What is Functional Dependency in DBMS?

Let us try to understand the concept thoroughly. If X is a relation that has attributes P and Q, then their functional dependency would be represented by -> (arrow sign)

Thus, here, the following would represent the functional dependency between the attributes using an arrow sign:

P -> Q

In this case, the left side of this arrow is a Determinant. The right side of this arrow is a Dependent. P will be the primary key attribute, while Q will be a dependent non-key attribute from a similar table as the primary key. It shows that the primary key attribute P is functionally dependent on the non-key attribute Q. In simpler words, If the column P attribute of a table identifies the column Q attribute of the very same table uniquely, then the functional dependency of column Q on column P is symbolized as P → Q.

Example

Let us look at an example that makes it easier to comprehend functional dependency.

Suppose we have a <Student> table with two separate attributes – Stu_Id and Stu_Name.

Stu_Id = Student ID

Stu_Name = Student Name

The StuId is our primary key. And StuId here identifies the StuName attribute uniquely. It is because if someone wants to know the student's name, then you need to have the StuId at first.

Stu_Id	**Stu_Name**
011	Marketing
022	HR
033	Finance

044	Accounting
055	Sales
066	Telecom

The functional dependency given above between Stu_Id and Stu_Name can be specified as Stu_Id is functionally dependent on Stu_Name.

Stu_Id -> Stu_Name

Types of Functional dependencies in DBMS

A functional dependency is a constraint that specifies the relationship between two sets of attributes where one set can accurately determine the value of other sets. It is denoted as **X** → **Y**, where X is a set of attributes that is capable of determining the value of Y. The attribute set on the left side of the arrow, **X** is called **Determinant**, while on the right side, **Y** is called the **Dependent**. Functional dependencies are used to mathematically express relations among database entities and are very important to understand advanced concepts in Relational Database System.

Example:

roll_no	name	dept_name	dept_building
42	abc	CO	A4
43	pqr	IT	A3
44	xyz	CO	A4
45	xyz	IT	A3
46	mno	EC	B2
47	jkl	ME	B2

From the above table we can conclude some valid functional dependencies:

- **roll_no → { name, dept_name, dept_building },→ Here, roll_no can determine values of fields name, dept_name and dept_building, hence a valid Functional dependency**
- **roll_no → dept_name , Since, roll_no can determine whole set of {name, dept_name, dept_building}, it can determine its subset dept_name also.**
- **dept_name → dept_building , Dept_name can identify the dept_building accurately, since departments with different dept_name will also have a different dept_building**
- **More valid functional dependencies: roll_no → name, {roll_no, name} ⇢ {dept_name, dept_building}, etc.**

Here are some invalid functional dependencies:

- **name → dept_name Students with the same name can have different dept_name, hence this is not a valid functional dependency.**
- **dept_building → dept_name There can be multiple departments in the same building, For example, in the above table departments ME and EC are in the same building B2, hence dept_building → dept_name is an invalid functional dependency.**
- **More invalid functional dependencies: name → roll_no, {name, dept_name} → roll_no, dept_building → roll_no, etc.**

Armstrong's axioms/properties of functional dependencies:

1. **Reflexivity: If Y is a subset of X, then X→Y holds by reflexivity rule For example, {roll_no, name} → name is valid.**
2. **Augmentation: If X → Y is a valid dependency, then XZ → YZ is also valid by the augmentation rule.**
   For example, If {roll_no, name} → dept_building is valid, hence {roll_no, name, dept_name} → {dept_building, dept_name} is also valid.→
3. **Transitivity: If X → Y and Y → Z are both valid dependencies, then X→Z is also valid by the Transitivity rule.**
   For example, roll_no → dept_name & dept_name → dept_building, then roll_no → dept_building is also valid.

Types of Functional dependencies in DBMS:

1. **Trivial functional dependency**

2. **Non-Trivial functional dependency**
3. **Multivalued functional dependency**
4. **Transitive functional dependency**

1. Trivial Functional Dependency

In Trivial Functional Dependency, a 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

For example,

For example,

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.

For example,

For example,

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 each other.

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

For example,

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 !)

4. Transitive Functional Dependency

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

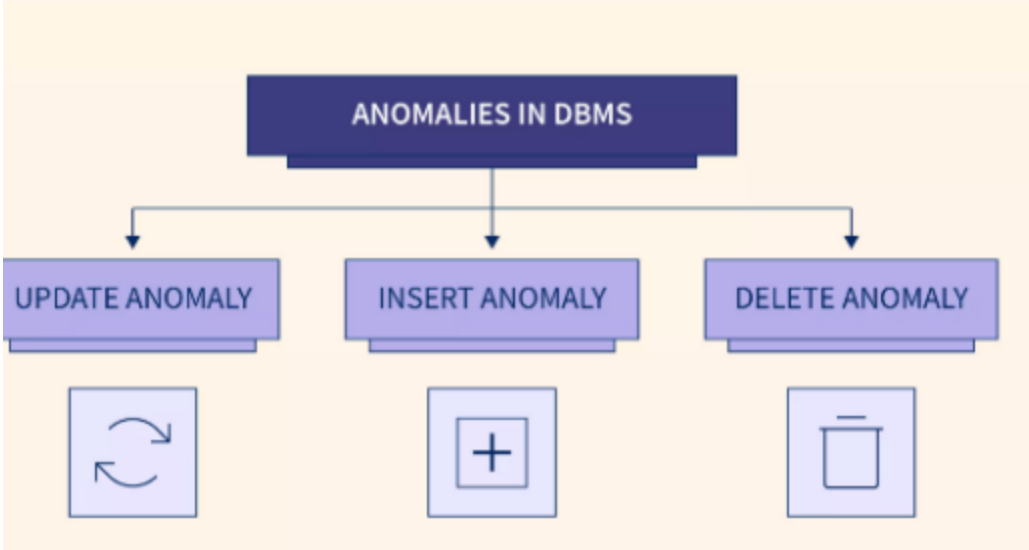
i.e. If a → b & b → c, then according to axiom of transitivity, a → c. This is a transitive functional dependency

For example,

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. This is an indirect functional dependency, hence called Transitive functional dependency.

Anomalies in DBMS



Overview

The term anomaly is used to describe a discrepancy between two parts of a database. In a retail database, for example, you may have a customer and an invoice table. If you are no longer selling to customers, you may want to purge them from your database periodically. If you delete a customer but keep their invoices for the purchases they have made, you have an anomaly. There is an invoice for a customer that doesn't exist anymore. By deleting all of a customer's invoices when you delete a customer, this anomaly can be avoided.

What are the Anomalies in DBMS?

A database anomaly is an inconsistency in the data resulting from an operation like an update, insertion, or deletion. There can be inconsistencies when a record is held in multiple places and not all of the copies are updated.

Anomalies occur when the data present in the database has too much redundancy and if the tables making up the database are poorly constructed

How are Anomalies Caused in DBMS?

Poorly constructed tables in a database are often the reason behind anomalies. How do you define "poor construction"? A poorly designed table becomes apparent if, when a designer is creating the database, he fails to identify the entities which are interdependent, such as rooms of a hostel and the hostel, and then minimizes the chances of an entity being independent of another.

A database anomaly is a fault within a database, which can occur because of poor planning or when everything is stored in a flat database. A normalization procedure, which combines and splits tables, is usually sufficient to remove this. By normalizing the database, we reduce the likelihood of creating tables that generate anomalies.

Type of Anomalies in DBMS

There are different types of anomalies that can occur in a database. Redundancy anomalies`, for instance, can cause problems during tests if you are a student, and during job interviews, if you are looking for work. However, they are easily spotted and fixed. These are the ones we need to pay attention to:

1. Update
2. Insert
3. Delete

Database anomalies fall into these three major categories:

1. Update anomaly:

Consider a college database that keeps student information in a table called student, which contains four columns: stu_id for the student's id, stu_name for the student's name, stu_address for the student's address, and stu_club for the student's club. Eventually, the table will appear as follows:

stu_id	stu_name	stu_address	stu_club
330	Muthu	Rajasthan	Literature
330	Muthu	Rajasthan	Finance
331	Mukesh	Mumbai	Crypto
332	Nanda	Karnataka	Public Speaking
332	Nanda	Karnataka	Arts

For student Muthu, we have two columns in the above table as he belongs to two clubs at the college. If we want to change Muthu's address, we must update it twice otherwise the data will be inconsistent.

When the correct address gets updated in one club but not in another, Muthu would possess two different addresses, which is not acceptable and could result in inconsistent data.

2. Insert anomaly:

Example

We use the same table in the previous example with modified data

stu_id	stu_name	stu_address	stu_club
220	Annamalai	Kerala	yoga
220	Muthu	Kerala	Music
231	Mukesh	Mumbai	Crypto
232	Muni	Karnataka	Public Speaking
232	Muni	Karnataka	Arts

For example, in the above table if a new student named Nanda has joined the college and he has no department affiliation as the club allows intake of students only from second year. Then we can't insert the data of Nanda into the table since the st_club field cannot accept null values.

3. Delete anomaly:

Example In this example, we use modified data from the previous example

stu_id	stu_name	stu_address	stu_club
120	Nanthu	Maharasthra	yoga
122	Nanthu	Maharashthra	Music
131	Mukesh	Mumbai	Crypto
132	Muni	Karnataka	Public Speaking
132	Muni	Karnataka	Arts

Suppose, for instance, the college at some point closes the club crypto, then deleting the rows that contain s_club as crypto would also delete the information of student Mukesh since he belongs only to this department.

Conclusion

- A database anomaly usually occurs as a result of poor planning and the use of flat databases.
- In this article we learned about insert, delete and update anomalies as well as the circumstances that can lead to them.
- Anomalies are usually removed by normalizing the tables by splitting or joining them
- Normalization provides structured data

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**

What is Normalization in DBMS (SQL)? 1NF, 2NF, 3NF Example

What is Database Normalization?

Normalization is a database design technique that reduces data redundancy and eliminates undesirable characteristics like Insertion, Update and Deletion Anomalies. Normalization rules divides larger

tables into smaller tables and links them using relationships. The purpose of Normalisation in SQL is to eliminate redundant (repetitive) data and ensure data is stored logically.
The inventor of the relational model Edgar Codd proposed the theory of normalization of data with the introduction of the First Normal Form, and he continued to extend theory with Second and Third Normal Form. Later he joined Raymond F. Boyce to develop the theory of Boyce-Codd Normal Form.

Database Normal Forms

Here is a list of Normal Forms in SQL:

- **1NF (First Normal Form)**
- **2NF (Second Normal Form)**
- **3NF (Third Normal Form)**
- **BCNF (Boyce-Codd Normal Form)**
- **4NF (Fourth Normal Form)**
- **5NF (Fifth Normal Form)**
- **6NF (Sixth Normal Form)**

Database Normalization With Examples

Database Normalization Example can be easily understood with the help of a case study. Assume, a video library maintains a database of movies rented out. Without any normalization in database, all information is stored in one table as shown below. Let's understand Normalization database with normalization example with solution:

FULL NAMES	PHYSICAL ADDRESS	MOVIES RENTED	SALUTATION
Janet Jones	First Street Plot No 4	Pirates of the Caribbean, Clash of the Titans	Ms.
Robert Phil	3rd Street 34	Forgetting Sarah Marshal, Daddy's Little Girls	Mr.
Robert Phil	5th Avenue	Clash of the Titans	Mr.

Here you see Movies Rented column has multiple values. Now let's move into 1st Normal Forms:

1NF (First Normal Form) Rules

- **Each table cell should contain a single value.**
- **Each record needs to be unique.**

The above table in 1NF-

1NF Example

FULL NAMES	PHYSICAL ADDRESS	MOVIES RENTED	SALUTATION
Janet Jones	First Street Plot No 4	Pirates of the Caribbean	Ms.
Janet Jones	First Street Plot No 4	Clash of the Titans	Ms.
Robert Phil	3rd Street 34	Forgetting Sarah Marshal	Mr.
Robert Phil	3rd Street 34	Daddy's Little Girls	Mr.
Robert Phil	5th Avenue	Clash of the Titans	Mr.

Example of 1NF in DBMS

Before we proceed let's understand a few things —

What is a KEY in SQL?

A KEY in SQL is a value used to identify records in a table uniquely. An SQL KEY is a single column or combination of multiple columns used to uniquely identify rows or tuples in the table. SQL Key is used to identify duplicate information, and it also helps establish a relationship between multiple tables in the database.
Note: Columns in a table that are NOT used to identify a record uniquely are called non-key columns.

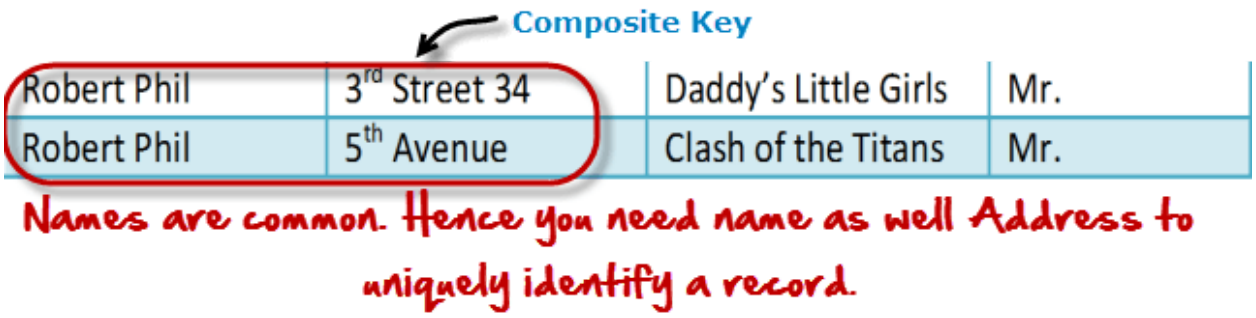
What is a Primary Key?

Primary Key in DBMS
A primary is a single column value used to identify a database record uniquely.
It has following attributes

- **A primary key cannot be NULL**
- **A primary key value must be unique**
- **The primary key values should rarely be changed**
- **The primary key must be given a value when a new record is inserted.**

What is Composite Key?

A composite key is a primary key composed of multiple columns used to identify a record uniquely
In our database, we have two people with the same name Robert Phil, but they live in different places.



Composite key in Database
Hence, we require both Full Name and Address to identify a record uniquely. That is a composite key.
Let's move into second normal form 2NF

2NF (Second Normal Form) Rules

- **Rule 1- Be in 1NF**
- **Rule 2- Single Column Primary Key that does not functionally dependant on any subset of candidate key relation**

It is clear that we can't move forward to make our simple database in 2nd Normalization form unless we partition the table above.

MEMBERSHIP ID	FULL NAMES	PHYSICAL ADDRESS	SALUTATION
1	Janet Jones	First Street Plot No 4	Ms.
2	Robert Phil	3rd Street 34	Mr.
3	Robert Phil	5th Avenue	Mr.

MEMBERSHIP ID	MOVIES RENTED
1	Pirates of the Caribbean
1	Clash of the Titans
2	Forgetting Sarah Marshal
2	Daddy's Little Girls
3	Clash of the Titans

We have divided our 1NF table into two tables viz. Table 1 and Table2. Table 1 contains member information. Table 2 contains information on movies rented.
We have introduced a new column called Membership_id which is the primary key for table 1. Records can be uniquely identified in Table 1 using membership id

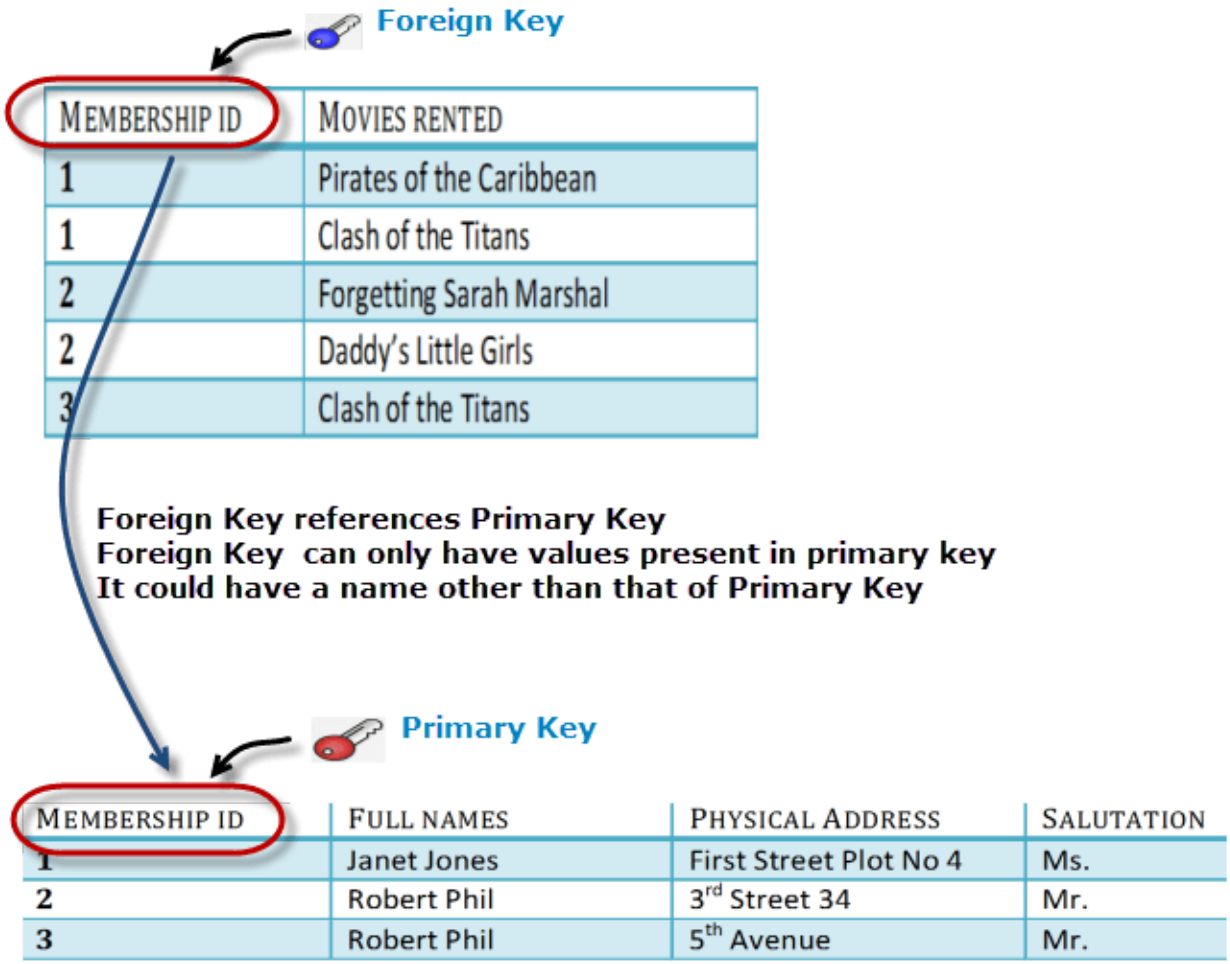
Database – Foreign Key

In Table 2, Membership_ID is the Foreign Key

MEMBERSHIP ID	MOVIES RENTED
1	Pirates of the Caribbean
1	Clash of the Titans
2	Forgetting Sarah Marshal
2	Daddy's Little Girls
3	Clash of the Titans

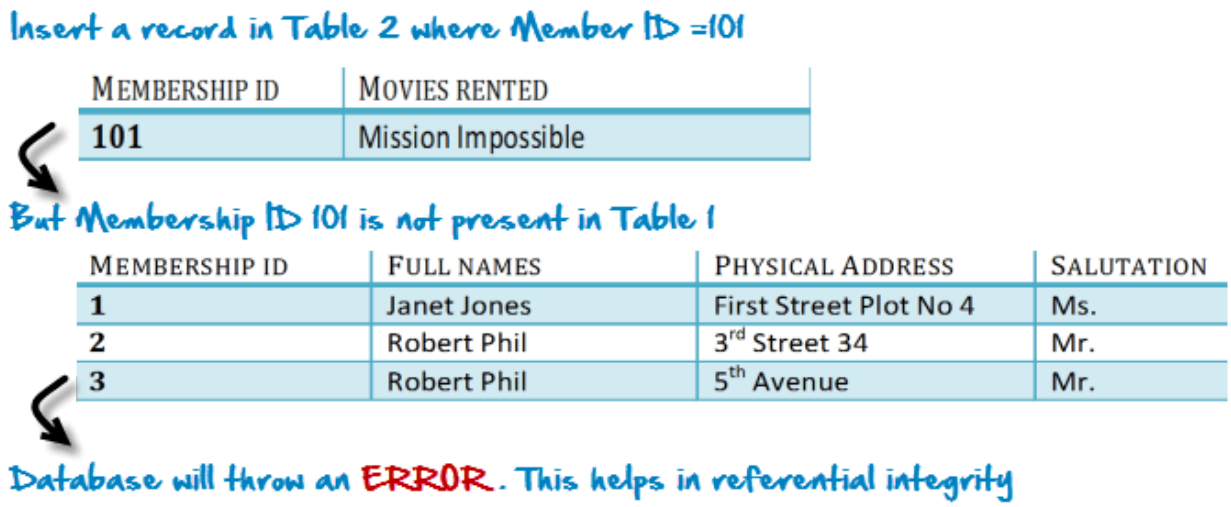
Foreign Key in DBMS
Foreign Key references the primary key of another Table! It helps connect your Tables

- **A foreign key can have a different name from its primary key**
- **It ensures rows in one table have corresponding rows in another**
- **Unlike the Primary key, they do not have to be unique. Most often they aren't**
- **Foreign keys can be null even though primary keys can not**



Why do you need a foreign key?

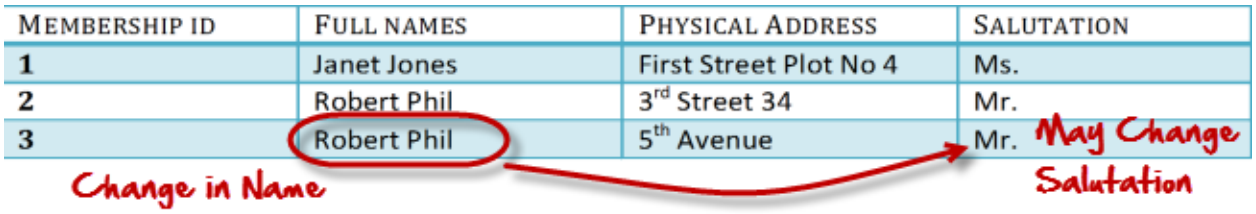
Suppose, a novice inserts a record in Table B such as



You will only be able to insert values into your foreign key that exist in the unique key in the parent table. This helps in referential integrity.
The above problem can be overcome by declaring membership id from Table2 as foreign key of membership id from Table1
Now, if somebody tries to insert a value in the membership id field that does not exist in the parent table, an error will be shown!

What are transitive functional dependencies?

A transitive functional dependency is when changing a non-key column, might cause any of the other non-key columns to change
Consider the table 1. Changing the non-key column Full Name may change Salutation.



Let's move into 3NF

3NF (Third Normal Form) Rules

- **Rule 1- Be in 2NF**

- **Rule 2- Has no transitive functional dependencies**

To move our 2NF table into 3NF, we again need to again divide our table.

3NF Example

Below is a 3NF example in SQL database:

MEMBERSHIP ID	FULL NAMES	PHYSICAL ADDRESS	SALUTATION ID
1	Janet Jones	First Street Plot No 4	2
2	Robert Phil	3rd Street 34	1
3	Robert Phil	5th Avenue	1

MEMBERSHIP ID	MOVIES RENTED
1	Pirates of the Caribbean
1	Clash of the Titans
2	Forgetting Sarah Marshal
2	Daddy's Little Girls
3	Clash of the Titans

SALUTATION ID	SALUTATION
1	Mr.
2	Ms.
3	Mrs.
4	Dr.

We have again divided our tables and created a new table which stores Salutations.
There are no transitive functional dependencies, and hence our table is in 3NF
In Table 3 Salutation ID is primary key, and in Table 1 Salutation ID is foreign to primary key in Table 3
Now our little example is at a level that cannot further be decomposed to attain higher normal form types of normalization in DBMS. In fact, it is already in higher normalization forms. Separate efforts for moving into next levels of normalizing data are normally needed in complex databases. However, we will be discussing next levels of normalisation in DBMS in brief in the following.

BCNF (Boyce-Codd Normal Form)

Even when a database is in 3rd Normal Form, still there would be anomalies resulted if it has more than one Candidate Key.
Sometimes is BCNF is also referred as 3.5 Normal Form.

4NF (Fourth Normal Form) Rules

If no database table instance contains two or more, independent and multivalued data describing the relevant entity, then it is in 4th Normal Form.

5NF (Fifth Normal Form) Rules

A table is in 5th Normal Form only if it is in 4NF and it cannot be decomposed into any number of smaller tables without loss of data.

6NF (Sixth Normal Form) Proposed

6th Normal Form is not standardized, yet however, it is being discussed by database experts for some time. Hopefully, we would have a clear & standardized definition for 6th Normal Form in the near future…
That's all to SQL Normalization!!!

Summary

- **Database designing is critical to the successful implementation of a database management system that meets the data requirements of an enterprise system.**
- **Normalization in DBMS is a process which helps produce database systems that are cost-effective and have better security models.**
- **Functional dependencies are a very important component of the normalize data process**
- **Most database systems are normalized database up to the third normal forms in DBMS.**
- **A primary key uniquely identifies are record in a Table and cannot be null**
- **A foreign key helps connect table and references a primary key**