Anomalies in DBMS

There are three types of anomalies that occur when the database is not normalized. These are – Insertion, update and deletion anomaly. Let's take an example to understand this.

Example: Suppose a manufacturing company stores the employee details in a table named employee that has four attributes: emp_id for storing employee's id, emp_name for storing employee's name, emp_address for storing employee's address and emp_dept for storing the department details in which the employee works. At some point of time the table looks like this:

emp_id	emp_name	emp_address	emp_dept
101	Rick	Delhi	D001
101	Rick	Delhi	D002
123	Maggie	Agra	D890
166	Glenn	Chennai	D900
166	Glenn	Chennai	D004

The above table is not normalized. We will see the problems that we face when a table is not normalized.

Update anomaly: In the above table we have two rows for employee Rick as he belongs to two departments of the company. If we want to update the address of Rick then we have to update the same in two rows or the data will become inconsistent. If somehow, the correct address gets updated in one department but not in other then as per the database, Rick would be having two different addresses, which is not correct and would lead to inconsistent data.

Insert anomaly: Suppose a new employee joins the company, who is under training and currently not assigned to any department then we would not be able to insert the data into the table if emp_dept field doesn't allow nulls.

Delete anomaly: Suppose, if at a point of time the company closes the department D890 then deleting the rows that are having emp_dept as D890 would also delete the information of employee Maggie since she is assigned only to this department.

To overcome these anomalies we need to normalize the data. In the next section we will discuss about normalization.

A functional dependency is an association between two attributes of the same relational database table. One of the attributes is called the determinant and the other attribute is called the determined. For each value of the determinant there is associated one and only one value of the determined.

If A is the determinant and B is the determined then we say that *A functionally determines B* and graphically represent this as A -> B. The symbols A à B· can also be expressed as *B is functionally determined by* A.

Example

The following table illustrates A ⟹ B:

A	B
1	1
2	4
3	9
4	16
2	4
7	9

Since for each value of A there is associated one and only one value of B.

Example

The following table illustrates that A does not functionally determine B:

A	B
1	1
2	4
3	9
4	16
3	10

Since for A = 3 there is associated more than one value of B.

Functional dependency can also be defined as follows:

An attribute in a relational model is said to be functionally dependent on another attribute in the table if it can take only one value for a given value of the attribute upon which it is functionally dependent.

Example: Consider the database having following tables:

The Supplier Table

SNo.	Sname	Status	City
S1	Suneet	20	Qadlan
S2	Ankit	10	Amritsar
S3	Amit	10	Amritsar

The Part Table

PNo.	Pname	Color	Weight	City
P1	Nut	Red	12	Qadlan
P2	Bolt	Green	17	Amritsar
P3	Screw	Blue	17	Amritsar
P4	Screw	Red	14	Qadlan

The Shipment Table

SNo.	Pno.	Qty
S1	P1	270
S1	P2	300
S1	P3	700
S2	P1	270
S2	P2	700
S3	P2	300

Here in Supplier table

Sno - Supplier number of supplier that is unique

Sname - Supplier name

City - City of the supplier

Status - Status of the city e.g. A grade cities may have status 10, B grad cities may have status 20 and so on.

Here, Sname is FD on Sno. Because, Sname can take only one value for the given value of Sno (e.g. S 1) or in other words there must be one Sname for supplier number S1.

FD is represented as:

Sno à Sname

FD is shown by à which means that Sname is functionally dependent on Sno.

Similarly, city and status are also FD on Sno, because for each value of Sno there will be only one city and status.

FD is represented as:

Sno - City

Sno - Status

S. Sno - S (Sname, City, Status)

Consider another database of shipment with following attributes:

Sno - Supplier number of the supplier

Pno - Part number supplied by supplier

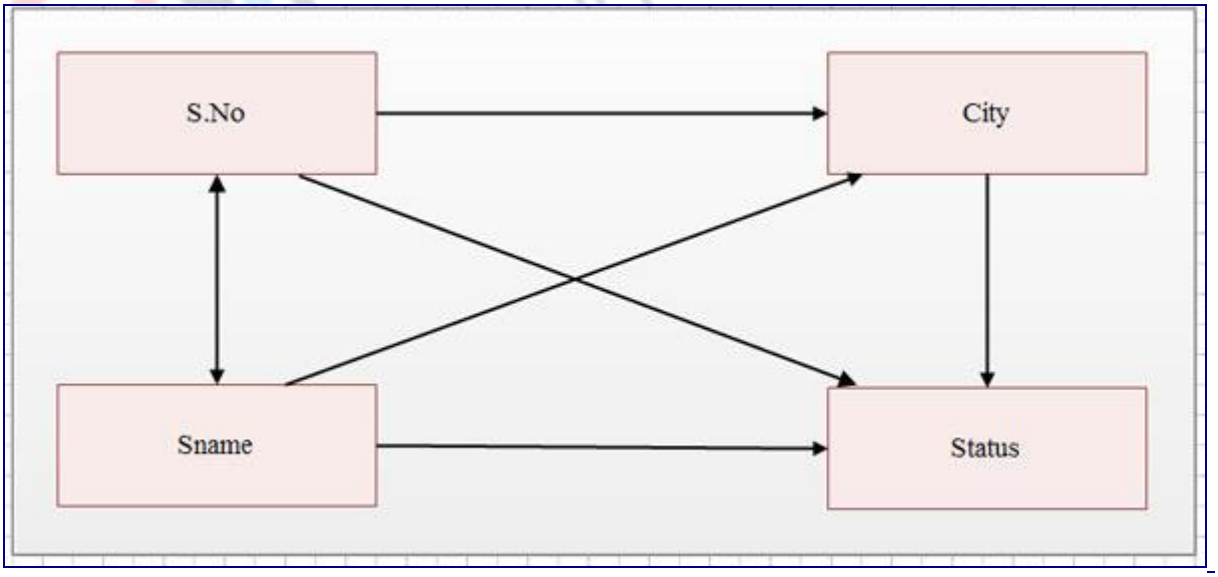
Qty - Quantity supplied by supplier for a particular Part no

In this case Qty is FD on combination of Sno, Pno because each combination of Sno and Pno results only for one Quantity.

SP (Sno, Pno) --> SP.QTY

Dependency Diagrams

A dependency diagram consists of the attribute names and all functional dependencies in a given table. The dependency diagram of Supplier table is.



Here, following functional dependencies exist in supplier table

Sno - Sname

Sname - Sno

Sno - City

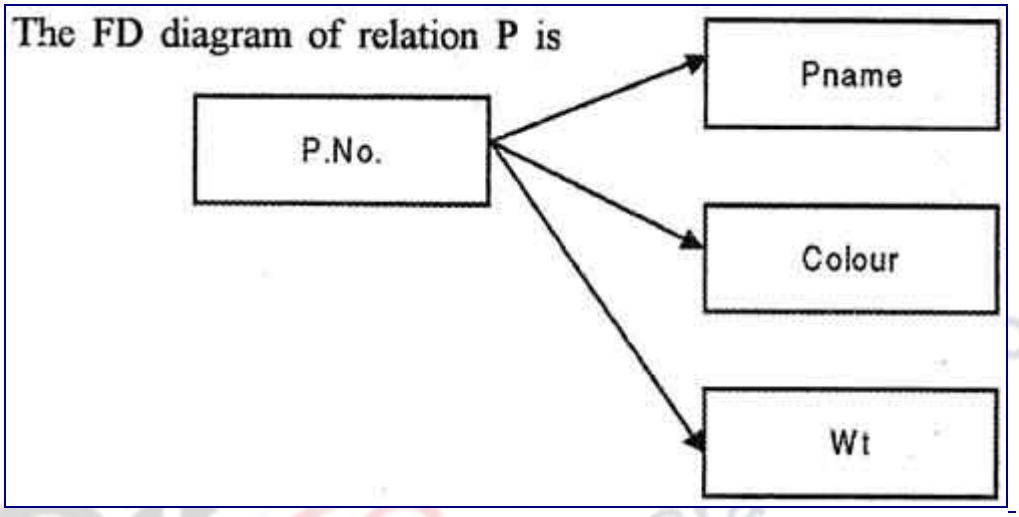
Sno - Status

Sname - City

Sname - Status

City - Status

The FD diagram of relation P is

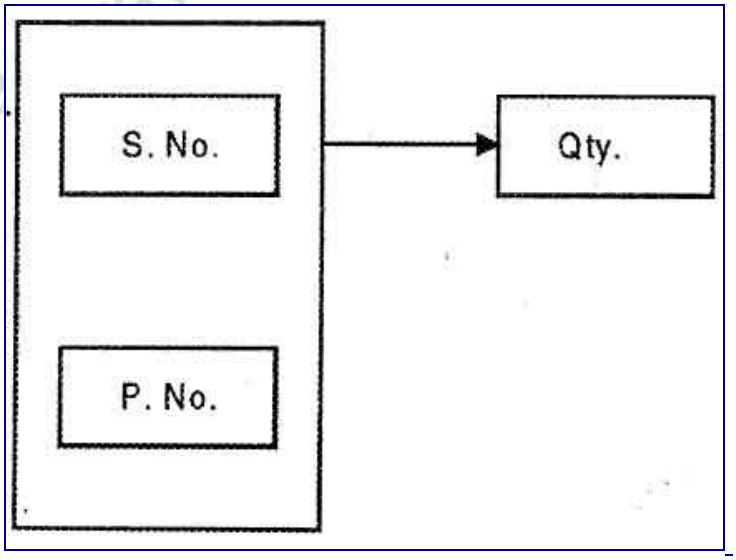


Here following functional dependencies exist in Part table:

Pno - Pname

Pno - Color

Pno - Wt



The FD diagram of relation Shipment is

Here following functional dependencies exist in parts table

SP (Sno, Pno) - SP.QTY

Fully Functional Dependence (FFD)

Fully Functional Dependence (FFD) is defined, as Attribute Y is FFD on attribute" X, if it is FD on X and not FD on any proper subset of X. For example, in relation Supplier, different cities may have the same status. It may be possible that cities like Amritsar, Jalandhar may have the same status 10.

So, the City is not FD on Status.

But, the combination of Sno, Status can give only one corresponding City ,because Sno" is unique. Thus,

(Sno, Status) à City

It means city is FD on composite attribute (Sno, Status) however City is not fully functional dependent on this composite attribute, which is explained below:

(Sno, Status) à City

X Y

Here Y is FD on X, but X has two proper subsets Sno and Status; city is· FD .on one proper subset .of X i.e. Sno

Sno à City

According to 'FFD definition Y must not be FD .on any proper subset of X, but here City is FD in one subset .of X i.e. Sno, so City is not FFD on (Sno, Status)

Consider another case of SP table:

Here, Qty is FD on combination of Sna, Pno.

(Sno, Pno) à Qty

X Y

Here, X has two proper subsets Sno and Pna

Qty is not FD on Sno, because one Sna can supply mare than .one quantity.

Qty is also not FD on Pno, because .one Pna may be supplied many times by different suppliers with different .or same quantities.

So, Qty is FFD and composite attribute of (Sno, Pno) à Qty.

Other Functional Dependencies

There are same rather types of functional dependencies, which play a vital rule during the process .of normalization of data.

Candidate Functional Dependency

A candidate functional dependency is a functional dependency that includes all attributes of the table. It should also be noted that a well-fanned dependency diagram must have at least one candidate functional dependency, and that there can be more than .one candidate functional dependency for a given dependency diagram.

Primary Functional Dependency

A primary functional dependency is a candidate functional dependency that is selected to determine the primary key. The determinant of the primary functional dependency is the primary key of the relational database table. Each dependency diagram must have one and only on primary functional dependency. If a relational database table has .only .one candidate functional dependency, then it automatically becomes the primary functional dependency

Once the primary key has been determined, there will be three possible types of functional dependencies:

Description

A à B A key attribute functionally determines a non-key attribute.

A à B A non-key attribute functionally determines a non-key attribute.

A à B A non-key attribute functionally determines a key attribute.

A **partial functional dependency** is a functional dependency where the determinant consists of key attributes, but not the entire primary key, and the determined consist~ of non-key attributes.

A **transitive functional dependency** is a functional dependency where the determinant consists of non-key attributes and the determined also consists of non-key attributes.

A **Boyce-Codd functional dependency** is a functional dependency where the determinant consists of non-key attributes and the determined consists of key attributes.

A **Multi-Value Dependency (MVD)** occurs when two or more independent multi valued facts about the same attribute occur within the same table. It means that if in a relation R having A, Band C as attributes, B and Care multi-value facts about A, which is represented as A ààB and A ààC ,then multi value dependency exist only if B and C are independent on each other.

A **Join Dependency** exists if a relation R is equal to the join of the projections X Z. where X, Y, Z projections of R.

Closure of set of dependencies

Let a relation *R* have some functional dependencies *F* specified. The *closure of F* (usually written as *F*+) is the set of all functional dependencies that may be logically derived from *F.* Often *F* is the set of most obvious and important functional dependencies and. *F*+, the closure, is the set of all the functional dependencies including *F* and those that can be deduced from *F.* The closure is important and may, for example, be needed in finding one or more candidate keys of the relation.

For example, the *student* relation has the following functional dependencies

sno à *Sname*

cno à *came*

sno à *address*

cno à *instructor*

Instructor à office

Let these dependencies be denoted by *F.* The closure of *F,* denoted by *F* +, includes *F* and all functional- dependencies that are implied by *F.*

To determine *F*+, we need rules for deriving all functional dependencies that are implied: by *F.* A set of rules that may be used to infer additional dependencies was proposed by Armstrong in 1974. These rules (or axioms) are a complete set of rules in· that all possible functional dependencies may be derived from them. The rules are:

1. Reflexivity Rule - If *X* is a set of attributes and Y is a subset of *X,* then *X àY* holds.

The reflexivity rule is the simplest (almost trivial) rule. It states that each subset of *X* is functionally dependent on *X.* In other words trivial dependence is defined as follows:

Trivial functional dependency: A trivial functional dependency is a functional dependency of an attribute on a superset of itself.

For example: {Employee ID, Employee Address} à {Employee Address} is trivial, here {Employee

Address} is a subset of {Employee ID, Employee Address}.

2. Augmentation Rule - If *X à Y* holds and *W* is a set of attributes, and then *WX à WY* holds.

The argumentation ('u rule is also quite simple. It states that if *Y* is determined by *X* then a set of attributes *W* and *Y* together will be determined by *W* and *X* together. Note that we use the notation *WX* to mean the collection of all attributes in *W* and *X* and write *WX* rather than the more conventional *(W, X)* for convenience.

For example: Rno - Name; Class and Marks is a set of attributes and act as

W. Then· {Rno, Class, Marks} -> {Name, Class, Marks}

3. Transitivity Rule - If *X* -> Y and Y -> *Z* hold, then *X* -> *Z* holds.

The transitivity rule is perhaps the most important one. It states that if *X* functionally determines Y and Y functionally determine *Z* then *X* functionally determines *Z.*

For example: Rno -> City and City -> Status, then Rno -> Status should be holding true.

These rules are called *Armstrong's Axioms.*

Further axioms may be derived from the above although the above three axioms are *sound and complete* in that they do not generate any incorrect functional dependencies (soundness) and they do generate all possible functional dependencies that can be inferred from *F* (completeness). The most important additional axioms are:

1. Union Rule - If *X* -> *Y* and *X* -> *Z* hold, then *X* -> *YZ* holds.

2. Decomposition Rule - If *X à YZ* holds, then so do *X à Y* and *X à* Z.

3. Pseudotransitivity Rule - If *X à Y* and *WY à* Z hold then so does *WX àZ.*

Based on the above axioms and the .functional dependencies specified for relation *student*, we may write a large number of functional dependencies. Some of these are:

(sno, cno) à sno (Rule 1)

(sno, cno) à cno (Rule 1)

(sno, cno) à (Sname, cname) (Rule 2)

cno à office (Rule 3)

sno à (Sname, address) (Union Rule)

Etc.

Often a very large list of dependencies can be derived from a given set *F* since Rule 1 itself will lead to a large number of dependencies. Since we have seven attributes *(sno, Sname, address, cno, cname, instructor, office),* there are 128 (that is, 2^7) subsets of these attributes. These 128 subsets could form 128 values of *X* in functional dependencies of the type *X ~ Y.* Of course, each value of *X* will then be associated with a number of values for *Y (Y* being a subset of x) Leading to several thousand dependencies. These large numbers of dependencies are not particularly helpful in achieving our aim of normalizing relations.

Although we could follow the present procedure and compute the closure of *F* to find all the functional dependencies, the computation requires exponential time and the list of dependencies is often very large and therefore not very useful. There are two possible approaches that can be taken to avoid dealing with the large number of dependencies in the closure. 'One' is to deal with one attribute or a set of attributes at a time and find its closure (i.e. all functional dependencies relating to them). The aim of this exercise is to find what attributes depend on a given set of attributes and therefore ought to be together. The other approach is to find the *minimal· covers.*

Minimal Functional Dependencies or Irreducible Set of Dependencies

In discussing the concept of equivalent FDs, it is useful to define the concept of *minimal functional dependencies* or *minimal cover* which is useful in eliminating necessary functional dependencies so that only the minimal numbers of dependencies need to be enforced by the system. The concept of minimal cover of *F* is sometimes called *irreducible Set* of *F.*

A functional depending set S is irreducible if the set has three following properties:

Each right set of a functional dependency of S contains only one attribute.

Each left set of a functional dependency of S is irreducible. It means that reducing anyone attribute from left set will change the content of S (S will lose some information).

Reducing any functional dependency will change the content of S.

Sets of functional dependencies with these properties are also called *canonical* or *minimal.*

If the value in a non-key attribute is determined by the value in another non-key attribute then that field has transitive dependency.

For example, look at the relation below:

```
course  (course_id
         course_title
         teacher_id
         teacher_name)
```

The attribute **teacher_name** is determined by the non-key attribute **teacher_id**, and not the primary key of **course_id**. This means that teacher_name is transitively dependent on the primary key of course_id.

In order to show a relation in 3NF, all transitive dependencies must be removed.

Functional Dependency

Functional dependency (FD) is a set of constraints between two attributes in a relation. Functional dependency says that if two tuples have same values for attributes A1, A2,..., An, then those two tuples must have to have same values for attributes B1, B2, ..., Bn.

Functional dependency is represented by an arrow sign (→) that is, X→Y, where X functionally determines Y. The left-hand side attributes determine the values of attributes on the right-hand side.

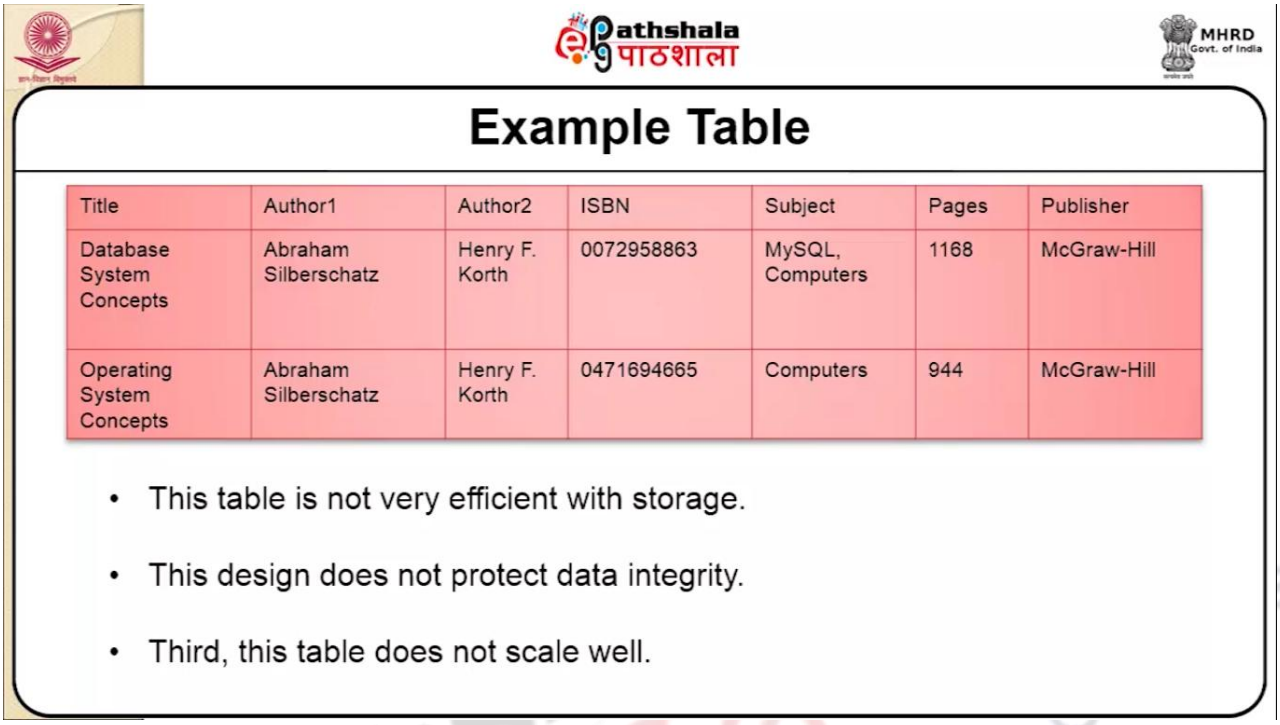
Armstrong's Axioms

If F is a set of functional dependencies then the closure of F, denoted as F^+, is the set of all functional dependencies logically implied by F. Armstrong's Axioms are a set of rules, that when applied repeatedly, generates a closure of functional dependencies.

- **Reflexive rule** − If alpha is a set of attributes and beta is_subset_of alpha, then alpha holds beta.
- **Augmentation rule** − If a → b holds and y is attribute set, then ay → by also holds. That is adding attributes in dependencies, does not change the basic dependencies.
- **Transitivity rule** − Same as transitive rule in algebra, if a → b holds and b → c holds, then a → c also holds. a → b is called as a functionally that determines b.

Trivial Functional Dependency

- **Trivial** − If a functional dependency (FD) X → Y holds, where Y is a subset of X, then it is called a trivial FD. Trivial FDs always hold.
- **Non-trivial** − If an FD X → Y holds, where Y is not a subset of X, then it is called a non-trivial FD.
- **Completely non-trivial** − If an FD X → Y holds, where x intersect Y = Φ, it is said to be a completely non-trivial FD.



Normalization

If a database design is not perfect, it may contain anomalies, which are like a bad dream for any database administrator. Managing a database with anomalies is next to impossible.

- **Update anomalies** − If data items are scattered and are not linked to each other properly, then it could lead to strange situations. For example, when we try to update one data item having its copies scattered over several places, a few instances get updated properly while a few others are left with old values. Such instances leave the database in an inconsistent state.
- **Deletion anomalies** − We tried to delete a record, but parts of it was left undeleted because of unawareness, the data is also saved somewhere else.
- **Insert anomalies** − We tried to insert data in a record that does not exist at all.

Normalization is a method to remove all these anomalies and bring the database to a consistent state.

Data redundancy is a condition created within a database or data storage technology in which the same piece of data is held in two separate places.

This can mean two different fields within a single database, or two different spots in multiple software environments or platforms. Whenever data is repeated, this basically constitutes data redundancy. This can occur by accident, but is also done deliberately for backup and recovery purposes.