UNIT-6 [Data Normalization]

Functional Dependencies

A functional Dependency in DBMS is a Fundamental Concept that describes the relationship between attributes (columns) in a table. It shows how the values in one or more attributes determine the value in another.

Functional dependency is represented in the form of an equation. Here, you have set of attributes (A, B, C, etc.) and an arrow (→) denoting the Dependency.

For example, if we have a table of employee data with columns "EmployeeID", "FirstName", and "LastName", we can express a functional dependency like this:

EmployeeID → FirstName, LastName.

Types of Functional Dependencies

1. Trivial Functional Dependency - In Trivial Functional Dependency, 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.

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.

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 eachother. i.e If a → {b, c} and there exists no functional dependency between b and c, then it is called a multivalued functional dependency

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!)

1. 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.

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, hence called Transitive Functional dependency.

5. Fully Functional Dependency – In a full functional dependency an attribute or a set of attributes uniquely determines another attribute or set of attributes. If a Relation R has attributes X, Y, Z with the dependencies X → Y and X → Z which states that those dependencies are fully functional.

Partial Functional Dependency – In partial functional dependency a non key attribute depends on a part of the composite key, rather than the whole key. If a Relation R has attributes X, Y, Z where X and Y are the Composite key and Z is non-key attribute. Then X → Z is a partial F.d in RDBMS.

25.10.

E.F Codd's Rules

EF Codd rules are proposed by a Computer Scientist named Dr. Edgar F. Codd and he also invent the relational model for database management. These rules are made to ensure data integrity, Consistency, and usability. This set of rules basically signifies the characteristics & requirements of a RDBMS.

Rule 1: The information Rule

A database contains various information, and this information must be stored in each cell of a table in the form of rows and columns.

Rule 2: Guaranteed Access Rule

Every single or precise data (atomic value) may be accessed logically from a relational database using the combination of primary key value, table name and Column name.

Rule 3: Systematic Treatment of Null Values

This rule defines the systematic treatment of Null values in database records. The null value has various meanings in database, like missing the data, no value in a cell, inappropriate information, unknown data & the primary keys should not be null.

Rule 4: Active/Dynamic Online Catalog based on the relational Model.

It represents the entire logical structure of the descriptive database that must be stored online is known as a database dictionary. It authorizes users to access the database and implement a similar query language to Access the database.

Rule 5: Comprehensive Data Sublanguage Rule

The relational database supports various languages, and if we want to Access the database, the language must be the explicit, linear or well-defined. Syntax, character strings and supports the Comprehensive: data definition, view definition, data manipulation, integrity constraints, & limit transaction Management operations. If the database allows access to the data without any language, it is considered a violation of the database.

Rule 6: View updating Rule

All views table can be theoretically updated and must be practically updated by the database systems.

Rule 7: Relational level operation (High-level Insert, Update and delete) Rule

A database system should follow high-level relational operations such as insert, update, and delete in each level or a single row. It also supports union, intersection and minus operation in the database system.

Rule 8: Physical Data Independence Rule

All stored data in a database or an application must be physically independent to access the database. Each data should not depend on other data or an application. If data is updated or the physical structure of the database is changed, it will not show any effect on external applications that are accessing the data from the database.

Rule 9: Logical Data Independence Rule

It is similar to physical data independence. It means, if any changes occured to the logical level (table structures), it should not affect the user's view (application). For ex, Suppose a table either split into two tables, or two table joins to create a single table, these changes should not be impacted on the user view application.

Rule 10: Integrity Independence Rule

A database must maintain integrity independence when inserting data into table's cell using the SQL query language. All entered values should not be changed or rely on any external factor or application to maintain integrity. It is also helpful in making the database independent for each front-end application.

Rule 11: Distribution Independence Rule

The distribution of data across multiple locations should be invisible to users, and the database system should handle the distribution transparently.

Rule 12: Non-Subversion Rule

If the interface of the system is providing access to low-level records, then the interface must not be able to damage the system and bypass security and integrity constraints.

Normalization

database management System (DBMS), normal forms are a ries of guidelines that help to ensure that the design of a base is efficient, organized, and free from data anomalies. re are several levels of normalization, each with its own set guidelines, known as normal Forms.

dvantages / Benefits of Normal Form

educed data redundancy - Normalization helps to eliminate duplicate data in tables, reducing the amount of storage space needed nd improving database efficiency.

mproved data Consistency - Normalization ensures the data is stored in a consistent and organized manner, reducing the risk of data inconsistencies and errors.

Simplified database design - Normalization provides guidelines for organizing tables and data relationships, making it easier o design and maintain a database.

Improved query performance - Normalized tables are typically easier to search and retrieve data from, resulting in faster query performance.

Easier database maintenance - Normalization reduces the complexity of a database by breaking it down into smaller, more manageable tables, making it easier to add, modify and delete data.

First Normal Form (1NF)

If a relation contain composite or multi-valued attribute, it violates first Normal form or a relation is in 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.

Example-

ID	Name	Courses
1	A	C1, C2
2	E	C3
3	M	C2, C3

→ In the above table Course is a multi-valued attribute So it is not in INF. Below Table is in 1NF as there is no multi-valued attribute.

ID	Name	Course
1	A	C1
1	A	C2
2	E	C3
3	M	C2
3	M	C3

Second Normal Form (2NF)

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 $X \rightarrow Y$ (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.

Example 1 – Consider the below table –

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 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 a proper subset of the Candidate key, which is 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 I: 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
C4	1000
C3	2000
C1	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.

Third Normal Form 3NF

A relation is said to be in third normal form, if we did not have any transitive dependency for non-prime attributes. The basic condition with the third Normal form is that, the relation must be in second Normal form.

Below mentioned is the basic condition that must be hold in the non-trivial functional dependency $X \rightarrow Y$:

- X is a Super Key.
- Y is a prime Attribute (this means that element of Y is some part of Candidate Key).

BCNF

BCNF (Boyce-Codd Normal Form) is just a advanced version of third Normal Form. Here we have some additional rules than Third Normal Form. The basic condition for any relation to be in BCNF is that it must be in Third Normal Form.

We have to focus on some basic rules that are for BCNF:

1. Table must be in 3NF
2. In relation $X \rightarrow Y$, must be a Super Key in a relation.

ACID Properties in DBMS

transaction is a single logical unit of work that accesses and possibly modifies the contents of a database. Transactions access data using read-write operations. To maintain consistency in a database, before and after the transaction, certain properties are followed. These are called ACID properties.

ACID

A = Atomicity

C = Consistency

I = Isolation

D = Durability

Atomicity:

By this, we mean that either the entire transaction takes place at once or doesn't happen at all. There is no midway i.e. transaction is do not occur partially. Each transaction is considered as one unit and either runs to completion or is not executed at all. It involves the following two operations:

- Abort : If a transaction aborts, changes made to the database are not visible.

- commit : If a transaction commits, changes made are visible.

Atomicity is also known as the 'All or nothing rule'.

Example. Consider the following transaction T consisting of T1 and T2 : Transfer of 100 from Account X to account Y.

Before : X : 500	Y : 200
Transaction T	
T1	T2
Read(X) X : = X − 100 write (X)	Read(Y) Y : Y + 100 Write (Y)
After : X : 400	Y : 300

If the transaction fails after completion of T1 but before Completion of T2. (say, after write(X) but before write(Y), then the amount has been deducted from X but not added to Y. This results in an inconsistent database state.
Therefore, the transaction must be executed in its entirety in order to ensure the correctness of the database state.

Consistency:

This means that integrity constraints must be maintained so that the database is consistent before and after a transaction. It refers to the correctness of a database.

Referring to the example above,

The total amount before and after the transaction must be maintained.

Total before T occurs = 500 + 200 = 700

Total after T occurs = 400 + 300 = 700

Therefore, the database is consistent Inconsistency occurs in case T1 completes but T2 fails. As a result, T is incomplete.

Isolation:

This property ensures that multiple transactions can occur concurrently without leading to the inconsistency of the database state. Transactions occur independently without interference. Changes occuring in a particular transaction will not be visible to any other transaction until that particular change in that transaction is written to memory or has been committed. This property ensures that the execution of transactions concurrently will result in a state that is equivalent to a state achieved these were executed serially in some order.

Durability:

This property ensures that once the transaction has completed execution, the updates and modifications to the database are stored in and written to disk and they persist even if a

System Failure occurs. These updates now become permanent and are stored in non-volatile memory. The effects of the transaction, thus, are never lost.

Concurrency Control in DBMS

Concurrency Control is a very important concept of DBMS which ensures the simultaneous execution or manipulation of data by several processes or user without resulting in data inconsistency.

Transaction

A transaction is a collection of operations that performs a single logical function in a database application. Each transaction is a unit of both atomicity and consistency. Thus, we require that transactions do not violate any database consistency constraints. That is, if database was consistent when a transaction started, the database must be consistent when the transaction successfully terminates.

A set of logically related operations is known as transaction. The main operations of a transaction are:

Read (A): Read operations Read (A) or R(A) reads the value of A from the database and stores it in a buffer in the main memory.

Write (A): Write operation Write (A) or W(A) writes the value back to the database from the buffer.

Example: let us take a debit transaction from an account which Consists of the following operations:

1 R(A);
2 A= A-1000;
3 W(A);

Assume A's value before starting the transaction is 5000.

→ The First operation reads the value of A from the database and Stores it in a buffer.

→ The second operation will decrease its value by 1000. So buffer will contain 4000.

→ The third operation will write the value from the buffer to the database. So A's final value will be 4000.

But it may also be possible that the transaction may fail after executing some of its operations. The failure can be because of hardware, Software or power, etc.

Schedule

A Schedule is a Series of operations from one or more transactions. A ~~Sch~~ Schedule can be of two types:

Serial Schedule: When one transaction completely executes before starting another transaction, the schedule is called a serial Schedule. A Serial Schedule is always Consistent.

Concurrent Schedule: When operations of a transaction are interleaved with operations of other transactions of a schedule the Schedule is called a Concurrent Schedule. eg; But Concurrency can lead to inconsistency in the database.

Concurrency Control Protocols

Concurrency control protocols are the set of rules which are maintained in order to solve the concurrency control problems in the database. It ensures that the concurrent transactions can execute properly while maintaining the database consistency.

Locked Based Protocol

In locked based Protocol, each transaction needs to acquire locks before they start accessing or modifying the data items. There are two types of locks used in databases.

Shared lock: Shared lock is also known as read lock which allows multiple transactions to read the data simultaneously. The transaction which is holding a shared lock can only read the data item but it can not modify the data item.

Exclusive lock: Exclusive lock is also known as the write lock. Exclusive lock allows a transaction to update a data item. Only one transaction can hold the exclusive lock on a data item at a time. While a transaction is holding an exclusive lock on a data item, no other transaction is allowed to acquire a shared/exclusive lock on the same data item.

Timestamp based Protocol

In this protocol each transaction has a timestamp attached to it. Timestamp is nothing but the time in which a transaction enters into the system.

The conflicting pairs of operations can be resolved by the timestamp ordering protocol through the utilization of the

timestamp values of the transactions.
Therefore, guaranteeing that the transactions take place in correct order.

Advantages of Concurrency

→ **Waiting time**

It means if a process is in ready state but still the process does not get the system to get execute is called waiting time. So Concurrency leads to less waiting time.

→ **Response time**

The time wasted in getting the response from the CPU for the time, is called response time. So, Concurrency leads to less Response Time.

→ **Resource Utilization**

The amount of Resource Utilization in a particular System is called Resource utilization. Multiple transactions can run parallel in the System. So, Concurrency leads to more Resource Utilization.

→ **Efficiency**

The amount of output produced in comparison to give input is called efficiency. So, Concurrency leads to more Efficiency.