What is RDBMS

**RDBMS** stands for *Relational Database Management Systems.*.

All modern database management systems like SQL, MS SQL Server, IBM DB2, ORACLE, My-SQL and Microsoft Access are based on RDBMS.

It is called Relational Data Base Management System (RDBMS) because it is based on relational model introduced by E.F. Codd.

How it works

Data is represented in terms of tuples (rows) in RDBMS.

Relational database is most commonly used database. It contains number of tables and each table has its own primary key.

Due to a collection of organized set of tables, data can be accessed easily in RDBMS.

Brief History of RDBMS

During 1970 to 1972, E.F. Codd published a paper to propose the use of relational database model.

RDBMS is originally based on that E.F. Codd's relational model invention.

What is table

The RDBMS database uses tables to store data. A table is a collection of related data entries and contains rows and columns to store data.

A table is the simplest example of data storage in RDBMS.

Let's see the example of student table.

|  |  |  |  |
| --- | --- | --- | --- |
| **ID** | **Name** | **AGE** | **COURSE** |
| 1 | Ajeet | 24 | B.Tech |
| 2 | aryan | 20 | C.A |
| 3 | Mahesh | 21 | BCA |
| 4 | Ratan | 22 | MCA |
| 5 | Vimal | 26 | BSC |

What is field

Field is a smaller entity of the table which contains specific information about every record in the table. In the above example, the field in the student table consist of id, name, age, course.

What is row or record

A row of a table is also called record. It contains the specific information of each individual entry in the table. It is a horizontal entity in the table. For example: The above table contains 5 records.

Let's see one record/row in the table.

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | Ajeet | 24 | B.Tech |

What is column

A column is a vertical entity in the table which contains all information associated with a specific field in a table. For example: "name" is a column in the above table which contains all information about student's name.

|  |
| --- |
| Ajeet |
| Aryan |
| Mahesh |
| Ratan |
| Vimal |

NULL Values

The NULL value of the table specifies that the field has been left blank during record creation. It is totally different from the value filled with zero or a field that contains space.

Data Integrity

There are the following categories of data integrity exist with each RDBMS:

**Entity integrity**: It specifies that there should be no duplicate rows in a table.

**Domain integrity**: It enforces valid entries for a given column by restricting the type, the format, or the range of values.

**Referential integrity**: It specifies that rows cannot be deleted, which are used by other records.

Difference between DBMS and RDBMS

Although DBMS and RDBMS both are used to store information in physical database but there are some remarkable differences between them.

|  |  |  |
| --- | --- | --- |
| **No.** | **DBMS** | **RDBMS** |
| 1) | DBMS applications store **data as file**. | RDBMS applications store **data in a tabular form**. |
| 2) | In DBMS, data is generally stored in either a hierarchical form or a navigational form. | In RDBMS, the tables have an identifier called primary key and the data values are stored in the form of tables. |
| 3) | **Normalization is not** present in DBMS. | **Normalization is** present in RDBMS. |
| 4) | DBMS does **not apply any security** with regards to data manipulation. | RDBMS **defines the integrity constraint** for the purpose of ACID (Atomocity, Consistency, Isolation and Durability) property. |
| 5) | DBMS uses file system to store data, so there will be **no relation between the tables**. | in RDBMS, data values are stored in the form of tables, so a **relationship** between these data values will be stored in the form of a table as well. |
| 6) | DBMS has to provide some uniform methods to access the stored information. | RDBMS system supports a tabular structure of the data and a relationship between them to access the stored information. |
| 7) | DBMS **does not support distributed database**. | RDBMS **supports distributed database**. |
| 8) | DBMS is meant to be for small organization and **deal with small data**. it supports **single user**. | RDBMS is designed to **handle large amount of data**. it supports **multiple users**. |
| 9) | Examples of DBMS are file systems, **xml** etc. | Example of RDBMS are **mysql**, **postgre**, **sql server**, **oracle** etc. |

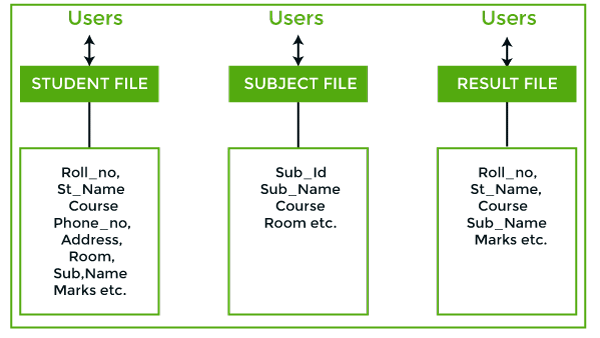
The main differences between DBMS and RDBMS are given below:

After observing the differences between DBMS and RDBMS, you can say that RDBMS is an extension of DBMS. There are many software products in the market today who are compatible for both DBMS and RDBMS. Means today a RDBMS application is DBMS application and vice-versa.

DBMS vs. File System

File System Approach

File based systems were an early attempt to computerize the manual system. It is also called a traditional based approach in which a decentralized approach was taken where each department stored and controlled its own data with the help of a data processing specialist. The main role of a data processing specialist was to create the necessary computer file structures, and also manage the data within structures and design some application programs that create reports based on file data.



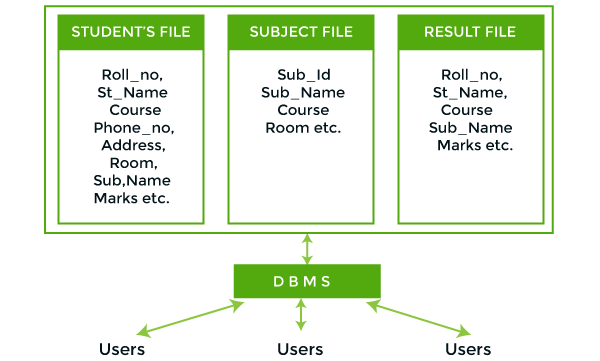
**In the above figure:**

Consider an example of a student's file system. The student file will contain information regarding the student (i.e. roll no, student name, course etc.). Similarly, we have a subject file that contains information about the subject and the result file which contains the information regarding the result.

Some fields are duplicated in more than one file, which leads to data redundancy. So to overcome this problem, we need to create a centralized system, i.e. DBMS approach.

## DBMS:

A database approach is a well-organized collection of data that are related in a meaningful way which can be accessed by different users but stored only once in a system. The various operations performed by the DBMS system are: Insertion, deletion, selection, sorting etc.



**In the above figure,**

In the above figure, duplication of data is reduced due to centralization of data.

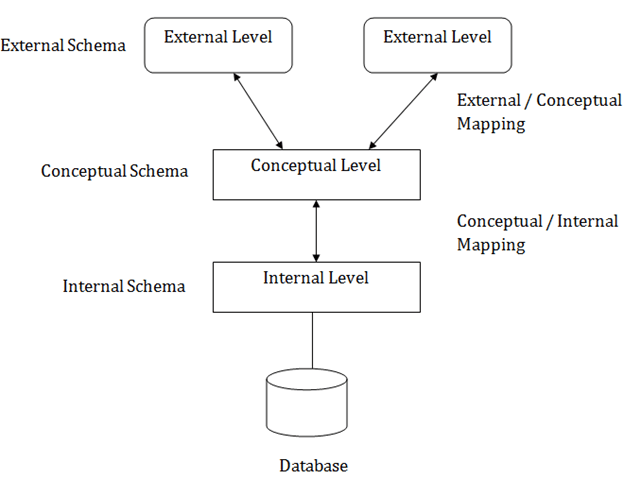
**There are the following differences between DBMS and File systems:**

|  |  |  |
| --- | --- | --- |
| **Basis** | **DBMS Approach** | **File System Approach** |
| **Meaning** | DBMS is a collection of data. In DBMS, the user is not required to write the procedures. | The file system is a collection of data. In this system, the user has to write the procedures for managing the database. |
| **Sharing of data** | Due to the centralized approach, data sharing is easy. | Data is distributed in many files, and it may be of different formats, so it isn't easy to share data. |
| **Data Abstraction** | DBMS gives an abstract view of data that hides the details. | The file system provides the detail of the data representation and storage of data. |
| **Security and Protection** | DBMS provides a good protection mechanism. | It isn't easy to protect a file under the file system. |
| **Recovery Mechanism** | DBMS provides a crash recovery mechanism, i.e., DBMS protects the user from system failure. | The file system doesn't have a crash mechanism, i.e., if the system crashes while entering some data, then the content of the file will be lost. |
| **Manipulation Techniques** | DBMS contains a wide variety of sophisticated techniques to store and retrieve the data. | The file system can't efficiently store and retrieve the data. |
| **Concurrency Problems** | DBMS takes care of Concurrent access of data using some form of locking. | In the File system, concurrent access has many problems like redirecting the file while deleting some information or updating some information. |
| **Where to use** | Database approach used in large systems which interrelate many files. | File system approach used in large systems which interrelate many files. |
| **Cost** | The database system is expensive to design. | The file system approach is cheaper to design. |
| **Data Redundancy and Inconsistency** | Due to the centralization of the database, the problems of data redundancy and inconsistency are controlled. | In this, the files and application programs are created by different programmers so that there exists a lot of duplication of data which may lead to inconsistency. |
| **Structure** | The database structure is complex to design. | The file system approach has a simple structure. |
| **Data Independence** | In this system, Data Independence exists, and it can be of two types.   * Logical Data Independence * Physical Data Independence | In the File system approach, there exists no Data Independence. |
| **Integrity Constraints** | Integrity Constraints are easy to apply. | Integrity Constraints are difficult to implement in file system. |
| **Data Models** | In the database approach, 3 types of data models exist:   * Hierarchal data models * Network data models * Relational data models | In the file system approach, there is no concept of data models exists. |
| **Flexibility** | Changes are often a necessity to the content of the data stored in any system, and these changes are more easily with a database approach. | The flexibility of the system is less as compared to the DBMS approach. |
| **Examples** | Oracle, SQL Server, Sybase etc. | Cobol, C++ etc. |

# Three schema Architecture

* The three schema architecture is also called ANSI/SPARC architecture or three-level architecture.
* This framework is used to describe the structure of a specific database system.
* The three schema architecture is also used to separate the user applications and physical database.
* The three schema architecture contains three-levels. It breaks the database down into three different categories.

**The three-schema architecture is as follows:**



**In the above diagram:**

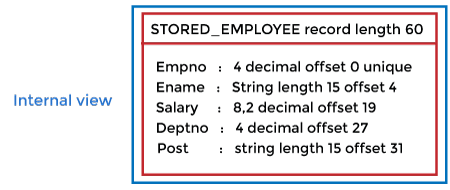
* It shows the DBMS architecture.
* Mapping is used to transform the request and response between various database levels of architecture.
* Mapping is not good for small DBMS because it takes more time.
* In External / Conceptual mapping, it is necessary to transform the request from external level to conceptual schema.
* In Conceptual / Internal mapping, DBMS transform the request from the conceptual to internal level.

## Objectives of Three schema Architecture

The main objective of three level architecture is to enable multiple users to access the same data with a personalized view while storing the underlying data only once. Thus it separates the user's view from the physical structure of the database. This separation is desirable for the following reasons:

* Different users need different views of the same data.
* The approach in which a particular user needs to see the data may change over time.
* The users of the database should not worry about the physical implementation and internal workings of the database such as data compression and encryption techniques, hashing, optimization of the internal structures etc.
* All users should be able to access the same data according to their requirements.
* DBA should be able to change the conceptual structure of the database without affecting the user's
* Internal structure of the database should be unaffected by changes to physical aspects of the storage.

### **1. Internal Level**



* The internal level has an internal schema which describes the physical storage structure of the database.
* The internal schema is also known as a physical schema.
* It uses the physical data model. It is used to define that how the data will be stored in a block.
* The physical level is used to describe complex low-level data structures in detail.

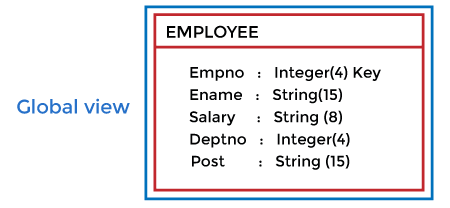
The internal level is generally is concerned with the following activities:

10 Sec

How to find Nth Highest Salary in SQL

* Storage space allocations.  
  **For Example:** B-Trees, Hashing etc.
* Access paths.  
  **For Example:** Specification of primary and secondary keys, indexes, pointers and sequencing.
* Data compression and encryption techniques.
* Optimization of internal structures.
* Representation of stored fields.

### **2. Conceptual Level**



* The conceptual schema describes the design of a database at the conceptual level. Conceptual level is also known as logical level.
* The conceptual schema describes the structure of the whole database.
* The conceptual level describes what data are to be stored in the database and also describes what relationship exists among those data.
* In the conceptual level, internal details such as an implementation of the data structure are hidden.
* Programmers and database administrators work at this level.

### **3. External Level**

DBMS Three schema Architecture

* At the external level, a database contains several schemas that sometimes called as subschema. The subschema is used to describe the different view of the database.
* An external schema is also known as view schema.
* Each view schema describes the database part that a particular user group is interested and hides the remaining database from that user group.
* The view schema describes the end user interaction with database systems.

## Mapping between Views

The three levels of DBMS architecture don't exist independently of each other. There must be correspondence between the three levels i.e. how they actually correspond with each other. DBMS is responsible for correspondence between the three types of schema. This correspondence is called Mapping.

**There are basically two types of mapping in the database architecture:**

* Conceptual/ Internal Mapping
* External / Conceptual Mapping

**Conceptual/ Internal Mapping**

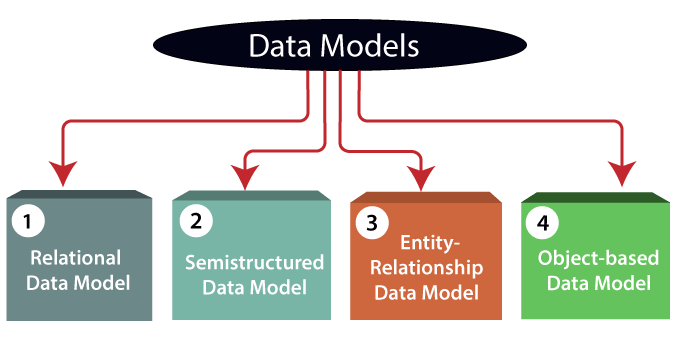
The Conceptual/ Internal Mapping lies between the conceptual level and the internal level. Its role is to define the correspondence between the records and fields of the conceptual level and files and data structures of the internal level.

**External/ Conceptual Mapping**

The external/Conceptual Mapping lies between the external level and the Conceptual level. Its role is to define the correspondence betweena particular external and the conceptual view.

# Data Models

Data Model is the modeling of the data description, data semantics, and consistency constraints of the data. It provides the conceptual tools for describing the design of a database at each level of data abstraction. Therefore, there are following four data models used for understanding the structure of the database:



**1) Relational Data Model:** This type of model designs the data in the form of rows and columns within a table. Thus, a relational model uses tables for representing data and in-between relationships. Tables are also called relations. This model was initially described by Edgar F. Codd, in 1969. The relational data model is the widely used model which is primarily used by commercial data processing applications.

**2) Entity-Relationship Data Model:** An ER model is the logical representation of data as objects and relationships among them. These objects are known as entities, and relationship is an association among these entities. This model was designed by Peter Chen and published in 1976 papers. It was widely used in database designing. A set of attributes describe the entities. For example, student\_name, student\_id describes the 'student' entity. A set of the same type of entities is known as an 'Entity set', and the set of the same type of relationships is known as 'relationship set'.

**3) Object-based Data Model:** An extension of the ER model with notions of functions, encapsulation, and object identity, as well. This model supports a rich type system that includes structured and collection types. Thus, in 1980s, various database systems following the object-oriented approach were developed. Here, the objects are nothing but the data carrying its properties.

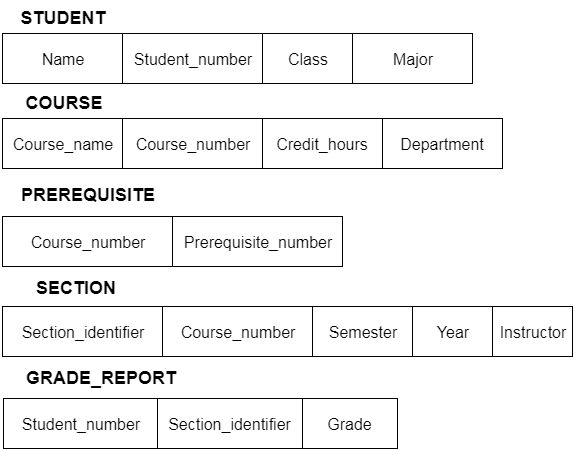
**4) Semistructured Data Model:** This type of data model is different from the other three data models (explained above). The semistructured data model allows the data specifications at places where the individual data items of the same type may have different attributes sets. The Extensible Markup Language, also known as XML, is widely used for representing the semistructured data. Although XML was initially designed for including the markup information to the text document, it gains importance because of its application in the exchange of data.

Data model Schema and Instance

* The data which is stored in the database at a particular moment of time is called an instance of the database.
* The overall design of a database is called schema.
* A database schema is the skeleton structure of the database. It represents the logical view of the entire database.
* A schema contains schema objects like table, foreign key, primary key, views, columns, data types, stored procedure, etc.
* A database schema can be represented by using the visual diagram. That diagram shows the database objects and relationship with each other.
* A database schema is designed by the database designers to help programmers whose software will interact with the database. The process of database creation is called data modeling.

A schema diagram can display only some aspects of a schema like the name of record type, data type, and constraints. Other aspects can't be specified through the schema diagram. For example, the given figure neither show the data type of each data item nor the relationship among various files.

In the database, actual data changes quite frequently. For example, in the given figure, the database changes whenever we add a new grade or add a student. The data at a particular moment of time is called the instance of the database.



Data Independence

* Data independence can be explained using the three-schema architecture.
* Data independence refers characteristic of being able to modify the schema at one level of the database system without altering the schema at the next higher level.

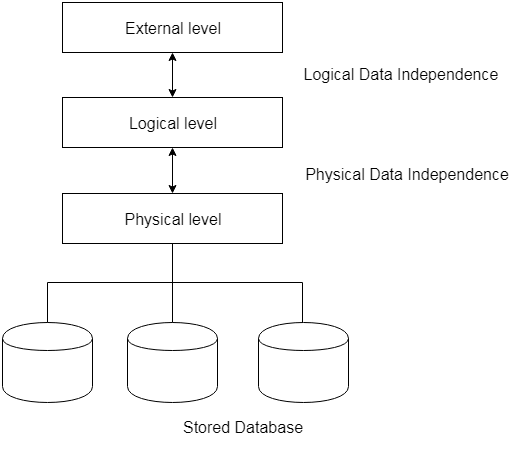
There are two types of data independence:

1. Logical Data Independence

* Logical data independence refers characteristic of being able to change the conceptual schema without having to change the external schema.
* Logical data independence is used to separate the external level from the conceptual view.
* If we do any changes in the conceptual view of the data, then the user view of the data would not be affected.
* Logical data independence occurs at the user interface level.

2. Physical Data Independence

* Physical data independence can be defined as the capacity to change the internal schema without having to change the conceptual schema.
* If we do any changes in the storage size of the database system server, then the Conceptual structure of the database will not be affected.
* Physical data independence is used to separate conceptual levels from the internal levels.
* Physical data independence occurs at the logical interface level.

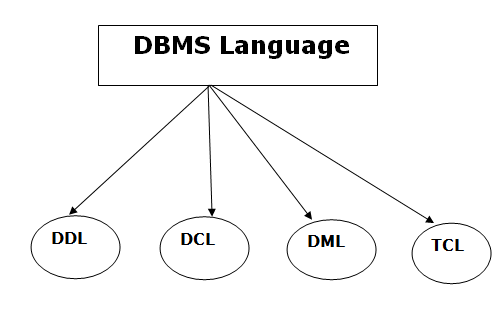


**Fig: Data Independence**

# Database Language

* A DBMS has appropriate languages and interfaces to express database queries and updates.
* Database languages can be used to read, store and update the data in the database.

## Types of Database Language



### **1. Data Definition Language**

* **DDL** stands for **D**ata **D**efinition **L**anguage. It is used to define database structure or pattern.
* It is used to create schema, tables, indexes, constraints, etc. in the database.
* Using the DDL statements, you can create the skeleton of the database.
* Data definition language is used to store the information of metadata like the number of tables and schemas, their names, indexes, columns in each table, constraints, etc.

Here are some tasks that come under DDL:

* **Create:** It is used to create objects in the database.
* **Alter:** It is used to alter the structure of the database.
* **Drop:** It is used to delete objects from the database.
* **Truncate:** It is used to remove all records from a table.
* **Rename:** It is used to rename an object.
* **Comment:** It is used to comment on the data dictionary.

These commands are used to update the database schema that's why they come under Data definition language.

### **2. Data Manipulation Language**

**DML** stands for **D**ata **M**anipulation **L**anguage. It is used for accessing and manipulating data in a database. It handles user requests.

Here are some tasks that come under DML:

* **Select:** It is used to retrieve data from a database.
* **Insert:** It is used to insert data into a table.
* **Update:** It is used to update existing data within a table.
* **Delete:** It is used to delete all records from a table.
* **Merge:** It performs UPSERT operation, i.e., insert or update operations.
* **Call:** It is used to call a structured query language or a Java subprogram.
* **Explain Plan:** It has the parameter of explaining data.
* **Lock Table:** It controls concurrency.

### **3. Data Control Language**

* **DCL** stands for **D**ata **C**ontrol **L**anguage. It is used to retrieve the stored or saved data.
* The DCL execution is transactional. It also has rollback parameters.

(But in Oracle database, the execution of data control language does not have the feature of rolling back.)

Here are some tasks that come under DCL:

* **Grant:** It is used to give user access privileges to a database.
* **Revoke:** It is used to take back permissions from the user.

There are the following operations which have the authorization of Revoke:

CONNECT, INSERT, USAGE, EXECUTE, DELETE, UPDATE and SELECT.

### **4. Transaction Control Language**

TCL is used to run the changes made by the DML statement. TCL can be grouped into a logical transaction.

Here are some tasks that come under TCL:

* **Commit:** It is used to save the transaction on the database.
* **Rollback:** It is used to restore the database to original since the last Commit.

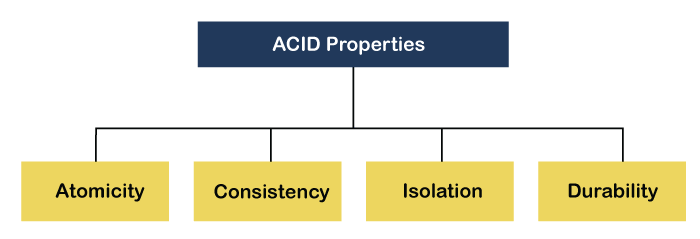
# ACID Properties in DBMS

DBMS is the management of data that should remain integrated when any changes are done in it. It is because if the integrity of the data is affected, whole data will get disturbed and corrupted. Therefore, to maintain the integrity of the data, there are four properties described in the database management system, which are known as the **ACID** properties. The ACID properties are meant for the transaction that goes through a different group of tasks, and there we come to see the role of the ACID properties.

In this section, we will learn and understand about the ACID properties. We will learn what these properties stand for and what does each property is used for. We will also understand the ACID properties with the help of some examples.

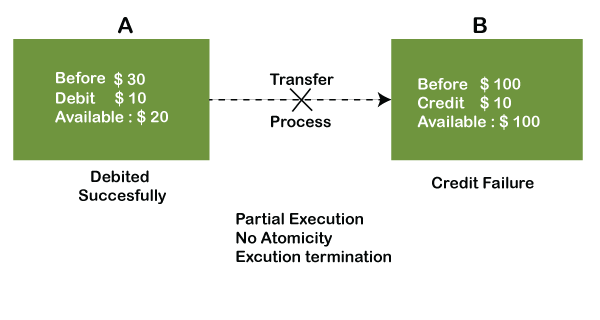
## ACID Properties

The expansion of the term ACID defines for:



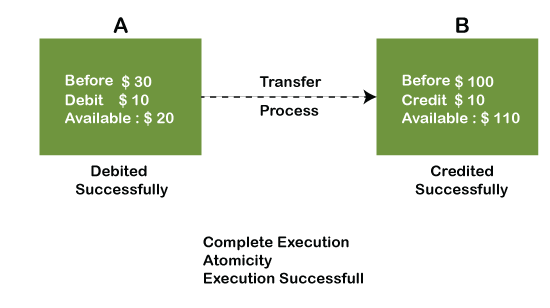
**1) Atomicity:** The term atomicity defines that the data remains atomic. It means if any operation is performed on the data, either it should be performed or executed completely or should not be executed at all. It further means that the operation should not break in between or execute partially. In the case of executing operations on the transaction, the operation should be completely executed and not partially.

**Example:** If Remo has account A having $30 in his account from which he wishes to send $10 to Sheero's account, which is B. In account B, a sum of $ 100 is already present. When $10 will be transferred to account B, the sum will become $110. Now, there will be two operations that will take place. One is the amount of $10 that Remo wants to transfer will be debited from his account A, and the same amount will get credited to account B, i.e., into Sheero's account. Now, what happens - the first operation of debit executes successfully, but the credit operation, however, fails. Thus, in Remo's account A, the value becomes $20, and to that of Sheero's account, it remains $100 as it was previously present.



In the above diagram, it can be seen that after crediting $10, the amount is still $100 in account B. So, it is not an atomic transaction.

The below image shows that both debit and credit operations are done successfully. Thus the transaction is atomic.

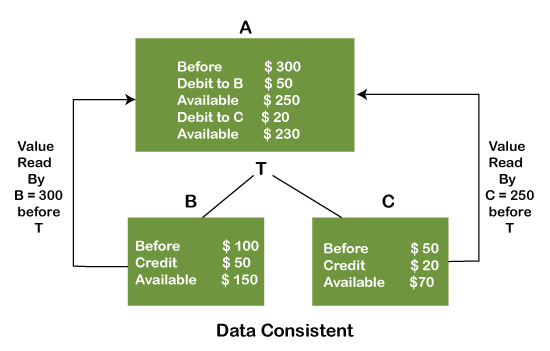


Thus, when the amount loses atomicity, then in the bank systems, this becomes a huge issue, and so the atomicity is the main focus in the bank systems.

**2) Consistency:** The word **consistency** means that the value should remain preserved always. In [DBMS](https://www.javatpoint.com/dbms-tutorial)

, the integrity of the data should be maintained, which means if a change in the database is made, it should remain preserved always. In the case of transactions, the integrity of the data is very essential so that the database remains consistent before and after the transaction. The data should always be correct.

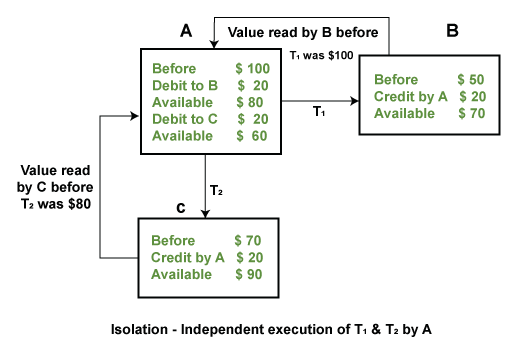
**Example:**



In the above figure, there are three accounts, A, B, and C, where A is making a transaction T one by one to both B & C. There are two operations that take place, i.e., Debit and Credit. Account A firstly debits $50 to account B, and the amount in account A is read $300 by B before the transaction. After the successful transaction T, the available amount in B becomes $150. Now, A debits $20 to account C, and that time, the value read by C is $250 (that is correct as a debit of $50 has been successfully done to B). The debit and credit operation from account A to C has been done successfully. We can see that the transaction is done successfully, and the value is also read correctly. Thus, the data is consistent. In case the value read by B and C is $300, which means that data is inconsistent because when the debit operation executes, it will not be consistent.

**4) Isolation:** The term 'isolation' means separation. In DBMS, Isolation is the property of a database where no data should affect the other one and may occur concurrently. In short, the operation on one database should begin when the operation on the first database gets complete. It means if two operations are being performed on two different databases, they may not affect the value of one another. In the case of transactions, when two or more transactions occur simultaneously, the consistency should remain maintained. Any changes that occur in any particular transaction will not be seen by other transactions until the change is not committed in the memory.

**Example:** If two operations are concurrently running on two different accounts, then the value of both accounts should not get affected. The value should remain persistent. As you can see in the below diagram, account A is making T1 and T2 transactions to account B and C, but both are executing independently without affecting each other. It is known as Isolation.



**4) Durability:** Durability ensures the permanency of something. In DBMS, the term durability ensures that the data after the successful execution of the operation becomes permanent in the database. The durability of the data should be so perfect that even if the system fails or leads to a crash, the database still survives. However, if gets lost, it becomes the responsibility of the recovery manager for ensuring the durability of the database. For committing the values, the COMMIT command must be used every time we make changes.

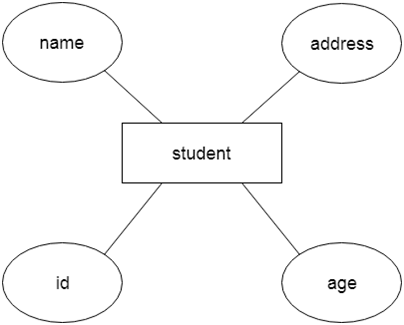
Therefore, the ACID property of DBMS plays a vital role in maintaining the consistency and availability of data in the database.

Thus, it was a precise introduction of ACID properties in DBMS. We have discussed these properties in the transaction section also.

# ER model

* ER model stands for an Entity-Relationship model. It is a high-level data model. This model is used to define the data elements and relationship for a specified system.
* It develops a conceptual design for the database. It also develops a very simple and easy to design view of data.
* In ER modeling, the database structure is portrayed as a diagram called an entity-relationship diagram.

**For example,** Suppose we design a school database. In this database, the student will be an entity with attributes like address, name, id, age, etc. The address can be another entity with attributes like city, street name, pin code, etc and there will be a relationship between them.



## Component of ER Diagram



### **1. Entity:**

An entity may be any object, class, person or place. In the ER diagram, an entity can be represented as rectangles.

Consider an organization as an example- manager, product, employee, department etc. can be taken as an entity.



**a. Weak Entity**

10 Sec

Prime Ministers of India | List of Prime Minister of India (1947-2020)

An entity that depends on another entity called a weak entity. The weak entity doesn't contain any key attribute of its own. The weak entity is represented by a double rectangle.



### **2. Attribute**

The attribute is used to describe the property of an entity. Eclipse is used to represent an attribute.

**For example,** id, age, contact number, name, etc. can be attributes of a student.



**a. Key Attribute**

The key attribute is used to represent the main characteristics of an entity. It represents a primary key. The key attribute is represented by an ellipse with the text underlined.



**b. Composite Attribute**

An attribute that composed of many other attributes is known as a composite attribute. The composite attribute is represented by an ellipse, and those ellipses are connected with an ellipse.



**c. Multivalued Attribute**

An attribute can have more than one value. These attributes are known as a multivalued attribute. The double oval is used to represent multivalued attribute.

**For example,** a student can have more than one phone number.



**d. Derived Attribute**

An attribute that can be derived from other attribute is known as a derived attribute. It can be represented by a dashed ellipse.

**For example,** A person's age changes over time and can be derived from another attribute like Date of birth.



### **3. Relationship**

A relationship is used to describe the relation between entities. Diamond or rhombus is used to represent the relationship.



Types of relationship are as follows:

**a. One-to-One Relationship**

When only one instance of an entity is associated with the relationship, then it is known as one to one relationship.

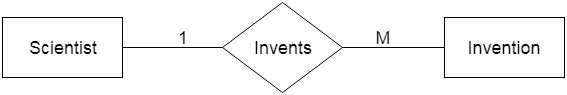
**For example,** A female can marry to one male, and a male can marry to one female.



**b. One-to-many relationship**

When only one instance of the entity on the left, and more than one instance of an entity on the right associates with the relationship then this is known as a one-to-many relationship.

**For example,** Scientist can invent many inventions, but the invention is done by the only specific scientist.

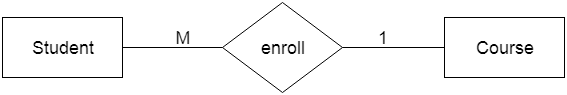


**c.**

**Many-to-one relationship**

When more than one instance of the entity on the left, and only one instance of an entity on the right associates with the relationship then it is known as a many-to-one relationship.

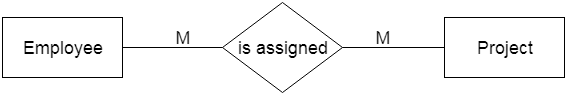
**For example,** Student enrolls for only one course, but a course can have many students.



**d. Many-to-many relationship**

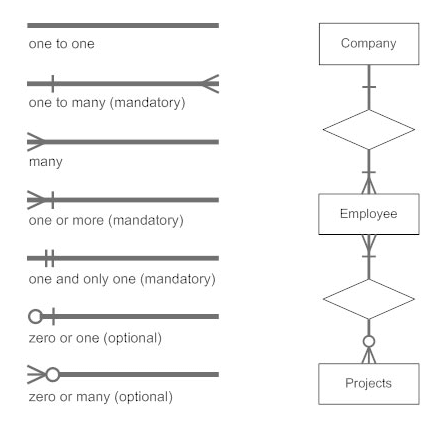
When more than one instance of the entity on the left, and more than one instance of an entity on the right associates with the relationship then it is known as a many-to-many relationship.

**For example,** Employee can assign by many projects and project can have many employees.



# Notation of ER diagram

Database can be represented using the notations. In ER diagram, many notations are used to express the cardinality. These notations are as follows:



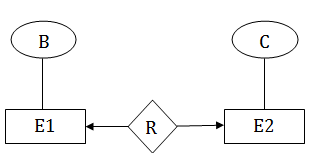
**Fig: Notations of ER diagram**

Mapping Constraints

* A mapping constraint is a data constraint that expresses the number of entities to which another entity can be related via a relationship set.
* It is most useful in describing the relationship sets that involve more than two entity sets.
* For binary relationship set R on an entity set A and B, there are four possible mapping cardinalities. These are as follows:
  1. One to one (1:1)
  2. One to many (1:M)
  3. Many to one (M:1)
  4. Many to many (M:M)

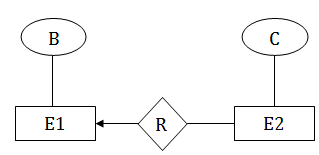
One-to-one

In one-to-one mapping, an entity in E1 is associated with at most one entity in E2, and an entity in E2 is associated with at most one entity in E1.



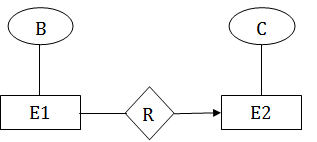
One-to-many

In one-to-many mapping, an entity in E1 is associated with any number of entities in E2, and an entity in E2 is associated with at most one entity in E1.



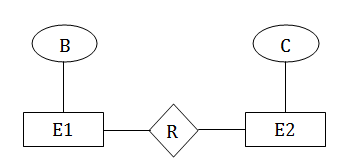
Many-to-one

In one-to-many mapping, an entity in E1 is associated with at most one entity in E2, and an entity in E2 is associated with any number of entities in E1.



Many-to-many

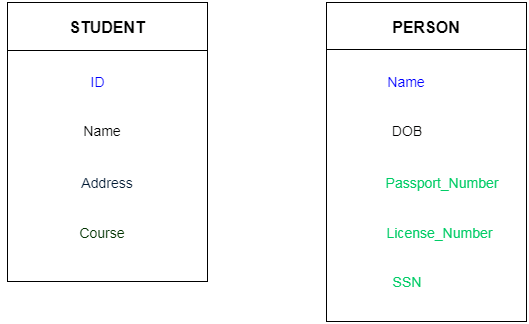
In many-to-many mapping, an entity in E1 is associated with any number of entities in E2, and an entity in E2 is associated with any number of entities in E1.



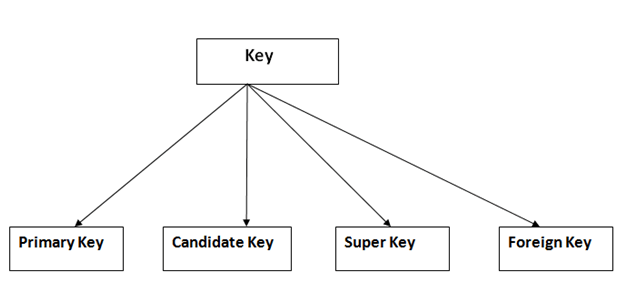
# Keys

* Keys play an important role in the relational database.
* It is used to uniquely identify any record or row of data from the table. It is also used to establish and identify relationships between tables.

**For example:** In Student table, ID is used as a key because it is unique for each student. In PERSON table, passport\_number, license\_number, SSN are keys since they are unique for each person.

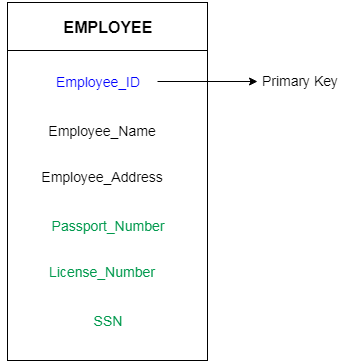


## Types of key:



### **1. Primary key**

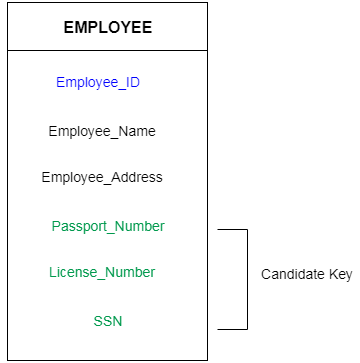
* It is the first key which is used to identify one and only one instance of an entity uniquely. An entity can contain multiple keys as we saw in PERSON table. The key which is most suitable from those lists become a primary key.
* In the EMPLOYEE table, ID can be primary key since it is unique for each employee. In the EMPLOYEE table, we can even select License\_Number and Passport\_Number as primary key since they are also unique.
* For each entity, selection of the primary key is based on requirement and developers.



### **2. Candidate key**

* A candidate key is an attribute or set of an attribute which can uniquely identify a tuple.
* The remaining attributes except for primary key are considered as a candidate key. The candidate keys are as strong as the primary key.

**For example:** In the EMPLOYEE table, id is best suited for the primary key. Rest of the attributes like SSN, Passport\_Number, and License\_Number, etc. are considered as a candidate key.



### **3. Super Key**

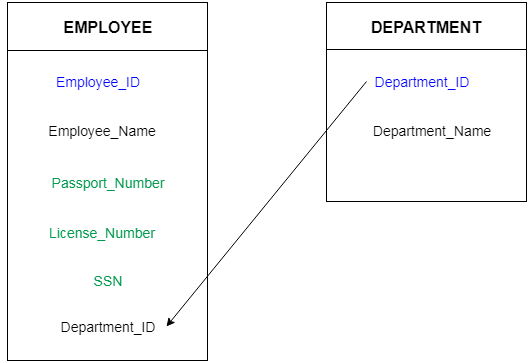
Super key is a set of an attribute which can uniquely identify a tuple. Super key is a superset of a candidate key.

**For example:** In the above EMPLOYEE table, for(EMPLOEE\_ID, EMPLOYEE\_NAME) the name of two employees can be the same, but their EMPLYEE\_ID can't be the same. Hence, this combination can also be a key.

The super key would be EMPLOYEE-ID, (EMPLOYEE\_ID, EMPLOYEE-NAME), etc.

### **4. Foreign key**

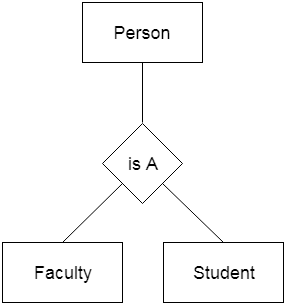
* Foreign keys are the column of the table which is used to point to the primary key of another table.
* In a company, every employee works in a specific department, and employee and department are two different entities. So we can't store the information of the department in the employee table. That's why we link these two tables through the primary key of one table.
* We add the primary key of the DEPARTMENT table, Department\_Id as a new attribute in the EMPLOYEE table.
* Now in the EMPLOYEE table, Department\_Id is the foreign key, and both the tables are related.



Generalization

* Generalization is like a bottom-up approach in which two or more entities of lower level combine to form a higher level entity if they have some attributes in common.
* In generalization, an entity of a higher level can also combine with the entities of the lower level to form a further higher level entity.
* Generalization is more like subclass and superclass system, but the only difference is the approach. Generalization uses the bottom-up approach.
* In generalization, entities are combined to form a more generalized entity, i.e., subclasses are combined to make a superclass.

**For example,** Faculty and Student entities can be generalized and create a higher level entity Person.

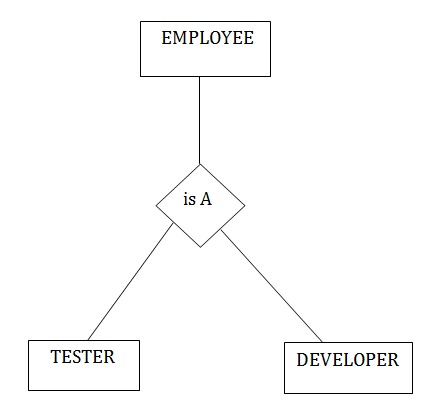


[**Next →**](https://www.javatpoint.com/dbms-aggregation)[**← Prev**](https://www.javatpoint.com/dbms-generalization)

Specialization

* Specialization is a top-down approach, and it is opposite to Generalization. In specialization, one higher level entity can be broken down into two lower level entities.
* Specialization is used to identify the subset of an entity set that shares some distinguishing characteristics.
* Normally, the superclass is defined first, the subclass and its related attributes are defined next, and relationship set are then added.

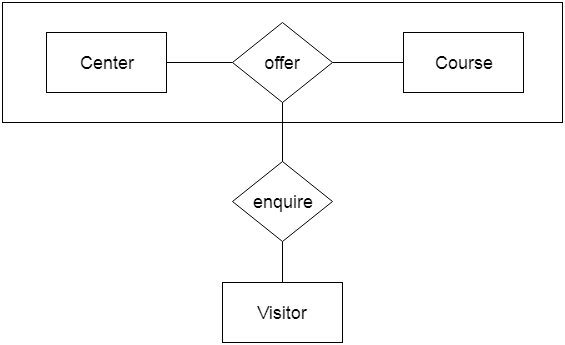
**For example:** In an Employee management system, EMPLOYEE entity can be specialized as TESTER or DEVELOPER based on what role they play in the company.



# Aggregation

In aggregation, the relation between two entities is treated as a single entity. In aggregation, relationship with its corresponding entities is aggregated into a higher level entity.

**For example:** Center entity offers the Course entity act as a single entity in the relationship which is in a relationship with another entity visitor. In the real world, if a visitor visits a coaching center then he will never enquiry about the Course only or just about the Center instead he will ask the enquiry about both.

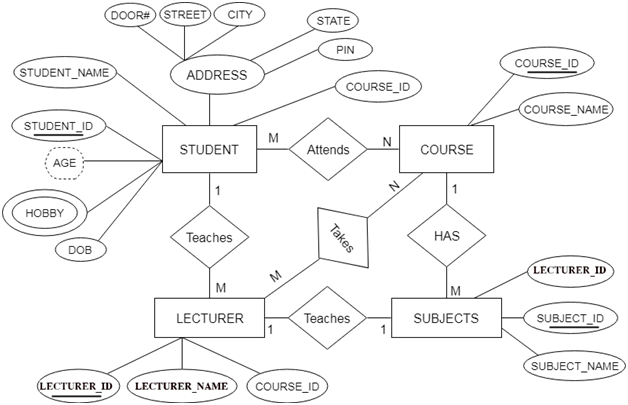


Reduction of ER diagram to Table

The database can be represented using the notations, and these notations can be reduced to a collection of tables.

In the database, every entity set or relationship set can be represented in tabular form.

**The ER diagram is given below:**



There are some points for converting the ER diagram to the table:

10 Sec

SQL CREATE TABLE

* **Entity type becomes a table.**

In the given ER diagram, LECTURE, STUDENT, SUBJECT and COURSE forms individual tables.

* **All single-valued attribute becomes a column for the table.**

In the STUDENT entity, STUDENT\_NAME and STUDENT\_ID form the column of STUDENT table. Similarly, COURSE\_NAME and COURSE\_ID form the column of COURSE table and so on.

* **A key attribute of the entity type represented by the primary key.**

In the given ER diagram, COURSE\_ID, STUDENT\_ID, SUBJECT\_ID, and LECTURE\_ID are the key attribute of the entity.

* **The multivalued attribute is represented by a separate table.**

In the student table, a hobby is a multivalued attribute. So it is not possible to represent multiple values in a single column of STUDENT table. Hence we create a table STUD\_HOBBY with column name STUDENT\_ID and HOBBY. Using both the column, we create a composite key.

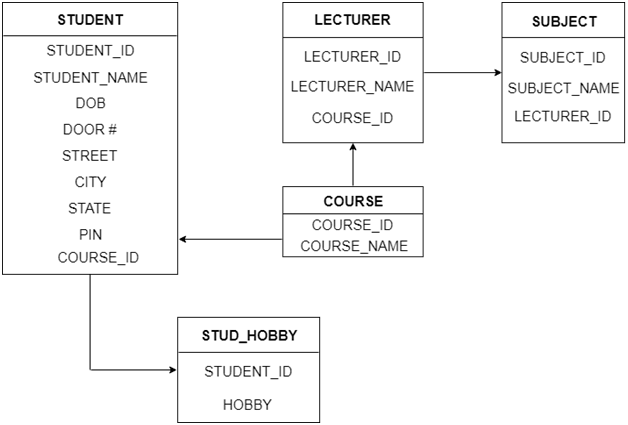
* **Composite attribute represented by components.**

In the given ER diagram, student address is a composite attribute. It contains CITY, PIN, DOOR#, STREET, and STATE. In the STUDENT table, these attributes can merge as an individual column.

* **Derived attributes are not considered in the table.**

In the STUDENT table, Age is the derived attribute. It can be calculated at any point of time by calculating the difference between current date and Date of Birth.

Using these rules, you can convert the ER diagram to tables and columns and assign the mapping between the tables. Table structure for the given ER diagram is as below:



**Figure: Table structure**

# Relationship of higher degree

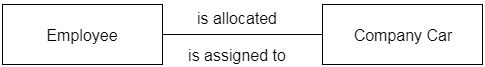
The degree of relationship can be defined as the number of occurrences in one entity that is associated with the number of occurrences in another entity.

There is the three degree of relationship:

1. One-to-one (1:1)
2. One-to-many (1:M)
3. Many-to-many (M:N)

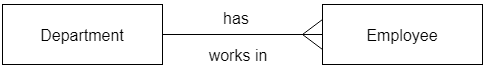
### **1. One-to-one**

* In a one-to-one relationship, one occurrence of an entity relates to only one occurrence in another entity.
* A one-to-one relationship rarely exists in practice.
* **For example:** if an employee is allocated a company car then that car can only be driven by that employee.
* Therefore, employee and company car have a one-to-one relationship.



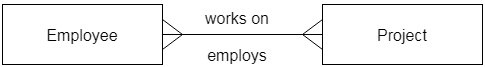
### **2. One-to-many**

* In a one-to-many relationship, one occurrence in an entity relates to many occurrences in another entity.
* **For example:** An employee works in one department, but a department has many employees.
* Therefore, department and employee have a one-to-many relationship.



### **3. Many-to-many**

* In a many-to-many relationship, many occurrences in an entity relate to many occurrences in another entity.
* Same as a one-to-one relationship, the many-to-many relationship rarely exists in practice.
* **For example:** At the same time, an employee can work on several projects, and a project has a team of many employees.
* Therefore, employee and project have a many-to-many relationship.



# Relational Model concept

Relational model can represent as a table with columns and rows. Each row is known as a tuple. Each table of the column has a name or attribute.

**Domain:** It contains a set of atomic values that an attribute can take.

**Attribute:** It contains the name of a column in a particular table. Each attribute Ai must have a domain, dom(Ai)

**Relational instance:** In the relational database system, the relational instance is represented by a finite set of tuples. Relation instances do not have duplicate tuples.

**Relational schema:** A relational schema contains the name of the relation and name of all columns or attributes.

**Relational key:** In the relational key, each row has one or more attributes. It can identify the row in the relation uniquely.

**Example: STUDENT Relation**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **NAME** | **ROLL\_NO** | **PHONE\_NO** | **ADDRESS** | **AGE** |
| Ram | 14795 | 7305758992 | Noida | 24 |
| Shyam | 12839 | 9026288936 | Delhi | 35 |
| Laxman | 33289 | 8583287182 | Gurugram | 20 |
| Mahesh | 27857 | 7086819134 | Ghaziabad | 27 |
| Ganesh | 17282 | 9028 9i3988 | Delhi | 40 |

* In the given table, NAME, ROLL\_NO, PHONE\_NO, ADDRESS, and AGE are the attributes.
* The instance of schema STUDENT has 5 tuples.
* t3 = <Laxman, 33289, 8583287182, Gurugram, 20>

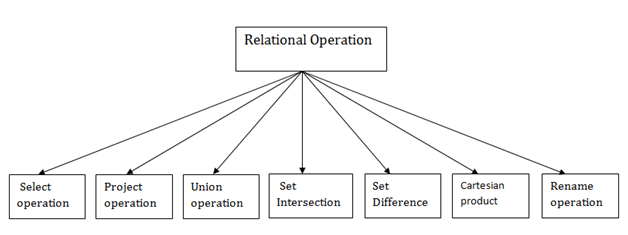
Properties of Relations

* Name of the relation is distinct from all other relations.
* Each relation cell contains exactly one atomic (single) value
* Each attribute contains a distinct name
* Attribute domain has no significance
* tuple has no duplicate value
* Order of tuple can have a different sequence

# Relational Algebra

Relational algebra is a procedural query language. It gives a step by step process to obtain the result of the query. It uses operators to perform queries.

## Types of Relational operation



### **1. Select Operation:**

* The select operation selects tuples that satisfy a given predicate.
* It is denoted by sigma (σ).

1. Notation:  σ p(r)

**Where:**

**σ** is used for selection prediction  
**r** is used for relation  
**p** is used as a propositional logic formula which may use connectors like: AND OR and NOT. These relational can use as relational operators like =, ≠, ≥, <, >, ≤.

**For example: LOAN Relation**

|  |  |  |
| --- | --- | --- |
| **BRANCH\_NAME** | **LOAN\_NO** | **AMOUNT** |
| Downtown | L-17 | 1000 |
| Redwood | L-23 | 2000 |
| Perryride | L-15 | 1500 |
| Downtown | L-14 | 1500 |
| Mianus | L-13 | 500 |
| Roundhill | L-11 | 900 |
| Perryride | L-16 | 1300 |

**Input:**

1. σ BRANCH\_NAME="perryride" (LOAN)

**Output:**

|  |  |  |
| --- | --- | --- |
| **BRANCH\_NAME** | **LOAN\_NO** | **AMOUNT** |
| Perryride | L-15 | 1500 |
| Perryride | L-16 | 1300 |

### **2. Project Operation:**

* This operation shows the list of those attributes that we wish to appear in the result. Rest of the attributes are eliminated from the table.
* It is denoted by ∏.

1. Notation: ∏ A1, A2, An (r)

**Where**

**A1**, **A2**, **A3** is used as an attribute name of relation **r**.

**Example: CUSTOMER RELATION**

|  |  |  |
| --- | --- | --- |
| **NAME** | **STREET** | **CITY** |
| Jones | Main | Harrison |
| Smith | North | Rye |
| Hays | Main | Harrison |
| Curry | North | Rye |
| Johnson | Alma | Brooklyn |
| Brooks | Senator | Brooklyn |

**Input:**

1. ∏ NAME, CITY (CUSTOMER)

**Output:**

|  |  |
| --- | --- |
| **NAME** | **CITY** |
| Jones | Harrison |
| Smith | Rye |
| Hays | Harrison |
| Curry | Rye |
| Johnson | Brooklyn |
| Brooks | Brooklyn |

### **3. Union Operation:**

* Suppose there are two tuples R and S. The union operation contains all the tuples that are either in R or S or both in R & S.
* It eliminates the duplicate tuples. It is denoted by ∪.

1. Notation: R ∪ S

A union operation must hold the following condition:

* R and S must have the attribute of the same number.
* Duplicate tuples are eliminated automatically.

### **Example:**

**DEPOSITOR RELATION**

|  |  |
| --- | --- |
| **CUSTOMER\_NAME** | **ACCOUNT\_NO** |
| Johnson | A-101 |
| Smith | A-121 |
| Mayes | A-321 |
| Turner | A-176 |
| Johnson | A-273 |
| Jones | A-472 |
| Lindsay | A-284 |

**BORROW RELATION**

|  |  |
| --- | --- |
| **CUSTOMER\_NAME** | **LOAN\_NO** |
| Jones | L-17 |
| Smith | L-23 |
| Hayes | L-15 |
| Jackson | L-14 |
| Curry | L-93 |
| Smith | L-11 |
| Williams | L-17 |

**Input:**

1. ∏ CUSTOMER\_NAME (BORROW) ∪ ∏ CUSTOMER\_NAME (DEPOSITOR)

**Output:**

|  |
| --- |
| **CUSTOMER\_NAME** |
| Johnson |
| Smith |
| Hayes |
| Turner |
| Jones |
| Lindsay |
| Jackson |
| Curry |
| Williams |
| Mayes |

### **4. Set Intersection:**

* Suppose there are two tuples R and S. The set intersection operation contains all tuples that are in both R & S.
* It is denoted by intersection ∩.

1. Notation: R ∩ S

**Example:** Using the above DEPOSITOR table and BORROW table

**nput:**

1. ∏ CUSTOMER\_NAME (BORROW) ∩ ∏ CUSTOMER\_NAME (DEPOSITOR)

**Output:**

|  |
| --- |
| **CUSTOMER\_NAME** |
| Smith |
| Jones |

### **5. Set Difference:**

* Suppose there are two tuples R and S. The set intersection operation contains all tuples that are in R but not in S.
* It is denoted by intersection minus (-).

1. Notation: R - S

**Example:** Using the above DEPOSITOR table and BORROW table

**Input:**

1. ∏ CUSTOMER\_NAME (BORROW) - ∏ CUSTOMER\_NAME (DEPOSITOR)

**Output:**

|  |
| --- |
| **CUSTOMER\_NAME** |
| Jackson |
| Hayes |
| Willians |
| Curry |

### **6. Cartesian product**

* The Cartesian product is used to combine each row in one table with each row in the other table. It is also known as a cross product.
* It is denoted by X.

1. Notation: E X D

### **Example:**

**EMPLOYEE**

|  |  |  |
| --- | --- | --- |
| **EMP\_ID** | **EMP\_NAME** | **EMP\_DEPT** |
| 1 | Smith | A |
| 2 | Harry | C |
| 3 | John | B |

**DEPARTMENT**

|  |  |
| --- | --- |
| **DEPT\_NO** | **DEPT\_NAME** |
| A | Marketing |
| B | Sales |
| C | Legal |

**Input:**

1. EMPLOYEE X DEPARTMENT

**Output:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **EMP\_ID** | **EMP\_NAME** | **EMP\_DEPT** | **DEPT\_NO** | **DEPT\_NAME** |
| 1 | Smith | A | A | Marketing |
| 1 | Smith | A | B | Sales |
| 1 | Smith | A | C | Legal |
| 2 | Harry | C | A | Marketing |
| 2 | Harry | C | B | Sales |
| 2 | Harry | C | C | Legal |
| 3 | John | B | A | Marketing |
| 3 | John | B | B | Sales |
| 3 | John | B | C | Legal |

### **7. Rename Operation:**

The rename operation is used to rename the output relation. It is denoted by **rho** (ρ).

**Example:** We can use the rename operator to rename STUDENT relation to STUDENT1.

1. ρ(STUDENT1, STUDENT)

# Join Operations:

A Join operation combines related tuples from different relations, if and only if a given join condition is satisfied. It is denoted by ⋈.

### **Example:**

**EMPLOYEE**

|  |  |
| --- | --- |
| **EMP\_CODE** | **EMP\_NAME** |
| 101 | Stephan |
| 102 | Jack |
| 103 | Harry |

**SALARY**

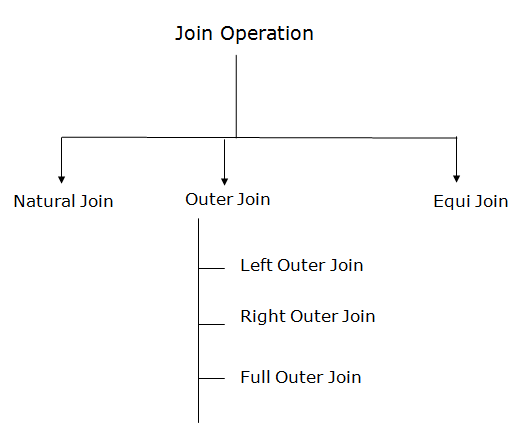
|  |  |
| --- | --- |
| **EMP\_CODE** | **SALARY** |
| 101 | 50000 |
| 102 | 30000 |
| 103 | 25000 |

1. Operation: (EMPLOYEE ⋈ SALARY)

**Result:**

|  |  |  |
| --- | --- | --- |
| **EMP\_CODE** | **EMP\_NAME** | **SALARY** |
| 101 | Stephan | 50000 |
| 102 | Jack | 30000 |
| 103 | Harry | 25000 |

## Types of Join operations:



### **1. Natural Join:**

* A natural join is the set of tuples of all combinations in R and S that are equal on their common attribute names.
* It is denoted by ⋈.

**Example:** Let's use the above EMPLOYEE table and SALARY table:

**Input:**

1. ∏EMP\_NAME, SALARY (EMPLOYEE ⋈ SALARY)

**Output:**

|  |  |
| --- | --- |
| **EMP\_NAME** | **SALARY** |
| Stephan | 50000 |
| Jack | 30000 |
| Harry | 25000 |

### **2. Outer Join:**

The outer join operation is an extension of the join operation. It is used to deal with missing information.

**Example:**

**EMPLOYEE**

|  |  |  |
| --- | --- | --- |
| **EMP\_NAME** | **STREET** | **CITY** |
| Ram | Civil line | Mumbai |
| Shyam | Park street | Kolkata |
| Ravi | M.G. Street | Delhi |
| Hari | Nehru nagar | Hyderabad |

**FACT\_WORKERS**

|  |  |  |
| --- | --- | --- |
| **EMP\_NAME** | **BRANCH** | **SALARY** |
| Ram | Infosys | 10000 |
| Shyam | Wipro | 20000 |
| Kuber | HCL | 30000 |
| Hari | TCS | 50000 |

**Input:**

1. (EMPLOYEE ⋈ FACT\_WORKERS)

**Output:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **EMP\_NAME** | **STREET** | **CITY** | **BRANCH** | **SALARY** |
| Ram | Civil line | Mumbai | Infosys | 10000 |
| Shyam | Park street | Kolkata | Wipro | 20000 |
| Hari | Nehru nagar | Hyderabad | TCS | 50000 |

An outer join is basically of three types:

1. Left outer join
2. Right outer join
3. Full outer join

### **a. Left outer join:**

* Left outer join contains the set of tuples of all combinations in R and S that are equal on their common attribute names.
* In the left outer join, tuples in R have no matching tuples in S.
* It is denoted by ⟕.

**Example:** Using the above EMPLOYEE table and FACT\_WORKERS table

**Input:**

1. EMPLOYEE ⟕ FACT\_WORKERS

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **EMP\_NAME** | **STREET** | **CITY** | **BRANCH** | **SALARY** |
| Ram | Civil line | Mumbai | Infosys | 10000 |
| Shyam | Park street | Kolkata | Wipro | 20000 |
| Hari | Nehru street | Hyderabad | TCS | 50000 |
| Ravi | M.G. Street | Delhi | NULL | NULL |

### **b. Right outer join:**

* Right outer join contains the set of tuples of all combinations in R and S that are equal on their common attribute names.
* In right outer join, tuples in S have no matching tuples in R.
* It is denoted by ⟖.

**Example:** Using the above EMPLOYEE table and FACT\_WORKERS Relation

**Input:**

1. EMPLOYEE ⟖ FACT\_WORKERS

**Output:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **EMP\_NAME** | **BRANCH** | **SALARY** | **STREET** | **CITY** |
| Ram | Infosys | 10000 | Civil line | Mumbai |
| Shyam | Wipro | 20000 | Park street | Kolkata |
| Hari | TCS | 50000 | Nehru street | Hyderabad |
| Kuber | HCL | 30000 | NULL | NULL |

### **c. Full outer join:**

* Full outer join is like a left or right join except that it contains all rows from both tables.
* In full outer join, tuples in R that have no matching tuples in S and tuples in S that have no matching tuples in R in their common attribute name.
* It is denoted by ⟗.

**Example:** Using the above EMPLOYEE table and FACT\_WORKERS table

**Input:**

1. EMPLOYEE ⟗ FACT\_WORKERS

**Output:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **EMP\_NAME** | **STREET** | **CITY** | **BRANCH** | **SALARY** |
| Ram | Civil line | Mumbai | Infosys | 10000 |
| Shyam | Park street | Kolkata | Wipro | 20000 |
| Hari | Nehru street | Hyderabad | TCS | 50000 |
| Ravi | M.G. Street | Delhi | NULL | NULL |
| Kuber | NULL | NULL | HCL | 30000 |

### **3. Equi join:**

It is also known as an inner join. It is the most common join. It is based on matched data as per the equality condition. The equi join uses the comparison operator(=).

**Example:**

**CUSTOMER RELATION**

|  |  |
| --- | --- |
| **CLASS\_ID** | **NAME** |
| 1 | John |
| 2 | Harry |
| 3 | Jackson |

**PRODUCT**

|  |  |
| --- | --- |
| **PRODUCT\_ID** | **CITY** |
| 1 | Delhi |
| 2 | Mumbai |
| 3 | Noida |

**Input:**

1. CUSTOMER ⋈ PRODUCT

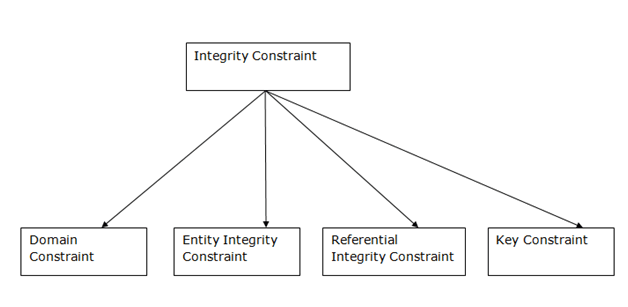
**Output:**

|  |  |  |  |
| --- | --- | --- | --- |
| **CLASS\_ID** | **NAME** | **PRODUCT\_ID** | **CITY** |
| 1 | John | 1 | Delhi |
| 2 | Harry | 2 | Mumbai |
| 3 | Harry | 3 | Noida |

# Integrity Constraints

* Integrity constraints are a set of rules. It is used to maintain the quality of information.
* Integrity constraints ensure that the data insertion, updating, and other processes have to be performed in such a way that data integrity is not affected.
* Thus, integrity constraint is used to guard against accidental damage to the database.

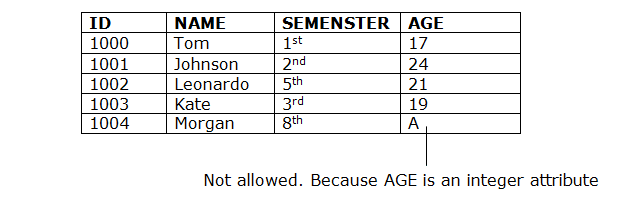
## Types of Integrity Constraint



### **1. Domain constraints**

* Domain constraints can be defined as the definition of a valid set of values for an attribute.
* The data type of domain includes string, character, integer, time, date, currency, etc. The value of the attribute must be available in the corresponding domain.

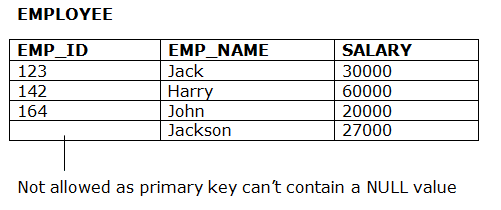
**Example:**



### **2. Entity integrity constraints**

* The entity integrity constraint states that primary key value can't be null.
* This is because the primary key value is used to identify individual rows in relation and if the primary key has a null value, then we can't identify those rows.
* A table can contain a null value other than the primary key field.

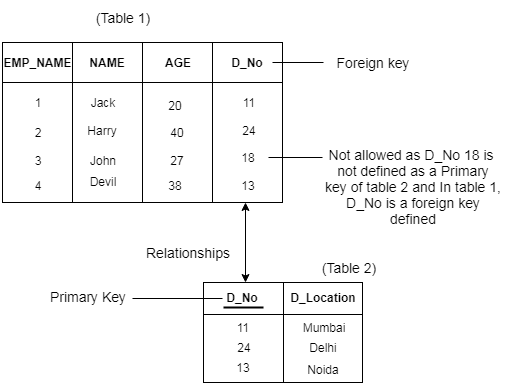
**Example:**



### **3. Referential Integrity Constraints**

* A referential integrity constraint is specified between two tables.
* In the Referential integrity constraints, if a foreign key in Table 1 refers to the Primary Key of Table 2, then every value of the Foreign Key in Table 1 must be null or be available in Table 2.

**Example:**

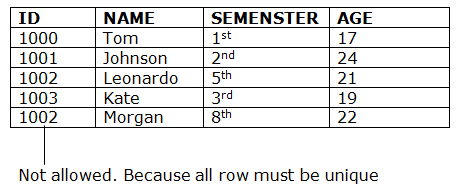


### **4. Key constraints**

* Keys are the entity set that is used to identify an entity within its entity set uniquely.
* An entity set can have multiple keys, but out of which one key will be the primary key. A primary key can contain a unique and null value in the relational table.

**Example:**

Competitive questions on Structures

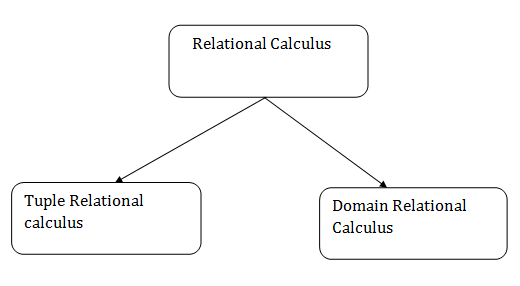


[**Next →**](https://www.javatpoint.com/dbms-sql-introduction)[**← Prev**](https://www.javatpoint.com/dbms-integrity-constraints)

# Relational Calculus

* Relational calculus is a non-procedural query language. In the non-procedural query language, the user is concerned with the details of how to obtain the end results.
* The relational calculus tells what to do but never explains how to do.

## Types of Relational calculus:



### **1. Tuple Relational Calculus (TRC)**

* The tuple relational calculus is specified to select the tuples in a relation. In TRC, filtering variable uses the tuples of a relation.
* The result of the relation can have one or more tuples.

**Notation:**

1. {T | P (T)}   or {T | Condition (T)}

Where

**T** is the resulting tuples

**P(T)** is the condition used to fetch T.

Java Try Catch

**For example:**

1. { T.name | Author(T) AND T.article = 'database' }

**OUTPUT:** This query selects the tuples from the AUTHOR relation. It returns a tuple with 'name' from Author who has written an article on 'database'.

TRC (tuple relation calculus) can be quantified. In TRC, we can use Existential (∃) and Universal Quantifiers (∀).

**For example:**

1. { R| ∃T ∈ Authors(T.article='database' AND R.name=T.name)}

**Output:** This query will yield the same result as the previous one.

### **2. Domain Relational Calculus (DRC)**

* The second form of relation is known as Domain relational calculus. In domain relational calculus, filtering variable uses the domain of attributes.
* Domain relational calculus uses the same operators as tuple calculus. It uses logical connectives ∧ (and), ∨ (or) and ┓ (not).
* It uses Existential (∃) and Universal Quantifiers (∀) to bind the variable.

**Notation:**

1. { a1, a2, a3, ..., an | P (a1, a2, a3, ... ,an)}

Where

**a1,a2** areattributes  
**P** stands for formula built by inner attributes

**For example:**

1. {< article, page, subject > |  ∈ javatpoint ∧ subject = 'database'}

**Output:** This query will yield the article, page, and subject from the relational javatpoint, where the subject is a database.

# SQL

* SQL stands for Structured Query Language. It is used for storing and managing data in relational database management system (RDMS).
* It is a standard language for Relational Database System. It enables a user to create, read, update and delete relational databases and tables.
* All the RDBMS like MySQL, Informix, Oracle, MS Access and SQL Server use SQL as their standard database language.
* SQL allows users to query the database in a number of ways, using English-like statements.

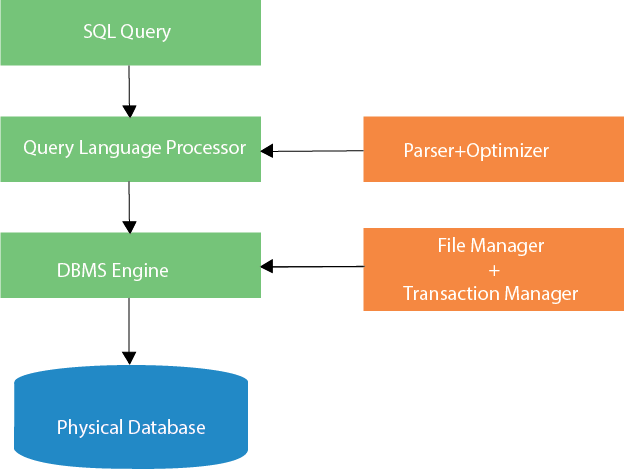
### **Rules:**

SQL follows the following rules:

* Structure query language is not case sensitive. Generally, keywords of SQL are written in uppercase.
* Statements of SQL are dependent on text lines. We can use a single SQL statement on one or multiple text line.
* Using the SQL statements, you can perform most of the actions in a database.
* SQL depends on tuple relational calculus and relational algebra.

### **SQL process:**

* When an SQL command is executing for any RDBMS, then the system figure out the best way to carry out the request and the SQL engine determines that how to interpret the task.
* In the process, various components are included. These components can be optimization Engine, Query engine, Query dispatcher, classic, etc.
* All the non-SQL queries are handled by the classic query engine, but SQL query engine won't handle logical files.



Characteristics of SQL

* SQL is easy to learn.
* SQL is used to access data from relational database management systems.
* SQL can execute queries against the database.
* SQL is used to describe the data.
* SQL is used to define the data in the database and manipulate it when needed.
* SQL is used to create and drop the database and table.
* SQL is used to create a view, stored procedure, function in a database.
* SQL allows users to set permissions on tables, procedures, and views.

Normalization:

# 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.

1. X   →   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.

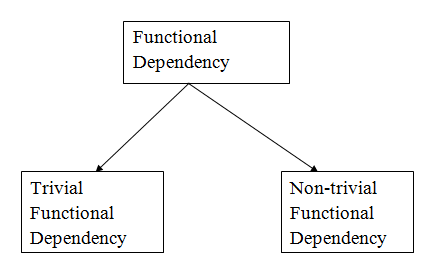
Here Emp\_Id attribute can uniquely identify the Emp\_Name attribute of employee table because if we know the Emp\_Id, we can tell that employee name associated with it.

Functional dependency can be written as:

1. Emp\_Id → Emp\_Name

We can say that Emp\_Name is functionally dependent on Emp\_Id.

## Types of 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:**

1. Consider a table with two columns Employee\_Id and Employee\_Name.
2. {Employee\_id, Employee\_Name}   →    Employee\_Id is a trivial functional dependency as
3. Employee\_Id is a subset of {Employee\_Id, Employee\_Name}.
4. 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:**

1. ID   →    Name,
2. 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 (IR1)

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

1. If X ⊇ Y then X  →    Y

**Example:**

1. X = {a, b, c, d, e}
2. Y = {a, b, c}

2. Augmentation Rule (IR2)

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

How to find Nth Highest Salary in SQL

1. If X    →  Y then XZ   →   YZ

**Example:**

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

3. Transitive Rule (IR3)

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

1. If X   →   Y and Y  →  Z then X  →   Z

4. Union Rule (IR4)

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

1. If X    →  Y and X   →  Z then X  →    YZ

**Proof:**

1. X → Y (given)  
2. X → Z (given)  
3. X → XY (using IR2 on 1 by augmentation with X. Where XX = X)  
4. XY → YZ (using IR2 on 2 by augmentation with Y)  
5. X → YZ (using IR3 on 3 and 4)

5. Decomposition Rule (IR5)

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.

1. If X   →   YZ then X   →   Y and X  →    Z

**Proof:**

1. X → YZ (given)  
2. YZ → Y (using IR1 Rule)  
3. X → Y (using IR3 on 1 and 2)

6. Pseudo transitive Rule (IR6)

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

1. If X   →   Y and YZ   →   W then XZ   →   W

**Proof:**

1. X → Y (given)  
2. WY → Z (given)  
3. WX → WY (using IR2 on 1 by augmenting with W)  
4. WX → Z (using IR3 on 3 and 2)

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 the undesirable characteristics like Insertion, Update and Deletion Anomalies.
* Normalization divides the larger table into the smaller table and links them using relationship.
* The normal form is used to reduce redundancy from the database table.

Types of Normal Forms

There are the four types of normal forms:



|  |  |
| --- | --- |
| **Normal Form** | **Description** |
| [1NF](https://www.javatpoint.com/dbms-first-normal-form) | A relation is in 1NF if it contains an atomic value. |
| [2NF](https://www.javatpoint.com/dbms-second-normal-form) | A relation will be in 2NF if it is in 1NF and all non-key attributes are fully functional dependent on the primary key. |
| [3NF](https://www.javatpoint.com/dbms-third-normal-form) | A relation will be in 3NF if it is in 2NF and no transition dependency exists. |
| [4NF](https://www.javatpoint.com/dbms-forth-normal-form) | A relation will be in 4NF if it is in Boyce Codd normal form and has no multi-valued dependency. |
| [5NF](https://www.javatpoint.com/dbms-fifth-normal-form) | A relation is in 5NF if it is in 4NF and not contains any join dependency and joining should be lossless. |

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.

**EMPLOYEE table:**

|  |  |  |  |
| --- | --- | --- | --- |
| **EMP\_ID** | **EMP\_NAME** | **EMP\_PHONE** | **EMP\_STATE** |
| 14 | John | 7272826385, 9064738238 | UP |
| 20 | Harry | 8574783832 | Bihar |
| 12 | Sam | 7390372389, 8589830302 | Punjab |

The decomposition of the EMPLOYEE table into 1NF has been shown below:

|  |  |  |  |
| --- | --- | --- | --- |
| **EMP\_ID** | **EMP\_NAME** | **EMP\_PHONE** | **EMP\_STATE** |
| 14 | John | 7272826385 | UP |
| 14 | John | 9064738238 | UP |
| 20 | Harry | 8574783832 | Bihar |
| 12 | Sam | 7390372389 | Punjab |
| 12 | Sam | 8589830302 | Punjab |

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.

**TEACHER table**

|  |  |  |
| --- | --- | --- |
| **TEACHER\_ID** | **SUBJECT** | **TEACHER\_AGE** |
| 25 | Chemistry | 30 |
| 25 | Biology | 30 |
| 47 | English | 35 |
| 83 | Math | 38 |
| 83 | Computer | 38 |

In the given table, non-prime attribute TEACHER\_AGE is dependent on TEACHER\_ID which is a proper subset of a candidate key. That's why it violates the rule for 2NF.

To convert the given table into 2NF, we decompose it into two tables:

Exception Handling in Java - Javatpoint

**TEACHER\_DETAIL table:**

|  |  |
| --- | --- |
| **TEACHER\_ID** | **TEACHER\_AGE** |
| 25 | 30 |
| 47 | 35 |
| 83 | 38 |

|  |  |
| --- | --- |
| **TEACHER\_ID** | **SUBJECT** |
| 25 | Chemistry |
| 25 | Biology |
| 47 | English |
| 83 | Math |
| 83 | Computer |

**TEACHER\_SUBJECT table:**

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.

**Example:**

**EMPLOYEE\_DETAIL table:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **EMP\_ID** | **EMP\_NAME** | **EMP\_ZIP** | **EMP\_STATE** | **EMP\_CITY** |
| 222 | Harry | 201010 | UP | Noida |
| 333 | Stephan | 02228 | US | Boston |
| 444 | Lan | 60007 | US | Chicago |
| 555 | Katharine | 06389 | UK | Norwich |
| 666 | John | 462007 | MP | Bhopal |

**Super key in the table above:**

* 1. {EMP\_ID}, {EMP\_ID, EMP\_NAME}, {EMP\_ID, EMP\_NAME, EMP\_ZIP}....so on

**Candidate key:** {EMP\_ID}

**Non-prime attributes:** In the given table, all attributes except EMP\_ID are non-prime.

Here, EMP\_STATE & EMP\_CITY dependent on EMP\_ZIP and EMP\_ZIP dependent on EMP\_ID. The non-prime attributes (EMP\_STATE, EMP\_CITY) transitively dependent on super key(EMP\_ID). It violates the rule of third normal form.

That's why we need to move the EMP\_CITY and EMP\_STATE to the new <EMPLOYEE\_ZIP> table, with EMP\_ZIP as a Primary key.

**EMPLOYEE table:**

|  |  |  |
| --- | --- | --- |
| **EMP\_ID** | **EMP\_NAME** | **EMP\_ZIP** |
| 222 | Harry | 201010 |
| 333 | Stephan | 02228 |
| 444 | Lan | 60007 |
| 555 | Katharine | 06389 |
| 666 | John | 462007 |

**EMPLOYEE\_ZIP table:**

|  |  |  |
| --- | --- | --- |
| **EMP\_ZIP** | **EMP\_STATE** | **EMP\_CITY** |
| 201010 | UP | Noida |
| 02228 | US | Boston |
| 60007 | US | Chicago |
| 06389 | UK | Norwich |
| 462007 | MP | Bhopal |

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:**

**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:**

1. EMP\_ID  →  EMP\_COUNTRY
2. 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:**

1. EMP\_ID   →    EMP\_COUNTRY
2. 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.

# Fourth normal form (4NF)

* A relation will be in 4NF if it is in Boyce Codd normal form and has no multi-valued dependency.
* For a dependency A → B, if for a single value of A, multiple values of B exists, then the relation will be a multi-valued dependency.

### **Example**

**STUDENT**

|  |  |  |
| --- | --- | --- |
| **STU\_ID** | **COURSE** | **HOBBY** |
| 21 | Computer | Dancing |
| 21 | Math | Singing |
| 34 | Chemistry | Dancing |
| 74 | Biology | Cricket |
| 59 | Physics | Hockey |

The given STUDENT table is in 3NF, but the COURSE and HOBBY are two independent entity. Hence, there is no relationship between COURSE and HOBBY.

In the STUDENT relation, a student with STU\_ID, **21** contains two courses, **Computer** and **Math** and two hobbies, **Dancing** and **Singing**. So there is a Multi-valued dependency on STU\_ID, which leads to unnecessary repetition of data.

So to make the above table into 4NF, we can decompose it into two tables:

Keep Watching

**STUDENT\_COURSE**

|  |  |
| --- | --- |
| **STU\_ID** | **COURSE** |
| 21 | Computer |
| 21 | Math |
| 34 | Chemistry |
| 74 | Biology |
| 59 | Physics |

**STUDENT\_HOBBY**

|  |  |
| --- | --- |
| **STU\_ID** | **HOBBY** |
| 21 | Dancing |
| 21 | Singing |
| 34 | Dancing |
| 74 | Cricket |
| 59 | Hockey |

# Fifth normal form (5NF)

* A relation is in 5NF if it is in 4NF and not contains any join dependency and joining should be lossless.
* 5NF is satisfied when all the tables are broken into as many tables as possible in order to avoid redundancy.
* 5NF is also known as Project-join normal form (PJ/NF).

### **Example**

|  |  |  |
| --- | --- | --- |
| **SUBJECT** | **LECTURER** | **SEMESTER** |
| Computer | Anshika | Semester 1 |
| Computer | John | Semester 1 |
| Math | John | Semester 1 |
| Math | Akash | Semester 2 |
| Chemistry | Praveen | Semester 1 |

In the above table, John takes both Computer and Math class for Semester 1 but he doesn't take Math class for Semester 2. In this case, combination of all these fields required to identify a valid data.

Suppose we add a new Semester as Semester 3 but do not know about the subject and who will be taking that subject so we leave Lecturer and Subject as NULL. But all three columns together acts as a primary key, so we can't leave other two columns blank.

So to make the above table into 5NF, we can decompose it into three relations P1, P2 & P3:

**P1**

|  |  |
| --- | --- |
| **SEMESTER** | **SUBJECT** |
| Semester 1 | Computer |
| Semester 1 | Math |
| Semester 1 | Chemistry |
| Semester 2 | Math |

**P2**

|  |  |
| --- | --- |
| **SUBJECT** | **LECTURER** |
| Computer | Anshika |
| Computer | John |
| Math | John |
| Math | Akash |
| Chemistry | Praveen |

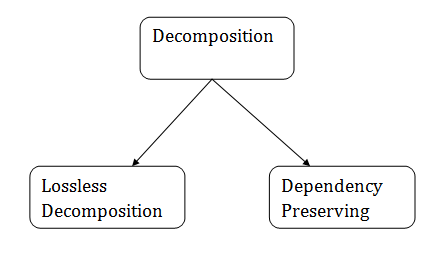
**P3**

|  |  |
| --- | --- |
| **SEMSTER** | **LECTURER** |
| Semester 1 | Anshika |
| Semester 1 | John |
| Semester 1 | John |
| Semester 2 | Akash |
| Semester 1 | Praveen |

# Relational Decomposition

* When a relation in the relational model is not in appropriate normal form then the decomposition of a relation is required.
* In a database, it breaks the table into multiple tables.
* If the relation has no proper decomposition, then it may lead to problems like loss of information.
* Decomposition is used to eliminate some of the problems of bad design like anomalies, inconsistencies, and redundancy.

## Types of Decomposition



### **Lossless Decomposition**

* If the information is not lost from the relation that is decomposed, then the decomposition will be lossless.
* The lossless decomposition guarantees that the join of relations will result in the same relation as it was decomposed.
* The relation is said to be lossless decomposition if natural joins of all the decomposition give the original relation.

**Example:**

**EMPLOYEE\_DEPARTMENT table:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **EMP\_ID** | **EMP\_NAME** | **EMP\_AGE** | **EMP\_CITY** | **DEPT\_ID** | **DEPT\_NAME** |
| 22 | Denim | 28 | Mumbai | 827 | Sales |
| 33 | Alina | 25 | Delhi | 438 | Marketing |
| 46 | Stephan | 30 | Bangalore | 869 | Finance |
| 52 | Katherine | 36 | Mumbai | 575 | Production |
| 60 | Jack | 40 | Noida | 678 | Testing |

The above relation is decomposed into two relations EMPLOYEE and DEPARTMENT

**EMPLOYEE table:**

|  |  |  |  |
| --- | --- | --- | --- |
| **EMP\_ID** | **EMP\_NAME** | **EMP\_AGE** | **EMP\_CITY** |
| 22 | Denim | 28 | Mumbai |
| 33 | Alina | 25 | Delhi |
| 46 | Stephan | 30 | Bangalore |
| 52 | Katherine | 36 | Mumbai |
| 60 | Jack | 40 | Noida |

**DEPARTMENT table**

|  |  |  |
| --- | --- | --- |
| **DEPT\_ID** | **EMP\_ID** | **DEPT\_NAME** |
| 827 | 22 | Sales |
| 438 | 33 | Marketing |
| 869 | 46 | Finance |
| 575 | 52 | Production |
| 678 | 60 | Testing |

Now, when these two relations are joined on the common column "EMP\_ID", then the resultant relation will look like:

**Employee ⋈ Department**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **EMP\_ID** | **EMP\_NAME** | **EMP\_AGE** | **EMP\_CITY** | **DEPT\_ID** | **DEPT\_NAME** |
| 22 | Denim | 28 | Mumbai | 827 | Sales |
| 33 | Alina | 25 | Delhi | 438 | Marketing |
| 46 | Stephan | 30 | Bangalore | 869 | Finance |
| 52 | Katherine | 36 | Mumbai | 575 | Production |
| 60 | Jack | 40 | Noida | 678 | Testing |

Hence, the decomposition is Lossless join decomposition.

### **Dependency Preserving**

* It is an important constraint of the database.
* In the dependency preservation, at least one decomposed table must satisfy every dependency.
* If a relation R is decomposed into relation R1 and R2, then the dependencies of R either must be a part of R1 or R2 or must be derivable from the combination of functional dependencies of R1 and R2.
* For example, suppose there is a relation R (A, B, C, D) with functional dependency set (A->BC). The relational R is decomposed into R1(ABC) and R2(AD) which is dependency preserving because FD A->BC is a part of relation R1(ABC).

Multivalued Dependency

* Multivalued dependency occurs when two attributes in a table are independent of each other but, both depend on a third attribute.
* A multivalued dependency consists of at least two attributes that are dependent on a third attribute that's why it always requires at least three attributes.

**Example:** Suppose there is a bike manufacturer company which produces two colors(white and black) of each model every year.

|  |  |  |
| --- | --- | --- |
| **BIKE\_MODEL** | **MANUF\_YEAR** | **COLOR** |
| M2011 | 2008 | White |
| M2001 | 2008 | Black |
| M3001 | 2013 | White |
| M3001 | 2013 | Black |
| M4006 | 2017 | White |
| M4006 | 2017 | Black |

Here columns COLOR and MANUF\_YEAR are dependent on BIKE\_MODEL and independent of each other.

In this case, these two columns can be called as multivalued dependent on BIKE\_MODEL. The representation of these dependencies is shown below:

1. BIKE\_MODEL   →  →  MANUF\_YEAR
2. BIKE\_MODEL   →  →  COLOR

This can be read as "BIKE\_MODEL multidetermined MANUF\_YEAR" and "BIKE\_MODEL multidetermined COLOR".