of Jones's address.

Definition and Importance of Normalization

- **Normalization** is the process of minimizing redundancy from a relation or set of relations. Redundancy in relation may cause insertion, deletion and updation anomalies. So, it helps to minimize the redundancy in relations. **Normal forms** are used to eliminate or reduce redundancy in database tables.
- Normalization divides the larger table into the smaller table and links them using relationship.
- The normal form is used to reduce redundancy from the database table.
- Database Normalization is a technique of organizing the data in the database. Normalization is a systematic approach of decomposing tables to eliminate data redundancy (repetition) and undesirable characteristics like Insertion, Update and Deletion Anomalies.

Importance of database normalization

——Improves system performance and accuracy.

——Supports integrity and consistency of the data.

——Saves space, minimize redundancy and eliminates anomalies.

——The goal of normalization is to create a set of relational tables that are free of redundant data and that can be consistently and correctly modified.

Functional Dependencies

If an attribute of a relation uniquely determines another attribute, it is called functional dependency.

- Functional dependencies are constraints on the set of legal relations. It defines attributes of relation, how they are related to each other.
- It determines unique value for a certain set of attributes to the value for another set of attributes that is functional dependency is a generalization of the notation of key.
- Functional dependencies are interrelationship among attributes of a relation.

Definition:

- 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** of the functional dependency (FD) and functional dependency denoted by X→ Y.
- The functional dependency X → Y holds if whenever two tuples have the same value for X, they must have the same value for Y. For any two tuples t1 and t2 in any relation instance r(R): If t1[X] = t2[X], then t1[Y] = t2[Y]
- In summary, A functional dependency is a constraint that specifies the relationship between two sets of attributes where one set can accurately determine the value of other sets. It is denoted as X → Y, where X is a set of attributes that is capable of determining the value of Y. The attribute set on the left side of the arrow, X is called **Determinant**, while on the right side, Y is called the **Dependent**.
- Example1: in the following relation

  Customer (Cid, Cname, Age, salary)

  - ✓ Cid → Cname

  It is true because Cid uniquely determine the customer name even in the case of duplicate Cname.

  - ✓ Cname→Cid

  It is false because the name of the customer may be same for different Cid. So Cname not uniquely determine the customer.

  - ✓ Cid → Age [True]
  - ✓ Cid →salary [True]
  - ✓ Age→cid [False]

A	B
1	5
1	7
4	9

Example 2: Consider R (A, B) with the following instance of R.

On this instance, A→B does NOT hold, but B →A does hold.

Example 3: consider a relation supplier : **Supplier(supplier_id,sname,status,city)**

S_id → {sname, status, city}

Here, sname, status and city are functionally dependent on supplier_id. Meaning is that each supplier id uniquely determines the value of attributes supplier name,supplier status and city.

This can be express by

supplier_id→ sname

supplier_id→ status

supplier_id→city

Types of functional dependency

There are many types of functional dependencies, depending on several criteria. Some of them are as follows:

Fully functional dependency

- ✓ For a given relation schema R, FD X→Y, Y is said to be fully functionally dependent on X if there is no Z (where Z is a proper subset of X) such that Z→Y.

  Example: Let us consider relational schema R=(A,B,C,D,E,H) with the FDs

  F={A→BC,CD→E,C→E,CD→E,CD→AH, ABH→BD,DH→BC}

  Here, the FD A→BC is left reduced, so clearly, BC is fully functionally dependent on A (because there is no possible proper subset of only element A)

  Here, the FDs CD→E, C→E where E is functionally dependent on CD and again E is functionally dependent on subset of CD. (i.e. C→E). Hence E is not fully functionally dependent on CD.

Example: Consider a relation StudentCourseInfo

StudentCourseInfo(<u>StudentId,</u> <u>CourseId</u>, CourseName, StudentName, Marks) with the following FDs

{ StudentId → StudentName,
StudentId, CoursetId→ Marks,
CoursetId→ CourseName
StudentId, CoursetId→ StudentName
StudentId, CoursetId→ CourseName
}

Here , StudentId, CoursetId→ Marks

So, Marks is fully functionally dependent on {StudentId, CoursetId}

Partial functional dependency

For a given relation schema R with set of functional dependency F on attribute of R. Let K as a candidiate key in R. if X is a proper subset of K and X→A then A is said to be partially dependent on K.

Example: Consider a relation schema 'StudentCourseInfo'

StudentCourseInfo(StudentId, CourseId, CourseName, StudentName,Marks) with the following FDs

{ StudentId → StudentName,

StudentId, CoursetId→ Marks,

CoursetId→ CourseName

StudentId, CoursetId→ StudentName

StudentId, CoursetId→ CourseName

}

Here *{ StudentId, CourseId }* is a candidate key.

We can see,

StudentId, CoursetId→ StudentName (Candidate key determines *StudentName)*

StudentId → StudentName(Proper subset of Candidate key determines *StudentName*)

Hence StudentName is partially dependent on *{StudentId, CoursetId}*

Trivial and Non-trivial dependencies

- A functional dependency X→ Y is said to be **trivial dependency** if Y is subset of X (not necessarily a proper subset).

  For example: Consider a table with two columns Student_id and Student_Name.

  {Student_Id, Student_Name} → Student_Id

  It is a trivial functional dependency as Student_Id is a subset of {Student_Id, Student_Name}.

  That makes sense because if we know the values of Student_Id and Student_Name then the value of Student_Id can be uniquely determined.

  Also, Student_Id → Student_Id & Student_Name → Student_Name are trivial dependencies too.

- A functional dependency X→ Y is said to be **non- trivial dependency** if Y is not a subset of X.

  For example:
  An **employee** table with three attributes: **emp_id, emp_name, emp_address**.
  The following functional dependencies are non-trivial:
  emp_id → emp_name (emp_name is not a subset of emp_id)
  emp_id → emp_address (emp_address is not a subset of emp_id)

  On the other hand, the following dependencies are **trivial:**
  {emp_id, emp_name} → emp_name [emp_name is a subset of {emp_id, emp_name}]

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 hold true:

- X→ Y
- Y does not → X
- Y→ Z

A transitive dependency can only occur in a relation of three or more attributes

Example1:

Let us consider relational schema 'Vendor'

Vender(VenderId,Name,AccountNumber,BankCode,BankName)

VenderId → BankCode

BankCode→ Bank Name

So BankName is transitively dependent on the key **(VenderId)**

Example2:

Let R=(A,B,C,D,E) and FDs F={AB→C,B→D,C→E}

Here AB acts a candidate key and E is transitively dependent on the key AB, Hence AB→C→E).

Inference Rules for FDs

- The Armstrong's axioms are the basic of inference rules.
- Armstrong's axioms (Developed by William W. Armstrong) are the set of inference rules used to infer all the functional dependencies on a relational database.
- Inference rules can be applied to a set of FD to derive other FD.
- Using the inference rules we can derive additional functional dependency from the initial set.
- Following are the inference rules (Armstrong's axioms)
    - **Reflexive rule :** If Y subset-of X, then X → Y
    - **Augmentation rule :** If X →Y, then XZ → YZ
    - **Transitive rule:** If X → Y and Y → Z, then X → Z
    - **Decomposition rule :** If X → YZ, then X → Y and X → Z
    - **Union rule :** If X →Y and X →Z, then X → YZ
    - **Pseudo- transitivity rule :** If X →Y and WY →Z, then WX → Z
- Reflexive, Augmentation and Transitive rules are considered as primary rules and others are considered as secondary rules

Multivalued Dependency

- Multivalued dependency occurs when two attributes in a table are independent of each other but, both depend on a third attribute.
- A multivalued dependency consists of at least two attributes that are dependent on a third attribute that's why it always requires at least three attributes.
- Multivalued dependency is represented by the double arrow sign (→→).

Example:

The relation student consists of three attributes student_id, Name, Course.

student_id	Name	Course
101	Ankit	Python
102	Kapil	Java
103	Krishna	DotNet
101	Ankit	JAVA
105	Akash	PHP

- In the above relation, Name and Course are two independent attributes in itself, but both are dependent on student_id.
- In this case, these two attributes are multivalued dependent on student_id. Following are the representation of these dependencies:

student_id →→ Name

student_id →→ Course

Normalization

- Database Normalization is a technique of organizing the data in the database. Normalization is a systematic approach of decomposing tables to eliminate data redundancy (repetition) and undesirable characteristics like Insertion, Update and Deletion Anomalies.

Properties of relation after normalization

- No data value should be duplicate in different row or tuple.
- A value must be specified for every attribute in a tuple.
- Important information should not be accidentally lost.
- When new data / record are inserted to relation, other relation in database should not be affected

Key and non-key attribute (prime and non-prime attribute)

- An attribute A in relation R is said to be key attribute if A is any part of candidate key; otherwise A is called non key attribute.
- Example: Let R=(A,B,C,D,E) and FDs F={AB →C,B →D,C→E}
- Here AB is composite primary key (composite candidate key) so attribute A and B are key attributes and attributes C, D and E are non-key attributes.

Attributes of the relation which exist in at least one of the possible candidate keys, are called prime or key attributes.

Attributes of the relation which does not exist in any of the possible candidate keys of the relation, such attributes are called non-prime or non-key attributes.

Normal Forms

- Certain rules in database management system design have been developed to better organize tables and minimize anomalies. The stage at which a table is organized is known as its normal form (or a stage of normalization).
- There are different normal forms which are as follows:
    - ❖ First Normal Form (1NF)
    - ❖ Second Normal Form (2NF)
    - ❖ Third Normal Form (3NF)
    - ❖ Fourth Normal Form (4NF)
    - ❖ Fifth Normal Form (5NF)
- A relation is said to in particular normal form if it satisfy the set of this particular normal form's constraints. In practical, third normal form is sufficient to design reliable database system.
- Normalization theory is based on:
    - Functional dependencies
    - Multi valued dependencies.

First Normal Form (1 NF)

A relation is said to be in 1NF if and only if all domains of the relation contains only atomic (indivisible) values. More simply a relation is in 1 NF if it does not have multi- valued attributes, composite attributes and their combinations. It states that the domain of an attribute must include only atomic (simple, indivisible) values.

For a table to be in the First Normal Form, it should follow the following rules:

- ✓ It should only have single (atomic) valued attributes/columns.
- ✓ Values stored in a column should be of the same domain
- ✓ All the columns in a table should have unique names.
- ✓ And the order in which data is stored, does not matter.

Example: let's take an un-normalized relation containing multivalued attributes as,

Here is our table, with some sample data added to it.

roll_no	name	subject
101	Akon	OS, CN
103	Ckon	Java
102	Bkon	C, C++

Here, out of the 3 different students in our table, 2 have opted for more than 1 subject. And we have stored the subject names in a single column. But as per the 1st Normal form each column must contain atomic value.

Here is our updated table and it now satisfies the First Normal Form.

roll_no	name	subject
101	Akon	OS
101	Akon	CN
103	Ckon	Java
102	Bkon	C
102	Bkon	C++

By doing so, although a few values are getting repeated but values for the subject column are now atomic for each record/row.

Using the First Normal Form, data redundancy increases, as there will be many columns with same data in multiple rows but each row as a whole will be unique.

Second Normal Form (2NF)

For a table to be in the Second Normal Form, it must satisfy two conditions:

- The table should be in the First Normal Form.
- There should be no Partial Dependency.(i.e. Every non-prime(non-key) attributes are fullyfunctionally dependent on the primary key of the relation)

If a table has some attributes which is partially dependent on the primary key of that table then it is not in 2NF. Example: let's take a relation in 1 NF but not in 2 NF as

Employee-Department

Emp-Id	Emp-Name	Emp-Salary	Dept-No	Dept-Name
1	Ram	40000	D1	BBA
1	Ram	40000	D2	CSIT
2	Bindu	30000	D3	BBS
3	Arjun	60000	D2	CSIT

In the above relation **{Emp-Id, Dept-No}** is the primary key. **Emp-Name**, **Emp-Salary** and **Dept-Name** all depend upon **{Emp-Id, Dept-No}.**

Again **Emp-Id→Emp-Name, Emp-Id→Emp-Salary** and **Dept-No→Dept-Name**, thus there also occur **partial dependency.** Due to which this relation is not in 2 NF.

Employee

Emp-Id	Emp-Name	Emp-Salary
1	Ram	40000
2	Bindu	30000
3	Arjun	60000

Emp-Dept

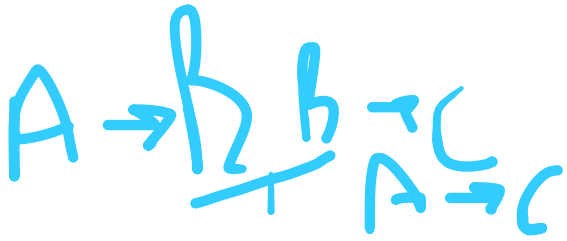
Emp-Id	Dept-No
1	D1
1	D2
2	D3
3	D2

Dept

Dept-No	Dept-Name
D1	BBA
D2	CSIT
D3	BBS

- ✓ For a table to be in the Second Normal form, it should be in the First Normal form and it should not have Partial Dependency.
- ✓ Partial Dependency exists, when for a composite primary key, any attribute in the table depends only on a part of the primary key and not on the complete primary key.

- ✓ To remove Partial dependency, we can divide the table, remove the attribute which is causing partial dependency, and move it to some other table where it fits in well.



Third Normal Form (3NF)

- For a table to be in the third normal form,
  - ✓ It should be in the Second Normal form.
  - ✓ And it should not have Transitive Dependency.(i.e. every non-prime attribute is non-transitively dependent on the primary key)
- If a table contains transitive dependency, then it is not in 3NF, and the table must be split to bring it into 3NF.

S-Id	S-Name	Age	Gender	Hostel-Name
1	Laxmi	21	Female	White house
2	Binita	22	Female	White house
3	Rajesh	32	Male	Red house
4	Aayan	21	Male	Red house

Here **S-Id→ Gender**, **Gender → Hostel** and **Gender → Hostel** This third dependency was not originally specified but we have derived it. **The derived dependency is called a transitive dependency.** In such a case the relation should be broken into two relations as

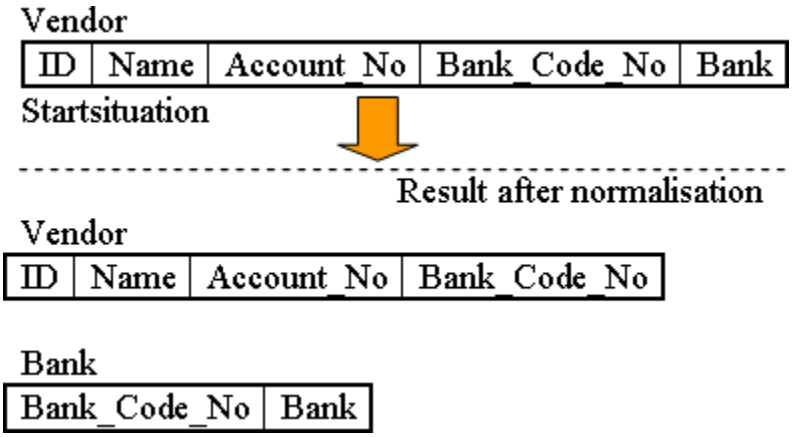
Student

S-Id	**S-Name**	**Age**	**Gender**
1	Laxmi	21	Female
2	Binita	22	Female
3	Rajesh	32	Male
4	Aayan	21	Male

Hostel

Gender	**Hostel-Name**
Female	White house
Male	Red house

Another example



Boyce Codd normal form (BCNF)

- BCNF is the advance version of 3NF. It is stricter than 3NF.
- A table is in BCNF if every functional dependency $X \rightarrow Y$, X is the super key of the table.
- For BCNF, the table should be in 3NF, and for every FD, LHS is super key.
- The BCNF imposes stronger constraints on the type of FDs allowed in a relation.
    - It allows only those non trivial functional dependencies whose determinants are super keys of relation. But in case of 3NF, it allow non trivial FDs whose determinant is not a candidate superkey if right-hand side of FDs contained a candidate key.
    - That is BCNF enforce more stronger constraints than 3NF.

Example: (not in BCNF)

(Suppose there is a college where one faculty teach in more than one department.)

FacultyID	FacultyAddress	CourseID	CourseName
101	Lalitpur	C1	BCA
101	Lalitpur	C2	BScCSIT
102	Bhaktapur	C1	BCA
102	Bhaktapur	C2	BScCSIT

In the above relation functional dependencies are as follows:

FacultyID → FacultyAddress

CourseID → CourseName

Candidate key: { FacultyID, CourseID }

Above relation is not in BCNF as neither FacultyID nor CourseID alone are keys.To make the relation in BCNF, we can break the table into three parts like this:

FacultyAddress

FacultyID	FacultyAddress
101	Lalitpur
102	Bhaktapur

Course

CourseId	CourseName
C1	MCA
C2	MBA

FacultyCourse

FacultyID	CourseId
101	C1
101	C2
102	C1
102	C2

Fourth Normal Form (4NF)

A relation is in fourth normal form (4NF) if it satisfies the following conditions:

- A relation is in BCNF.
- And, there is no multivalued dependency exists in the relation.

Multivalued Dependency exists in a relation when two attributes depend on the third attribute but independent to each other.

Example:

The relation student consists of three attributes: student_id, Name, and Course.

student_id	Name	Course
101	Ankit	Python
102	Kapil	Java
103	Krishna	DotNet
101	Ankit	JAVA
105	Akash	PHP

In the above relation, Name and Course are two independent attributes and both are dependent on student_id.

In this case, these two attributes can be called as multivalued dependent on student_id. Following are the representation of these dependencies:

Student_id →→ Name

Student_id →→ Course

So, to make the above relation into the fourth normal form (4NF), decompose it into two tables:

Student_name

student_id	Name
101	Ankit
102	Kapil
103	Krishna
105	Akash

Student_course

student_id	Course
101	Python
102	Java
103	DotNet
101	JAVA
105	PHP