Module – 4

Database Programming

Introduction of Database Normalization:

- **Normalization** is an important process in database design that helps to improve the database's efficiency, consistency, and accuracy. It makes it easier to manage and maintain the data and ensures that the database is adaptable to changing business needs.
- **Database normalization** is the process of organizing the attributes of the database to reduce or eliminate data redundancy (having the same data but at different places).

Features of Database Normalization:

- **Elimination of Data Redundancy:** Data redundancy refers to the repetition of data in different parts of the database. Normalization helps in reducing or eliminating this redundancy, which can improve the efficiency and consistency of the database.
- **Ensuring Data Consistency:** Normalization helps in ensuring that the data in the database is consistent and accurate. By eliminating redundancy, normalization helps in preventing inconsistencies and contradictions that can arise due to different versions of the same data.
- **Simplification of Data Management:** Normalization simplifies the process of managing data in a database. By breaking down a complex data structure into simpler tables, normalization makes it easier to manage the data, update it, and retrieve it.
- **Improved Database Design:** Normalization helps in improving the overall design of the database. By organizing the data in a structured and systematic way, normalization makes it easier to design and maintain the database. It also makes the database more flexible and adaptable to changing business needs.
- **Avoiding Update Anomalies:** Normalization helps in avoiding update anomalies, which can occur when updating a single record in a table affects multiple records in other tables. Normalization ensures that each table contains only one type of data and that the relationships between the tables are clearly defined, which helps in avoiding such anomalies.
- **Standardization:** Normalization helps in standardizing the data in the database. By organizing the data into tables and defining relationships between them, normalization helps in ensuring that the data is stored in a consistent and uniform manner.

Nonloss/Lossless Decomposition:

- Lossless join decomposition is a decomposition of a relation R into relations R1, and R2 such that if we perform a natural join of relation R1 and R2, it will return the original relation R. This is effective in removing redundancy from databases while preserving the original data.
- **Example of Lossless Decomposition**
    - Employee (Employee_Id, Ename, Salary, Department_Id, Dname)
- Can be decomposed using lossless decomposition as,
    - Employee_desc (Employee_Id, Ename, Salary, Department_Id)
    - Department_desc (Department_Id, Dname)

What is Functional Dependency and its Types?

- Functional Dependency is a constraint between two sets of attributes about a database. A function dependency A -> B means for all instances of a particular value of A, there is the same value of B. For example , A -> B is true, but B -> A is not true as there are different values of A for B.
- roll_no → { name, dept_name, dept_building },→ Here, roll_no can determine values of fields name, dept_name and dept_building, hence a valid Functional dependency

Example:

roll_no	name	dept_name	dept_building
42	abc	CO	A4
43	pqr	IT	A3
44	xyz	CO	A4
45	xyz	IT	A3
46	mno	EC	B2
47	jkl	ME	B2

Types of Functional Dependencies in DBMS:

1. Trivial functional dependency
2. Non-Trivial functional dependency
3. Multivalued functional dependency
4. Transitive functional dependency

1. **Trivial Functional Dependency:**
   - In **Trivial Functional Dependency**, a dependent is always a subset of the determinant. i.e. If **X → Y** and **Y is the subset of X**, then it is called trivial functional dependency.

Example:

roll_no	name	age
42	abc	17
43	pqr	18
44	xyz	18

Here, **{roll_no, name} → name** is a trivial functional dependency, since the dependent **name** is a subset of determinant set **{roll_no, name}**. Similarly, **roll_no → roll_no** is also an example of trivial functional dependency.

2. **Non-trivial Functional Dependency:**
   - In **Non-trivial functional dependency**, the dependent is strictly not a subset of the determinant. i.e. If **X → Y** and **Y is not a subset of X**, then it is called Non-trivial functional dependency.

Example:

roll_no	name	age
42	abc	17
43	pqr	18
44	xyz	18

Here, **roll_no → name** is a non-trivial functional dependency, since the dependent **name** is **not a subset of** determinant **roll_no**. Similarly, **{roll_no, name} → age** is also a non-trivial functional dependency, since **age** is **not a subset of {roll_no, name}**

3. **Multivalued Functional Dependency:**
   - In **Multivalued functional dependency**, entities of the dependent set are **not dependent on each other.** i.e. If **a → {b, c}** and there exists **no functional dependency** between **b and c**, then it is called a **multivalued functional dependency.**

For example,

roll_no	name	age
42	abc	17
43	pqr	18
44	xyz	18
45	abc	19

Here, **roll_no → {name, age}** is a multivalued functional dependency, since the dependents **name** & **age** are **not dependent** on each other(i.e. **name → age** or **age → name doesn't exist !**)

4. **Transitive Functional Dependency:**
   - In transitive functional dependency, dependent is indirectly dependent on determinant. i.e. If **a → b** & **b → c**, then according to axiom of transitivity, **a → c**. This is a **transitive functional dependency.**

For example,

enrol_no	name	dept	building_no
42	abc	CO	4
43	pqr	EC	2
44	xyz	IT	1
45	abc	EC	2

Here, **enrol_no → dept** and **dept → building_no.** Hence, according to the axiom of transitivity, **enrol_no → building_no** is a valid functional dependency. This is an indirect functional dependency, hence called Transitive functional dependency.

Advantages of Functional Dependencies:

- Functional dependencies having numerous applications in the field of database management system. Here are some applications listed below:

1. Data Normalization

Data normalization is the process of organizing data in a database in order to minimize redundancy and increase data integrity. It plays an important part in data

normalization. With the help of functional dependencies we are able to identify the primary key, candidate key in a table which in turns helps in normalization.

2. Query Optimization

With the help of functional dependencies we are able to decide the connectivity between the tables and the necessary attributes need to be projected to retrieve the required data from the tables. This helps in query optimization and improves performance.

3. Consistency of Data

Functional dependencies ensures the consistency of the data by removing any redundancies or inconsistencies that may exist in the data. Functional dependency ensures that the changes made in one attribute does not affect inconsistency in another set of attributes thus it maintains the consistency of the data in database.

4. Data Quality Improvement

Functional dependencies ensure that the data in the database to be accurate, complete and updated. This helps to improve the overall quality of the data, as well as it eliminates errors and inaccuracies that might occur during data analysis and decision making, thus functional dependency helps in improving the quality of data in database.

Normal Forms in DBMS:

- **Normalization** is the process of minimizing **redundancy** from a relation or set of relations. Redundancy in relation may cause insertion, deletion, and update anomalies. So, it helps to minimize the redundancy in relations. **Normal forms** are used to eliminate or reduce redundancy in database tables.
- In database management systems (DBMS), normal forms are a series of guidelines that help to ensure that the design of a database is efficient, organized, and free from data anomalies. There are several levels of normalization, each with its own set of guidelines, known as normal forms.

First Normal Form:

- If a relation contains a composite or multi-valued attribute, it violates the first normal form, or the relation is in the first normal form if it does not contain any **composite** or **multi-valued attribute**. A relation is in first normal form if every attribute in that relation is single-valued attribute.
- A table is in 1 NF if:
- There are only Single Valued Attributes.
- Attribute Domain does not change.

- There is a unique name for every Attribute/Column.
- The order in which data is stored does not matter.
- **Example 1 –** Relation STUDENT in table 1 is not in 1NF because of multi-valued attribute STUD_PHONE. Its decomposition into 1NF has been shown in table 2.

STUD_NO	STUD_NAME	STUD_PHONE	STUD_STATE	STUD_COUNTRY
1	RAM	9716271721, 9871717178	HARYANA	INDIA
2	RAM	9898297281	PUNJAB	INDIA
3	SURESH		PUNJAB	INDIA

Table 1

Conversion to first normal form

STUD_NO	STUD_NAME	STUD_PHONE	STUD_STATE	STUD_COUNTRY
1	RAM	9716271721	HARYANA	INDIA
1	RAM	9871717178	HARYANA	INDIA
2	RAM	9898297281	PUNJAB	INDIA
3	SURESH		PUNJAB	INDIA

Table 2

Second Normal Form:

- A relation is in 2NF if it is in 1NF and any non-prime attribute (attributes which are not part of any candidate key) is not partially dependent on any proper subset of any candidate key of the table. In other words, we can say that, every non-prime attribute must be fully dependent on each candidate key.
- A functional dependency X->Y (where X and Y are set of attributes) is said to be in **partial dependency**, if Y can be determined by any proper subset of X.
- However, in 2NF it is possible for a prime attribute to be partially dependent on any candidate key, but every non-prime attribute must be fully dependent(or not partially dependent) on each candidate key of the table.
- *In other words, A relation that is in First Normal Form and every non-primary-key attribute is fully functionally dependent on the primary key, then the relation is in Second Normal Form (2NF).*
- **Example 1 –** Consider table-3 as following below.

STUD_NO	COURSE_NO	COURSE_FEE
1	C1	1000
2	C2	1500
1	C4	2000
4	C3	1000
4	C1	1000
2	C5	2000

- {Note that, there are many courses having the same course fee} Here, COURSE_FEE cannot alone decide the value of COURSE_NO or STUD_NO; COURSE_FEE together with STUD_NO cannot decide the value of COURSE_NO; COURSE_FEE together with COURSE_NO cannot decide the value of STUD_NO; Hence, COURSE_FEE would be a non-prime attribute, as it does not belong to the one only candidate key {STUD_NO, COURSE_NO} ; But, COURSE_NO -> COURSE_FEE, i.e., COURSE_FEE is dependent on COURSE_NO, which is a proper subset of the candidate key. Non-prime attribute COURSE_FEE is dependent on a proper subset of the candidate key, which is a partial dependency and so this relation is not in 2NF. To convert the above relation to 2NF, we need to split the table into two tables such as : Table 1: STUD_NO, COURSE_NO Table 2: COURSE_NO, COURSE_FEE

Table 1

STUD_NO	COURSE_NO
1	C1
2	C2
1	C4
4	C3
4	C1

Table 2

COURSE_NO	COURSE_FEE
C1	1000
C2	1500
C3	1000
C4	2000
C5	2000

- **NOTE:** 2NF tries to reduce the redundant data getting stored in memory. For instance, if there are 100 students taking C1 course, we don't need to store its Fee as 1000 for all the 100 records, instead, once we can store it in the second table as the course fee for C1 is 1000.
- **Example-2: Consider following functional dependencies in relation R (A, B, C, D)**
- AB -> C [A and B together determine C]
  BC -> D [B and C together determine D]
- First, we can check if there are any partial dependencies. A partial dependency occurs when a non-prime attribute (not part of any candidate key) depends on only part of a candidate key.
- The candidate keys for relation R can be determined by finding the closure of each attribute:
- AB determines every keys.

Third Normal Form (3NF):

- A relation is in the third normal form, if there is no transitive dependency for non-prime attributes as well as it is in the second normal form. A relation is in 3NF if at least one of the following conditions holds in every non-trivial function dependency X –> Y.

- X is a super key.
- Y is a prime attribute (each element of Y is part of some candidate key).
- *In other words, A relation that is in First and Second Normal Form and in which no non-primary-key attribute is transitively dependent on the primary key, then it is in Third Normal Form (3NF).*
- **Example 1: In relation STUDENT given in Table 4,**

STUD_NO	STUD_NAME	STUD_STATE	STUD_COUNTRY	STUD_AGE
1	RAM	HARYANA	INDIA	20
2	RAM	PUNJAB	INDIA	19
3	SURESH	PUNJAB	INDIA	21

Table 4

FD set: {STUD_NO -> STUD_NAME, STUD_NO -> STUD_STATE, STUD_STATE -> STUD_COUNTRY, STUD_NO -> STUD_AGE} Candidate Key: {STUD_NO} For this relation in table 4, STUD_NO -> STUD_STATE and STUD_STATE -> STUD_COUNTRY are true. So STUD_COUNTRY is transitively dependent on STUD_NO. It violates the third normal form. To convert it in third normal form, we will decompose the relation STUDENT (STUD_NO, STUD_NAME, STUD_PHONE, STUD_STATE, STUD_COUNTRY_STUD_AGE) as:

```
STUDENT (STUD_NO, STUD_NAME, STUD_PHONE, STUD_STATE, STUD_AGE)
STATE_COUNTRY (STATE, COUNTRY)
```

- **Example 2: Consider Relation R(A, B, C, D, E)**
- A -> BC,
  CD -> E,
  B -> D,
  E -> A
- All possible candidate keys in above relation are {A, E, CD, BC} All attribute are on right sides of all functional dependencies are prime.

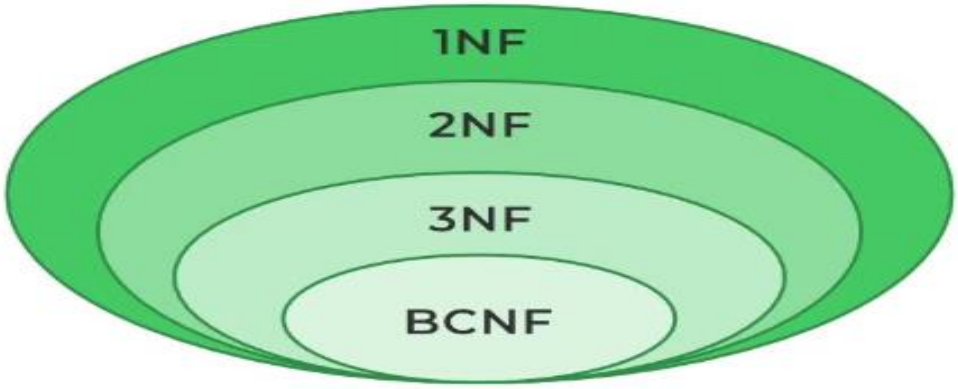
What is Transitive Dependency?

- A transitive dependency occurs when a non-key attribute depends on the another non-key attribute rather than directly on the primary key. For instance, consider a table with the attributes (A, B, C) where A is the primary key and B and C are non-key

attributes. If B determines C then C is transitively dependent on the A through B. This can lead to data anomalies and redundancy which 3NF aims to eliminate by the ensuring that all non-key attributes depend only on the primary key.

Boyce-Codd Normal Form (BCNF):

- Boyce–Codd Normal Form (BCNF) is based on functional dependencies that take into account all candidate keys in a relation; however, BCNF also has additional constraints compared with the general definition of 3NF.
- **Rules for BCNF**
- ***Rule 1:*** *The table should be in the 3rd Normal Form.*
- ***Rule 2:*** *X should be a superkey for every functional dependency (FD) X−>Y in a given relation.*
- **Note:** To test whether a relation is in BCNF, we identify all the determinants and make sure that they are candidate keys.



BCNF in DBMS

- **Example 1:** Let us consider the student database, in which data of the student are mentioned.

Stu_ID	Stu_Branch	Stu_Course	Branch_Number	Stu_Course_No
101	Computer Science & Engineering	DBMS	B_001	201
101	Computer Science & Engineering	Computer Networks	B_001	202
102	Electronics & Communication Engineering	VLSI Technology	B_003	401
102	Electronics & Communication Engineering	Mobile Communication	B_003	402

- Functional Dependency of the above is as mentioned:
- Stu_ID –> Stu_Branch
  Stu_Course –> {Branch_Number, Stu_Course_No}
- Candidate Keys of the above table are: **{Stu_ID, Stu_Course}**
- **Why this Table is Not in BCNF?**
- The table present above is not in BCNF, because as we can see that neither Stu_ID nor Stu_Course is a Super Key. As the rules mentioned above clearly tell that for a table to be in BCNF, it must follow the property that for functional dependency X–>Y, X must be in Super Key and here this property fails, that's why this table is not in BCNF.
- **How to Satisfy BCNF?**
- For satisfying this table in BCNF, we have to decompose it into further tables. Here is the full procedure through which we transform this table into BCNF. Let us first divide this main table into two tables **Stu_Branch** and **Stu_Course** Table.
- **Stu_Branch Table:**

Stu_ID	Stu_Branch
101	Computer Science & Engineering
102	Electronics & Communication Engineering

- Candidate Key for this table: **Stu_ID**.
- **Stu_Course Table:**

Stu_Course	Branch_Number	Stu_Course_No
DBMS	B_001	201
Computer Networks	B_001	202
VLSI Technology	B_003	401
Mobile Communication	B_003	402

- Candidate Key for this table: **Stu_Course**.
- **Stu_ID to Stu_Course_No Table:**

Stu_ID	Stu_Course_No
101	201
101	202
102	401
102	402

- Candidate Key for this table: **{Stu_ID, Stu_Course_No}.**
- After decomposing into further tables, now it is in BCNF, as it is passing the condition of Super Key, that in functional dependency X–>Y, X is a **Super Key.**

Introduction of 4th and 5th Normal Form in DBMS:

- Two of the highest levels of database normalization are the fourth normal form (4NF) and the fifth normal form (5NF). Multivalued dependencies are handled by 4NF, whereas join dependencies are handled by 5NF.
- If two or more independent relations are kept in a single relation or we can say multivalue dependency occurs when the presence of one or more rows in a table implies the presence of one or more other rows in that same table. Put another way, two attributes (or columns) in a table are independent of one another, but both depend on a third attribute. A **multivalued dependency** always requires at least

three attributes because it consists of at least two attributes that are dependent on a third.

- For a dependency A -> B, if for a single value of A, multiple values of B exist, then the table may have a multi-valued dependency. The table should have at least 3 attributes and B and C should be independent for A ->> B multivalued dependency.

Fourth Normal Form (4NF):

- The Fourth Normal Form (4NF) is a level of database normalization where there are no non-trivial multivalued dependencies other than a candidate key. It builds on the first three normal forms (1NF, 2NF, and 3NF) and the Boyce-Codd Normal Form (BCNF). It states that, in addition to a database meeting the requirements of BCNF, it must not contain more than one multivalued dependency.
- **Properties**
- A relation R is in 4NF if and only if the following conditions are satisfied:

1. It should be in the Boyce-Codd Normal Form (BCNF).
2. The table should not have any Multi-valued Dependency.

- A table with a multivalued dependency violates the normalization standard of the Fourth Normal Form (4NF) because it creates unnecessary redundancies and can contribute to inconsistent data. To bring this up to 4NF, it is necessary to break this information into two tables.
- **Example:** Consider the database table of a class that has two relations R1 contains student ID(SID) and student name (SNAME) and R2 contains course id(CID) and course name (CNAME).
- **Table R1** **Table R2**

SID	SNAME
S1	A
S2	B

CID	CNAME
C1	C
C2	D

- When their cross-product is done it resulted in multivalued dependencies.

- **Table R1 X R2**

SID	SNAME	CID	CNAME
S1	A	C1	C
S1	A	C2	D
S2	B	C1	C
S2	B	C2	D

- Multivalued dependencies (MVD) are:

  SID->->CID; SID->->CNAME; SNAME->->CNAME

Join Dependency:

- Join decomposition is a further generalization of Multivalued dependencies. If the join of R1 and R2 over C is equal to relation R then we can say that a join dependency (JD) exists, where R1 and R2 are the decomposition R1(A, B, C) and R2(C, D) of a given relations R (A, B, C, D). Alternatively, R1 and R2 are a **lossless decomposition** of R. A JD ⋈ {R1, R2, …, Rn} is said to hold over a relation R if R1, R2, ….., Rn is a lossless-join decomposition. The *(A, B, C, D), (C, D) will be a JD of R if the join of joins attribute is equal to the relation R. Here, *(R1, R2, R3) is used to indicate that relation R1, R2, R3 and so on are a JD of R. Let R is a relation schema R1, R2, R3……..Rn be the decomposition of R.
- **Example:**
- **Table R1:** **Table R2** **Table R3**

Company	Product
C1	Pendrive
C1	mic
C2	speaker
C2	speaker

Company->->Product

Agent	Company
Aman	C1
Aman	C2
Mohan	C1

Agent->->Company

Agent	Product
Aman	Pendrive
Aman	Mic
Aman	speaker
Mohan	speaker

Agent->->Product

- **Table R1⋈R2⋈R3**

Company	Product	Agent
C1	Pendrive	Aman
C1	mic	Aman
C2	speaker	speaker
C1	speaker	Aman

Agent->->Product

Fifth Normal Form/Projected Normal Form (5NF):

- A relation R is in Fifth Normal Form if and only if everyone joins dependency in R is implied by the candidate keys of R. A relation decomposed into two relations must have lossless join Property, which ensures that no spurious or extra tuples are generated when relations are reunited through a natural join.
- **Properties**
- 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).

- **Example –** Consider the above schema, with a case as “if a company makes a product and an agent is an agent for that company, then he always sells that product for the company”. Under these circumstances, the ACP table is shown as:
- **Table ACP**

Agent	Company	Product
A1	PQR	Nut
A1	PQR	Bolt
A1	XYZ	Nut
A1	XYZ	Bolt
A2	PQR	Nut

The relation ACP is again decomposed into 3 relations. Now, the natural Join of all three relations will be shown as:

- **Table R1** **Table R2** **Table R3**

Agent	Company
A1	PQR
A1	XYZ
A2	PQR

Agent	Product
A1	Nut
A1	Bolt
A2	Nut

Company	Product
PQR	Nut
PQR	Bolt
XYZ	Nut
XYZ	Bolt

- The result of the Natural Join of R1 and R3 over ‘Company’ and then the Natural Join of R13 and R2 over ‘Agent’ and ‘Product’ will be **Table ACP**.
- Hence, in this example, all the redundancies are eliminated, and the decomposition of ACP is a lossless join decomposition. Therefore, the relation is in 5NF as it does not violate the property of lossless join.

Important Points Regarding Normal Forms in DBMS:

- **First Normal Form (1NF):** This is the most basic level of normalization. In 1NF, each table cell should contain only a single value, and each column should have a unique name. The first normal form helps to eliminate duplicate data and simplify queries.
- **Second Normal Form (2NF):** 2NF eliminates redundant data by requiring that each non-key attribute be dependent on the primary key. This means that each column should be directly related to the primary key, and not to other columns.
- **Third Normal Form (3NF):** 3NF builds on 2NF by requiring that all non-key attributes are independent of each other. This means that each column should be directly related to the primary key, and not to any other columns in the same table.
- **Boyce-Codd Normal Form (BCNF):** BCNF is a stricter form of 3NF that ensures that each determinant in a table is a candidate key. In other words, BCNF ensures that each non-key attribute is dependent only on the candidate key.
- **Fourth Normal Form (4NF):** 4NF is a further refinement of BCNF that ensures that a table does not contain any multi-valued dependencies.
- **Fifth Normal Form (5NF):** 5NF is the highest level of normalization and involves decomposing a table into smaller tables to remove data redundancy and improve data integrity.