SNS COLLEGE OF TECHNOLOGY

(An Autonomous Institution)

Re-accredited by NAAC with A+ grade, Accredited by NBA(CSE, IT, ECE, EEE & Mechanical)
Approvedy by AICTE, New Delhi, Recognized by UGC, Affiliated to Anna University, Chennai

Department of MCA

DBMS Dependency in DBMS

Course Name : 23CAT603 - DATA BASE MANAGEMENT SYSTEM

Class : I Year / I Semester

Unit III – Dependency in DBMS

Dependency in DBMS

What are Dependencies in DBMS?

- A dependency is a constraint that governs or defines the relationship between two or more attributes.
- In a database, it happens when information recorded in the same table uniquely determines other information stored in the same table.
- This may also be described as a relationship in which knowing the value of one attribute (or collection of attributes) in the same table tells you the value of another attribute (or set of attributes).
- It's critical to understand database dependencies since they serve as the foundation for database normalization.

Types of Dependencies in DBMS

In DBMS, it has the following types:

- Functional Dependency
- Fully-Functional Dependency
- Transitive Dependency
- Multivalued Dependency
- Partial Dependency

Now, let's get started with Functional Dependency.

Functional Dependency

Functional Dependencies:

A functional dependency (FD) is a relationship that exists between two attributes in a database, typically the primary key and additional non-key attributes. Consider it a link between two qualities of the same relation.

A dependency is denoted by an arrow "→".

If C determines D functionally, then **C→D.**

Functional dependency, indicated as C→ D, is a relationship between two sets of attributes, C and D. In this case, C is referred to as the "**determinant**", and D is referred to as the "**dependent**".

Functional Dependency aids in the maintenance of database data quality.

Functional Dependency Rules

Inference Rules

Axioms:

A relational database's functional dependencies can be inferred using Armstrong's axioms, a set of inference principles. Armstrong, William W., created them.

Functional Dependencies Axioms:

The reflexive rule states that if D is a subset of C, then D is determined by C., i.e. C→D.

The augmentation rule, also known as the partial dependency rule, states that if D is determined by C, then CZ determines DZ for any Z.Every non-key attribute is required to be totally dependent on the Primary Key, according to it.
i.e. If C→D, then CZ→DZ for any Z.

Transitivity rule states that if D is determined by C and Z is determined by D, then C must also determine Z., i.e. if C→D and D→Z, then C→Z.

Decomposition:

It is a rule that stipulates that if a table appears to contain two entities determined by the same primary key, it should be split into two independent tables.

Functional Dependency Rules

According to this rule, if C determines D and Z, C also determines D and Z individually.i.e. if C→DZ then C→D and C→Z.

Union

It suggests that if two tables are independent yet have the same Primary Key, they should be combined.It states that C must determine D and Z if C determines D and C determines Z.

i.e. if C→D and C→Z then C→DZ.

Functional Dependency TERMS

Terms:

Dependent: It is shown on the functional dependency diagram's right side.

Determinant: It is shown on the functional dependency Diagram's left side.

Non-normalized table: A table containing redundant data.

Examples

Example 1: Here, we have a table named **Student**.

<Student>

StuID	StuName	StuAge
E01	Rose	14
E02	Rolly	13

Here, **StuName** in the preceding table is functionally dependent on **StuID** since **StuName** can only accept one value for the specified value of StuID, i.e. Because a student's name can be uniquely determined from an ID, StuName can be considered to be dependent on StuID.

1.StuID→StuName

However, the converse assertion (StuName?>StuID) is false because multiple students can have the same name but have different StuID's.

Functional Dependency TERMS

Example 2: We have a table **Employee.**
<Employee>

Employee_No	E_Name	E_Salary	Address
1	Dolly	60000	Seoul
2	Flora	48000	BukchonHanok
3	Anni	35000	Seoul

We can Deduce Several Valid Functional Dependencies from the Preceding Table:
In this case, knowing the value of Employee_No allows us to access E_Name, Address, E_Salary, and so on. As a result, the Address, E Name, and E Salary are all functionally dependent on Employee No.

- **Employee_No→ {E_Name, E_Salary, Address}:** Employee _No can decide the values of fields E_Name, E_Salary, and Address in this case, resulting in a legal Functional dependence.
- **Employee_No→E_Salary**, Because Employee_No can determine the entire set of {E_Name, E_Salary, and Address}, it can also determine its subset E_Salary.
- More valid functional dependents include: Employee_No→name, {Employee_No, E_Name }→(E_Salary, Address}, and so on.

Functional Dependency TERMS

Here are Some Invalid Functional Dependencies:

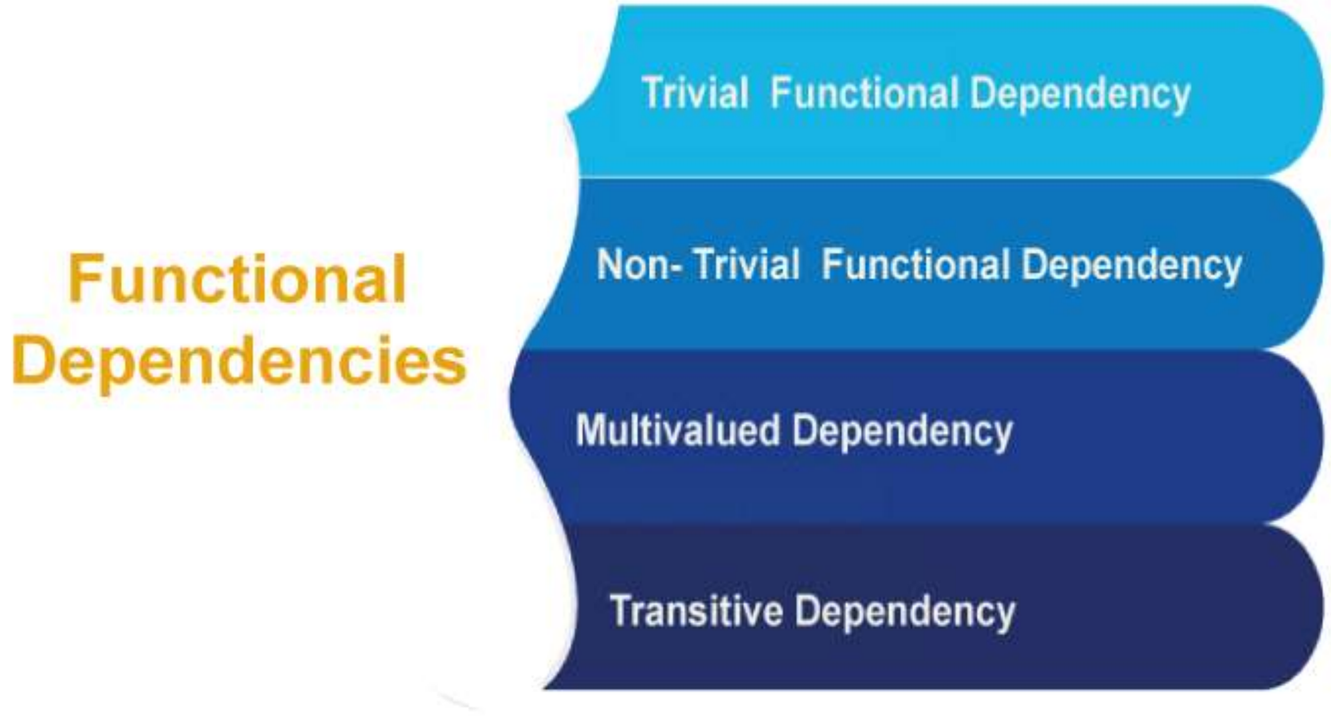
E_Name→E_Salary: This is not an acceptable functional dependency because employees with the same name can have different salaries.

Address→E_Salary: Different salaries can be given to the employees of the same Address; for example, E_Salary 60000 and 35000 in the preceding table belong to employees of the same address, "Seoul"; hence **Address→E_Salary** is an incorrect functional dependency.

More **invalid functional dependencies** include: **E_Name→Employee_No, {E_Name, E_Salary}→Employee_No**, and so on.

Types of Functional Dependencies

Types of Functional Dependencies



1. Trivial Functional Dependency

1. Trivial Functional Dependency:

A "dependent" in Trivial functional dependency is always a subset of the "determinant".

A functional dependency is said to be trivial if the attributes on its right side are a subset of the attributes on its left side.

If D is a subset of C, C→D is referred to as a Trivial Functional Dependency.

Example: Take a look at the Student table below.

<Student>

Roll_No	S_Name	S_Age
1	John	13
2	Riya	12
3	Giya	15
4	Jolly	16

- **{Roll_No, S_ Name} →S_Name** is a Trivial functional dependency in this case because the dependant S_Name is a subset of the determinant {Roll_No, S_Name}.
- **{ Roll_No } → { Roll_No }, { S_Name } → { S_Name }** and
- **{ S_Age } → { S_Age }** are also Trivial.

2. Non-Trivial Functional Dependency

It is the inverse of Trivial functional dependence. Formally, a Dependent is a Non-Trivial functional dependency if it is not a subset of the determinant.

If D is not a subset of C, C→D is said to have a non-trivial functional dependency. Non-trivial functional dependency is defined as a functional dependency C→ D where C is a set of attributes and D is also a set of attributes but not a subset of C.

Example: Consider the Student table below.

<Student>

Roll_No	S_Name	S_Age
1	John	13
2	Riya	12
3	Giya	15
4	Jolly	16

Roll_No→S_Name is a non-trivial functional dependency in this case since S_Name(dependent) is not a subset of Roll_No (determinant).

Similarly, **{Roll_No, Name}→ Age** are non-trivial functional dependencies.

3. Multivalued Functional Dependency

In multivalued functional dependency, attributes in the dependent set are not dependent on one another.

For example, C {D, Z}is referred to as a Multivalued functional dependency if there is no functional dependency between D and Z.

Example: Take a look at the **Student** table below.

<Student>

Roll_No	S_Name	S_Age
1	John	13
2	Riya	12
3	Giya	15
4	Jolly	16

{Roll_No}→ {S_Name, S_Age) is a Multivalued functional dependency in this case because the "dependent values" S_ Name and S_Age are not functionally dependent (i.e. **S_Name→S_Age** or **S_ Age→S_ Name** does not exist).

4. Transitive Functional Dependency

Consider two functional dependencies, C→ D and D→Z; C→Z must exist according to the transitivity principle. This is referred to as a **Transitive Functional dependency**.

In transitive functional dependency, the dependent is dependent on the determinant indirectly.

Example: Consider the Student table below.

<Student>

Roll_No	S_Name	S_Department	Street_No
1	John	AC	12
2	Riya	BH	11
3	Giya	MV	14
4	Jolly	CD	18

Roll_No→S_Department and **S_Department→Street_No** are correct here. As a result, **Roll_No→Street_No** is a valid functional dependency, according to the principle of transitivity.

Benefits of Functional Dependency

- Functional Dependency prevents data duplication. As a result, the same data does not appear several times in that database.
- It assists you in maintaining the database's data quality.
- It assists you in defining database semantics and constraints.
- It aids you in spotting flawed designs.
- It aids you in locating database design information.
- The Normalization method begins with identifying the potential keys in the relation. It is impossible to locate candidate keys and normalize the database without functional dependencies.

Fully Functional Dependency

A functional dependency C→D,a fully functional dependency is one in which, if any attribute x from C is removed, the "dependency" no longer exists.
If D is "fullyfunctional dependent" on C, it is not functionally dependent on any of the valid subsets of C.
i.e.Attribute Z in the relation CDE->Z is "fully functionally dependent"on CDE and not on any appropriate subset of CDE. That is, CDE subsets such as CD, DE, C, D, and so on cannot determine Z.
Also;

- Full Functional Dependency corresponds to the Second Normal Form normalization standard.
- Functional dependency improves the data quality of our database.
- In this dependency, the non-prime property is functionally reliant on the candidate key.
- The full dependency on database attributes helps to assure data integrity and eliminate data abnormalities.

Fully Functional Dependency

Example: Here, we have a table named **Supply**.
<Supply>

Seller_Id	Product_id	T_price
1	1	530
2	1	535
1	2	100
2	2	101
3	1	342

According to the Table, neither **Seller_id** nor **Product_id** can uniquely determine the price, but both **Seller_id** and **Product_id** combined can.
As a result, we can say that **T_price** is "fully functionally dependent" on **Seller_id** nor **Product_id**.
This outlines and demonstrates our fully functional dependency:
1.{ Seller_id , Product_id } →T_Price

Partial Functional Dependency

A functional dependency C → D, If the dependency doeshold after removing any attribute x from C, then it is said to be a **Partial Functional Dependency**.

A functional dependency C→D, If D is functionally dependent on C and may be determined by any appropriate subset of C, there is a partial dependency.

i.e. We have an CF->D, C->E, and E->D relation. Now, let us compute the closure of {C+}=CED. In this case, C can determine D on its own, implying that D is partially dependent on CF.

Also;

In partial functional dependency, the non-prime attribute is functionally dependent on a component of a candidate key.

The normalizing standard of the Second Normal Form does not apply to Partial Functional Dependency. 2NF, on the other hand, eliminates Partial Dependency.

Partially dependent data does not improve data quality. It must be removed before normalization in the second normal form may occur.

Partial Functional Dependency

Cause of Partial Dependency:

Partial dependency happens when a non-prime attribute is functionally dependent on a portion of the given candidate key, as we saw in the preceding section.

In other words, partial dependency arises when an attribute in a table is dependent on only a portion of the primary key rather than the entire key.

Example: We have a table called **Student** here.

<Student>

Roll_No	S_Name	S_Course
1	John	DBMS
2	Riya	C++
3	Giya	Java
4	Jolly	C

We can see that the attributes S_Name and Roll_No can both uniquely identify a S_Course. As a result, we might argue that the relationship is partly dependent.

Transitive Dependency

A transitive dependence is any non-prime attribute other than the candidate key that is reliant on another non-prime attribute that is wholly dependent on the candidate key.

Transitive Dependency occurs when an indirect interaction results in functional dependency. As a result, if C→ D and D ->Z are true, then C ->Z is a transitive dependency.

Transitive dependency causes deletion, update, and insertion errors in the database and is regarded as poor database design.

To reach 3NF, one must first eliminate Transitive Dependency.

Only when two Functional Dependencies establish an indirect functional dependency can it be transitive. As an example,

When the following functional dependencies hold true, C →E is a transitive dependency:

C ->D

D does not imply C

C→E

Only in the case of some given relation of three or more attributes can transitive dependency occur effortlessly. Such a dependency aids us in normalizing the database in its 3rd Normal Form (3NF).

Transitive Dependency

Example: Here, we have a table **Telecast_show**.

<Telecast_show>

d_show	Id_telecast	Type_telecast	Cost_CD
F01	S01	Romantic	30
F02	S02	Thriller	50
F03	S03	Comedy	20

(Because of a transitive functional relationship, the table above is not in its 3NF.)

Id_show→Id_telecast

Id_telecast→Type_telecast

As a result, the following functional dependency is transitive.

1.Id_show→Type_telecast

Transitive Dependency

Avoiding Transitive Functional Dependency

According to the preceding statement, the relation <Telecast> violates the 3NF (3rd Normal Form). To address this violation, we must split the tables in order to remove the transitive functional relationship.

<show>

Id_show	Id_telecast	Cost_CD
F01	S01	30
F02	S02	50
F03	S03	20

<telecast>

Id_telecast	Type_telecast
S09	Thriller
S05	Romantic
S09	Comedy

The preceding relationship is now in the Third Normal Form (3NF) of Normalization.

Multivalued Dependency

The term Multivalued Dependency refers to having several rows in a particular table. As a result, it implies that there are multiple other rows in the same table. A multivalued dependency would thus preclude the 4NF. Any multivalued dependency would involve at least three table attributes.

When two separate attributes in a given table are independent of each other, multivalued dependency occurs. However, both of these are dependent on a third factor. At least two of the attributes are reliant on the third attribute in the multivalued dependence. This is why it always includes at least three of the qualities.

Example: Here we have a table **Car.**

<Car>

Model_car	Month_manu	Col_or
S2001	Jan	Yellow
S2002	Feb	Red
S2003	March	Yellow
S2004	April	Red
S2005	May	Yellow
S2006	June	Red

In this scenario, the columns Col_or and Month_manu are both dependent on Model-car but independently of one another. As a result, we can refer to both of these columns as multivalued. Thus, they are dependent on Model_car. Here is a diagram of the dependencies we covered earlier:

1.Model_car → →Month_manu

2.Model_car → → Col_or

Why Do We Use Multivalued Dependency in DBMS?

When we face these two different ways, we always employ multivalued conditions:
When we wish to test the relationships or determine whether they are legal under certain arrangements of practical and multivalued dependencies.
When we want to know what restrictions exist on the arrangement of legal relationships; as a result, we will only be concerned with the relationships that fulfil a specific arrangement of practical and multivalued dependencies.

Occurrence:

•When two qualities in a table are independent of each other yet reliant on a third property, this is referred to as multivalued dependence.

•Because multivalued Dependency requires a minimum of two variables that are independent of each other in order to be dependent on the third variable, the minimum number of variables necessary is two.

DBMS Dependency Conditions for Multivalued Dependency

If all of the following conditions are met, we can state that multivalued dependency exists.
If any attribute 'C' has many dependencies on 'D,' for any relation R, for all the pair data values in table row R1 and table row R2, such that the relation
R1[C]=R2[C]
exists, and there is a relationship between row R3 and row R4 in the table such that
R1[C] = R2[C] = R3[C] = R4[C]
R1[D] = R3[D], R2[D] = R4[D]
Then we can assert the existence of Multivalued Dependency (MVD).
That is, in Rows R1, R2, R3, and R4,
R1[C], R2[C], R3[C], and R4[C] must all have the same value.
The value of R1[D] should be equal to R3[D], and the value of R2[D] should be equal to R4[D].

DBMS Dependency Conditions for Multivalued Dependency

Example: Here, we have a table **Course.**

<Course>

Row_	Name_	Course_work_	Hobby_
R1	Ronit	Java	Dancing
R2	Ronit	Python	Singing
R3	Ronit	Java	Dancing
R4	Ronit	Python	Singing

DBMS Dependency Conditions for Multivalued Dependency

Because we have distinct values of Course_work_ and Hobby_ for the same value of Name "Ronit," we have multivalued dependents on Name_.

Verificationof <Course> table.

Let us now examine the condition of MVD(Multivalued Dependency) in our table.

Condition 1:

R1[C] = R2[C] = R3[C] = R4[C]

From the table;

R1[C] = R2[C] = R3[C] = R4[C] = 'Ronit'.

As a result, condition 1 appears to be met.

DBMS Dependency Conditions for Multivalued Dependency

Condition 2:

R1[D] = R3[D], R2[D] = R4[D]

From the table;

R1[D] = R3[D] = 'Java', R2[D] = R4[D]= 'Python'.

As a result, condition 2 appears to be met as well.

Condition 3:

R1[e] = R3[e], R2[e] = R4[e]

We can draw a conclusion from the table.

R1[E] = R3[E] = 'Dancing', R2[E] = R4[E] = 'Singing'.

As a result, condition 3 is likewise satisfied, indicating that MVD occurs in the provided situation.

We have now;

C →→ D

And from the table, we obtained the following;

Name_ →→Course_work_

And for C →→ E, we have

Name_ → Hobby_

Finally, in the given table, we can conclude with the conditional relation as

Name_ →→Course_work_

Name_ →→ Hobby_

Normalization

What Normalization Stands for?

Normalization is a method of organizing data in a database that helps to reduce data redundancy, insertion, update, and deletion errors. It is the process of assessing relation schemas based on functional relationships and primary keys.

This allows you to limit the amount of space a database takes up while also ensuring that the data is kept correctly.

Normalization Need

As previously stated, normalization is used to eliminate data redundancy. It offers a mechanism for removing the following anomalies from the database and making it more consistent:

A database anomaly is a fault in the database caused by insufficient preparation and redundancy.

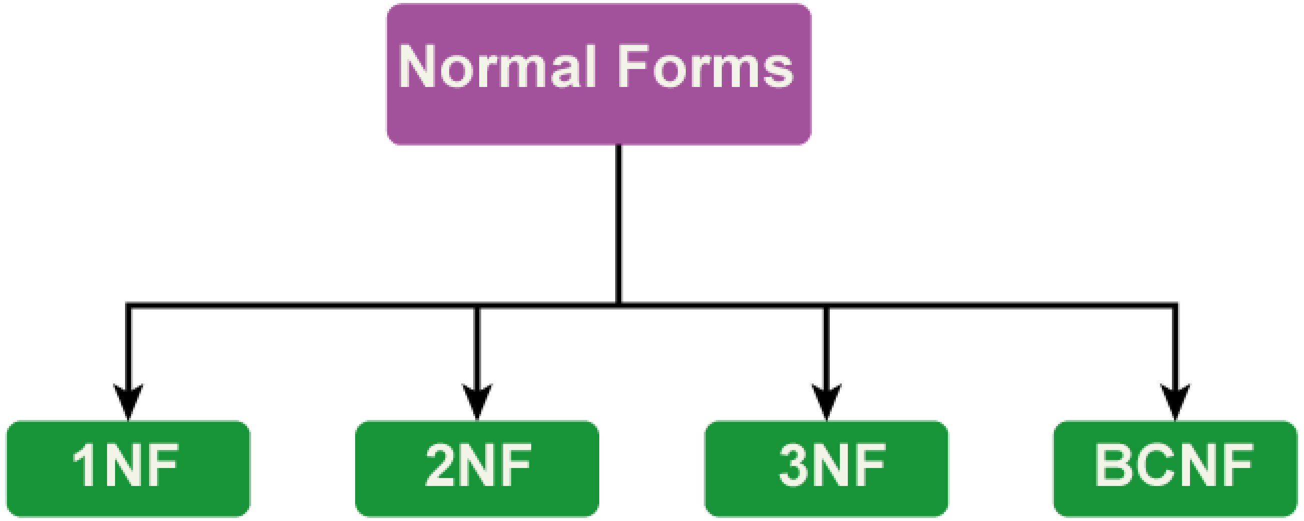
Insertion anomalies occur when we are unable to insert data into a database due to the absence of particular attributes at the moment of insertion.

Updation anomalies occur when the same data items with the same values are repeated but are not related to each other.

A **deletion anomaly** happens when deleting one part of the data results in the deletion of the other necessary information from the database.

Normal Forms

As illustrated in the image below, there are four types of normal forms that are commonly used in relational databases:



1NF, 2NF

First Normal Form (1NF):

A relation is in 1NF if all of its attributes are single-valued or if it lacks any multi-valued or composite attributes, i.e., every attribute is an atomic attribute.

The 1NF is violated if there is a composite or multi-valued attribute. To resolve this, we can construct a new row for each of the multi-valued attribute values in order to transform the table into the 1NF.

Second Normal Form (2NF):

Normalization of 1NF to 2NF relations entails the removal of incomplete dependencies. When any non-prime attributes, i.e., qualities that are not part of the candidate key, are not totally functionally reliant on one of the candidate keys, a partial dependency occurs.

To be in second normal form, a relational table must obey the following rules:

- The table must be presented in the first normal form.
- It must not have any partial dependencies, which means that all non-prime attributes must be totally functionally dependent on the primary key.

3NF

Third Normal Form (3NF):

Normalization of 2NF to 3NF relations entails the removal of transitive dependencies.

To be in the third normal form, a relational table must obey the following rules:

- The table should be in second normal form.
- There are no non-prime attributes that are transitively dependent on the primary key.
- At least one of the following conditions must be met for each functional dependency X -> Z:
- The table's super key is X.
- Z is a key feature of the table.

4NF

Boyce-Codd Normal Form (BCNF):

Boyce-Codd Normal Form is a more advanced variant of 3NF since it has more limitations than 3NF.

To be in Boyce-Codd normal form, a relational table must fulfil the following rules:

- The table must be in the "Third Normal Form".
- For every non-trivial functional dependency X -> Y, X is the table's superkey. That is, if Y is a prime attribute, X cannot be a non-prime attribute.

References

1. https://www.javatpoint.com/dependency-in-dbms
2. https://www.geeksforgeeks.org/types-of-functional-dependencies-in-dbms/