CP & Proj.

Languages (SQL) Concept of Functional dependencies & Multi-valued dependencies

What is Functional Dependency

Functional dependency in DBMS, as the name suggests is a relationship between attributes of a table dependent on each other. Introduced by E. F. Codd, it helps in preventing data redundancy and gets to know about bad designs.

Or we can define a *functional dependency* (FD) as a relationship between two attributes, typically between the PK and other non-key attributes within a table. For any relation R, attribute Y is functionally dependent on attribute X (usually the PK), if for every valid instance of X, that value of X uniquely determines the value of Y. This relationship is indicated by the representation below:

X ———> Y

The left side of the above FD diagram is called the *determinant*, and the right side is the *dependent*. Here are a few examples.

In the first example, below, SIN determines Name, Address and Birthdate. Given SIN, we can determine any of the other attributes within the table.

SIN ———> Name, Address, Birthdate

For the second example, SIN and Course determine the date completed (DateCompleted). This must also work for a composite PK.

SIN, Course ———> DateCompleted

The third example indicates that ISBN determines Title.

ISBN ———> Title

To understand the concept thoroughly, let us consider P is a relation with attributes A and B. Functional Dependency is represented by -> (arrow sign)

Then the following will represent the functional dependency between attributes with an arrow sign –

A -> B

Above suggests the following:

Functional Dependency

A -> B

B - functionally dependent on **A**
A - determinant set
B - dependent attribute

Other Examples

The following is an example that would make it easier to understand functional dependency –

We have a **<Department>** table with two attributes – **DeptId** and **DeptName.**

DeptId = Department ID
DeptName = Department Name

The **DeptId** is our primary key. Here, **DeptId** uniquely identifies the **DeptName** attribute. This is because if you want to know the department name, then at first you need to have the **DeptId.**

DeptId	**DeptName**
001	Finance
002	Marketing
003	HR

Therefore, the above functional dependency between **DeptId** and **DeptName** can be determined as **DeptId** is functionally dependent on **DeptName** –

Example:

Employee number	**Employee Name**	**Salary**	**City**
1	Dana	50000	San Francisco
2	Francis	38000	London
3	Andrew	25000	Tokyo

In this example, if we know the value of Employee number, we can obtain Employee Name, city, salary, etc. By this, we can say that the city, Employee Name, and salary are functionally depended on Employee number.

Key terms

Here, are some key terms for functional dependency:

Key Terms	Description
Axiom	Axioms is a set of inference rules used to infer all the functional dependencies on a relational database.
Decomposition	It is a rule that suggests if you have a table that appears to contain two entities which are determined by the same primary key then you should consider breaking them up into two different tables.
Dependent	It is displayed on the right side of the functional dependency diagram.
Determinant	It is displayed on the left side of the functional dependency Diagram.
Union	It suggests that if two tables are separate, and the PK is the same, you should consider putting them. together

Rules of Functional Dependencies

Consider the following table of data r(R) of the relation schema R(ABCDE) shown in Table 1

A	B	C	D	E
a1	b1	c1	d1	e1
a2	b1	c2	d2	e1
a3	b2	c1	d1	e1
a4	b2	c2	d2	e1
a5	b3	c3	d1	e1

Table R

Table 1 Functional dependency example, by A. Watt.

As you look at this table, ask yourself: *What kind of dependencies can we observe among the attributes in Table R?* Since the values of A are unique (a1, a2, a3, etc.), it follows from the FD definition that:

$A \rightarrow B$, $A \rightarrow C$, $A \rightarrow D$, $A \rightarrow E$

- It also follows that $A \rightarrow BC$ (or any other subset of ABCDE).
- This can be summarized as $A \rightarrow BCDE$.
- From our understanding of primary keys, A is a primary key.

Since the values of E are always the same (all e1), it follows that:

$A \rightarrow E$, $B \rightarrow E$, $C \rightarrow E$, $D \rightarrow E$

However, we cannot generally summarize the above with $ABCD \rightarrow E$ because, in general, $A \rightarrow E$, $B \rightarrow E$, $AB \rightarrow E$.

Other observations:

1. Combinations of BC are unique, therefore $BC \rightarrow ADE$.
2. Combinations of BD are unique, therefore $BD \rightarrow ACE$.
3. If C values match, so do D values.
    1. Therefore, $C \rightarrow D$
    2. However, D values don't determine C values
    3. So C does not determine D, and D does not determine C.

Looking at actual data can help clarify which attributes are dependent and which are determinants.

Inference Rules

Armstrong's axioms are a set of inference rules used to infer all the functional dependencies on a relational database. They were developed by William W. Armstrong. The following describes what will be used, in terms of notation, to explain these axioms.

Let R(U) be a relation scheme over the set of attributes U. We will use the letters X, Y, Z to represent any subset of and, for short, the union of two sets of attributes, instead of the usual $X \cup Y$.

Axiom of reflexivity

This axiom says, if Y is a subset of X, then X determines Y (see Figure 2).

IF $Y \subseteq X$, then $X \longrightarrow Y$

Figure 2 Equation for axiom of reflexivity.

For example, **PartNo —> NT123** where X (PartNo) is composed of more than one piece of information; i.e., Y (NT) and partID (123).

Axiom of augmentation

The axiom of augmentation, also known as a partial dependency, says if X determines Y, then XZ determines YZ for any Z (see Figure 3)

IF X ⟶ Y, then XZ ⟶ YZ for any Z

Figure 3. Equation for axiom of augmentation.

The axiom of augmentation says that every non-key attribute must be fully dependent on the PK. In the example shown below, StudentName, Address, City, Prov, and PC (postal code) are only dependent on the StudentNo, not on the StudentNo and Grade.

StudentNo, Course —> StudentName, Address, City, Prov, PC, Grade, DateCompleted

This situation is not desirable because every non-key attribute has to be fully dependent on the PK. In this situation, student information is only partially dependent on the PK (StudentNo).

To fix this problem, we need to break the original table down into two as follows:

- Table 1: StudentNo, Course, Grade, DateCompleted
- Table 2: StudentNo, StudentName, Address, City, Prov, PC

Axiom of transitivity

The axiom of transitivity says if X determines Y, and Y determines Z, then X must also determine Z (see Figure 4).

IF X ⟶ Y and Y ⟶ Z then X ⟶ Z

Figure 4. Equation for axiom of transitivity.

The table below has information not directly related to the student; for instance, ProgramID and ProgramName should have a table of its own. ProgramName is not dependent on StudentNo; it's dependent on ProgramID.

StudentNo —> StudentName, Address, City, Prov, PC, ProgramID, ProgramName

This situation is not desirable because a non-key attribute (ProgramName) depends on another non-key attribute (ProgramID).

To fix this problem, we need to break this table into two: one to hold information about the student and the other to hold information about the program.

- Table 1: StudentNo —> StudentName, Address, City, Prov, PC, ProgramID
- Table 2: ProgramID —> ProgramName

However we still need to leave an FK in the student table so that we can identify which program the student is enrolled in.

Union

This rule suggests that if two tables are separate, and the PK is the same, you may want to consider putting them together. It states that if X determines Y and X determines Z then X must also determine Y and Z (see Figure 5).

IF X → Y and X → Z, then X → YZ

Figure 5. Equation for the Union rule.

For example, if:

- SIN —> EmpName
- SIN —> SpouseName

You may want to join these two tables into one as follows:

SIN –> EmpName, SpouseName

Some database administrators (*DBA*) might choose to keep these tables separated for a couple of reasons. One, each table describes a different entity so the entities should be kept apart. Two, if SpouseName is to be left NULL most of the time, there is no need to include it in the same table as EmpName.

Decomposition

Decomposition is the reverse of the Union rule. If you have a table that appears to contain two entities that are determined by the same PK, consider breaking them up into two tables. This rule states that if X determines Y and Z, then X determines Y and X determines Z separately (see Figure 6).

IF X → YZ the X → Y and X → Z

Figure 6. Equation for decomposition rule.

Dependency Diagram

A dependency diagram, shown in Figure 11.6, illustrates the various dependencies that might exist in a *non-normalized table*. A non-normalized table is one that has data redundancy in it.

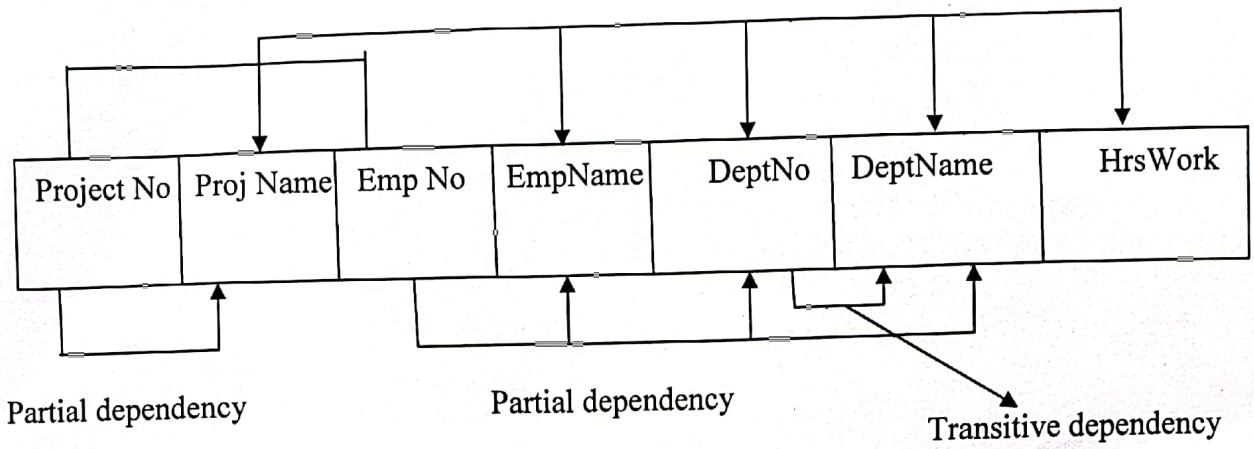

Figure 7. Dependency diagram.

The following dependencies are identified in this table:

- ProjectNo and EmpNo, combined, are the PK.
- Partial Dependencies:
  - ProjectNo —> ProjName
  - EmpNo —> EmpName, DeptNo,
  - ProjectNo, EmpNo —> HrsWork
- Transitive Dependency:
  - DeptNo —> DeptName

Key Terms

Armstrong's axioms: a set of inference rules used to infer all the functional dependencies on a relational database**DBA**: database administrator

decomposition: a rule that suggests if you have a table that appears to contain two entities that are determined by the same PK, consider breaking them up into two tables

dependent: the right side of the functional dependency diagram

determinant: the left side of the functional dependency diagram

functional dependency (FD): a relationship between two attributes, typically between the PK and other non-key attributes within a table

non-normalized table: a table that has data redundancy in it

Union: a rule that suggests that if two tables are separate, and the PK is the same, consider putting them together

Below given are the Three most important rules for Functional Dependency:

- Reflexive rule –. If X is a set of attributes and Y is_subset_of X, then X holds a value of Y.
- Augmentation rule: When x -> y holds, and c is attribute set, then ac -> bc also holds. That is adding attributes which do not change the basic dependencies.
- Transitivity rule: This rule is very much similar to the transitive rule in algebra if x -> y holds and y -> z holds, then x -> z also holds. X -> y is called as functionally that determines y.

Types of Functional Dependencies

- **Multivalued dependency:**
- **Trivial functional dependency:**
- **Non-trivial functional dependency:**
- **Transitive dependency:**

Multivalued dependency in DBMS

Multivalued dependency occurs in the situation where there are multiple independent multivalued attributes in a single table. A multivalued dependency is a complete constraint between two sets of attributes in a relation. It requires that certain tuples be present in a relation.

Example:

Car_model	Maf_year	Color
H001	2017	Metallic
H001	2017	Green
H005	2018	Metallic
H005	2018	Blue
H010	2015	Metallic
H033	2012	Gray

In this example, maf_year and color are independent of each other but dependent on car_model. In this example, these two columns are said to be multivalue dependent on car_model.

This dependence can be represented like this:

car_model -> maf_year

car_model-> colour

Trivial Functional dependency:

The Trivial dependency is a set of attributes which are called a trivial if the set of attributes are included in that attribute.

So, X -> Y is a trivial functional dependency if Y is a subset of X.

For example:

Emp_id	Emp_name
AS555	Harry
AS811	George
AS999	Kevin

Consider this table with two columns Emp_id and Emp_name.

{Emp_id, Emp_name} -> Emp_id is a trivial functional dependency as Emp_id is a subset of {Emp_id,Emp_name}.

Non trivial functional dependency in DBMS

Functional dependency which also known as a nontrivial dependency occurs when A->B holds true where B is not a subset of A. In a relationship, if attribute B is not a subset of attribute A, then it is considered as a non-trivial dependency.

Company	CEO	Age
Microsoft	Satya Nadella	51
Google	Sundar Pichai	46
Apple	Tim Cook	57

Example:

(Company} -> {CEO} (if we know the Company, we knows the CEO name)

But CEO is not a subset of Company, and hence it's non-trivial functional dependency.

Transitive dependency:

A transitive is a type of functional dependency which happens when t is indirectly formed by two functional dependencies.

Example:

Company	CEO	Age
Microsoft	Satya Nadella	51
Google	Sundar Pichai	46
Alibaba	Jack Ma	54

{Company} -> {CEO} (if we know the compay, we know its CEO's name)

{CEO } -> {Age} If we know the CEO, we know the Age

Therefore according to the rule of rule of transitive dependency:

{ Company} -> {Age} should hold, that makes sense because if we know the company name, we can know his age.

Note: You need to remember that transitive dependency can only occur in a relation of three or more attributes.

What is Normalization?

Normalization is a method of organizing the data in the database which helps you to avoid data redundancy, insertion, update & deletion anomaly. It is a process of analyzing the relation schemas based on their different functional dependencies and primary key.

Normalization is inherent to relational database theory. It may have the effect of duplicating the same data within the database which may result in the creation of additional tables.

Advantages of Functional Dependency

- Functional Dependency avoids data redundancy. Therefore same data do not repeat at multiple locations in that database
- It helps you to maintain the quality of data in the database
- It helps you to defined meanings and constraints of databases
- It helps you to identify bad designs
- It helps you to find the facts regarding the database design

Transaction Management in DBMS

A **transaction** is a set of logically related operations. For example, you are transferring money from your bank account to your friend's account, the set of operations would be like this:

Simple Transaction Example

1. Read your account balance
2. Deduct the amount from your balance
3. Write the remaining balance to your account
4. Read your friend's account balance
5. Add the amount to his account balance
6. Write the new updated balance to his account

This whole set of operations can be called a transaction. Although I have shown you read, write and update operations in the above example but the transaction can have operations like read, write, insert, update, delete.

In DBMS, we write the above 6 steps transaction like this:
Lets say your account is A and your friend's account is B, you are transferring 10000 from A to B, the steps of the transaction are:

```
1. R(A);
2. A = A - 10000;
3. W(A);
4. R(B);
5. B = B + 10000;
6. W(B);
```

In the above transaction **R** refers to the **Read operation** and **W** refers to the **write operation**.

Transaction failure in between the operations

Now that we understand what is transaction, we should understand what are the problems associated with it.

The main problem that can happen during a transaction is that the transaction can fail before finishing the all the operations in the set. This can happen due to power failure, system crash etc. This is a serious problem that can leave database in an inconsistent state. Assume that transaction fail after third operation (see the example above) then the amount would be deducted from your account but your friend will not receive it.

To solve this problem, we have the following two operations

Commit: If all the operations in a transaction are completed successfully then commit those changes to the database permanently.

Rollback: If any of the operation fails then rollback all the changes done by previous operations.

Even though these operations can help us avoiding several issues that may arise during transaction but they are not sufficient when two transactions are running concurrently. To handle those problems we need to understand database ACID properties.