DATABASE MANAGEMENT SYSTEM

UNIT-3

Functional Dependency

The functional dependency is a relationship that exists between two attributes. It typically exists between the primary key and non-key attribute within a table.

$$X \rightarrow Y$$

The left side of FD is known as a determinant, the right side of the production is known as a dependent.

For example:

Assume we have an employee table with attributes: Emp_Id, Emp_Name, Emp_Address.

Here Emp_Id attribute can uniquely identify the Emp_Name attribute of employee table because if we know the Emp_Id, we can tell that employee 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.

Types of Functional dependency

1. Trivial functional dependency
2. Non-trivial functional dependency

1. Trivial functional dependency

- A → B has trivial functional dependency if B is a subset of A.
- The following dependencies are also trivial like: A → A, B → B

Example:

Consider a table with two columns Employee_Id and Employee_Name.

{Employee_id, Employee_Name} → Employee_Id is a trivial functional dependency as

Employee_Id is a subset of {Employee_Id, Employee_Name}.

Also, Employee_Id → Employee_Id and Employee_Name → Employee_Name are trivial dependencies too.

2. Non-trivial functional dependency

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

Example:

ID → Name,

Name → DOB

Inference Rule (IR):

- The Armstrong's axioms are the basic inference rule.
- Armstrong's axioms are used to conclude functional dependencies on a relational database.
- The inference rule is a type of assertion. It can apply to a set of FD(functional dependency) to derive other FD.
- Using the inference rule, we can derive additional functional dependency from the initial set.

The Functional dependency has 6 types of inference rule:

1. Reflexive Rule (IR_1)

In the reflexive rule, if Y is a subset of X, then X determines Y.

If X ⊇Y then X → Y

Example:

X = {a, b, c, d, e}

Y = {a, b, c}

2. Augmentation Rule (IR_2)

The augmentation is also called as a partial dependency. In augmentation, if X determines Y, then XZ determines YZ for any Z.

If X → Y then XZ → YZ

Example:

For R(ABCD), **if** A → B then AC → BC

3. Transitive Rule (IR_3)

In the transitive rule, if X determines Y and Y determine Z, then X must also determine Z.

If X → Y and Y → Z then X → Z

4. Union Rule (IR_4)

Union rule says, if X determines Y and X determines Z, then X must also determine Y and Z.

If X → Y and X → Z then X → YZ

Proof:

1. X → Y (given)
2. X → Z (given)
3. X → XY (using IR_2 on 1 by augmentation with X. Where XX = X)
4. XY → YZ (using IR_2 on 2 by augmentation with Y)
5. X → YZ (using IR_3 on 3 and 4)

5. Decomposition Rule (IR_5)

Decomposition rule is also known as project rule. It is the reverse of union rule.

This Rule says, if X determines Y and Z, then X determines Y and X determines Z separately.

If X → YZ then X → Y and X → Z

Proof:

1. X → YZ (given)
2. YZ → Y (using IR_1 Rule)
3. X → Y (using IR_3 on 1 and 2)

6. Pseudo transitive Rule (IR_6)

In Pseudo transitive Rule, if X determines Y and YZ determines W, then XZ determines W.

If X → Y and YZ → W then XZ → W

Proof:

1. X → Y (given)
2. WY → Z (given)
3. WX → WY (using IR_2 on 1 by augmenting with W)
4. WX → Z (using IR_3 on 3 and 2)

Lossless decomposition:

Lossless join decomposition is a decomposition of a relation R into relations R1, 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…

In other words, by lossless decomposition, it becomes feasible to reconstruct the relation R from decomposed tables R1 and R2 by using Joins.

In Lossless Decomposition, we select the common attribute and the criteria for selecting a common attribute is that the common attribute must be a candidate key or super key in either relation R1, R2, or both.

Decomposition of a relation R into R1 and R2 is a lossless-join decomposition if at least one of the following functional dependencies are in F+ (Closure of functional dependencies)

R1 ∩ R2 → R1
OR
R1 ∩ R2 → R2

Normalization:

A large database defined as a single relation may result in data duplication. This repetition of data may result in:

- Making relations very large.
- It isn't easy to maintain and update data as it would involve searching many records in relation.
- Wastage and poor utilization of disk space and resources.
- The likelihood of errors and inconsistencies increases.

So, to handle these problems, we should analyze and decompose the relations with redundant data into smaller, simpler, and well-structured relations that are satisfy desirable properties. Normalization is a process of decomposing the relations into relations with fewer attributes.

What is Normalization?

- Normalization is the process of organizing the data in the database.
- Normalization is used to minimize the redundancy from a relation or set of relations. It is also used to eliminate undesirable characteristics like Insertion, Update, and Deletion Anomalies.
- Normalization divides the larger table into smaller and links them using relationships.
- The normal form is used to reduce redundancy from the database table.

Data modification anomalies can be categorized into three types:

- **Insertion Anomaly:** Insertion Anomaly refers to when one cannot insert a new tuple into a relationship due to lack of data.
- **Deletion Anomaly:** The delete anomaly refers to the situation where the deletion of data results in the unintended loss of some other important data.

- **Updatation Anomaly:** The update anomaly is when an update of a single data value requires multiple rows of data to be updated.

Advantages of Normalization

- Normalization helps to minimize data redundancy.
- Greater overall database organization.
- Data consistency within the database.
- Much more flexible database design.
- Enforces the concept of relational integrity.

Disadvantages of Normalization

- You cannot start building the database before knowing what the user needs.
- The performance degrades when normalizing the relations to higher normal forms, i.e., 4NF, 5NF.
- It is very time-consuming and difficult to normalize relations of a higher degree.
- Careless decomposition may lead to a bad database design, leading to serious problems.

Types of Normalization:

1. First Normal Form (1NF)

- A relation will be 1NF if it contains an atomic value.
- It states that an attribute of a table cannot hold multiple values. It must hold only single-valued attribute.
- First normal form disallows the multi-valued attribute, composite attribute, and their combinations.

Example: Relation EMPLOYEE is not in 1NF because of multi-valued attribute EMP_PHONE.

2. Second Normal Form (2NF)

- In the 2NF, relational must be in 1NF.
- In the second normal form, all non-key attributes are fully functional dependent on the primary key

Example: Let's assume, a school can store the data of teachers and the subjects they teach. In a school, a teacher can teach more than one subject.

3. Third Normal Form (3NF)

- A relation will be in 3NF if it is in 2NF and not contain any transitive partial dependency.
- 3NF is used to reduce the data duplication. It is also used to achieve the data integrity.
- If there is no transitive dependency for non-prime attributes, then the relation must be in third normal form.

A relation is in third normal form if it holds atleast one of the following conditions for every non-trivial function dependency X → Y.

1. X is a super key.
2. Y is a prime attribute, i.e., each element of Y is part of some candidate key.

4. 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 → 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.

Example: Let's assume there is a company where employees work in more than one department.

EMPLOYEE table:

EMP_ID	EMP_COUNTRY	EMP_DEPT	DEPT_TYPE	EMP_DEPT_NO
264	India	Designing	D394	283
264	India	Testing	D394	300
364	UK	Stores	D283	232
364	UK	Developing	D283	549

In the above table Functional dependencies are as follows:

3. EMP_ID → EMP_COUNTRY
4. EMP_DEPT → {DEPT_TYPE, EMP_DEPT_NO}

Candidate key: {EMP-ID, EMP-DEPT}

The table is not in BCNF because neither EMP_DEPT nor EMP_ID alone are keys.

To convert the given table into BCNF, we decompose it into three tables:

EMP_COUNTRY table:

EMP_ID	EMP_COUNTRY
264	India
264	India

EMP_DEPT table:

EMP_DEPT	DEPT_TYPE	EMP_DEPT_NO

Designing	D394	283
Testing	D394	300
Stores	D283	232
Developing	D283	549

EMP_DEPT_MAPPING table:

EMP_ID	EMP_DEPT
D394	283
D394	300
D283	232
D283	549

Functional dependencies:

5. EMP_ID → EMP_COUNTRY
6. EMP_DEPT → {DEPT_TYPE, EMP_DEPT_NO}

Candidate keys:

For the first table: EMP_ID
For the second table: EMP_DEPT
For the third table: {EMP_ID, EMP_DEPT}

Now, this is in BCNF because left side part of both the functional dependencies is a key.

Dependency Preservation:

This decomposition property can only be done by maintaining the functional dependency. When an update is made to the database, the system should be able to check that the update will not create an illegal relation.

A Decomposition D = { R1, R2, R3….Rn } of R is dependency preserving wrt a set F of Functional dependency if

(F1 ∪F2 ∪… ∪Fm)+ = F+.
Consider a relation R
R ---> F{...with some functional dependency(FD)....}

R is decomposed or divided into R1 with FD { f1 } and R2 with { f2 }, then
there can be three cases:

f1 U f2 = F -----> Decomposition is dependency preserving.
f1 U f2 is a subset of F -----> Not Dependency preserving.
f1 U f2 is a super set of F -----> This case is not possible.

Transaction:

Transaction is an action, or series of actions that are being performed by a single user or application program, which reads or updates the contents of the database.

Types of transaction

Read only transaction: If the database operations in a transaction do not update the database but only retrieve data.

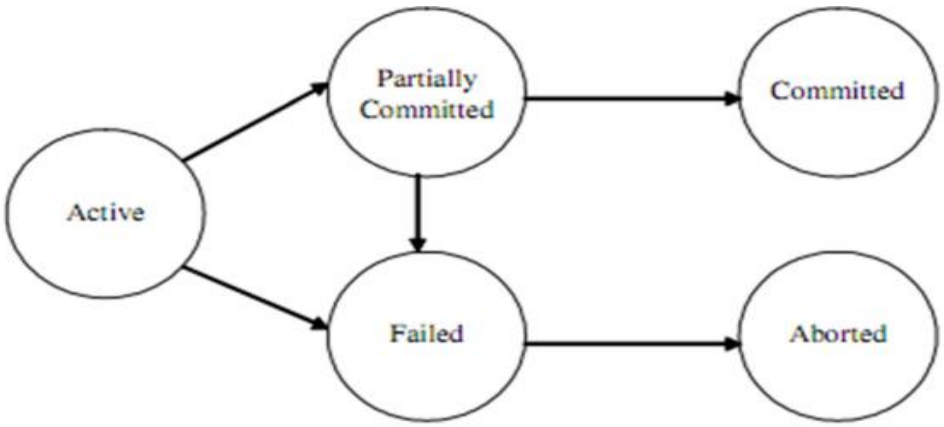
Read write transaction: If the database operation in a transaction retrieve as well as update the database.

Transaction Properties

A transaction must have the following four properties, called ACID properties, to ensure that a database remains stable state after the transaction is executed:

1. **Atomicity:** This property states that each transaction must be considered as a single unit and must be completed fully or not completed at all.
2. **Consistency:** The database must remain in a consistent state after any transaction.
3. **Isolation:** Isolation property of a transaction means that the data used during the execution of a transaction cannot be used by a second transaction until the first one is completed.
4. **Durability:** The database should be strong enough to handle any system failure.

States of Transactions:



Active: In this state, the transaction is being executed. This is the initial state of every transaction.

Partially-Committed: When a transaction executes its final operation, it is said to be in a partially committed state.

Committed: If a transaction executes all its operations successfully, it is said to be committed.

Failed: If a transaction cannot proceed to the execution state because of the failure of the system or database, then the transaction is said to be in failed state.

Aborted: If a transaction is failed to execute, then the transaction has been rolled back and the database has been restored to its state prior to the start of the transaction.