# Basic Concepts of DBMS

In this section, we will read about:

* Purpose of database systems
* Data abstraction
* Database Users
* Data Independence (Logical & Physical)
* Instance & Schemes
* Three layered Architecture of DBMS
* Different Levels of Abstraction.
* Data Modeling
* E-R Modeling
* Logical Model: Object & Record based – Object oriented model - Entity relationship models
* Entity sets & relationships sets
* Concept of attributes and relationships
* Introduction to mapping constraints.
* Basic Concepts of ER Model in DBMS
* Introduction to DBMS
* Structure of DBMS
* Relational Models
* Introduction to Hierarchical Model and Network Model
* Introduction to RDBMS and Relational Models
* Introduction to relational algebra and relational calculus
* Understanding database technologies
* Relational Data Structure
* Keys and Relational Data Manipulation
* Relational Algebra
* Relational Algebraic Operations
* Set Operations
* Fundamental Operations
* Relational Calculus
* Data Definition Language
* Operators: select, project, join, rename etc.

# Purpose of database systems

## What is Database?

A **database** is an organized collection of data, so that it can be easily accessed and managed.

You can organize data into tables, rows, columns, and index it to make it easier to find relevant information.

**Database handlers** create a database in such a way that only one set of software program provides access of data to all the users.

The **main purpose** of the database is to operate a large amount of information by storing, retrieving, and managing data.

There are many **dynamic websites** on the World Wide Web nowadays which are handled through databases. For example, a model that checks the availability of rooms in a hotel. It is an example of a dynamic website that uses a database.

There are many **databases available** like MySQL, Sybase, Oracle, MongoDB, Informix, PostgreSQL, SQL Server, etc.

Modern databases are managed by the database management system (DBMS).

**SQL** or Structured Query Language is used to operate on the data stored in a database. SQL depends on relational algebra and tuple relational calculus.

A cylindrical structure is used to display the image of a database.



Image1: Database

Reference: https://static.javatpoint.com/sqlpages/images/database.png

## What is DBMS?

**Database Management System (DBMS)** refers to the technology solution used to optimize and manage the storage and retrieval of data from databases. DBMS offers a systematic approach to manage databases via an interface for users as well as workloads accessing the databases via apps. The management responsibilities for DBMS encompass information within the databases, the processes applied to databases (such as access and modification), and the database’s logic structure. DBMS also facilitates additional administrative operations such as change management, disaster recovery, compliance, and performance monitoring, among others.

In order to facilitate these functions, DBMS has the following key components:

**Software.** DBMS is primarily a software system that can be considered as a management console or an interface to interact with and manage databases. The interfacing also spreads across real-world physical systems that contribute data to the backend databases. The OS, networking software, and the hardware infrastructure is involved in creating, accessing, managing, and processing the databases.

**Data.** DBMS contains operational data, access to database records and metadata as a resource to perform the necessary functionality. The data may include files with such as index files, administrative information, and data dictionaries used to represent data flows, ownership, structure, and relationships to other records or objects.

**Procedures.** While not a part of the DBMS software, procedures can be considered as instructions on using DBMS. The documented guidelines assist users in designing, modifying, managing, and processing databases.

**Database languages.** These are components of the DBMS used to access, modify, store, and retrieve data items from databases; specify database schema;

control user access; and perform other associated database management operations. Types of DBMS languages include Data Definition Language (DDL), Data Manipulation Language (DML), Database Access Language (DAL) and Data Control Language (DCL).

**Query processor.** As a fundamental component of the DBMS, the query processor acts as an intermediary between users and the DBMS data engine in order to communicate query requests. When users enter an instruction in SQL language, the command is executed from the high-level language instruction to a low-level language that the underlying machine can understand and process to perform the appropriate DBMS functionality. In addition to instruction parsing and translation, the query processor also optimizes queries to ensure fast processing and accurate results.

**Runtime database manager.** A centralized management component of DBMS that handles functionality associated with runtime data, which is commonly used for context-based database access. This component checks for user authorization to request the query; processes the approved queries; devises an optimal strategy for query execution; supports concurrency so that multiple users can simultaneously work on same databases; and ensures integrity of data recorded into the databases.

**Database manager.** Unlike the runtime database manager that handles queries and data at runtime, the database manager performs DBMS functionality associated with the data within databases. Database manager allows a set of commands to perform different DBMS operations that include creating, deleting, backup, restoring, cloning, and other database maintenance tasks. The database manager may also be used to update the database with patches from vendors.

**Database engine.** This is the core software component within the DBMS solution that performs the core functions associated with data storage and retrieval. A database engine is also accessible via APIs that allow users or apps to create, read, write, and delete records in databases.

**Reporting.** The report generator extracts useful information from DBMS files and displays it in structured format based on defined specifications. This

information may be used for further analysis, decision making, or business intelligence.

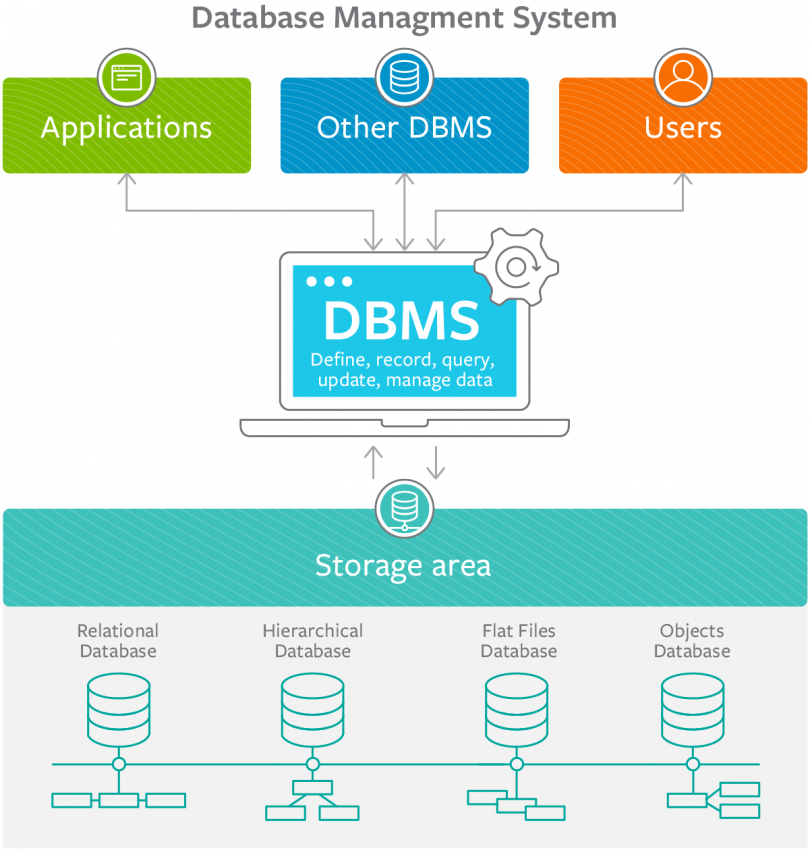


Image2: DBMS

Reference: https://blogs.bmc.com/wp-content/uploads/2018/08/dbms-database-management-systems-810x898.png

## What are the types of DBMS?

Depending upon the usage requirements, there are following types of databases available in the market:

* Centralised database.
* Distributed database.
* Personal database.
* End-user database.
* Commercial database.
* NoSQL database.
* Operational database.
* Relational database.
* Cloud database.
* Object-oriented database.
* Graph database.

### Centralised Database

The information(data) is stored at a centralized location and the users from different locations can access this data. This type of database contains application procedures that help the users to access the data even from a remote location.

Various kinds of authentication procedures are applied for the verification and validation of end users, likewise, a registration number is provided by the application procedures which keeps a track and record of data usage. The local area office handles this thing.

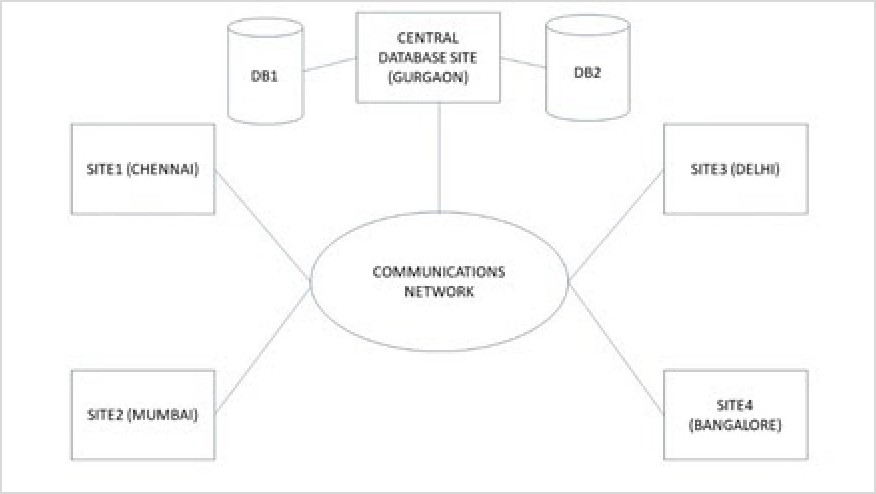


Image3: Centralised Database

Reference: https://[www.tutorialspoint.com/assets/questions/images/109302-1532341956.jpg](http://www.tutorialspoint.com/assets/questions/images/109302-1532341956.jpg)

### Distributed Database

Just opposite of the centralized database concept, the distributed database has contributions from the common database as well as the information captured by local computers also. The data is not at one place and is distributed at various sites of an organization. These sites are connected to each other with the help of communication links which helps them to access the distributed data easily.

You can imagine a distributed database as a one in which various portions of a database are stored in multiple different locations(physical) along with the application procedures which are replicated and distributed among various points in a network.

There are two kinds of distributed database, viz. homogenous and heterogeneous. The databases which have same underlying hardware and run over same operating systems and application procedures are known as homogeneous DDB, for eg. All physical locations in a DDB. Whereas, the operating systems, underlying hardware as well as application procedures can be different at various sites of a DDB which is known as heterogeneous DDB.

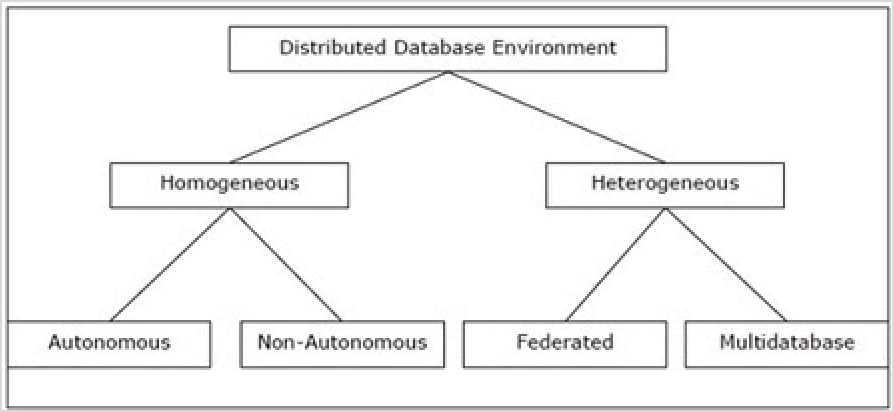


Image4: Distributed Database

Reference: https://[www.tutorialspoint.com/assets/questions/images/104606-1532341968.jpg](http://www.tutorialspoint.com/assets/questions/images/104606-1532341968.jpg)

### Personal Database

Data is collected and stored on personal computers which is small and easily manageable. The data is generally used by the same department of an organization and is accessed by a small group of people.

### End User Database

The end user is usually not concerned about the transaction or operations done at various levels and is only aware of the product which may be a software or an application. Therefore, this is a shared database which is specifically designed for the end user, just like different levels’ managers. Summary of whole information is collected in this database.

### Commercial Database

These are the paid versions of the huge databases designed uniquely for the users who want to access the information for help. These databases are subject specific, and one cannot afford to maintain such a huge information. Access to such databases is provided through commercial links.

### NoSQL Database

These are used for large sets of distributed data. There are some big data performance issues which are effectively handled by relational databases, such kind of issues are easily managed by NoSQL databases. There are very efficient in analyzing large size unstructured data that may be stored at multiple virtual servers of the cloud.

### Operational Database

Information related to operations of an enterprise is stored inside this database. Functional lines like marketing, employee relations, customer service etc. require such kind of databases.

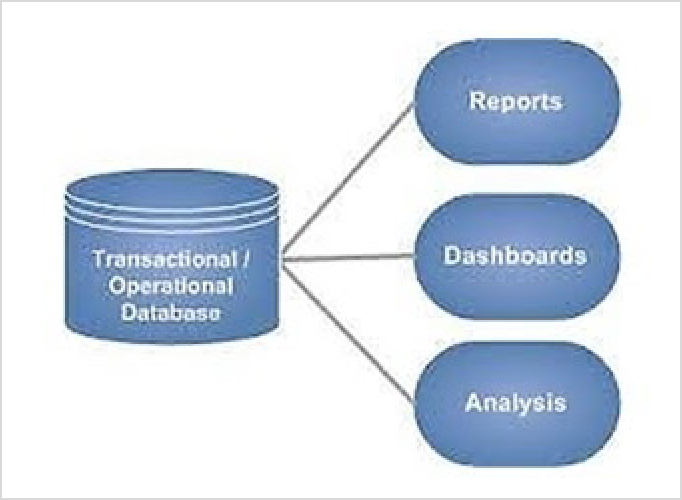


Image5: Operational Database

Reference: https://[www.tutorialspoint.com/assets/questions/images/104532-1532341980.jpg](http://www.tutorialspoint.com/assets/questions/images/104532-1532341980.jpg)

### Relational Databases

These databases are categorized by a set of tables where data gets fit into a pre-defined category. The table consists of rows and columns where the column has an entry for data for a specific category and rows contains instance for that data defined according to the category. The Structured Query Language (SQL) is the standard user and application program interface for a relational database.

There are various simple operations that can be applied over the table which makes these databases easier to extend, join two databases with a common relation and modify all existing applications.

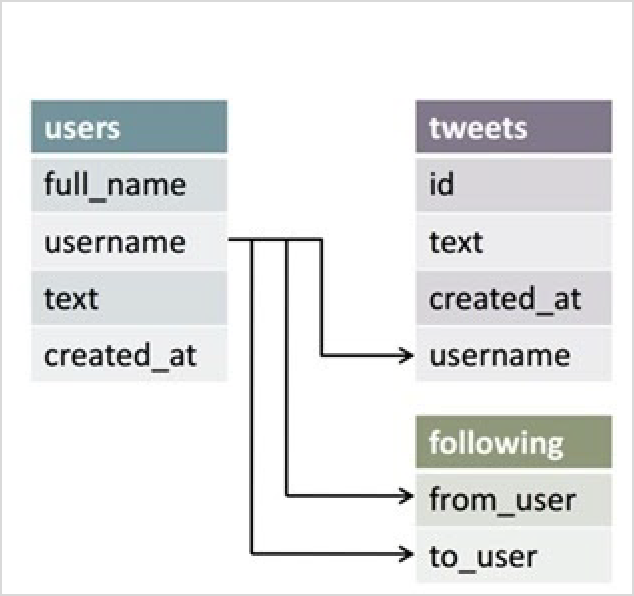


Image6: Relational Database

Reference: https://[www.tutorialspoint.com/assets/questions/images/112548-1532342000.jpg](http://www.tutorialspoint.com/assets/questions/images/112548-1532342000.jpg)

### Cloud Databases

Now a day, data has been specifically getting stored over clouds also known as a virtual environment, either in a hybrid cloud, public or private cloud. A cloud database is a database that has been optimized or built for such a virtualized environment. There are various benefits of a cloud database, some of which are the ability to pay for storage capacity and bandwidth on a per-user basis, and they provide scalability on demand, along with high availability.

A cloud database also gives enterprises the opportunity to support business applications in a software-as-a-service deployment.

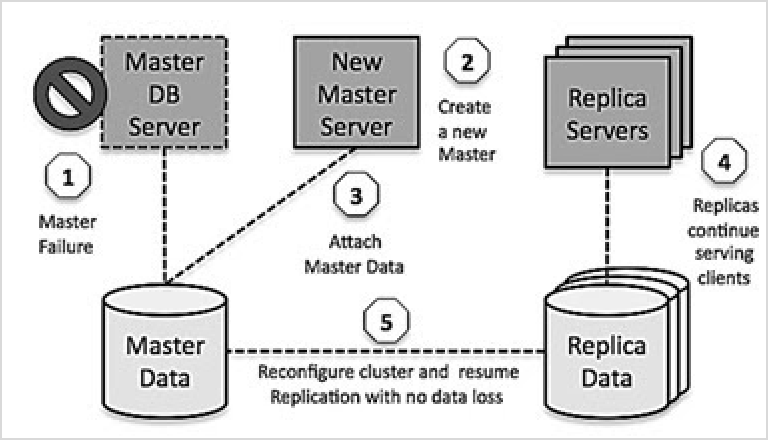


Image7: Cloud Database

Reference: https://[www.tutorialspoint.com/assets/questions/images/112415-1532342015.jpg](http://www.tutorialspoint.com/assets/questions/images/112415-1532342015.jpg)

### Object-Oriented Databases

An object-oriented database is a collection of object-oriented programming and relational database. There are various items which are created using object-oriented programming languages like C++, Java which can be stored in relational databases, but object-oriented databases are well-suited for those items.

An object-oriented database is organized around objects rather than actions, and data rather than logic. For example, a multimedia record in a relational database can be a definable data object, as opposed to an alphanumeric value.

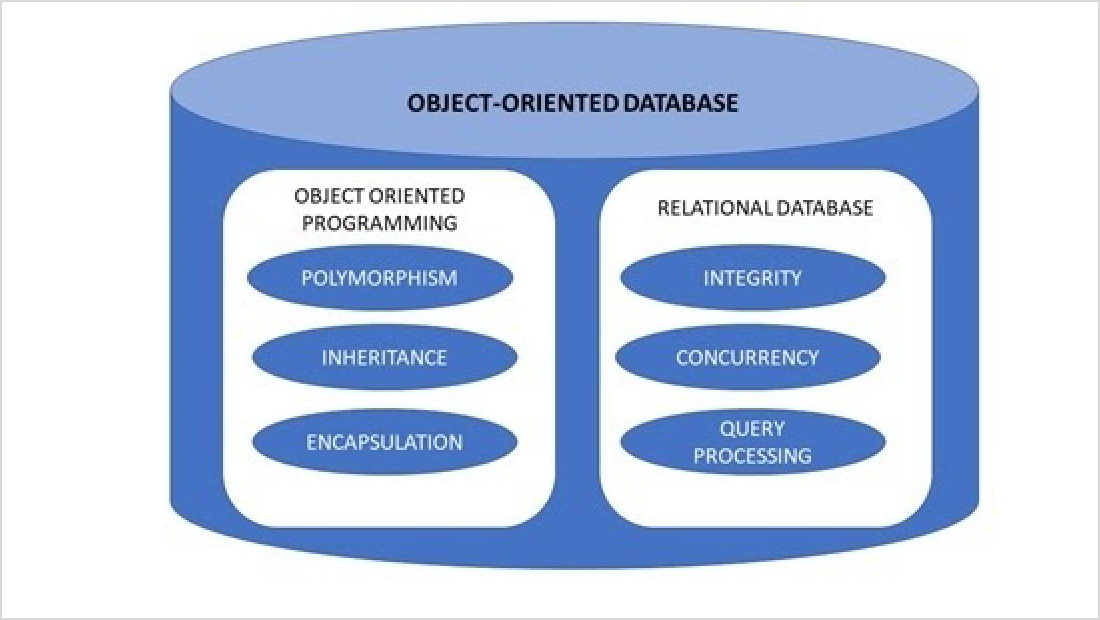


Image8: Object OrientedDatabase

Reference: https://[www.tutorialspoint.com/assets/questions/images/113587-1532342027.jpg](http://www.tutorialspoint.com/assets/questions/images/113587-1532342027.jpg)

### Graph Databases

The graph is a collection of nodes and edges where each node is used to represent an entity and each edge describes the relationship between entities. A graph-oriented database, or graph database, is a type of NoSQL database that uses graph theory to store, map and query relationships.

Graph databases are basically used for analyzing interconnections. For example, companies might use a graph database to mine data about customers from social media.

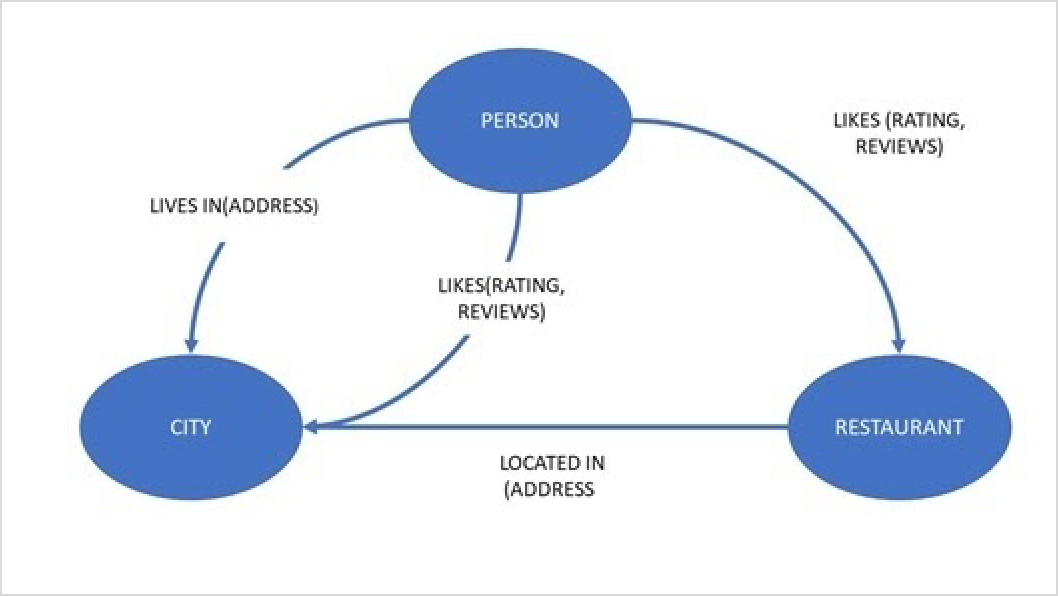


Image9: Graph Database

Reference: https://[www.tutorialspoint.com/assets/questions/images/114315-1532342037.jpg](http://www.tutorialspoint.com/assets/questions/images/114315-1532342037.jpg)

## DBMS vs. File System

There are following differences between DBMS and File system:

|  |  |
| --- | --- |
| **DBMS** | **File System** |
| DBMS is a collection of data. In DBMS, the user is not required to write the  procedures. | File system is a collection of data. In this system, the user has to write the procedures for managing the database. |
| DBMS gives an abstract view of data that  hides the details. | File system provides the detail of the data representation  and storage of data. |
| DBMS provides a crash recovery mechanism, i.e., DBMS protects the user  from the system failure. | 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 lost. |
| DBMS provides a good protection  mechanism. | It is very difficult to protect a file under the file system. |
| DBMS contains a wide variety of sophisticated techniques to store and  retrieve the data. | File system can't efficiently store and retrieve the data. |
| 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. |

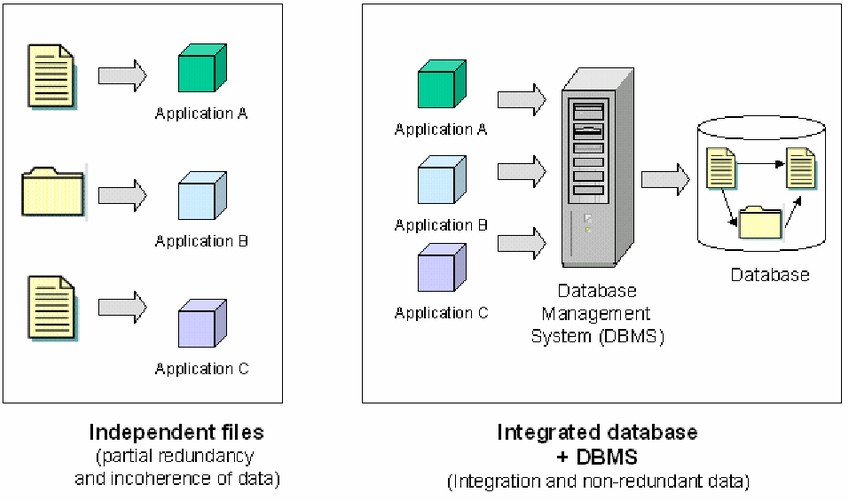


Image10: DBMS vs File System

Reference: https://www.researchgate.net/profile/Yvan\_Bedard/publication/11041878/figure/fig1/AS:277384914849800@1443145125825/independent-files- vs-integrated-database-DBMS.png

# Data Abstraction

It is one of the main and important characteristics of database approach. Data abstraction is the concept of hiding the details like data definition, data organization and storage of data from the end users and showing them only the essential things as per their requirement.

For example, an end user may be naïve user, application programmer, or an expert in DBMS. Their requirement in viewing data is different for different user. The users may not be interested in everything about a database. Since most of the end users of any database may not have enough idea about the implementation, the database developers have hidden many of the unwanted (for end users) information through several levels of data abstraction.

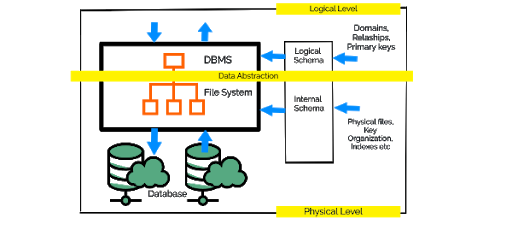


Image11: Data Abstraction Reference:

https://lh3.googleusercontent.com/proxy/SPb-TFbsIwUqDqf3AzqIpABk6nb\_6eIt5QB3howul7kzRsiYJQ29r3JOoJMYGRtrHkZsRrmhe9osOTm Hv77fBwl1knff7cExBkZHBSI7bi1c10wW5vv5NYA

# Database Users

**Database Users** are the one who interacts with the system. There can be four **different types of users** according to the way they interact with the system and for all the different users, different kind of user interfaces are designed as well. The four **types of users** are:

1. Naive users
2. Application Programmers
3. Sophisticated Users
4. DBA

Let us have a look over all the four users and their interfaces:

## Naive Users

Naive users are also termed as unsophisticated users and they interact with the system by calling anyone application program that has been written previously.

**For e.g. –** To transfer the program from one account to another, there is a need for an application program called transfer.

The user interface that is required for the naïve users is a forms interface, in which the user can fill the required fields.

Naive users can also easily read the reports that are generated from the database.

## Application programmers

Application programmers are the one who is responsible to write the application programs. They develop user interfaces through different tools. To construct the forms and the reports such that there is no need to write the program, there is a tool named Rapid Application Development (RAD).

Some special type of programming languages is also available such that includes vital control structures such as for loops, while loops and many others with the data manipulation language's

statements. These special programming languages are termed as fourth-generation languages and they include the special features to provide the ability for the generation of the forms and to display the data on the screen.

In today's world, a large variety of commercial database systems includes these fourth generation languages.

## Sophisticated users

Sophisticated users aren't interested in writing programs and they interact with the system without writing any programs. Contrary, they use database query languages to interact with the system.

Sophisticated Users submit their queries to a query processor. Query Processor provides the facility to break the DML statements into instructions that can be understood by the storage manager.

Analysts are one among the sophisticated users. They use the tools to perform their task such as:

1. **Online analytical processing (OLAP)** - It helps the analysts to view them the summaries of the data in different ways.
2. **Data Mining Tools** – It helps the analysts find a certain kind of pattern in the given data.

**DBA [Database Administrators]**

* In any organization where many people use the same resources, there is a need for a chief administrator to oversee and manage these resources.
* In a database environment, the primary resource is the database itself, and the secondary resource is the DBMS and related software.
* Administering these resources is the responsibility of the database administrator (DBA).
* The DBA is responsible for authorizing access to the database, coordinating and monitoring its use, and acquiring software and hardware resources as needed.
* The DBA is accountable for problems such as security breaches and poor system response time. In large organizations, the DBA is assisted by a staff that carries out these functions.

# Data Independence (Logical & Physical)

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

**Types of Data Independence**

In DBMS there are two types of data independence

1. Physical data independence
2. Logical data independence.

**Levels of Database**

Before we learn Data Independence, a refresher on Database Levels is important. The database has 3 levels as shown in the diagram below

1. Physical/Internal
2. Conceptual
3. External

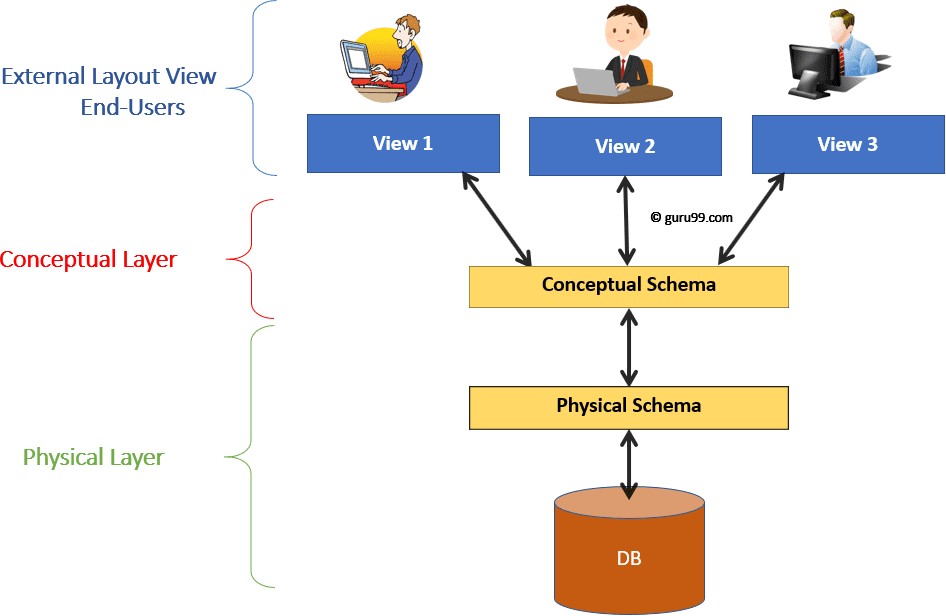


Image12: Levels of DBMS Architecture Diagram

Reference: https://[www.guru99.com/images/1/042919\_0417\_DataIndepen1.png](http://www.guru99.com/images/1/042919_0417_DataIndepen1.png)

Consider an Example of a University Database. At the different levels this is how the implementation will look like:

External Schema **View 1**: Course info(cid:int,cname:string)

Type of Schema Implementation

**View 2**: studeninfo(id:int. name:string)

Conceptual Shema

Students(id: int, name: string, login: string, age: integer)

Courses(id: int, cname.string, credits:integer) Enrolled(id: int, grade:string)

Physical Schema

* Relations stored as unordered files.
* Index on the first column of Students.

**Physical Data Independence**

Physical data independence helps you to separate conceptual levels from the internal/physical levels. It allows you to provide a logical description of the database without the need to specify physical structures. Compared to Logical Independence, it is easy to achieve physical data independence.

With Physical independence, you can easily change the physical storage structures or devices with an effect on the conceptual schema. Any change done would be absorbed by the mapping between the conceptual and internal levels. Physical data independence is achieved by the presence of the internal level of the database and then the transformation from the conceptual level of the database to the internal level.

### Examples of changes under Physical Data Independence

Due to Physical independence, any of the below changes will not affect the conceptual layer.

* + Using a new storage device like Hard Drive or Magnetic Tapes
  + Modifying the file organization technique in the Database
  + Switching to different data structures.
  + Changing the access method.
  + Modifying indexes.
  + Changes to compression techniques or hashing algorithms.
  + Change of Location of Database from say C drive to D Drive

**Logical Data Independence**

Logical Data Independence is the ability to change the conceptual scheme without changing

1. External views
2. External API or programs

Any change made will be absorbed by the mapping between external and conceptual levels.

When compared to Physical Data independence, it is challenging to achieve logical data independence.

### Examples of changes under Logical Data Independence

Due to Logical independence, any of the below change will not affect the external layer.

1. Add/Modify/Delete a new attribute, entity or relationship is possible without a rewrite of existing application programs
2. Merging two records into one
3. Breaking an existing record into two or more records

**Difference between Physical and Logical Data Independence**

Logical Data Independence is mainly concerned with the structure or changing the data definition.

**Logical Data Independence Physical Data Independence**

It is difficult as the retrieving of data is mainly dependent on the logical structure of data.

It is easy to retrieve.

Mainly concerned with the storage of the data.

Compared to Logic Physical independence it is difficult to achieve logical data independence.

Compared to Logical Independence it is easy to achieve physical data independence.

You need to make changes in the Application program if new fields are added or deleted from the database.

A change in the physical level usually does not need change at the Application program level.

Modification at the logical levels is significant whenever the logical structures of the database are changed.

Concerned with conceptual schema Concerned with internal schema

Modifications made at the internal levels may or may not be needed to improve the performance of the structure.

Example: Add/Modify/Delete a new attribute Example: change in compression techniques, hashing

algorithms, storage devices, etc

**Importance of Data Independence**

* + Helps you to improve the quality of the data
  + Database system maintenance becomes affordable
  + Enforcement of standards and improvement in database security
  + You don't need to alter data structure in application programs
  + Permit developers to focus on the general structure of the Database rather than worrying about the internal implementation
  + It allows you to improve state which is undamaged or undivided
  + Database incongruity is vastly reduced.
  + Easily making modifications in the physical level is needed to improve the performance of the system.

# Instance & Schemes

* 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 shows 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.

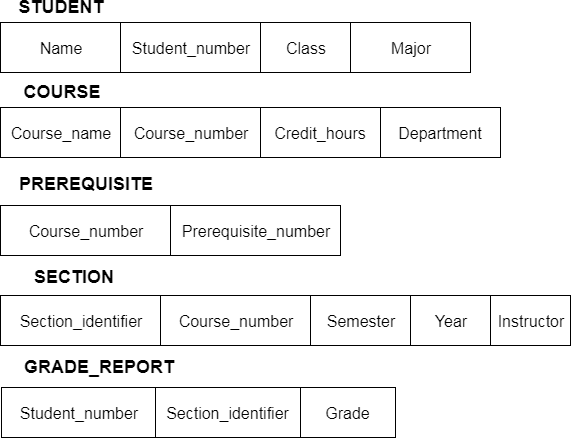


Image13: Data Model Schema & Instance

Reference: https://static.javatpoint.com/dbms/images/dbms-data-model-schema-and-instance.png

**Differences Between Schema and Instance**

* 1. A schema is the design representation of a database whereas instance is the snapshot of a database at a particular moment.
  2. Instance changes very frequently, whenever data is removed or added in the database. As against, the changes in schema occurs rarely.
  3. For example, schema and instance can be easily perceived by analogy to a program. At the time of writing a program in a programming language, the variables of that program is declared at first, this is analogous to the schema definition. Additionally, each variable in a program must have some values associated at a particular time; this is similar to an instance.

## Sub-Schema

A subschema lets the user have access to different areas of applications in which the user designed. The areas that are included in an application are set, types, record types, data items, and data aggregates. Schemas may have many different subschemas’ that are all very different. There are several different reasons for subschema’s. One reason for a subschema is to let a user or programmer see different views without having to see all the data contained in the database. Another reason would be to improve security so that someone who is not authorized can change or add to the data.

Data definition language describes the database in which there are possibly many programs that have been writing in different program languages. The description is in the form of names and characteristics of data items, data aggregates, records, areas, and sets included in the database, and the relationships that exist and that have to be maintained between occurrences of elements within the database.

A data item is represented by a value in a database, which is an occurrence of the smallest unit of the named data.

A Data aggregate is the occurrence of named collections of data items that are inside the record. There are two kinds, the first is a vector that is a one-dimensional sequence of data items. These all have identical characteristics. The other is a repeating group and is a collection of data that occurs several times within a record occurrence.

Record is an occurrence of a named collection of zero, one, or more items. Keys are a way to uniquely identify a record.

Set is an occurrence of a named collection of records. For every record in the schema there must be one occurrence.

Area is a named collection of records but it does not have to be linked between each record.

Database contains all the records within the schema; however each database must contain a separate schema.

# Three Layered of Architecture in DBMS

* The DBMS design depends upon its architecture. The basic client/server architecture is used to deal with a large number of PCs, web servers, database servers and other components that are connected with networks.
* The client/server architecture consists of many PCs and a workstation which are connected via the network.
* DBMS architecture depends upon how users are connected to the database to get their request done.

## Types of DBMS Architecture

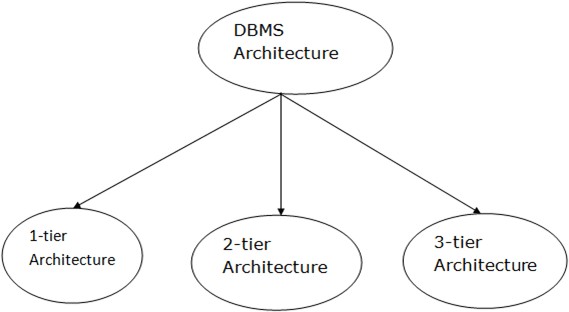


Image14: DBMS Architecture

Reference: https://static.javatpoint.com/dbms/images/dbms-architecture.png

Database architecture can be seen as a single tier or multi-tier. But logically, database architecture is of two types like: **2-tier architecture** and **3-tier architecture**.

## Tier Architecture

* + In this architecture, the database is directly available to the user. It means the user can directly sit on the DBMS and uses it.
  + Any changes done here will directly be done on the database itself. It doesn't provide a handy tool for end users.
  + The 1-Tier architecture is used for development of the local application, where programmers can directly communicate with the database for the quick response.

## Tier Architecture

* + The 2-Tier architecture is same as basic client-server. In the two-tier architecture, applications on the client end can directly communicate with the database at the server side. For this interaction, API's like: **ODBC**, **JDBC** are used.
  + The user interfaces and application programs are run on the client-side.
  + The server side is responsible to provide the functionalities like: query processing and transaction management.
  + To communicate with the DBMS, client-side application establishes a connection with the server side.

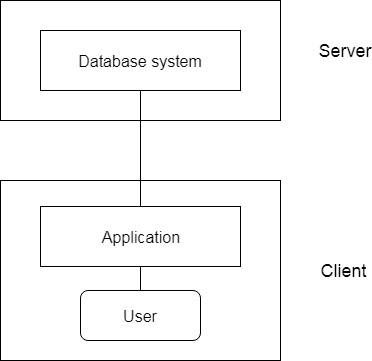


Image15: 2-Tier Architecture

Reference: https://static.javatpoint.com/dbms/images/dbms-2-tier-architecture.png

## Tier Architecture

* + The 3-Tier architecture contains another layer between the client and server. In this architecture, client can't directly communicate with the server.
  + The application on the client-end interacts with an application server which further communicates with the database system.
  + End user has no idea about the existence of the database beyond the application server. The database also has no idea about any other user beyond the application.
  + The 3-Tier architecture is used in case of large web application.

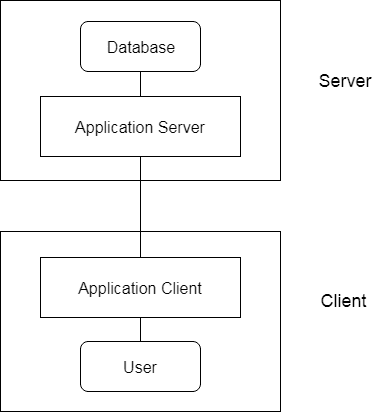


Image16: 3-Tier Architecture

Reference: https://static.javatpoint.com/dbms/images/dbms-3-tier-architecture.png

# Different Levels of Abstraction

* + The three schema architecture is also called ANSI/SPARC architecture or different-level of Abstraction.
  + 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 level of Abstraction is as follows

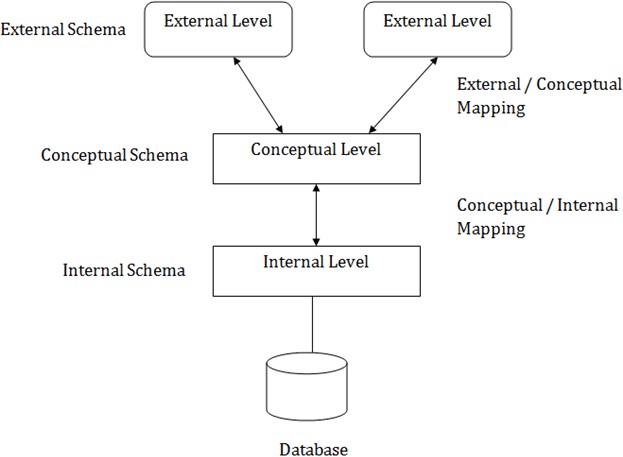


Image17: Level of Abstraction

Reference: https://static.javatpoint.com/dbms/images/dbms-three-schema-architecture.png

### 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 transforms the request from the conceptual to internal level.

## Physical level or 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.

## Logical Level or 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.

## View Level or External Level

* + At the external level, a database contains several schemas that are 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 in and hides the remaining database from that user group.
  + The view schema describes the end user interaction with database systems.

# Data Modelling

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:

## 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.

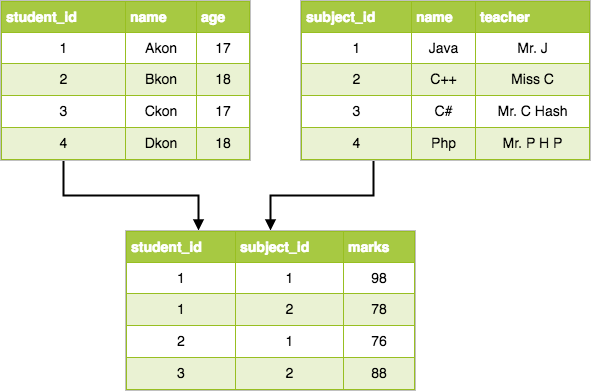


Image18: Relational Data Models

Reference: https://[www.studytonight.com/dbms/images/relational-dbms-model.png](http://www.studytonight.com/dbms/images/relational-dbms-model.png)

## 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'.

## 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.

## 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.

## Hierarchical Model

This database model organises data into a tree-like-structure, with a single root, to which all the other data is linked. The hierarchy starts from the Root data, and expands like a tree, adding child nodes to the parent nodes.

In this model, a child node will only have a single parent node.

This model efficiently describes many real-world relationships like indexes of a book, recipes etc.

In hierarchical models, data is organised into a tree-like structure with one one-to-many relationship between two different types of data, for example, one department can have many courses, many professors and of-course many students.

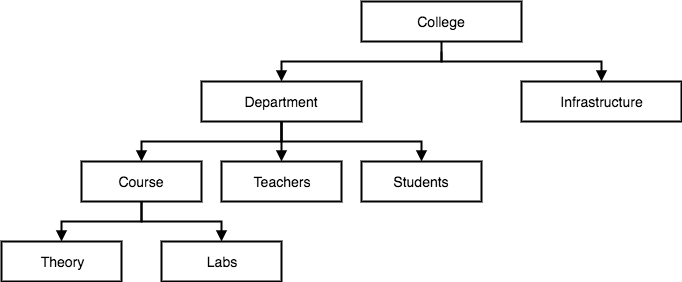


Image19: Hierarchical Model

## Network Model

Reference: https://[www.studytonight.com/dbms/images/hierarchical-dbms-model.png](http://www.studytonight.com/dbms/images/hierarchical-dbms-model.png)

This is an extension of the Hierarchical model. In this model data is organised more like a graph, and are allowed to have more than one parent node.

In this database model data is more related as more relationships are established in this database model. Also, as the data is more related, hence accessing the data is also easier and faster. This database model was used to map many-to-many data relationships.

This was the most widely used database model, before the Relational Model was introduced.

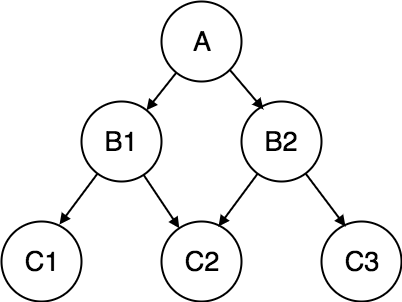


Image20: Network Model

## Flat Data Model

Reference: https://[www.studytonight.com/dbms/images/network-dbms-model.png](http://www.studytonight.com/dbms/images/network-dbms-model.png)

Flat data model is the first and foremost introduced model and in this all the data used is kept in the same plane. Since it was used earlier this model was not so scientific.

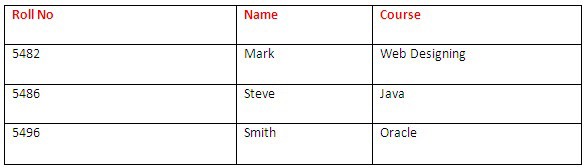


Image21: Flat Data Model Reference:https://whatisdbms.com/wp-content/uploads/2016/06/Flat-Data-Model-in-DBMS.jpg

## Record base Data Model

Record base model is used to specify the overall structure of the database and in this there are many record types. Each record type has fixed no. of fields having the fixed length.

# ER modeling

* + 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.

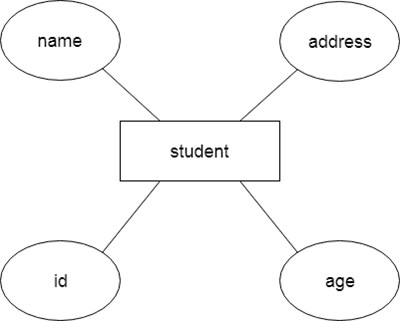


Image22: ER Model

Reference: https://static.javatpoint.com/dbms/images/dbms-er-model-concept.png

### Component of ER Diagram

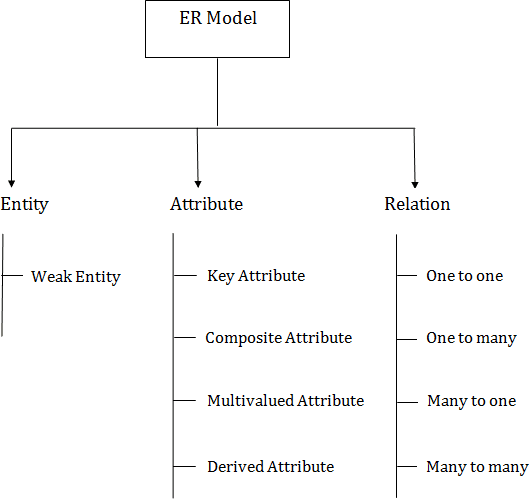


Image23: ER Model Concept Diagram

Reference: https://static.javatpoint.com/dbms/images/dbms-er-model-concept-diagram.png

# Logical Data Model

The logical data model is the one used most in designing BI applications. It builds upon the requirements provided by the business group. It includes a further level of detail, supporting both the business system-related and data requirements.

The business rules are appropriated into the logical data model, where they form relationships between the various data objects and entities. As opposed to a conceptual data model, which may have very general terms, the logical data model is the first step in designing and building out the architecture of the applications.

Like the conceptual data model, the logical data model is independent of specific database and data storage structures. It uses indexes and foreign keys to represent data relationships, but these are defined in a generic database context independent of any specific DBMS product.

The characteristics of the logical data model include:

•Features independent of specific database and data storage structures.

•Specific entities and attributes to be implemented.

•Identification of the business rules and relationships between those entities and attributes.

•Definitions of the primary keys, foreign keys, alternate keys, and inversion entities.

The logical model is used as a bridge from the application designer’s view to the database design and the developer’s specifications. This model should be used to validate whether the resulting applications that are built fulfill business and data requirements.

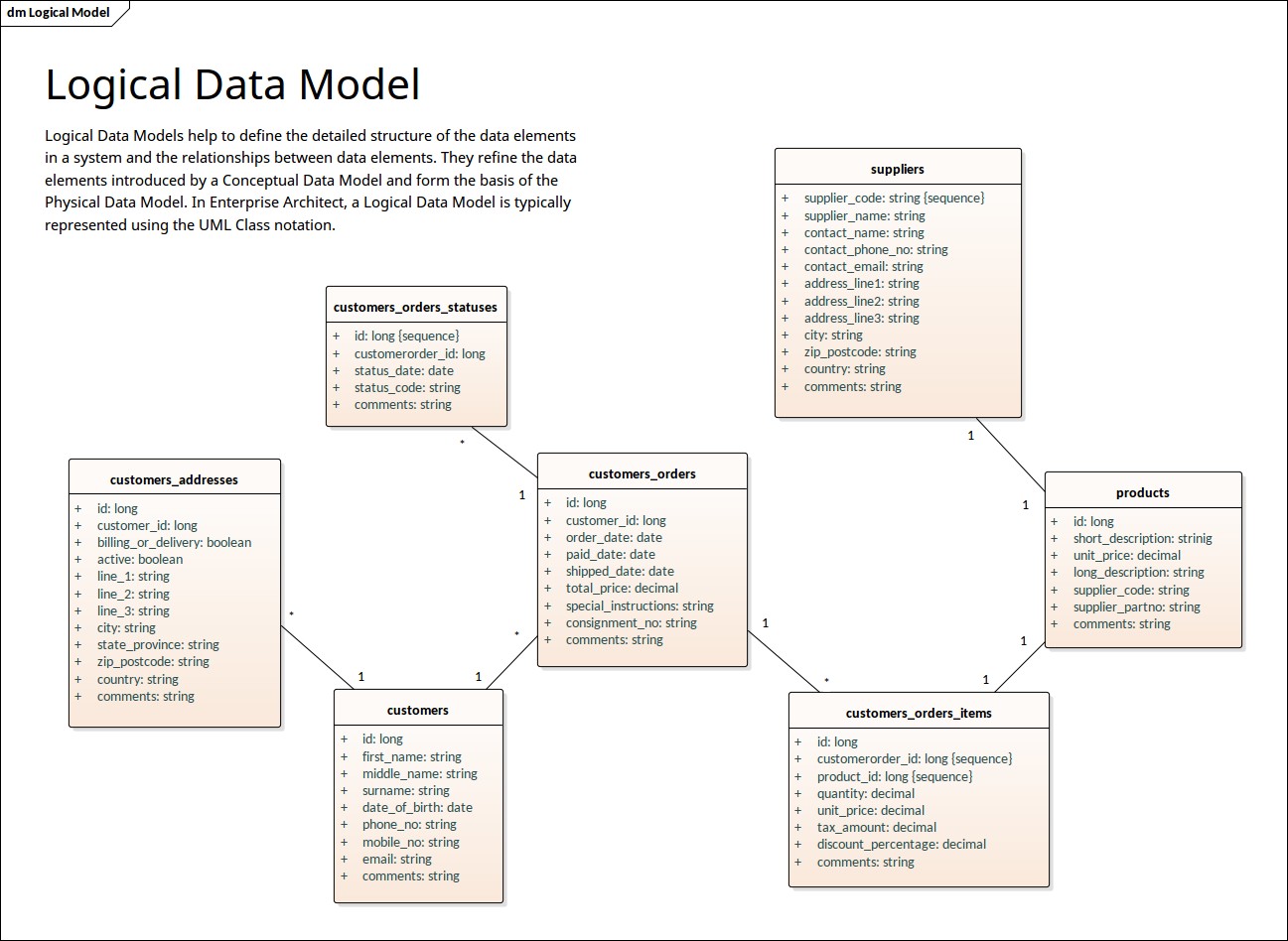


Image24: Logical Data Model Diagram

Reference: https://sparxsystems.com/resources/gallery/diagrams/images/logical-data-model-uml-notation.png

# Entity Sets

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.

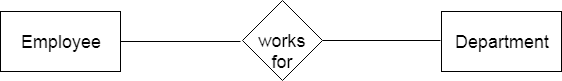


Image25: ER Model for Entity

Reference: https://static.javatpoint.com/dbms/images/dbms-er-model-concept2.png

## Definition of Strong Entity

The **Strong Entity** is the one whose existence does not depend on the existence of any other entity in a schema. It is denoted by a **single rectangle**. A strong entity always has the **primary key** in the set of attributes that describes the strong entity. It indicates that each entity in a strong entity set can be uniquely identified.

Set of similar types of strong entities together forms the **Strong Entity Set**. A strong entity holds the relationship with the weak entity via an **Identifying Relationship**, which is denoted by double diamond in the ER diagram. On the other hand, the relationship between two strong entities is denoted by a single diamond and it is simply called a **relationship**.

Let us understand this concept with the help of an example; a customer borrows a loan. Here we have two entities first a customer entity, and second a loan entity.

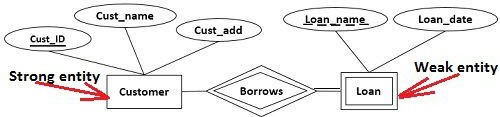


Image26: Strong & Weak Entity Diagram

Reference: https://techdifferences.com/wp-content/uploads/2016/12/Strong-entity-and-weak-entity.jpg

Observing the ER-diagram above, for each loan, there should be at least one borrower otherwise that loan would not be listed in the Loan entity set. But even if a customer does not borrow any loan it would be listed in the Customer entity set. So we can conclude that a customer entity does not depend on a loan entity.



Image27: Customer EntitySet Diagram

Reference: https://techdifferences.com/wp-content/uploads/2016/12/Strong\_Entity-set.jpg

The second thing you can observe is that the Customer entity has a primary key Cust\_ID which uniquely identifies each entity in Customer Entity set. This makes the Customer entity a strong entity on which a loan entity depends.

## Definition of Weak Entity

A **Weak entity** is the one that depends on its owner entity i.e. a strong entity for its existence. A weak entity is denoted by the **double rectangle**. Weak entities do **not** have the **primary key** instead it has a **partial key** that uniquely discriminates against the weak entities. The **primary key of a weak entity** is a composite key formed from the **primary key of the strong entity** and **partial key of the weak entity**.

The collection of similar weak entities is called **Weak Entity Set**. The relationship between a weak entity and a strong entity is always denoted with an **Identifying Relationship** i.e. **double diamond**.

For further illustration let us discuss the above example, this time from a weak entity’s point of view. We have Loan as our weak entity, and as I said above for each loan there must be at least one borrower. You can observe in the loan entity set, no customer has borrowed a car loan and hence, it has totally vanished from the loan entity set. For the presence of a car loan in the loan entity set, it must have been borrowed by a customer. In this way, the weak Loan entity is dependent on the strong Customer entity.

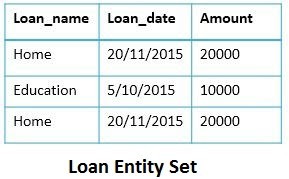


Image28: Loan EntitySet Diagram

Reference: https://techdifferences.com/wp-content/uploads/2016/12/Weak\_Entity\_set.jpg

The second thing, we know is a weak entity does not have a primary key. So here Loan\_name, the partial key of the weak entity and Cust\_ID primary key of the customer entity makes the primary key of the loan entity.

In the Loan entity set, we have two exactly same entities i.e. a **Home loan on date 20/11/2015 with amount 20000.** Now how to identify who had borrowed them this can be done with the help of the primary key of the weak entity (Loan\_name + Cust\_ID). So, it will be determined that one home loan is borrowed by Customer 101 Jhon and other by Customer 103 Ruby. This is how the composite primary key of weak entities identifies each entity in the weak entity set.

**Difference between Strong and Weak Entity**

|  |  |  |
| --- | --- | --- |
| **S.NO** | **STRONG ENTITY** | **WEAK ENTITY** |
| 1. | Strong entity always has a primary key. | While a weak entity has a partial discriminator key. |

|  |  |  |
| --- | --- | --- |
| 2. | Strong entity is not dependent on any other entity. | Weak entities depend on strong entities. |
| 3. | Strong entity is represented by a single rectangle. | Weak entity is represented by a double rectangle. |
| 4. | Two strong entity’s relationships are represented by a single diamond. | While the relation between one strong and one weak entity is represented by a double diamond. |
| 5. | Strong entities have either total participation or not. | While a weak entity always has total participation. |

# Relationship Sets

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



Image29: Relationship

Reference: https://static.javatpoint.com/dbms/images/dbms-er-model-concept9.png

Types of relationship are as follows:

## 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.



Image30: One-to-One Relationship

Reference: https://static.javatpoint.com/dbms/images/dbms-er-model-concept10.png

## 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,** scientists can invent many inventions, but the invention is done by only specific scientist.

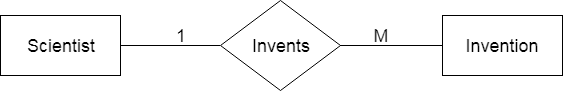


Image31: One-to-Many Relationship

Reference: https://static.javatpoint.com/dbms/images/dbms-er-model-concept11.png

## 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.

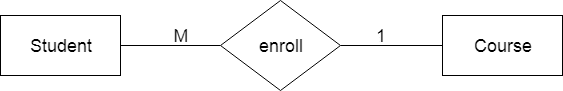


Image32: Many-to-one Relationship

Reference: https://static.javatpoint.com/dbms/images/dbms-er-model-concept12.png

## 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,** employees can be assigned to many projects and projects can have many employees.

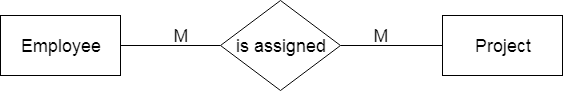


Image33: Many-to-Many Relationship

Reference: https://static.javatpoint.com/dbms/images/dbms-er-model-concept13.png

# Concept of Attributes

## Introduction

Attributes are the descriptive properties which are owned by each entity of an Entity Set.

There exists a specific domain or set of values for each attribute from where the attribute can take its values.

It is a single-valued property of either an entity-type or a relationship-type. For example, a lecture might have attributes: time, date, duration, place, etc. An attribute is represented by an Ellipse

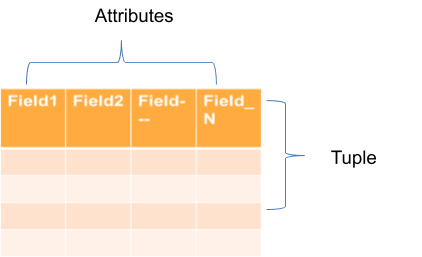
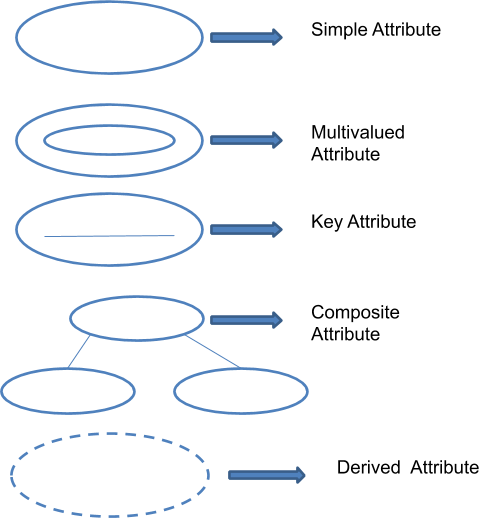


Image 1 – Attributes identification

**Symbols used to represent attributes**



## Types of Attributes

* Simple Attributes
* Composite Attributes
* Single Valued Attributes
* Key Attributes
* Derived Attributes
* Multivalued Attributes

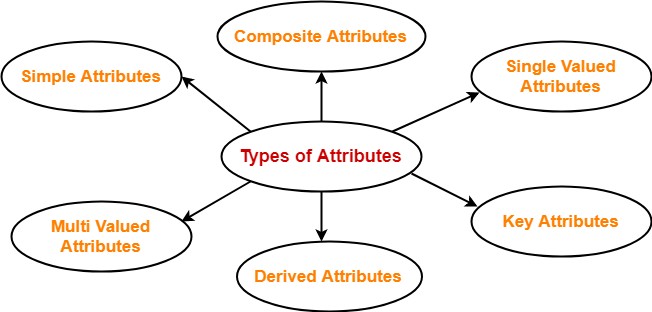


Image 2 – Type of Attributes

Reference - https://[www.gatevidyalay.com/wp-content/uploads/2018/06/Attributes-in-DBMS-Types.png](http://www.gatevidyalay.com/wp-content/uploads/2018/06/Attributes-in-DBMS-Types.png)

### Simple Attributes

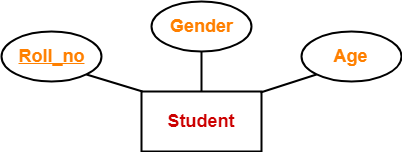
A simple attribute is identify entity from an entity set. Simple attribute represent oval symbol with its value.

Image 3 – Simple Attribute

Reference - https://[www.gatevidyalay.com/wp-content/uploads/2018/06/Key-Attributes-Example.png](http://www.gatevidyalay.com/wp-content/uploads/2018/06/Key-Attributes-Example.png)

Here, all the attributes are simple attributes as they cannot be divided further.

### Single Valued Attributes

Single valued attributes are those attributes which can take only one value for a given entity from an entity set.

### Key Attributes

A key attribute is uniquely identify entity from an entity set.

Key attribute represent oval symbol same as like other with underline.

### Derived Attributes

Derived attributes are those attributes which can be derived from other attribute(s).

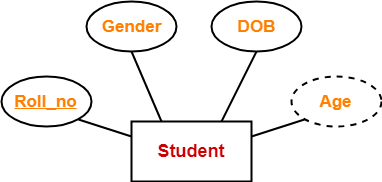


Image 4 – Key Attribute

Reference - https://[www.gatevidyalay.com/wp-content/uploads/2018/06/Derived-Attributes-Example.png](http://www.gatevidyalay.com/wp-content/uploads/2018/06/Derived-Attributes-Example.png)

Here, the attribute “Age” is a derived attribute as it can be derived from the attribute “DOB”.

### Multivalued Attributes

Multi valued attributes are those attributes which can take more than one value for a given entity from an entity set.

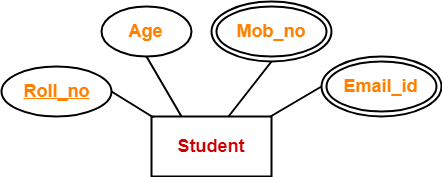


Image 5 – Multivalued Attribute

Reference - https://[www.gatevidyalay.com/wp-content/uploads/2018/06/Multi-Valued-Attributes-Example.png](http://www.gatevidyalay.com/wp-content/uploads/2018/06/Multi-Valued-Attributes-Example.png)

Here, the attributes “Mob\_no” and “Email\_id” are multi valued attributes as they can take more than one values for a given entity.

### Example 1 – Student Book Relation

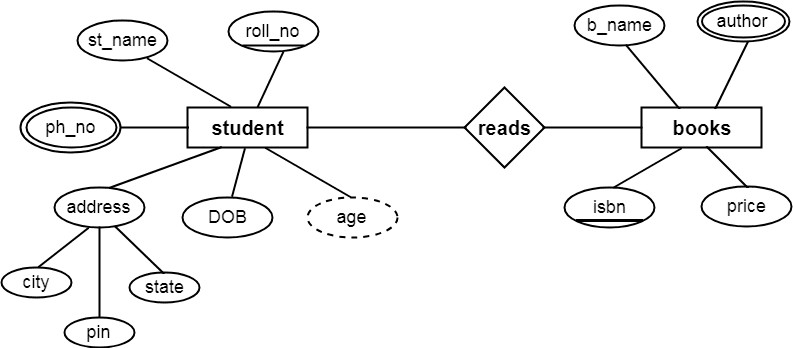


Image 6 – Student Book Relation

Reference - https://[www.csetutor.com/wp-content/uploads/2018/09/ER-Diagram-in-DBMS-Example.png](http://www.csetutor.com/wp-content/uploads/2018/09/ER-Diagram-in-DBMS-Example.png)

### Example 2 – Product Order System

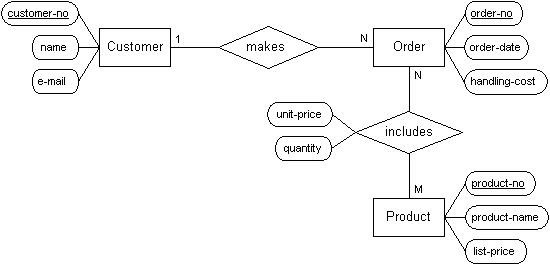


Image 7 – Product Order System

Reference - https://codeandwork.github.io/courses/cs/media/erdiagram2.jpg

### Example 3 –Employee Management

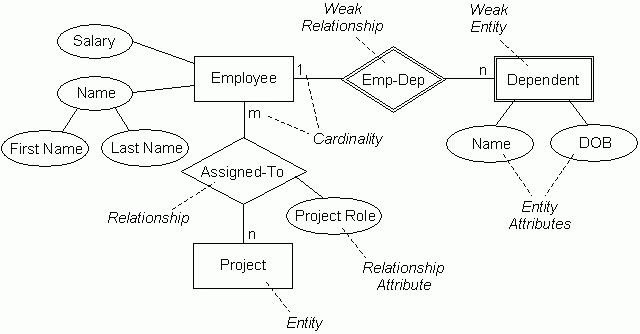


Image 8 – Employee Management

Reference - https://codeandwork.github.io/courses/cs/media/erd-employee.jpg

# Concept of Relationship

## Introduction

An entity relationship diagram (ERD) shows the relationships of entity sets stored in a database. These entities can have attributes that define its properties. By defining the entities, their attributes, and showing the relationships between them, an ER diagram illustrates the logical structure of databases

Relationship is nothing but an association among two or more entities.

Entities take part in relationships. We can often identify relationships with verbs or verb phrases.

### Symbol



**Example**

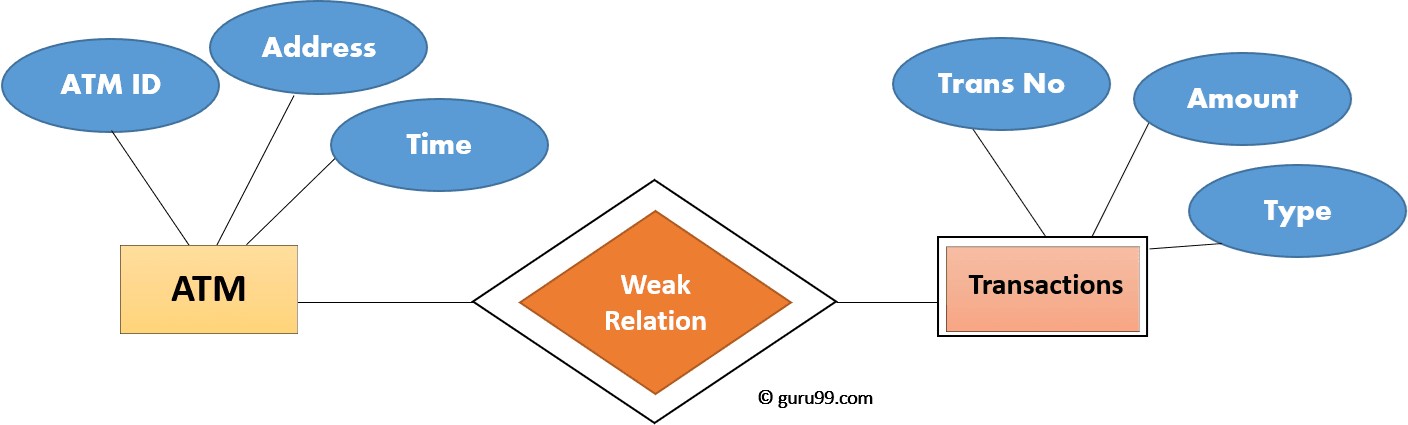


Image 9 – Transaction

## Types of Keys

Reference - https://[www.guru99.com/images/1/100518\_0621\_ERDiagramTu5.png](http://www.guru99.com/images/1/100518_0621_ERDiagramTu5.png)

### Keys

Keys play an important role in the relational database.

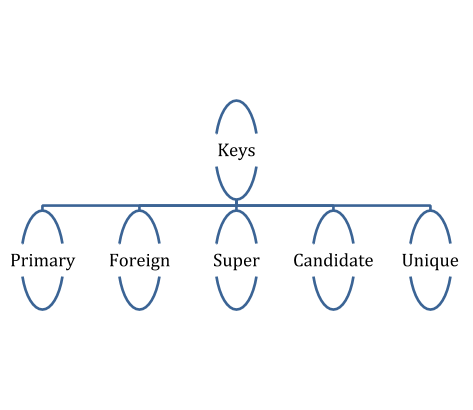
It is used to uniquely identify any record or row of data from the table. It is also used to establish and identify relationships between tables.

Image 10 – Transaction

### Primary Key

PRIMARY KEY It is the first key which is used to identify one and only one instance of an entity uniquely.

Rules Defining Primary Key –

* Two rows can't have the same primary key value
* It must for every row to have a primary key value.
* The primary key field cannot be null.
* The value in a primary key column can never be modified or updated if any foreign key refers to that primary key.

There can be more than one candidate key in relation out of which one can be chosen as the primary key. For Example, STUD\_NO, as well as STUD\_PHONE both, are candidate keys for relation STUDENT but STUD\_NO can be chosen as the primary key (only one out of many candidate keys).

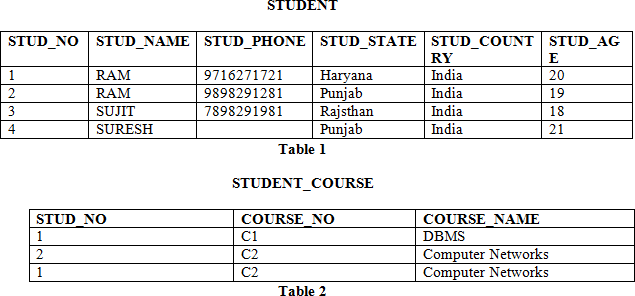


Image 11 – Student table

Reference - https://media.geeksforgeeks.org/wp-content/uploads/image7.png

### Foreign Key

FOREIGN KEY is a column that creates a relationship between two tables.

The purpose of Foreign keys is to maintain data integrity and allow navigation between two different instances of an entity.

Rules defining foreign key –

* Foreign key columns must use their referenced column's type.
* Each column cannot belong to more than 1 Foreign Key constraint.
* Cannot be a computed column.
* Foreign key columns must be indexed.

Based of Image 9 f an attribute can only take the values which are present as values of some other attribute, it will be a foreign key to the attribute to which it refers. The relation which is being referenced is called referenced relation and the corresponding attribute is called referenced attribute and the relation which refers to the referenced relation is called referencing relation and the corresponding attribute is called referencing attribute. The referenced attribute of the referenced relation should be the primary key for it. For Example, STUD\_NO in STUDENT\_COURSE is a foreign key to STUD\_NO in STUDENT relation.

It may be worth noting that unlike, Primary Key of any given relation, Foreign Key can be NULL as well as may contain duplicate tuples i.e. it need not follow uniqueness constraint.

For Example, STUD\_NO in STUDENT\_COURSE relation is not unique. It has been repeated for the first and third tuple. However, the STUD\_NO in STUDENT relation is a primary key and it needs to be always unique and it cannot be null.

### 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.

Rules defining candidate key –

* It must contain unique values
* Candidate key may have multiple attributes
* Must not contain null values
* It should contain minimum fields to ensure uniqueness
* Uniquely identify each record in a table

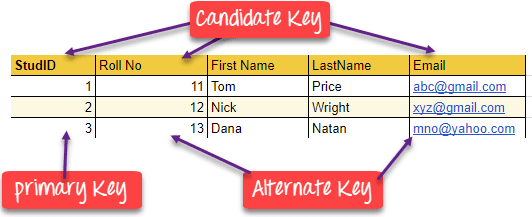


Image 12 – Candidate Key

Reference - https://[www.guru99.com/images/1/100518\_0517\_DBMSKeysPri1.png](http://www.guru99.com/images/1/100518_0517_DBMSKeysPri1.png)

### Super Key

A superkey is a group of single or multiple keys which identifies rows in a table.

A Super key may have additional attributes that are not needed for unique identification. In the above-given example, EmpNo and Emp\_Name are superkeys.

The set of attributes which can uniquely identify a tuple is known as Super Key. For Example, STUD\_NO, (STUD\_NO, STUD\_NAME) etc.

Adding zero or more attributes to candidate key generates super key. A candidate key is a super key but vice versa is not true.

### Compound Key

COMPOUND KEY has two or more attributes that allow you to uniquely recognize a specific record.

It is possible that each column may not be unique by itself within the database Order ID and Product ID could be used as it uniquely identified each record.

In database design, a composite key is a candidate key that consists of two or more attributes (table columns) that together uniquely identify an entity occurrence (table row). A compound key is a composite key for which each attribute that makes up the key is a simple (foreign) key in its own right.

### Alternate Key

ALTERNATE KEYS is a column or group of columns in a table that uniquely identify every row in that table.

A table can have multiple choices for a primary key but only one can be set as the primary key. All the keys which are not primary key are called an Alternate Key.

The candidate key other than the primary key is called an alternate key. For Example, STUD\_NO, as well as STUD\_PHONE both, are candidate keys for relation STUDENT but STUD\_PHONE will be alternate key (only one out of many candidate keys).

### Unique Key

A unique key is a set of one or more than one fields/columns of a table that uniquely identify a record in a database table.

In database relational modeling and implementation, a unique key (also known as a candidate key or just a key) is a set of attributes (columns) within a relational database table (also called a relation), such that:

the table does not have two distinct rows or records with the same values for these columns;

this set of columns is minimal; i.e., removing any column from the key would result in duplicate values in the resulting subset.

When a column or set of columns is defined as unique to the database management system, the system verifies that each set of values is unique before assigning the constraint. After the columns are defined as unique, an error will occur if an insertion is attempted with values that already exist. Some systems will not allow key values to be updated, all systems will not allow duplicates. This ensures that uniqueness is maintained in both the primary table and any relations that are later bound to it.

# Mapping Constraints

## Introduction

A mapping constraint is a data constraint that expresses the number of entities to which another entity can be related via a relationship set.

Mapping constraints defines how many entities can be related to another entity to a relationship.

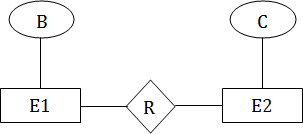


Image 13 – Mapping Constraint

Reference - https://static.javatpoint.com/dbms/images/dbms-mapping-constraints4.png

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:

### Types of Mapping Constraints

* One to one (1:1)
* One to many (1:M)
* Many to one (M:1)
* Many to many (M:M)

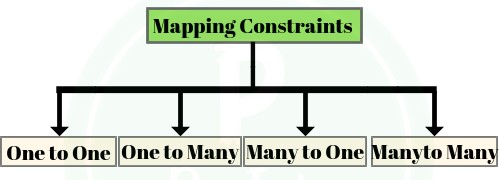


Image 14 – Mapping Constraint

Reference - https://prepinsta.com/wp-content/uploads/2019/07/30-300x300.png

### 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.

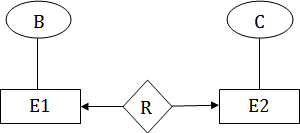


Image 15 – One to One

Reference - https://static.javatpoint.com/dbms/images/dbms-mapping-constraints.png

### 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.

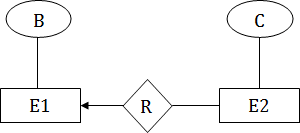


Image 16 – One to Many

Many-to-One

Reference - https://static.javatpoint.com/dbms/images/dbms-mapping-constraints2.png

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.

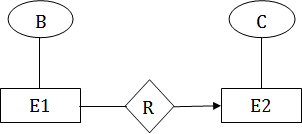


Image 17 – One to Many

Reference - https://static.javatpoint.com/dbms/images/dbms-mapping-constraints2.png

### 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.

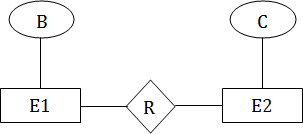


Image 18 – One to Many

Reference - https://static.javatpoint.com/dbms/images/dbms-mapping-constraints4.png

### Example of Employee Management System

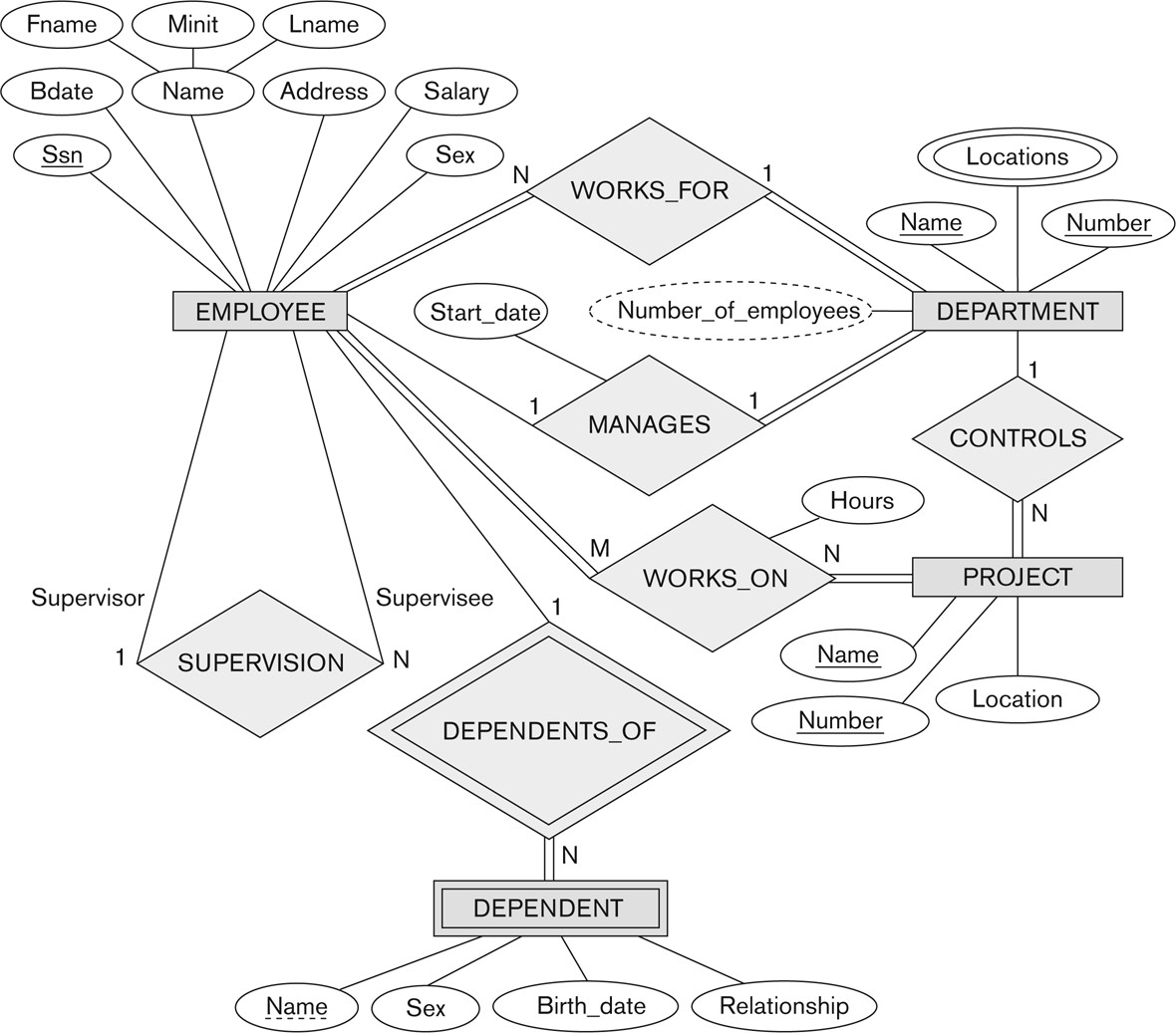


Image 19 – Employee Management

Reference - <http://pld.cs.luc.edu/database/images/fig7.2.png>

### Example of Movie Ticket Management

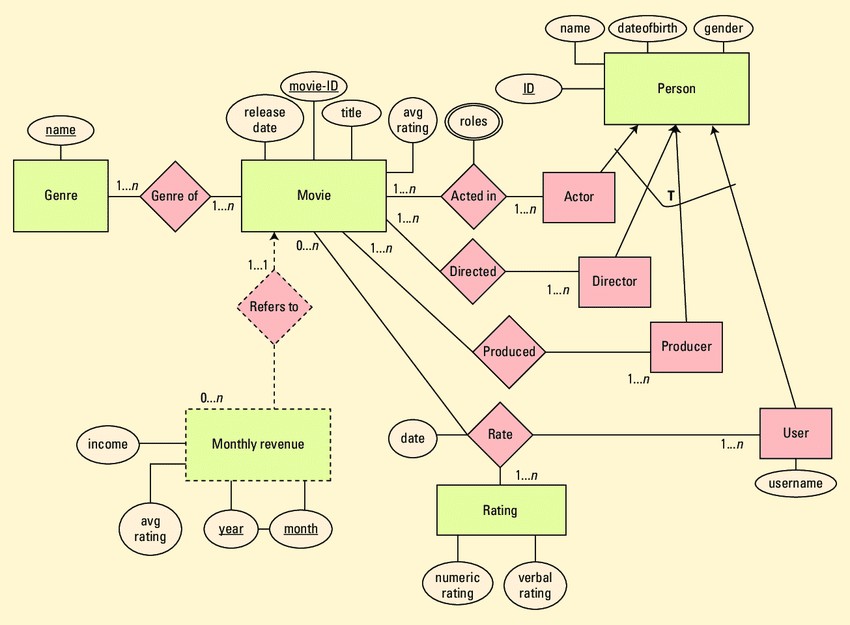


Image 20 – Movie Ticket Management Reference -

https://www.researchgate.net/profile/Peretz\_Shoval/publication/321352935/figure/fig1/AS:571569942142976@1513284301301/The-entity-relati onship-diagram-for-the-movie-recommendation-system-Subtypes-are-not.png

# Entity Relationship Diagram

## Introduction

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.

### Components of ER Diagram

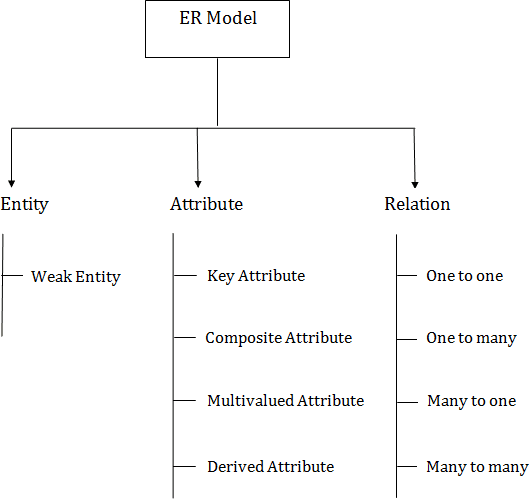


Image 21 – Components of ERD

Reference - https://static.javatpoint.com/dbms/images/dbms-er-model-concept-diagram.png

### 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.

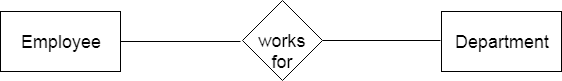


Image 22 – Entity

### a. Weak Entity

Reference - https://static.javatpoint.com/dbms/images/dbms-er-model-concept2.png

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.



Image 23 – Week Entity

Reference - https://static.javatpoint.com/dbms/images/dbms-er-model-concept3.png

### Examples of entities:

Person: Employee, Student, Patient Place: Store, Building

Object: Machine, product, and Car

Event: Sale, Registration, Renewal Concept: Account, Course

Difference between Strong and Week Entity

|  |  |
| --- | --- |
| **Strong Entity Set** | **Weak Entity Set** |
| Strong entity set always has a primary key. | It does not have enough attributes to build a primary key. |
| It is represented by a rectangle symbol. | It is represented by a double rectangle symbol. |
| It contains a Primary key represented by the underline symbol. | It contains a Partial Key which is represented by a dashed underline symbol. |
| The member of a strong entity set is called as dominant entity set. | The member of a weak entity set called as a subordinate entity set. |
| Primary Key is one of its attributes which helps to identify its member. | In a weak entity set, it is a combination of primary key and partial key of the strong entity set. |
| In the ER diagram the relationship between two strong entity set shown by using a diamond symbol. | The relationship between one strong and a weak entity set shown by using the double diamond symbol. |
| The connecting line of the strong entity set with the relationship is single. | The line connecting the weak entity set for identifying relationship is double. |

### 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.

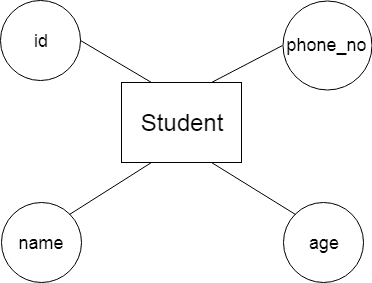


Image 23 – Attribute

### Key Attribute

Reference - https://static.javatpoint.com/dbms/images/dbms-er-model-concept4.png

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.

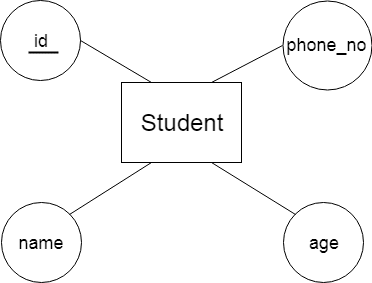


Image 24 – Key Attribute

Reference - https://static.javatpoint.com/dbms/images/dbms-er-model-concept5.png

### 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.

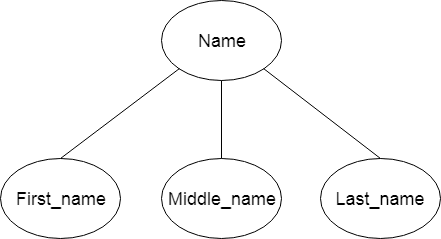


Image 25 – Composite Attribute

Reference - https://static.javatpoint.com/dbms/images/dbms-er-model-concept6.png

### 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.

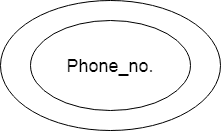


Image 26 – Multivalued Attribute

Reference - https://static.javatpoint.com/dbms/images/dbms-er-model-concept7.png

### 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.

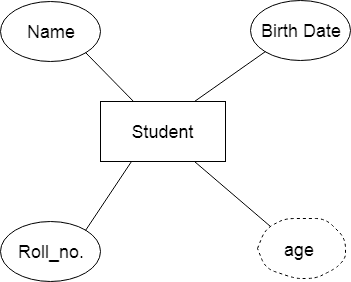


Image 27 – Derived Attribute

Reference - https://static.javatpoint.com/dbms/images/dbms-er-model-concept8.png

**3. Relationship**

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

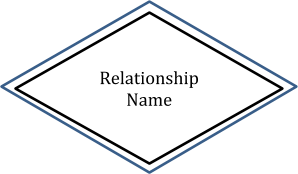


Image 28 – Relationship Attribute

Reference - https://static.javatpoint.com/dbms/images/dbms-er-model-concept9.png

### Week Relationship

Transaction ID is week relation between bank and customer account money withdrawal



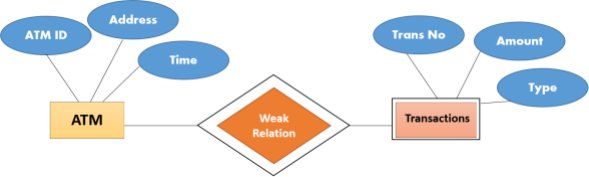


Image 29 – Relationship Attribute

Reference - https://[www.guru99.com/images/1/100518\_0621\_ERDiagramTu5.png](http://www.guru99.com/images/1/100518_0621_ERDiagramTu5.png)

### Example : Library Book Management

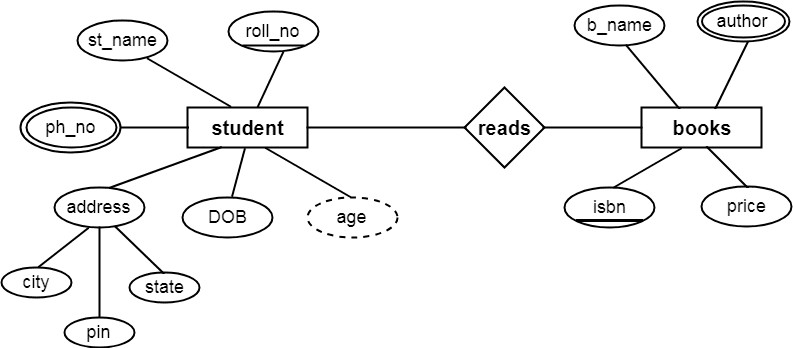


Image 30 – Library Book Management

Reference - https://[www.csetutor.com/wp-content/uploads/2018/09/ER-Diagram-in-DBMS-Example.png](http://www.csetutor.com/wp-content/uploads/2018/09/ER-Diagram-in-DBMS-Example.png)

Example: Employee Management System

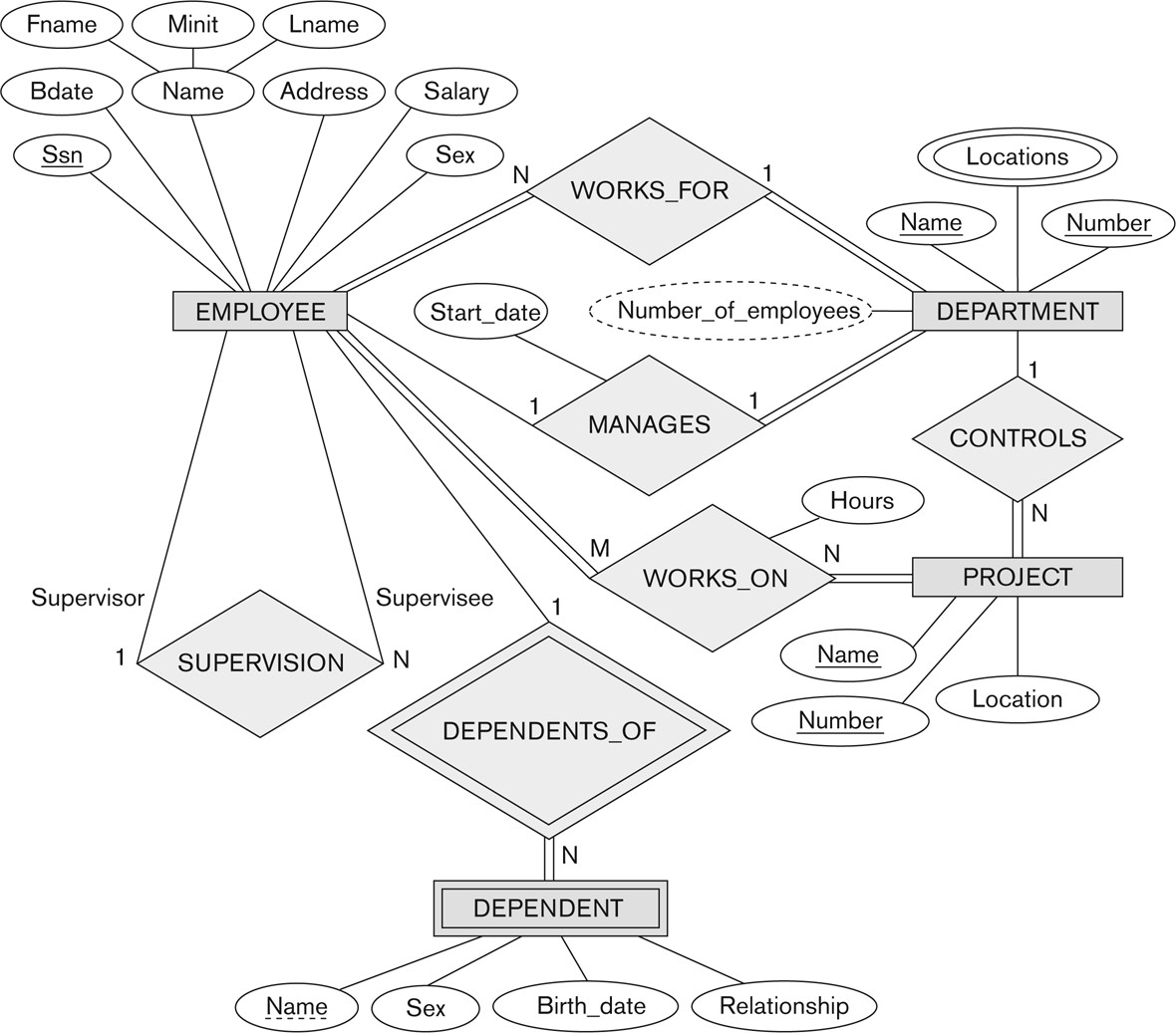


Image 31 – Employee Management

Reference - <http://pld.cs.luc.edu/database/images/fig7.2.png>

### Example: Banking Organization

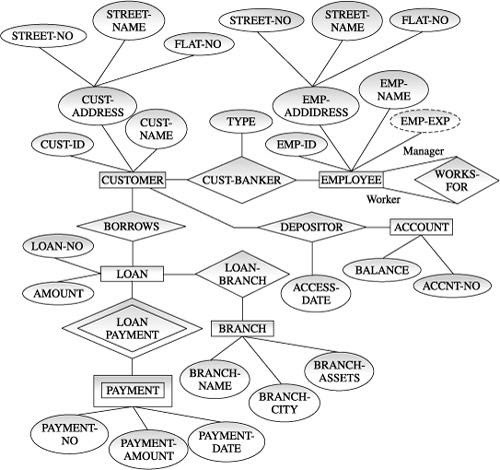


Image 32 – Banking Management

Reference - https://[www.oreilly.com/library/view/database-systems-concepts/9788177585674/9788177585674\_ch06lev1sec5\_image01.jpeg](http://www.oreilly.com/library/view/database-systems-concepts/9788177585674/9788177585674_ch06lev1sec5_image01.jpeg)

### Example: Hospital Management

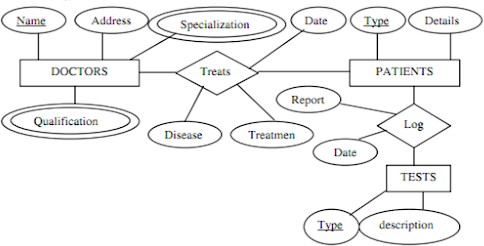


Image 33 – Banking Management Reference -

https://lh3.googleusercontent.com/proxy/ApY5GXZPpliyg9sRl18JwNH8N4Dyd3wAZQNwwtlIEwk4Q89tIlJq\_P4RwtlKUvVEQw0zaS4VydBz1 soUEARVNFpHv9YLu5y17jQ9EdTrGGD1zgWnJYvx09MMc9C7Y70

# Relational Model

## Introduction

Relational Model was proposed by E.F. Codd to model data in the form of relations or tables. After designing the conceptual model of Database using ER diagram, we need to convert the conceptual model in the relational model which can be implemented using any RDMBS languages like Oracle SQL, MySQL etc. So we will see what Relational Model is.

### What is Relational Model?

Relational Model represents how data is stored in Relational Databases. A relational database stores data in the form of relations (tables). Consider a relation STUDENT with attributes ROLL\_NO, NAME, ADDRESS, PHONE and AGE shown in Table 1.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **STUDENT**  ROLL\_NO | NAME | ADDRESS | PHONE | AGE |
| 1 | RAM | DELHI | 9455123451 | 18 |
| 2 | RAMESH | GURGAON | 9652431543 | 18 |
| 3 | SUJIT | ROHTAK | 9156253131 | 20 |
| 4 | SURESH | DELHI | **9156253132** | 18 |

### IMPORTANT TERMINOLOGIES

**Attribute:** Attributes are the properties that define a relation. e.g.; ROLL\_NO, NAME

**Relation Schema:** A relation schema represents name of the relation with its attributes. e.g.; STUDENT (ROLL\_NO, NAME, ADDRESS, PHONE and AGE) is relation schema for STUDENT. If a schema has more than 1 relation, it is called Relational Schema.

**Tuple:** Each row in the relation is known as tuple. The above relation contains 4 tuples, one of which is shown as:

1 RAM DELHI 9455123451 18

**Relation Instance:** The set of tuples of a relation at a particular instance of time is called as relation instance. Table 1 shows the relation instance of STUDENT at a particular time. It can change whenever there is insertion, deletion or updation in the database.

**Degree:** The number of attributes in the relation is known as degree of the relation. The STUDENT relation defined above has degree 5.

**Cardinality:** The number of tuples in a relation is known as cardinality. The STUDENT relation defined above has cardinality 4.

**Column:** Column represents the set of values for a particular attribute. The column ROLL\_NO is extracted from relation STUDENT.

ROLL\_NO 1

2

3

4

**NULL Values:** The value which is not known or unavailable is called NULL value. It is represented by blank space. e.g.; PHONE of STUDENT having ROLL\_NO 4 is NULL.

### Constraints in Relational Model

While designing Relational Model, we define some conditions which must hold for data present in database are called Constraints. These constraints are checked before performing any operation (insertion, deletion and updation) in database. If there is a violation in any of constrains, operation will fail.

**Domain Constraints:** These are attribute level constraints. An attribute can only take values which lie inside the domain range. e.g,; If a constrains AGE>0 is applied on STUDENT relation, inserting negative value of AGE will result in failure.

**Key Integrity:** Every relation in the database should have atleast one set of attributes which defines a tuple uniquely. Those set of attributes is called key. e.g.; ROLL\_NO in STUDENT is a key. No two students can have same roll number. So a key has two properties:

It should be unique for all tuples. It can’t have NULL values.

**Referential Integrity:** When one attribute of a relation can only take values from other attribute of same relation or any other relation, it is called referential integrity. Let us suppose we have 2 relations

**STUDENT**

**ROLL\_NO NAME ADDRESS PHONE AGE BRANCH\_CODE**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | RAM | DELHI | 9455123451 | 18 | CS |
| 2 | RAMESH | GURGAON | 9652431543 | 18 | CS |
| 3 | SUJIT | ROHTAK | 9156253131 | 20 | ECE |
| 4 | SURESH | DELHI |  | 18 | IT |
| **BRANCH** |  |  |  |  |  |

**BRANCH\_CODE BRANCH\_NAME**

CS COMPUTER SCIENCE

IT INFORMATION TECHNOLOGY

ECE ELECTRONICS AND COMMUNICATION ENGINEERING

CV CIVIL ENGINEERING

BRANCH\_CODE of STUDENT can only take the values which are present in BRANCH\_CODE of BRANCH which is called referential integrity constraint. The relation which is referencing to other relation is called REFERENCING RELATION (STUDENT in this

case) and the relation to which other relations refer is called REFERENCED RELATION (BRANCH in this case).

An anomaly is an irregularity, or something which deviates from the expected or normal state. When designing databases, we identify three types of anomalies: Insert, Update and Delete.

### Insertion Anomaly in Referencing Relation:

We can’t insert a row in REFERENCING RELATION if referencing attribute’s value is not present in referenced attribute value. e.g.; Insertion of a student with BRANCH\_CODE ‘ME’ in STUDENT relation will result in error because ‘ME’ is not present in BRANCH\_CODE of BRANCH.

### Deletion/ Updation Anomaly in Referenced Relation:

We can’t delete or update a row from REFERENCED RELATION if value of REFRENCED ATTRIBUTE is used in value of REFERENCING ATTRIBUTE. e.g; if we try to delete tuple from BRANCH having BRANCH\_CODE ‘CS’, it will result in error because ‘CS’ is referenced by BRANCH\_CODE of STUDENT, but if we try to delete the row from BRANCH with BRANCH\_CODE CV, it will be deleted as the value is not been used by referencing relation. It can be handled by following method:

**ON DELETE CASCADE:** It will delete the tuples from REFERENCING RELATION if value used by REFERENCING ATTRIBUTE is deleted from REFERENCED RELATION. e.g;, if we delete a row from BRANCH with BRANCH\_CODE ‘CS’, the rows in STUDENT relation with BRANCH\_CODE CS (ROLL\_NO 1 and 2 in this case) will be deleted.

**ON UPDATE CASCADE:** It will update the REFERENCING ATTRIBUTE in REFERENCING RELATION if attribute value used by REFERENCING ATTRIBUTE is updated in REFERENCED RELATION. e.g;, if we update a row from BRANCH with BRANCH\_CODE ‘CS’ to ‘CSE’, the rows in STUDENT relation with BRANCH\_CODE CS (ROLL\_NO 1 and 2 in this case) will be updated with BRANCH\_CODE ‘CSE’.

### SUPER KEYS:

Any set of attributes that allows us to identify unique rows (tuples) in a given relation are known as super keys. Out of these super keys we can always choose a proper subset among these which

can be used as a primary key. Such keys are known as Candidate keys. If there is a combination of two or more attributes which is being used as the primary key then we call it as a Composite key.

**Attribute -** Each column in a Table. Attributes are the properties which define a relation. e.g., Student\_Rollno, NAME,etc.

**Tables –** In the Relational model the, relations are saved in the table format. It is stored along with its entities. A table has two properties rows and columns. Rows represent records and columns represent attributes.

**Tuple –** It is nothing but a single row of a table, which contains a single record.

**Relation Schema:** A relation schema represents the name of the relation with its attributes. **Degree -** The total number of attributes which in the relation is called the degree of the relation. **Cardinality -** Total number of rows present in the Table.

**Column -** The column represents the set of values for a specific attribute.

**Relation instance –** Relation instance is a finite set of tuples in the RDBMS system. Relation instances never have duplicate tuples.

**Relation Key -** Ev**e**ry row has one, two or multiple attributes, which is called relation key.

**Attribute Domain** – Every attribute has some pre-defined value and scope which is known as attribute domain.

### Properties of Relation

* 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

Example 1

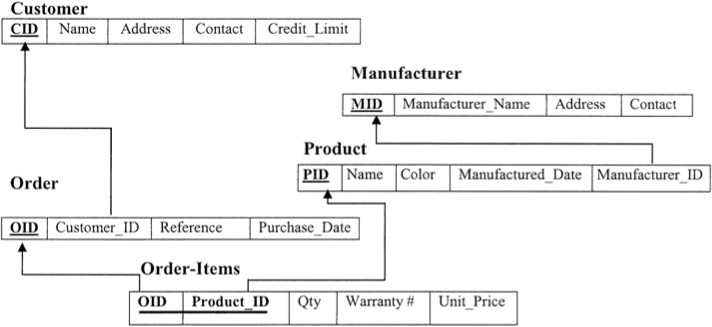


Image 34 – Production Management

Reference - https://d1whtlypfis84e.cloudfront.net/guides/wp-content/uploads/2019/01/01111318/Relational-databse.png

Example 2

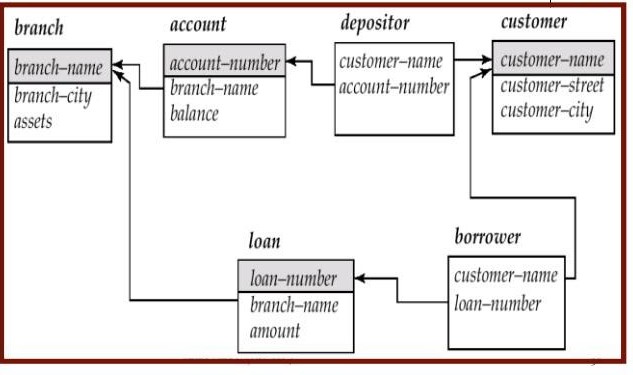


Image 35 – Production Management Reference -

https://[www.google.com/url?sa=i&url=https%3A%2F%2Fwww.slideshare.net%2FProsantaGhosh%2Fdbms-ii-mcach4relational-model2013&psi](http://www.google.com/url?sa=i&url=https%3A%2F%2Fwww.slideshare.net%2FProsantaGhosh%2Fdbms-ii-mcach4relational-model2013&psi) g=AOvVaw1UFOBY4OLPKvIkomoOiJh9&ust=1587013292216000&source=images&cd=vfe&ved=0CAIQjRxqFwoTCJC5\_K\_T6egCFQAAA AAdAAAAABAD

# Network Model

## Introduction

Charles Bachman was the original inventor of the network model. In 1969, the Conference on Data Systems Languages (CODASYL) Consortium developed the network model into a standard specification. A second publication was introduced in 1971, which later turned into the basis for virtually all implementations.

### The benefits of the network model include:

Simple Concept: Similar to the hierarchical model, this model is simple and the implementation is effortless.

Ability to Manage More Relationship Types: The network model has the ability to manage one-to-one (1:1) as well as many-to-many (N: N) relationships.

Easy Access to Data: Accessing the data is simpler when compared to the hierarchical model.

Data Integrity: In a network model, there's always a connection between the parent and the child segments because it depends on the parent-child relationship.

Data Independence: Data independence is better in network models as opposed to the hierarchical models.

### The drawbacks of the network model include:

System Complexity: Each and every record has to be maintained with the help of pointers, which makes the database structure more complex.

Functional Flaws: Because a great number of pointers is essential, insertion, updates, and deletion become more complex.

Lack of Structural Independence: A change in structure demands a change in the application as well, which leads to lack of structural independence.

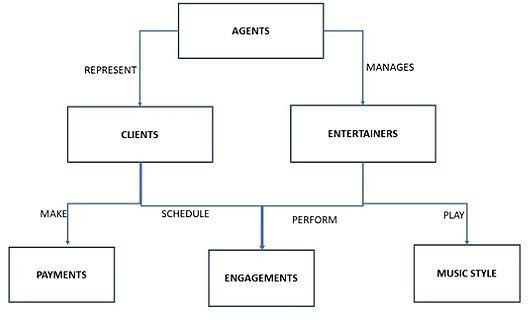


Image 36 – Network Model

Reference - https://[www.tutorialspoint.com/assets/questions/images/120543-1532343127.jpg](http://www.tutorialspoint.com/assets/questions/images/120543-1532343127.jpg)

The network model is the extension of the hierarchical structure because it allows many-to-many relationships to be managed in a tree-like structure that allows multiple parents.

It can represent redundancy in data more efficiently than that in the hierarchical model. There can be more than one path from a previous node to successor node/s.

The operations of the network model are maintained by indexing structure of linked list (circular) where a program maintains a current position and navigates from one record to another by following the relationships in which the record participates.

Records can also be located by supplying key values.

# Hierarchical Model

## Introduction

A hierarchical model represents the data in a tree-like structure in which there is a single parent for each record. To maintain order there is a sort field which keeps sibling nodes into a recorded manner. These types of models are designed basically for the early mainframe database management systems, like the Information Management System (IMS) by IBM.

This model structure allows the one-to-one and a one-to-many relationship between two/ various types of data. This structure is very helpful in describing many relationships in the real world; table of contents, any nested and sorted information.

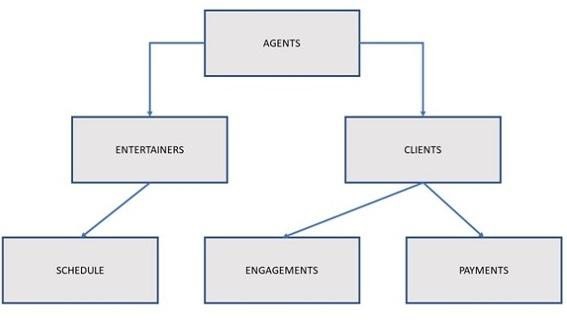


Image 37 – Hierarchical Model

Reference - https://[www.tutorialspoint.com/assets/questions/images/154411-1532346635.jpg](http://www.tutorialspoint.com/assets/questions/images/154411-1532346635.jpg)

### Advantages

A user can retrieve data very quickly due to the presence of explicit links between the table structures.

The referential integrity is built in and automatically enforced due to which a record in a child table must be linked to an existing record in a parent table, along with that if a record deleted in the parent table then that will cause all associated records in the child table to be deleted as well.

### Disadvantages

When a user needs to store a record in a child table that is currently unrelated to any record in a parent table, it gets difficulty in recording and user must record an additional entry in the parent table.

This type of database cannot support complex relationships, and there is also a problem of redundancy, which can result in producing inaccurate information due to the inconsistent recording of data at various sites.

Example

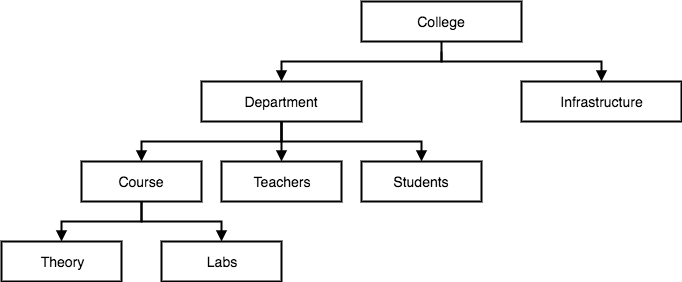


Image 38 – Hierarchical Model

Reference - https://[www.studytonight.com/dbms/images/hierarchical-dbms-model.png](http://www.studytonight.com/dbms/images/hierarchical-dbms-model.png)

# Relational Database Management System

## Introduction

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.

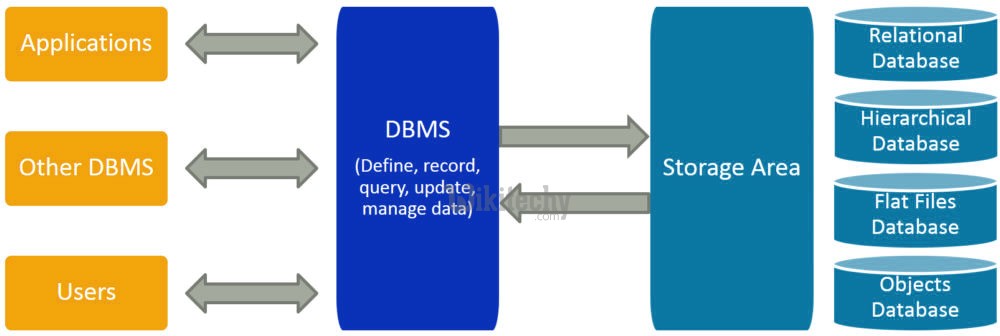


Image 39 – RDBMS Model

Reference - https://d5ngkkf53wl41.cloudfront.net/interview-questions/dbms/what-is-dbms.png

It is the process of making a description of the execution of the database on secondary storage, which describes the base relations, file organizations as well as indexes used to gain efficient access to the data and any associated integrity constraints and security measures.

### Characteristics of physical database design

It typically illustrates data requirements for a single project or application. Sometimes even a part of an application

May be incorporated into other physical data models by means of a repository of shared entities

It typically includes 10-1000 tables; although these numbers are highly variable, depending on the scope of the data model

It has the relationships between tables that address cardinality and null ability (optionality) of the relationships

Designed and developed to be reliant on a specific version of a DBMS, storage location of data or technology

Database columns will have data types with accurate precisions and lengths assigned to them. Columns will have null ability (optional) assigned

Tables and columns will have specific definitions Steps required to implement physical database

The steps of the physical database design methodology are as follows:

### Transform the logical data model for target DBMS

* Design base relations
* Design representation of derived data
* Design general constraints
* Design file organizations and indexes
* Analyze transactions
* Choose file organizations
* Choose indexes
* Estimate disk space requirements
* Design user views
* Design security mechanisms
* Consider the introduction of controlled redundancy
* Monitor and tune the operational system

### Database Engine

A database engine (or storage engine) is the underlying software component that a database management system (DBMS) uses to create, read, update and delete (CRUD) data from a database. Most database management systems include their own application programming interface (API) that allows the user to interact with their underlying engine without going through the user interface of the DBMS.

The term "database engine" is frequently used interchangeably with "database server" or "database management system". A 'database instance' refers to the processes and memory structures of the running database engine.

### Database Schema

The database schema of a database is its structure described in a formal language supported by the database management system (DBMS). The term "schema" refers to the organization of data as a blueprint of how the database is constructed (divided into database tables in the case of relational databases). The formal definition of a database schema is a set of formulas (sentences) called integrity constraints imposed on a database.[citation needed] These integrity constraints ensure compatibility between parts of the schema. All constraints are expressible in the same language. A database can be considered a structure in realization of the database language. The states of a created conceptual schema are transformed into an explicit mapping, the database schema. This describes how real-world entities are modeled in the database.

"A database schema specifies, based on the database administrator's knowledge of possible applications, the facts that can enter the database, or those of interest to the possible end-users."

The notion of a database schema plays the same role as the notion of theory in predicate calculus. A model of this "theory" closely corresponds to a database, which can be seen at any instant of time as a mathematical object. Thus a schema can contain formulas representing integrity constraints specifically for an application and the constraints specifically for a type of database, all expressed in the same database language.

In a relational database, the schema defines the tables, fields, relationships, views, indexes, packages, procedures, functions, queues, triggers, types, sequences, materialized views, synonyms, database links, directories, XML schemas, and other elements.

A database generally stores its schema in a data dictionary. Although a schema is defined in text database language, the term is often used to refer to a graphical depiction of the database structure. In other words, schema is the structure of the database that defines the objects in the database.

### Advantages of relational database management system

The use of an RDBMS can be beneficial to most organizations; the systematic view of raw data helps companies better understand and execute the information while enhancing the

decision-making process. The use of tables to store data also improves the security of information stored in the databases. Users are able to customize access and set barriers to limit the content that is made available. This feature makes the RDBMS particularly useful to companies in which the manager decides what data is provided to employees and customers.

Furthermore, RDBMSes make it easy to add new data to the system or alter existing tables while ensuring consistency with the previously available content.

### Other advantages of the RDBMS include:

Flexibility -- updating data is more efficient since the changes only need to be made in one place.

Maintenance -- database administrators can easily maintain, control and update data in the database. Backups also become easier since automation tools included in the RDBMS automate these tasks.

Data structure -- the table format used in RDBMSes is easy to understand and provides an organized and structural manner through which entries are matched by firing queries.

### Disadvantages of RDBMS

Software is expensive.

Complex software refers to expensive hardware and hence increases overall cost to avail the RDBMS service.

It requires skilled human resources to implement. Certain applications are slow in processing.

It is difficult to recover the lost data.

### Applications of RDBMS

**Sector Use of DBMS**

Banking For customer information, account activities, payments, deposits, loans, etc.

Airlines For reservations and schedule information.

Universities For student information, course registrations, colleges and

grades.

Telecommunication

It helps to keep call records, monthly bills, maintaining

balances, etc.

Finance For storing information about stock, sales, and purchases of financial instruments like stocks and bonds.

Sales Use for storing customer, product & sales information.

Manufacturing

It is used for the management of supply chain and for tracking production of items. Inventories status in warehouses.

HR Management

For information about employees, salaries, payroll,

deduction, generation of paychecks, etc.

### Structure of DBMS

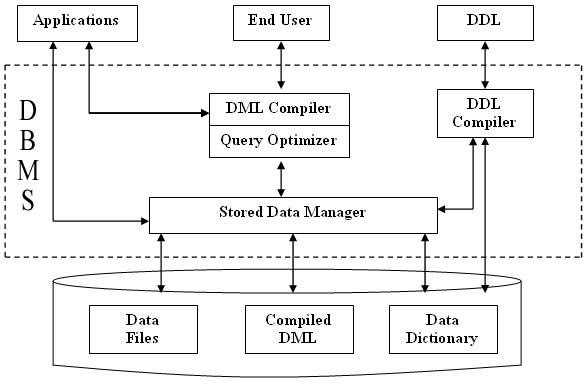


Image 40 – Structure of DBMS

Reference -

<http://1.bp.blogspot.com/-GmlmEbglhWk/Vc12mfuuTYI/AAAAAAAABak/jYO7eiIdprw/s1600/Structure_of_DBMS_DBMSbasics.blogspot.co> m.jpg

At very high level, a database is considered as shown in below diagram. Let us see them in detail below.

**Applications: –** It can be considered as a user friendly web page where the user enters the requests. Here he simply enters the details that he needs and presses buttons to get the data.

**End User: –** They are the real users of the database. They can be developers, designers, administrator or the actual users of the database.

**DDL: –** Data Definition Language (DDL) is a query fired to create database, schema, tables, mappings etc in the database. These are the commands used to create the objects like tables, indexes in the database for the first time. In other words, they create structure of the database.

**DDL Compiler: –** This part of database is responsible for processing the DDL commands. That means these compiler actually breaks down the command into machine understandable codes. It is also responsible for storing the metadata information like table name, space used by it, number of columns in it, mapping information etc.

**DML Compiler: –** When the user inserts, deletes, updates or retrieves the record from the database, he will be sending request which he understands by pressing some buttons. But for the database to work/understand the request, it should be broken down to object code. This is done by this compiler. One can imagine this as when a person is asked some question, how this is broken down into waves to reach the brain!

**Query Optimizer: –** When user fires some request, he is least bothered how it will be fired on the database. He is not all aware of database or its way of performance. But whatever be the request, it should be efficient enough to fetch, insert, update or delete the data from the database. The query optimizer decides the best way to execute the user request which is received from the DML compiler. It is similar to selecting the best nerve to carry the waves to brain!

**Stored Data Manager: –** This is also known as Database Control System. It is one the main central system of the database. It is responsible for various tasks

It converts the requests received from query optimizer to machine understandable form. It makes actual request inside the database. It is like fetching the exact part of the brain to answer.

It helps to maintain consistency and integrity by applying the constraints. That means, it does not allow inserting / updating / deleting any data if it has child entry. Similarly it does not allow entering any duplicate value into database tables.

It controls concurrent access. If there is multiple users accessing the database at the same time, it makes sure, all of them see correct data. It guarantees that there is no data loss or data mismatch happens between the transactions of multiple users.

It helps to backup the database and recover data whenever required. Since it is a huge database and when there is any unexpected exploit of transaction, and reverting the changes are not easy. It maintains the backup of all data, so that it can be recovered.

**Data Files: –** It has the real data stored in it. It can be stored as magnetic tapes, magnetic disks or optical disks.

**Compiled DML: –** Some of the processed DML statements (insert, update, delete) are stored in it so that if there is similar requests, it will be re-used.

**Data Dictionary: –** It contains all the information about the database. As the name suggests, it is the dictionary of all the data items. It contains description of all the tables, view, materialized views, constraints, indexes, triggers etc.

**DBMS VS File System**

|  |  |
| --- | --- |
| **DBMS** | **File System** |
| In DBMS, the user is not required to write the procedures. | In this system, the user has to write the procedures for managing the database. |
| DBMS gives an abstract view of data that hides the details. | File system provides the detail of the data representation and storage of data. |
| DBMS provides a crash recovery mechanism, i.e., DBMS protects the user from the system failure. | 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 lost. |
| DBMS provides a good protection mechanism. | It is very difficult to protect a file under the file system. |
| DBMS contains a wide variety of sophisticated techniques to store and retrieve the data. | File system can't efficiently store and retrieve the data. |

|  |  |
| --- | --- |
| 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 other deleting some information or updating some information. |

**DBMS VS RDBMS**

|  |  |  |
| --- | --- | --- |
| **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 does not support distributed database. | RDBMS supports distributed database. |
| 7) | 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. |
| 8) | Examples of DBMS are file systems, xml etc. | Example of RDBMS are mysql, postgre, sql server, oracle etc. |

1. **Tier Architecture**

* In this architecture, the database is directly available to the user. It means the user can directly sit on the DBMS and uses it.
* Any changes done here will directly be done on the database itself. It doesn't provide a handy tool for end users.
* The 1-Tier architecture is used for development of the local application, where programmers can directly communicate with the database for the quick response.

### Tier Architecture

* + The 2-Tier architecture is same as basic client-server. In the two-tier architecture, applications on the client end can directly communicate with the database at the server side. For this interaction, API's like: **ODBC**, **JDBC** are used.
  + The user interfaces and application programs are run on the client-side.
  + The server side is responsible to provide the functionalities like: query processing and transaction management.
  + To communicate with the DBMS, client-side application establishes a connection with the server side.

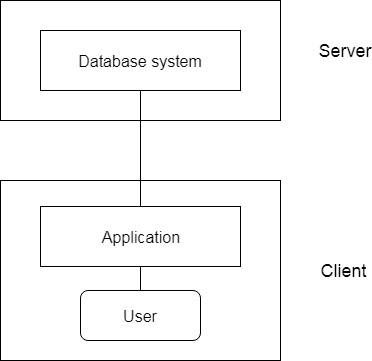


Image 41 – 2 tier Architecture

Reference - https://static.javatpoint.com/dbms/images/dbms-2-tier-architecture.png

1. **Tier Architecture**
   * The 3-Tier architecture contains another layer between the client and server. In this architecture, client can't directly communicate with the server.
   * The application on the client-end interacts with an application server which further communicates with the database system.
   * End user has no idea about the existence of the database beyond the application server. The database also has no idea about any other user beyond the application.
   * The 3-Tier architecture is used in case of large web application.

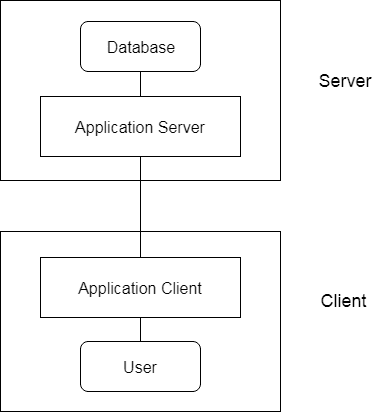


Image 42 – 3 tier Architecture

Reference - https://static.javatpoint.com/dbms/images/dbms-3-tier-architecture.png

### Three Schema Architecture

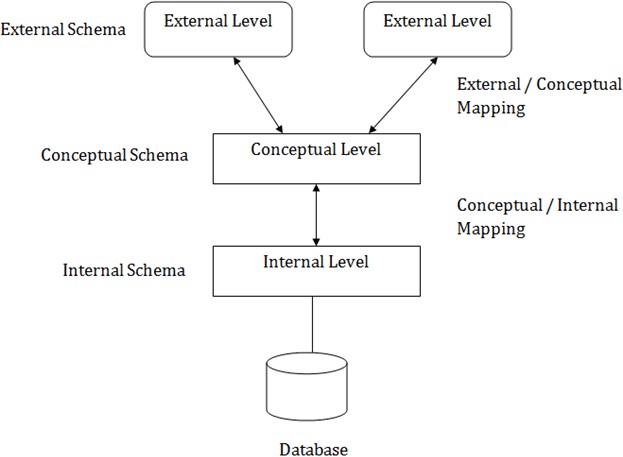


Image 43 – 3 Schema Architecture

Reference - https://static.javatpoint.com/dbms/images/dbms-three-schema-architecture.png

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

### 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.

### 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.

Programmers and database administrators work at this level.

### External Level

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.

The view schema describes the end user interaction with database systems.

# Relational Algebra

Relational Algebra is procedural query language, which takes Relation as input and generate relation as output.

Relational algebra operations are performed recursively on a relation.

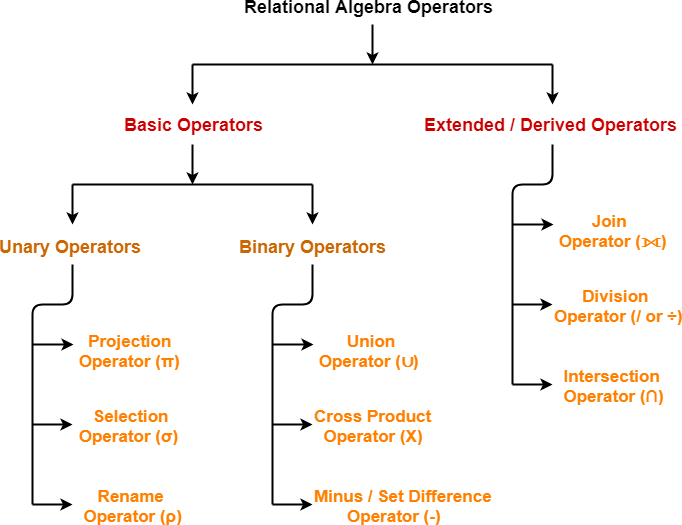


Image 44 – Relational Algebra Reference -

https://encrypted-tbn0.gstatic.com/images?q=tbn%3AANd9GcRrt8Z79xBi\_fs1wYROuo33pDaGmXFbTyRBWowkH\_UGkPdrf2Ft&usqp=CAU

### Select Operation

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

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:** |  |  |
| **σ BRANCH\_NAME="perryride" (LOAN)** | | |
| Output: |  |  |
| BRANCH\_NAME | LOAN\_NO | AMOUNT |
| Perryride | L-15 | 1500 |
| Perryride | L-16 | 1300 |

**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 ∏. 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:** |  |  |

### ∏ NAME, CITY (CUSTOMER)

Output:

NAME CITY

Jones Harrison

Smith Rye

Hays Harrison

### 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 ∪. 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:** |  |

### ∏ CUSTOMER\_NAME (BORROW) ∪ ∏ CUSTOMER\_NAME (DEPOSITOR)

Output:

CUSTOMER\_NAME

Johnson Smith Hayes Turner Jones Lindsay Jackson Curry Williams Mayes

### 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 ∩. Notation: R ∩ S

Example: Using the above DEPOSITOR table and BORROW table

### Input:

**∏ CUSTOMER\_NAME (BORROW) ∩ ∏ CUSTOMER\_NAME (DEPOSITOR)**

Output:

CUSTOMER\_NAME

Smith Jones

### 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 (-). Notation: R - S

Example: Using the above DEPOSITOR table and BORROW table

### Input:

**∏ CUSTOMER\_NAME (BORROW) - ∏ CUSTOMER\_NAME (DEPOSITOR)**

Output:

CUSTOMER\_NAME

Jackson Hayes Willians Curry

### 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.

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

1. Marketing
2. Sales
3. Legal

**Input:**

**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 |

**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. **ρ(STUDENT1, STUDENT)**

### Join Operation

Join operation is essentially a Cartesian product followed by a selection criterion. Join operation denoted by ⋈.

JOIN operation also allows joining variously related tuples from different relations.

### Inner Join

Natural join between two or more relations will result in all the combination of tuples where they have equal values for the common attribute

### Theta Join

The general case of JOIN operation is called a Theta join. It is denoted by symbol θ For example:

A ⋈ A.column 2 > B.column 2 (B)

### EQUI Join

When a theta join uses only equivalence condition, it becomes a equi join. For example:

A ⋈ A.column 2 = B.column 2 (B) Example

A ⋈θ B Theta join can use any conditions in the selection criteria.

### Left Outer Join

In the left outer join, operation allows keeping all tuple in the left relation. However, if there is no matching tuple is found in right relation, then the attributes of right relation in the join result are filled with null values.

Example : A ⟕ B

Example: Select students whose ROLL\_NO is greater than EMP\_NO of employees and details of other students as well

STUDENT⟕STUDENT.ROLL\_NO>EMPLOYEE.EMP\_NOEMPLOYEE

### Right Outer Join

In the left outer join, operation allows keeping all tuple in the left relation. However, if there is no matching tuple is found in right relation, then the attributes of right relation in the join result are filled with null values.

Example: Select students whose ROLL\_NO is greater than EMP\_NO of employees and details of other Employees as well

STUDENT⟖STUDENT.ROLL\_NO>EMPLOYEE.EMP\_NOEMPLOYEE

Full Outer Join

In the left outer join, operation allows keeping all tuple in the left relation. However, if there is no matching tuple is found in right relation, then the attributes of right relation in the join result are filled with null values.

Example: Select students whose ROLL\_NO is greater than EMP\_NO of employees and details of other Employees as well and other Students as well

STUDENT⟗STUDENT.ROLL\_NO>EMPLOYEE.EMP\_NOEMPLOYEE

### Division Operator

Division operator A÷B can be applied if and only if:

Attributes of B is proper subset of Attributes of A.

The relation returned by division operator will have attributes = (All attributes of A – All Attributes of B)

The relation returned by division operator will return those tuples from relation A which are associated to every B’s tuple.

Consider the relation STUDENT\_SPORTS and ALL\_SPORTS given in Table 2 and Table 3 above.

To apply division operator as

**STUDENT\_SPORTS÷ ALL\_SPORTS**

# Relational Calculus

## Introduction

Relational calculus is a non-procedural query language, and instead of algebra, it uses mathematical predicate calculus.

Tuple relational calculus which was originally proposed by Codd in the year 1972 and Domain relational calculus which was proposed by Lacroix and Pirotte in the year 1977

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

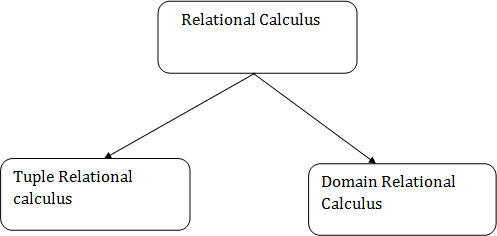


Image 45 – Relational Calculus

Reference - https://static.javatpoint.com/dbms/images/dbms-relational-calculus.png

### Tuple Relational Calculus

In tuple relational calculus, we work on filtering tuples based on the given condition.

Syntax: { T | Condition }

In domain relational calculus, filtering is done based on the domain of the attributes and not based on the tuple values.

Syntax: { c1, c2, c3, ..., cn | F(c1, c2, c3, ... ,cn)}

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:

{T | P (T)} or {T | Condition (T)} Where

T is the resulting tuples

P(T) is the condition used to fetch T. For example:

{ 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'.

The symbols used for logical operators are: ∧ **for AND,** ∨ **for OR and** ┓ **for NOT. Variables** can be constrained by quantified statements to tuples in a single relation: –

**Existential Quantifier**. ∃T ∈ R(Cond) will succeed if Cond succeeds for at least one tuple in T.

–

**Universal Quantifier.** ∀T ∈ R(Cond) will succeed if Cond succeeds for at all tuples in T

### Examples

{t | P (t)} or {t | condition (t)} —

this is also known as expression of relational calculus

Where t is the resulting tuples, P(t) is the condition used to fetch t.

{t | EMPLOYEE (t) and t.SALARY>10000} –

implies that it selects the tuples from EMPLOYEE relation such that resulting employee tuples will have salary greater than 10000. It is example of selecting a range of values.

{t | EMPLOYEE (t) AND t.DEPT\_ID = 10} –

this select all the tuples of employee name who work for Department 10.

{ T.name | F ACULT Y (T) AND T.DeptId = 0 CS0 }

can be read as: “Find all tuples T field such that T is a tuple in the FACULTY relation and the value of DeptId field is ’CS’. Return a tuple with a single field name which is equivalent to the name field of one such T tuple”.

{R | ∃T ∈ F ACULT Y (T.DeptId = 0 CS0 AND R.name = T.name)}

can be read as: “Find all tuples R such that there exists a tuple T in FACULTY with the DeptId field value ’CS’, and the value of the name field of R is equivalent to the name field of this tuple T.”

### Example 1 Find the loan number, branch, amount of loans of greater than or equal to 10000 amount.

{t| t ∈ loan ∧ t[amount]>=10000}

### Example 2 Find the loan number for each loan of an amount greater or equal to 10000.

{t| ∃ s ∈ loan(t[loan number] = s[loan number] ∧ s[amount]>=10000)}

### Example 3 Find the names of all customers who have a loan and an account at the bank.

{t | ∃ s ∈ borrower( t[customer-name] = s[customer-name]) ∧ ∃ u ∈ depositor( t[customer-name]

= u[customer-name])}

### Domain Relational Calculus

In domain relational calculus, filtering variable uses the domain of attributes.

omain 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.

The domain variables those will be in resulting relation must appear before | within ≺ and ≻ and all the domain variables must appear in which order they are in original relation or table..

Notation:

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

Where

a1, a2 are attributes

P stands for formula built by inner attributes

### Examples

Example 1

### Find the loan number, branch, amount of loans of greater than or equal to 100 amount.

{≺l, b, a≻ | ≺l, b, a≻ ∈ loan ∧ (a ≥ 100)}

### Example 2

**Find the loan number for each loan of an amount greater or equal to 150.**

{≺l≻ | ∃ b, a (≺l, b, a≻ ∈ loan ∧ (a ≥ 150)}

### Example 3

**Find the names of all customers having a loan at the “Main” branch and find the loan amount .**

{≺c, a≻ | ∃ l (≺c, l≻ ∈ borrower ∧ ∃ b (≺l, b, a≻ ∈ loan ∧ (b = “Main”)))}

### Example 4

{< name, age > | ∈ Student ∧ age <21}

Again, the above query will return the names and ages of the students in the table Student who not greater than 21 years old

### Example 5

{< Fname, Emp\_ID > | ∈ Employee ∧ Salary > 10000}

The result here will be returning the Fname and Emp\_ID values for all the rows in the employee table where salary is greater than 10000.

# RDBMS Technologies

## Oracle Database Technology Features

* Data Concurrency and Consistency
* Manageability
* Backup & Recovery
* Business Intelligence
* High Availability
* Very Large Databases
* Content Management

## MySQL Database Technology Features

* Data Concurrency and Consistency
* Scalability and Limit
* Backup & Recovery
* Connectivity
* High Availability
* Clients and Tools
* Workbench tool

## MongoDB Database Technology Features

* Indexing
* Replication
* Backup & Recovery
* Load Balancing
* Map Reducing and Aggregation
* Stores files of any size easily without complicating your stack.
* Cloud Support

## Microsoft SQL Server Database Technology Features

* Highest performing data warehouses
* End to End Mobile BI
* Backup & Recovery
* Load Balancing
* Built-in Analytics
* Mission Critical Availability
* Cloud Support

# Relational Data Structure/Relational Model

The **relational model** (**RM**) for database management is an approach to managing data using a structure and language consistent with first-order-predicate logic, first described in 1969 by English computer scientist Edgar F Codd, where all data is represented in terms of tuples, grouped into relations. A database organized in terms of the relational model is a relational database.

**Note:** First-order logic—also known as predicate logic, is a collection of formal systems used in mathematics, philosophy, linguistics, and computer science.

## What is a Relational Model?

**RELATIONAL MODEL (RM)** represents the database as a collection of relations. A relation is nothing but a table of values. Every row in the table represents a collection of related data values. These rows in the table denote a real-world entity or relationship.

* The table name and column names are helpful to interpret the meaning of values in each row.
* The data are represented as a set of relations.
* In the relational model, data is stored as tables.
* However, the physical storage of the data is independent of the way the data are logically organized.

Some popular Relational Database management systems are:

* DB2 and Informix Dynamic Server - IBM
* Oracle and RDB – Oracle
* SQL Server and Access - Microsoft

## Relational Model Concepts

* 1. **Attribute:** Each column in a Table. Attributes are the properties which define a relation. e.g., Student\_Rollno, NAME,etc.
  2. **Tables:** In the Relational model the, relations are saved in the table format. It is stored along with its entities. A table has two properties: rows and columns. Rows represent records and columns represent attributes.
  3. **Tuple:** It is nothing but a single row of a table, which contains a single record.
  4. **Relation Schema:** A relation schema represents the name of the relation with its attributes.
  5. **Degree:** The total number of attributes which in the relation is called the degree of the relation.
  6. **Cardinality:** Total number of rows present in the Table.
  7. **Column:** The column represents the set of values for a specific attribute.
  8. **Relation instance:** Relation instance is a finite set of tuples in the RDBMS system. Relation instances never have duplicate tuples.
  9. **Relation key:** Every row has one, two or multiple attributes, which is called relation key.
  10. **Attribute domain:** Every attribute has some predefined value and scope which is known as attribute domain

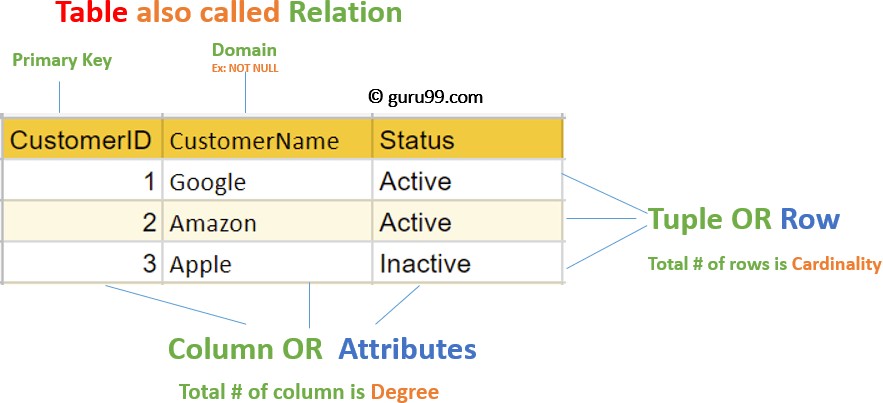


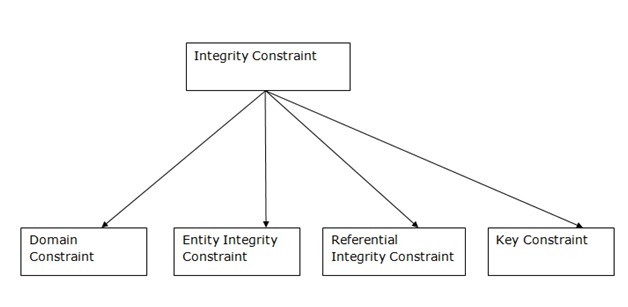
Image 1- Shows some of the **Relational Model Concepts Reference- https://**[**www.guru99.com/relational-data-model-dbms.html**](http://www.guru99.com/relational-data-model-dbms.html)

## Relational Integrity constraints

Relational Integrity constraints are referred to conditions which must be present for a valid relation. These integrity constraints are derived from the rules in the mini-world that the database represents.

* 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 constraints

1. Domain constraints
2. Entity Integrity constraints
3. Referential integrity constraints
4. Key constraints

© Edunet Foundation. All rights reserved. 380

Image 2- Types of integrity constraints

Reference- https://[www.javatpoint.com/dbms-integrity-constraints](http://www.javatpoint.com/dbms-integrity-constraints)

### 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:

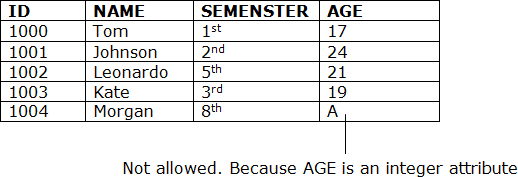


Image 3- Example of Domain Constraint

Reference- https://[www.javatpoint.com/dbms-integrity-constraints](http://www.javatpoint.com/dbms-integrity-constraints)

### 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:

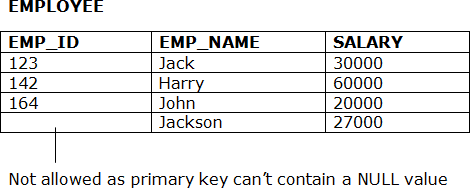


Image 4- Example of Entity integrity constraint Reference- https://[www.javatpoint.com/dbms-integrity-constraints](http://www.javatpoint.com/dbms-integrity-constraints)

### 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:

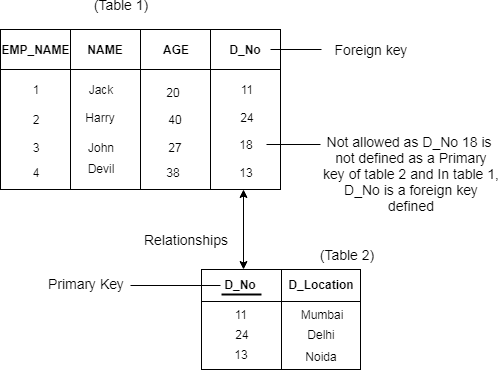


Image 5- referential integrity constraint

Reference- https://[www.javatpoint.com/dbms-integrity-constraints](http://www.javatpoint.com/dbms-integrity-constraints)

### 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:

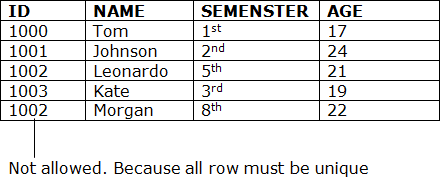


Image 6- Example of Key Constraint

Reference- https://[www.javatpoint.com/dbms-integrity-constraints](http://www.javatpoint.com/dbms-integrity-constraints)

## Operations in the Relational Model

Four basic update operations performed on the relational database model are Insert, update, delete and select.

* Insert is used to insert data into the relation
* Delete is used to delete tuples from the table.
* Modify allows you to change the values of some attributes in existing tuples.
* Select allows you to choose a specific range of data.

Whenever one of these operations are applied, integrity constraints specified on the relational database schema must never be violated.

### Insert Operation:

The insert operation gives values of the attribute for a new tuple which should be inserted into a relation.

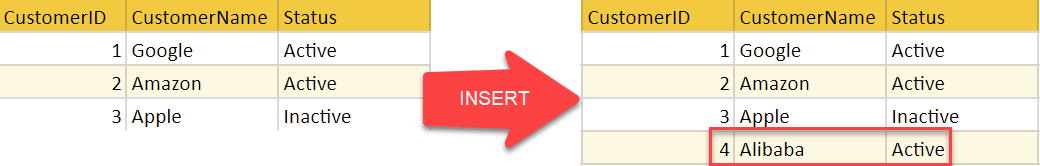


Image 7- Insert

Reference- https://[www.guru99.com/relational-data-model-dbms.html](http://www.guru99.com/relational-data-model-dbms.html)

### Update Operation

You can see that in the below-given relation table CustomerName= 'Apple' is updated from Inactive to Active.



Image 8- Update

Reference- https://[www.guru99.com/relational-data-model-dbms.html](http://www.guru99.com/relational-data-model-dbms.html)

### Delete Operation

To specify deletion, a condition on the attributes of the relation selects the tuple to be deleted.

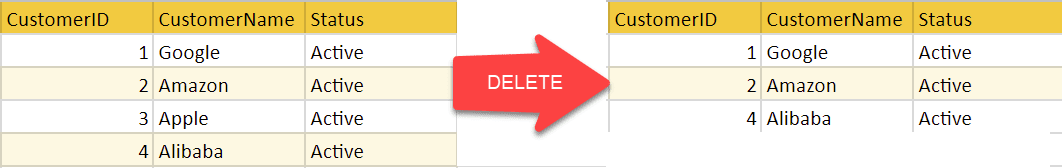


Image 9- Delete

Reference- https://[www.guru99.com/relational-data-model-dbms.html](http://www.guru99.com/relational-data-model-dbms.html)

In the above-given example, CustomerName= "Apple" is deleted from the table.

The Delete operation could violate referential integrity if the tuple which is deleted is referenced by foreign keys from other tuples in the same database.

### Select Operation



Image 10- Select

Reference- https://[www.guru99.com/relational-data-model-dbms.html](http://www.guru99.com/relational-data-model-dbms.html)

In the above-given example, CustomerName="Amazon" is selected.

## Advantages of using the Relational model

* **Simplicity**: A relational data model is simpler than the hierarchical and network model.
* **Structural Independence**: The relational database is only concerned with data and not with a structure. This can improve the performance of the model.
* **Easy to use**: The relational model is easy as tables consisting of rows and columns is quite natural and simple to understand
* **Query capability**: It makes possible for a high-level query language like SQL to avoid complex database navigation.
* **Data independence**: The structure of a database can be changed without having to change any application.
* **Scalable**: Regarding a number of records, or rows, and the number of fields, a database should be enlarged to enhance its usability.

## Disadvantages of using Relational model

* Few relational databases have limits on field lengths which can't be exceeded.
* Relational databases can sometimes become complex as the amount of data grows, and the relations between pieces of data become more complicated.
* Complex relational database systems may lead to isolated databases where the information cannot be shared from one system to another.

# Key and Relational Data Manipulation

## Keys

### What are Keys?

A DBMS key is an attribute or set of an attribute which helps you to identify a row(tuple) in a relation(table). They allow you to find the relation between two tables. Keys help you uniquely identify a row in a table by a combination of one or more columns in that table.

|  |  |  |  |
| --- | --- | --- | --- |
| **Example:** |  | | |
|  | Student ID | FirstName | LastName |
|  | 1 | Qanith | Khan |
|  | 2 | Rajesh | Singh |
|  | 3 | john | Hale |

In the above-given example, Student ID is a primary key because it uniquely identifies an Student record. In this table, no other Student can have the same Student ID.

### Why do we need a Key?

Here are reasons for using Keys in the DBMS system.

* Keys help you to identify any row of data in a table. In a real-world application, a table could contain thousands of records. Moreover, the records could be duplicated. Keys ensure that you can uniquely identify a table record despite these challenges.
* Allows you to establish a relationship between and identify the relation between tables
* Help you to enforce identity and integrity in the relationship.

### Various Keys

Each keys have their different functionality:

* Super Key
* Primary Key
* Candidate Key
* Alternate Key
* Foreign Key
* Compound Key
* Composite Key
* Surrogate Key

### Super key:

A superkey is a group of single or multiple keys which identifies rows in a table. A Super key may have additional attributes that are not needed for unique identification.

### Example:

|  |  |  |
| --- | --- | --- |
| **EmpSSN** | **EmpNum** | **Empname** |
| 9812345098 | AB05 | Shown |
| 9876512345 | AB06 | Roslyn |

199937890 AB07 James

In the above-given example, EmpSSN and EmpNum name are superkeys.

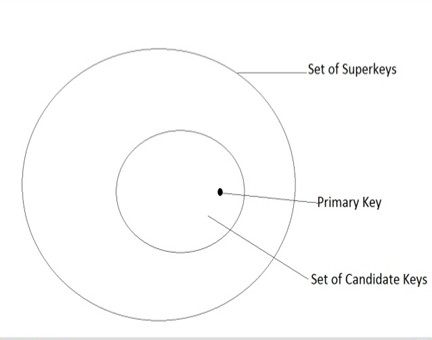


Image 11- Relation b/w keys Reference-

https://[www.slideshare.net/TechtudNetwork/relation-between-super-key-candidate-key-and-primary-key?qid=c7bb9eac-8135-480c-92ee-890bf4c](http://www.slideshare.net/TechtudNetwork/relation-between-super-key-candidate-key-and-primary-key?qid=c7bb9eac-8135-480c-92ee-890bf4c) 744aa&v=&b=&from\_search=1

### Primary Key

**PRIMARY KEY** is a column or group of columns in a table that uniquely identify every row in that table. The Primary Key can't be a duplicate meaning the same value can't appear more than once in the table. A table cannot have more than one primary key.

### Rules for defining Primary key:

* Two rows can't have the same primary key value
* It must for every row to have a primary key value.
* The primary key field cannot be null.
* The value in a primary key column can never be modified or updated if any foreign key refers to that primary key.

### Example:

In the following example, <code>StudID</code> is a Primary Key.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **StudID** | Roll No | First Name | LastName | Email |
| 1 | 11 | Tom | Price | [abc@gmail.com](mailto:abc@gmail.com) |
| 2 | 12 | Nick | Wright | [xyz@gmail.com](mailto:xyz@gmail.com) |
| 3 | 13 | Dana | Natan | [mno@yahoo.com](mailto:mno@yahoo.com) |

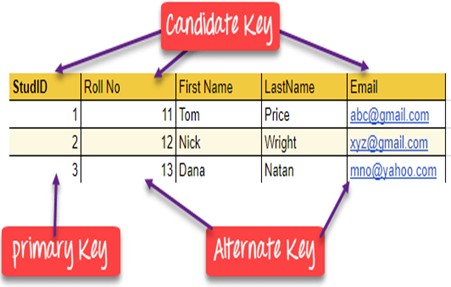


Image 12- Primary Key

### Alternate key

**ALTERNATE KEYS** is a column or group of columns in a table that uniquely identify every row in that table. A table can have multiple choices for a primary key but only one can be set as the primary key. All the keys which are not primary key are called an Alternate Key.

### Example:

In this table, StudID, Roll No, Email are qualified to become a primary key. But since StudID is the primary key, Roll No, Email becomes the alternative key.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **StudID** | Roll No | First Name | LastName | Email |
| 1 | 11 | Tom | Price | [abc@gmail.com](mailto:abc@gmail.com) |
| 2 | 12 | Nick | Wright | [xyz@gmail.com](mailto:xyz@gmail.com) |
| 3 | 13 | Dana | Natan | [mno@yahoo.com](mailto:mno@yahoo.com) |

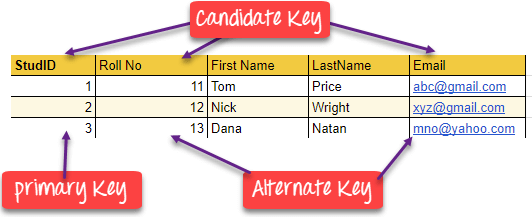


Image 13- Alternate Key

### Candidate Key

**CANDIDATE KEY** is a set of attributes that uniquely identify tuples in a table. Candidate Key is a super key with no repeated attributes. The Primary key should be selected from the candidate keys. Every table must have at least a single candidate key. A table can have multiple candidate keys but only a single primary key.

### Properties of Candidate key:

* It must contain unique values
* Candidate key may have multiple attributes
* Must not contain null values
* It should contain minimum fields to ensure uniqueness
* Uniquely identify each record in a table

Example: In the given table Stud ID, Roll No, and email are candidate keys which help us to uniquely identify the student record in the table.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **StudID** | Roll No | First Name | LastName | Email |
| 1 | 11 | Tom | Price | [abc@gmail.com](mailto:abc@gmail.com) |
| 2 | 12 | Nick | Wright | [xyz@gmail.com](mailto:xyz@gmail.com) |
| 3 | 13 | Dana | Natan | [mno@yahoo.com](mailto:mno@yahoo.com) |

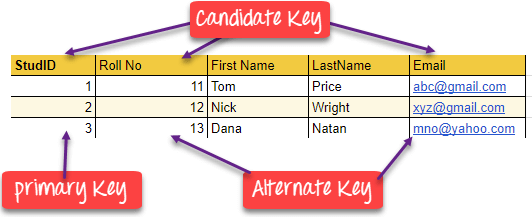


Image 14- Candidate Key

Reference-https://[www.guru99.com/dbms-keys.html](http://www.guru99.com/dbms-keys.html)

### Foreign key

**FOREIGN KEY** is a column that creates a relationship between two tables. The purpose of Foreign keys is to maintain data integrity and allow navigation between two different instances of an entity. It acts as a cross-reference between two tables as it references the primary key of another table.

### Example:

**DeptCode DeptName**

001 Science

002 English

|  |  |  |  |
| --- | --- | --- | --- |
| 005 |  | Computer |  |
| **Teacher ID** | **Fname** |  | **Lname** |
| B002 | David |  | Warner |
| B017 | Sara |  | Joseph |
| B009 | Mike |  | Brunton |

In this example, we have two table, teach and department in a school. However, there is no way to see which search work in which department.

In this table, adding the foreign key in Deptcode to the Teacher name, we can create a relationship between the two tables.

|  |  |  |  |
| --- | --- | --- | --- |
| **Teacher ID** | **DeptCode** | **Fname** | **Lname** |
| B002 | 002 | David | Warner |
| B017 | 002 | Sara | Joseph |

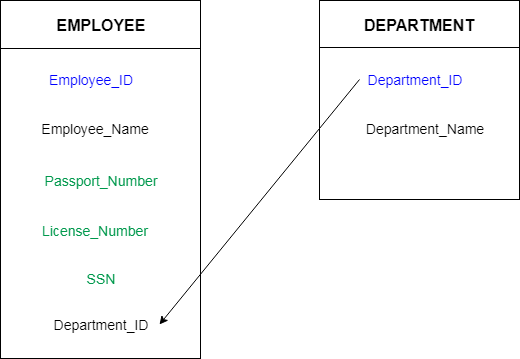
B009 001 Mike Brunton This concept is also known as Referential Integrity.

Image 15- Foreign key

Reference- https://[www.javatpoint.com/dbms-keys](http://www.javatpoint.com/dbms-keys)

### Compound key

**COMPOUND KEY** has two or more attributes that allow you to uniquely recognize a specific record. It is possible that each column may not be unique by itself within the database. However, when combined with the other column or columns the combination of composite keys become unique. The purpose of compound key is to uniquely identify each record in the table.

### Example:

|  |  |  |  |
| --- | --- | --- | --- |
| **OrderNo** | **PorductID** | **Product Name** | **Quantity** |
| B005 | JAP102459 | Mouse | 5 |
| B005 | DKT321573 | USB | 10 |
| B005 | OMG446789 | LCD Monitor | 20 |
| B004 | DKT321573 | USB | 15 |
| B002 | OMG446789 | Laser Printer | 3 |

In this example, OrderNo and ProductID can't be a primary key as it does not uniquely identify a record. However, a compound key of Order ID and Product ID could be used as it uniquely identified each record.

### Composite key

**COMPOSITE KEY** is a combination of two or more columns that uniquely identify rows in a table. The combination of columns guarantees uniqueness, though individually uniqueness is not guaranteed. Hence, they are combined to uniquely identify records in a table.

The difference between compound and the composite key is that any part of the compound key can be a foreign key, but the composite key may or maybe not a part of the foreign key.

### Surrogate Key

An artificial key which aims to uniquely identify each record is called a surrogate key. These kind of key are unique because they are created when you don't have any natural primary key. They do not lend any meaning to the data in the table. Surrogate key is usually an integer.

|  |  |  |  |
| --- | --- | --- | --- |
| **Fname** | **Lastname** | **Start Time** | **End Time** |
| Anne | Smith | 09:00 | 18:00 |
| Jack | Francis | 08:00 | 17:00 |
| Anna | McLean | 11:00 | 20:00 |
| Shown | Willam | 14:00 | 23:00 |

Above, given example, shown shift timings of the different employee. In this example, a surrogate key is needed to uniquely identify each employee.

Surrogate keys are allowed when

* No property has the parameter of the primary key.
* In the table when the primary key is too big or complicated.

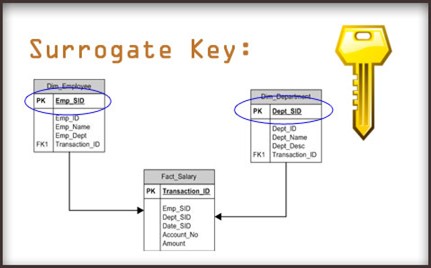


Image 16- Surrogate Key Reference-

https://[www.google.com/search?q=Surrogate+Key+&tbm=isch&ved=2ahUKEwiX99ycj-roAhUI5DgGHffQBnQQ2-cCegQIABAA&oq=Surroga](http://www.google.com/search?q=Surrogate%2BKey%2B&tbm=isch&ved=2ahUKEwiX99ycj-roAhUI5DgGHffQBnQQ2-cCegQIABAA&oq=Surroga) te+Key+&gs\_lcp=CgNpbWcQAzIECAAQQzICCAAyAggAMgIIADICCAAyAggAMgIIADICCAAyAggAMgIIAFD-nBpY\_pwaYJuhGmgAcA B4AIABswGIAbMBkgEDMC4xmAEAoAEBqgELZ3dzLXdpei1pbWc&sclient=img&ei=89OWXtexAojI4-EP96GboAc&bih=657&biw=1366# imgrc=JViJUIE4gOGs0M

### Let’s see the difference between keys Primary Key and Foreign Key:

**S.NO. PRIMARY KEY FOREIGN KEY**

1. A primary key is used to ensure data in the

A foreign key is a column or group of columns in a relational database table that provides a link between data in two tables.

specific column is unique.

It uniquely identifies a

1. record in the relational database table.

It refers to the field in a table which is the primary key of another table.

Only one primary key

3

is allowed in a table.

Whereas more than one foreign key are allowed in a table.

It is a combination of

4 UNIQUE and Not Null constraints.

It can contain duplicate values and a table in a relational database.

It does not allow

5

NULL values.

It can also contain NULL values.

Its value cannot be

1. deleted from the parent table.

Its value can be deleted from the child table.

It constraint can be

1. implicitly defined on the temporary tables.

It constraint cannot be defined on the local or global temporary tables.

### Primary and Candidate Key:

**S.NO PRIMARY KEY CANDIDATE KEY**

Primary key is a minimal super key. So

1. there is one and only one primary key in

a relation.

While in a relation there can be more than one candidate key.

Any attribute of Primary key can not

2.

contain NULL value.

While in Candidate key any attribute can contain NULL value.

Primary key can be optional to specify

3.

any relation.

But without candidate key there can’t be specified any relation.

Primary key specifies the important

4.

attribute for the relation.

Candidate specifies the key which can qualify for primary key.

Its confirmed that a primary key is a

5.

candidate key.

But Its confirmed that a candidate key can be a primary key.

### Super Key and Candidate Key

**S.NO SUPER KEY CANDIDATE KEY**

Super Key is an attribute (or set of

1. attributes) that is used to uniquely identifies all attributes in a relation.

Candidate Key is a proper subset of a super key.

All super keys can’t be candidate

2.

keys.

But all candidate keys are super keys.

Various super keys together makes the

3.

criteria to select the candidate keys.

Various candidate keys together makes the criteria to select the primary keys.

In a relation, number of super keys are

4.

more than number of candidate keys.

While in a relation, number of candidate keys are less than number of super keys.

Super key’s attributes can contain

5.

NULL values.

Candidate key’s attributes can also contain NULL values.

### Primary key and Unique key

**PARAMETER PRIMARY KEY UNIQUE KEY**

Basic

Used to serve as a unique identifier for each row in a table.

Uniquely determines a row which isn’t primary key.

NULL value acceptance Cannot accept NULL values. Can accept one NULL value.

Number of keys that

can be defined in the table

Only one primary key More than one unique key

Index Creates clustered index Creates non-clustered index

# Data Manipulation Language (DML)

## Introduction to DML

* DML stands for **Data Manipulation Language.**
* It is a language used for selecting, inserting, deleting and updating data in a database.
* It is used to retrieve and manipulate data in a relational database.

### DDL commands are as follows:

1. SELECT
2. INSERT
3. UPDATE
4. DELETE

DML performs read-only queries of data.

### SELECT COMMAND

* + **SELECT command** is used to retrieve data from the database.
  + This command allows database users to retrieve the specific information they desire from an operational database.
  + It returns a result set of records from one or more tables.

### SELECT Command has many optional clauses are as stated below:

|  |  |
| --- | --- |
| **Clause** | **Description** |
| WHERE | It specifies which rows to retrieve. |

|  |  |
| --- | --- |
| GROUP BY | It is used to arrange the data into groups. |
| HAVING | It selects among the groups defined by the GROUP BY clause. |
| ORDER BY | It specifies an order in which to return the rows. |
| AS | It provides an alias which can be used to temporarily rename tables or columns. |

**Syntax:**

SELECT \* FROM <table\_name>;

**Example:** Select Command

SELECT \* FROM employee; OR

SELECT \* FROM employee where salary >=10,000;

### INSERT COMMAND

* + **INSERT command** is used for inserting data into a table.
  + Using this command, you can add one or more records to any single table in a database.
  + It is also used to add records to an existing code.

### Syntax:

INSERT INTO <table\_name> (`column\_name1` <datatype>, `column\_name2` <datatype>, . . . ,

`column\_name\_n` <database>) VALUES (`value1`, `value2`, . . . , `value n`);

### Example:

INSERT INTO employee (`eid` int, `ename` varchar(20), `city` varchar(20)) VALUES (`1`, `ABC`, `PUNE`);

### UPDATE COMMAND

* + **UPDATE command** is used to modify the records present in existing table.
  + This command updates existing data within a table.
  + It changes the data of one or more records in a table.

### Syntax:

UPDATE <table\_name>

SET <column\_name = value> WHERE condition;

**Example:** Update Command UPDATE employee

SET salary=20000 WHERE ename='ABC';

### DELETE COMMAND

* + **DELETE command** is used to delete some or all records from the existing table.
  + It deletes all the records from a table.

### Syntax:

DELETE FROM <table\_name> WHERE <condition>;

**Example:** Delete Command DELETE FROM employee WHERE emp\_id = '001';

If we does not write the WHERE condition, then all rows will get deleted.

# Relational Algebra

## Relational Algebra

**RELATIONAL ALGEBRA** is a widely used procedural query language. It collects instances of relations as input and gives occurrences of relations as output. It uses various operations to perform this action. Relational algebra operations are performed recursively on a relation. The output of these operations is a new relation, which might be formed from one or more input relations.

* + First created by Edgar F. Codd
  + The main application of relational algebraic providing a theoretical foundation for relational databases, particularly query languages for such databases, chief among which is SQL.

# Relational Algebraic Operations

Relational Algebra divide in various groups

### Unary Relational Operations

* 1. SELECT (symbol: σ)
  2. PROJECT (symbol: π)
  3. RENAME (symbol: ρ)

### Relational Algebra Operations From Set Theory

* 1. UNION (υ)
  2. INTERSECTION ( ∩),
  3. DIFFERENCE (-)
  4. CARTESIAN PRODUCT ( x )

### Binary Relational Operations

* 1. JOIN

## Unary Relational Operations

### SELECT (σ)

The SELECT operation is used for selecting a subset of the tuples according to a given selection condition. Sigma(σ)Symbol denotes it. It is used as an expression to choose tuples which meet the selection condition. Select operation selects tuples that satisfy a given predicate.

σp(r)

σ is the predicate

r stands for relation which is the name of the table p is prepositional logic

### Example 1

σ topic = "Database" (Tutorials)

Output - Selects tuples from Tutorials where topic = 'Database'.

### Example 2

σ topic = "Database" and author = "Edunet"( Tutorials)

Output - Selects tuples from Tutorials where the topic is 'Database' and 'author' is Edunet.

### Example 3

σ sales > 50000 (Customers)

Output - Selects tuples from Customers where sales is greater than 50000

### Projection(π)

The projection eliminates all attributes of the input relation but those mentioned in the projection list. The projection method defines a relation that contains a vertical subset of Relation.

This helps to extract the values of specified attributes to eliminate duplicate values. (pi) The symbol used to choose attributes from a relation. This operation helps you to keep specific columns from a relation and discards the other columns.

### Example of Projection:

Consider the following table

|  |  |  |
| --- | --- | --- |
| **CustomerID** | **CustomerName** | **Status** |
| 1 | Google | Active |
| 2 | Amazon | Active |

1. Apple Inactive

4 Alibaba Active

Here, the projection of CustomerName and status will give Π CustomerName, Status (Customers)

**CustomerName Status**

Google Active

Amazon Active

Apple Inactive

Alibaba Active

### Rename(ρ)

The results of relational algebra are also relations but without any name. The rename operation allows us to rename the output relation. 'rename' operation is denoted with small Greek letter rho *ρ*.

Notation − *ρ* x (E)

Where the result of expression E is saved with the name of x.

**Example:** ρ(RelationNew, RelationOld)

### Relational Algebra Operations From Set Theory Union operation (υ)

UNION is symbolized by ∪ symbol. It includes all tuples that are in tables A or in B. It also eliminates duplicate tuples. So, set A UNION set B would be expressed as:

The result <- A ∪ B

For a union operation to be valid, the following conditions must hold -

* + R and S must be the same number of attributes.
  + Attribute domains need to be compatible.
  + Duplicate tuples should be automatically removed.

### Example

Consider the following tables.

|  |  |  |  |
| --- | --- | --- | --- |
| **Table A** |  | **Table B** |  |
| **column 1** | **column 2** | **column 1** | **column 2** |
| 1 | 1 | 1 | 1 |
| 1 | 2 | 1 | 3 |

A ∪ B gives



|  |  |
| --- | --- |
| **Table A** ∪ **B** |  |
| **column 1** | **column 2** |
| 1 | 1 |
| 1 | 2 |
| 1 | 3 |

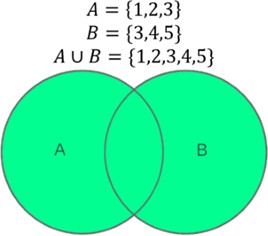


Image 17- Union Example

Reference- https://[www.codeproject.com/articles/1172312/just-enough-set-theory-when-sets-collide-part-of](http://www.codeproject.com/articles/1172312/just-enough-set-theory-when-sets-collide-part-of)

### Intersection

An intersection is defined by the symbol ∩ A ∩ B

Defines a relation consisting of a set of all tuple that are in both A and B. However, A and B must be union-compatible.

**Example:** A ∩ B

**Table A ∩ B**

### column 1 column 2

1 1

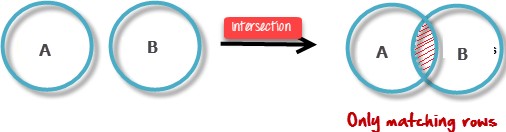


Image 18- Intersection Example

Reference- https://[www.guru99.com/relational-algebra-dbms.html](http://www.guru99.com/relational-algebra-dbms.html)

### Set Difference (-)

* Symbol denotes it. The result of A - B, is a relation which includes all tuples that are in A but not in B.
  + The attribute name of A has to match with the attribute name in B.
  + The two-operand relations A and B should be either compatible or Union compatible.
  + It should be a defined relation consisting of the tuples that are in relation A, but not in B.

### Example

A-B

**Table A - B**

### column 1 column 2

1 2

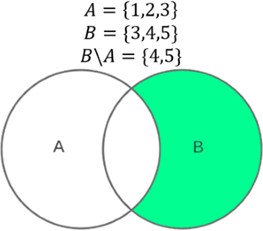


Image 19- Set Difference

### Cartesian product(X)

This type of operation is helpful to merge columns from two relations. Generally, a Cartesian product is never a meaningful operation when it performs alone. However, it becomes meaningful when it is followed by other operations.

### Example – Cartesian product

σ column 2 = '1' (A X B)

Output – The above example shows all rows from relation A and B whose column 2 has value 1

**σ column 2 = '1' (A X B)**

### column 1 column 2

1 1

1 1

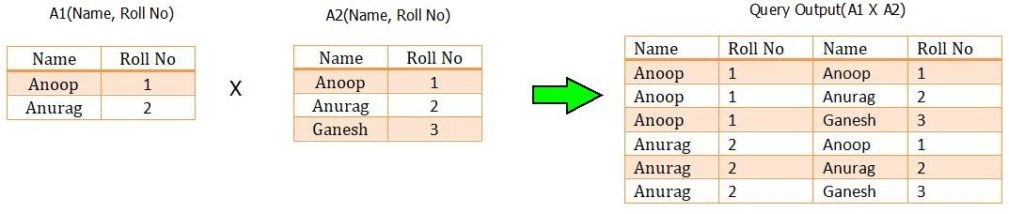


Image 20- Cartesian Product

Reference- https://[www.minigranth.com/dbms-tutorial/relational-algebra/](http://www.minigranth.com/dbms-tutorial/relational-algebra/)

## Binary Relational Operations

### JOIN

Join operation is essentially a cartesian product followed by a selection criterion. Join operation denoted by ⋈.

JOIN operation also allows joining variously related tuples from different relations.

### Types of JOIN

Various forms of join operation are:

* 1. Inner Joins:
     + Theta join
     + EQUI join
     + Natural join
  2. Outer join:
     + Left Outer Join
     + Right Outer Join
     + Full Outer Join

### Inner Join:

In an inner join, only those tuples that satisfy the matching criteria are included, while the rest are excluded. Let's study various types of Inner Joins:

### Theta Join:

The general case of JOIN operation is called a Theta join. It is denoted by symbol **θ Example:** A ⋈θ B

Theta join can use any conditions in the selection criteria.

### For example:

A ⋈ A.column 2 > B.column 2 (B)



**A** ⋈ **A.column 2 > B.column 2 (B)**

### column 1 column 2

1 2

**EQUI join:**

When a theta join uses only equivalence condition, it becomes a equi join.

### For example:

A ⋈ A.column 2 = B.column 2 (B)



**A** ⋈ **A.column 2 = B.column 2 (B)**

### column 1 column 2

1 1

EQUI join is the most difficult operations to implement efficiently in an RDBMS and one reason why RDBMS have essential performance problems.

### NATURAL JOIN (⋈)

Natural join can only be performed if there is a common attribute (column) between the relations. The name and type of the attribute must be same.

Example

Consider the following two tables

**C**

### Num Square

2 4

3 9

**D**

### Num Cube

2 8

3 27

C ⋈ D



|  |  |  |
| --- | --- | --- |
| **C** ⋈ **D** |  | |
| **Num** | **Square** | **Cube** |
| 2 | 4 | 4 |
| 3 | 9 | 27 |

### OUTER JOIN

In an outer join, along with tuples that satisfy the matching criteria, we also include some or all tuples that do not match the criteria.

### Left Outer Join(A B)

In the left outer join, operation allows keeping all tuple in the left relation. However, if there is no matching tuple is found in right relation, then the attributes of right relation in the join result are filled with null values.



Image 21- Left Outer Join

Reference- https://[www.guru99.com/relational-algebra-dbms.html](http://www.guru99.com/relational-algebra-dbms.html)

Consider the following 2 Tables

|  |  |
| --- | --- |
| **A** |  |
| **Num** | **Square** |
| 2 | 4 |
| 3 | 9 |
| 4 | 16 |
|  |  |
| **B** |  |
| **Num** | **Cube** |
| 2 | 8 |
| 3 | 18 |
| 5 | 75 |

A  B



|  |  |  |
| --- | --- | --- |
| **A** ⋈ **B** |  | |
| **Num** | **Square** | **Cube** |
| 2 | 4 | 4 |
| 3 | 9 | 9 |
| 4 | 16 | - |

### Right Outer Join: ( A B )

In the right outer join, operation allows keeping all tuple in the right relation. However, if there is no matching tuple is found in the left relation, then the attributes of the left relation in the join result are filled with null values.

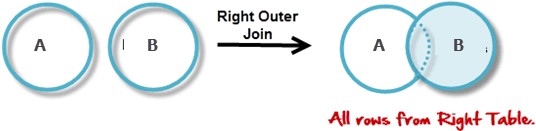


Image 22- Right Outer Join

Reference- https://[www.guru99.com/relational-algebra-dbms.html](http://www.guru99.com/relational-algebra-dbms.html)

A  B



**A** ⋈ **B**

### Num Cube Square

|  |  |  |
| --- | --- | --- |
| 2 | 8 | 4 |
| 3 | 18 | 9 |
| 5 | 75 | - |

**Full Outer Join: ( A B)**

In a full outer join, all tuples from both relations are included in the result, irrespective of the matching condition.

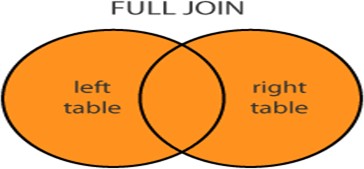


Image 23- Full Outer Join

A B



|  |  |  |
| --- | --- | --- |
| **A** ⋈ **B** |  | |
| **Num** | **Cube** | **Square** |
| 2 | 4 | 8 |
| 3 | 9 | 18 |

|  |  |  |
| --- | --- | --- |
| 4 | 16 | - |
| 5 | - | 75 |

# Set Operations

The SQL Set operation is used to combine the two or more SQL SELECT statements.

## Types of Set Operation

1. Union
2. UnionAll
3. Intersect
4. Minus



Image 24- Set Operations

Reference- https://[www.javatpoint.com/dbms-sql-set-operation](http://www.javatpoint.com/dbms-sql-set-operation)

### Union

* + The SQL Union operation is used to combine the result of two or more SQL SELECT queries.
  + In the union operation, all the number of datatype and columns must be same in both the tables on which UNION operation is being applied.
  + The union operation eliminates the duplicate rows from its resultset.

### Syntax

1. SELECT column\_name FROM table1
2. UNION
3. SELECT column\_name FROM table2;

**Example:**

**The First table**

|  |  |
| --- | --- |
| ID | NAME |
| 1 | Jack |
| 2 | Harry |
| 3 | Jackson |

**The Second table**

|  |  |
| --- | --- |
| ID | NAME |
| 3 | Jackson |
| 4 | Stephan |
| 5 | David |

Union SQL query will be:

* 1. SELECT \* FROM First
  2. UNION
  3. SELECT \* FROM Second;

### The resultset table will look like:

|  |  |
| --- | --- |
| ID | NAME |
| 1 | Jack |
| 2 | Harry |

|  |  |
| --- | --- |
| 3 | Jackson |
| 4 | Stephan |
| 5 | David |

1. **Union All**

Union All operation is equal to the Union operation. It returns the set without removing duplication and sorting the data.

### Syntax:

* 1. SELECT column\_name FROM table1
  2. UNION ALL
  3. SELECT column\_name FROM table2; **Example:** Using the above First and Second table. Union All query will be like:

1. SELECT \* FROM First
2. UNION ALL
3. SELECT \* FROM Second;

### The resultset table will look like:

|  |  |
| --- | --- |
| ID | NAME |
| 1 | Jack |
| 2 | Harry |
| 3 | Jackson |
| 3 | Jackson |
| 4 | Stephan |
| 5 | David |

1. **Intersect**
   * It is used to combine two SELECT statements. The Intersect operation returns the common rows from both the SELECT statements.
   * In the Intersect operation, the number of datatype and columns must be the same.
   * It has no duplicates and it arranges the data in ascending order by default.

### Syntax

1. SELECT column\_name FROM table1
2. INTERSECT
3. SELECT column\_name FROM table2;

### Example:

**Using the above First and Second table.**

Intersect query will be:

1. SELECT \* FROM First
2. INTERSECT
3. SELECT \* FROM Second;

### The resultset table will look like:

|  |  |
| --- | --- |
| **ID**  **NAME** | |
| **3** | **Jackson** |

1. **Minus**
   * It combines the result of two SELECT statements. Minus operator is used to display the rows which are present in the first query but absent in the second query.
   * It has no duplicates and data arranged in ascending order by default.

### Syntax:

* 1. SELECT column\_name FROM table1
  2. MINUS
  3. SELECT column\_name FROM table2;

### Example

**Using the above First and Second table.**

Minus query will be:

1. SELECT \* FROM First
2. MINUS
3. SELECT \* FROM Second;

**The resultset table will look like:**

ID NAME

|  |  |
| --- | --- |
| 1 | Jack |
| 2 | Harry |

# Fundamental Operations

## SELECT (σ)

The SELECT operation is used for selecting a subset of the tuples according to a given selection condition. Sigma(σ)Symbol denotes it. It is used as an expression to choose tuples which meet the selection condition. Select operation selects tuples that satisfy a given predicate.

σp(r)

σ is the predicate

r stands for relation which is the name of the table p is prepositional logic

### Example 1

σ topic = "Database" (Tutorials)

Output - Selects tuples from Tutorials where topic = 'Database'.

### Example 2

σ topic = "Database" and author = "Edunet"( Tutorials)

Output - Selects tuples from Tutorials where the topic is 'Database' and 'author' is Edunet.

### Example 3

σ sales > 50000 (Customers)

Output - Selects tuples from Customers where sales is greater than 50000

## Projection(π)

The projection eliminates all attributes of the input relation but those mentioned in the projection list. The projection method defines a relation that contains a vertical subset of Relation.

This helps to extract the values of specified attributes to eliminate duplicate values. (pi) The symbol used to choose attributes from a relation. This operation helps you to keep specific columns from a relation and discards the other columns.

### Example of Projection:

Consider the following table

|  |  |  |
| --- | --- | --- |
| **CustomerID** | **CustomerName** | **Status** |
| 1 | Google | Active |
| 2 | Amazon | Active |
| 3 | Apple | Inactive |
| 4 | Alibaba | Active |

Here, the projection of CustomerName and status will give Π CustomerName, Status (Customers)

**CustomerName Status**

Google Active

Amazon Active

Apple Inactive

Alibaba Active

## Cartesian product(X)

This type of operation is helpful to merge columns from two relations. Generally, a Cartesian product is never a meaningful operation when it performs alone. However, it becomes meaningful when it is followed by other operations.

### Example – Cartesian product

σ column 2 = '1' (A X B)

Output – The above example shows all rows from relation A and B whose column 2 has value 1

**σ column 2 = '1' (A X B)**

### column 1 column 2

1 1

1 1

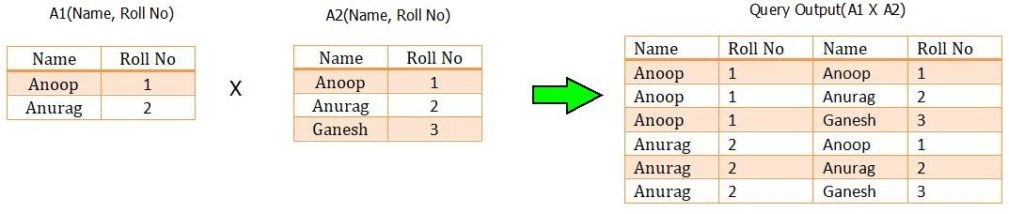


Image 20- Cartesian Product

Reference- https://[www.minigranth.com/dbms-tutorial/relational-algebra/](http://www.minigranth.com/dbms-tutorial/relational-algebra/)

## Union operation (υ)

UNION is symbolized by ∪ symbol. It includes all tuples that are in tables A or in B. It also eliminates duplicate tuples. So, set A UNION set B would be expressed as:

The result <- A ∪ B

For a union operation to be valid, the following conditions must hold -

* R and S must be the same number of attributes.
* Attribute domains need to be compatible.
* Duplicate tuples should be automatically removed.

### Example

Consider the following tables.

|  |  |  |  |
| --- | --- | --- | --- |
| **Table A** |  | **Table B** |  |
| **column 1** | **column 2** | **column 1** | **column 2** |

1

1

1

1

1 2 1 3

A ∪ B gives



**Table A** ∪ **B**

### column 1 column 2

1

1

1 2

1

3

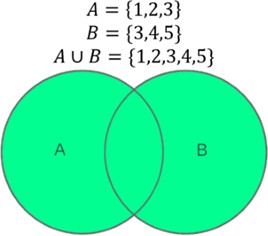


Image 17- Union Example

Reference- https://[www.codeproject.com/articles/1172312/just-enough-set-theory-when-sets-collide-part-of](http://www.codeproject.com/articles/1172312/just-enough-set-theory-when-sets-collide-part-of)

## Set Difference (-)

* Symbol denotes it. The result of A - B, is a relation which includes all tuples that are in A but not in B.
  + The attribute name of A has to match with the attribute name in B.
  + The two-operand relations A and B should be either compatible or Union compatible.
  + It should be a defined relation consisting of the tuples that are in relation A, but not in B.

### Example

A-B

**Table A - B**

**column 1 column 2**

1 2

# Relational Calculus

## What is Relational Calculus?

Contrary to Relational Algebra which is a procedural query language to fetch data and which also explains how it is done, **Relational Calculus** in non-procedural query language and has no description about how the query will work or the data will be fetched. It only focuses on what to do, and not on how to do it.

Relational Calculus exists in two forms:

1. Tuple Relational Calculus (TRC)
2. Domain Relational Calculus (DRC)

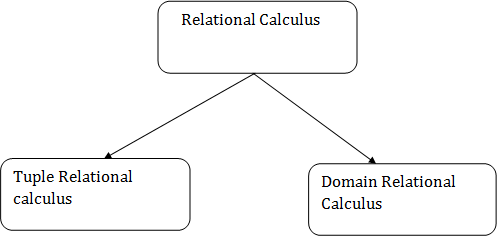


Image 25- Types of Relational Calculus

Reference- https://[www.javatpoint.com/dbms-relational-calculus](http://www.javatpoint.com/dbms-relational-calculus)

## Tuple Relational Calculus (TRC)

Tuple Relational Calculus is a **non-procedural query language** unlike relational algebra. Tuple Calculus provides only the description of the query but it does not provide the methods to solve it. Thus, it explains what to do but not how to do.

In Tuple Calculus, a query is expressed as

{t| P(t)}

where t = resulting tuples,

P(t) = known as Predicate and these are the conditions that are used to fetch t Thus, it generates a set of all tuples t, such that Predicate P(t) is true for t.

P(t) may have various conditions logically combined with OR (∨), AND (∧), NOT(¬).

It also uses quantifiers:

∃ t ∈ r (Q(t)) = ”there exists” a tuple in t in relation r such that predicate Q(t) is true.

∀ t ∈ r (Q(t)) = Q(t) is true “for all” tuples in relation r.

### Example:

**Table-1: Customer**

**CITY**

**STREET**

**CUSTOMER NAME**

Saurabh A7 Patiala

Mehak B6 Jalandhar

Sumiti D9 Ludhiana

Ria A5 Patiala

### Table-2: Branch

**BRANCH CITY**

**BRANCH NAME**

ABC Patiala

DEF Ludhiana

GHI Jalandhar

### Table-3: Account

**BALANCE**

**BRANCH NAME**

**ACCOUNT NUMBER**

1111 ABC 50000

1112 DEF 10000

1113 GHI 9000

1114 ABC 7000

### Table-4: Loan

**AMOUNT**

**BRANCH NAME**

**LOAN NUMBER**

L33 ABC 10000

L35 DEF 15000

L49 GHI 9000

L98 DEF 65000

**Table-5: Borrower**

**LOAN NUMBER**

**CUSTOMER NAME**

Saurabh L33

Mehak L49

Ria L98

### Table-6: Depositor

**ACCOUNT NUMBER**

**CUSTOMER NAME**

Saurabh 1111

Mehak 1113

Sumiti 1114

**Queries-1:** Find the loan number, branch, amount of loans of greater than or equal to 10000 amount.

{t| t ∈ loan ∧ t[amount]>=10000}

### Resulting relation:

**AMOUNT**

**BRANCH NAME**

**LOAN NUMBER**

L33 ABC 10000

L35 DEF 15000

L98 DEF 65000

In the above query, t[amount] is known as tupple variable.

**Queries-2:** Find the loan number for each loan of an amount greater or equal to 10000.

{t| ∃ s ∈ loan(t[loan number] = s[loan number]

∧ s[amount]>=10000)}

### Resulting relation:

**LOAN NUMBER**

L33

L35

`````L98

**Queries-3:** Find the names of all customers who have a loan and an account at the bank.

{t | ∃ s ∈ borrower( t[customer-name] = s[customer-name])

∧ ∃ u ∈ depositor( t[customer-name] = u[customer-name])}

### Resulting relation:

**CUSTOMER NAME**

Saurabh

Mehak

**Queries-4:** Find the names of all customers having a loan at the “ABC” branch.

{t | ∃ s ∈ borrower(t[customer-name] = s[customer-name]

∧ ∃ u ∈ loan(u[branch-name] = “ABC” ∧ u[loan-number] = s[loan-number]))} Resulting relation:

**CUSTOMER NAME**

Saurabh

## Domain Relational Calculus(DRC)

**Domain Relational Calculus** is a non-procedural query language equivalent in power to Tuple Relational Calculus. Domain Relational Calculus provides only the description of the query but it does not provide the methods to solve it. In Domain Relational Calculus, a query is expressed as,

{ < x1, x2, x3, ..., xn > | P (x1, x2, x3, ..., xn ) }

where, < x1, x2, x3, …, xn > represents resulting domains variables and P (x1, x2, x3, …, xn ) represents the condition or formula equivalent to the Predicate calculus.

### Predicate Calculus Formula:

* 1. Set of all comparison operators
  2. Set of connectives like and, or, not
  3. Set of quantifiers

### Example:

**Table-1: Customer**

**CITY**

**STREET**

**CUSTOMER NAME**

Debomit Kadamtala Alipurduar

Sayantan Udaypur Balurghat

Soumya Nutanchati Bankura

Ritu Juhu Mumbai

### Table-2: Loan

**AMOUNT**

**BRANCH NAME**

**LOAN NUMBER**

L01 Main 200

L03 Main 150

L10 Sub 90

L08 Main 60

**Table-3: Borrower**

**LOAN NUMBER**

**CUSTOMER NAME**

Ritu L01

Debomit L08

Soumya L03

**Query-1:** Find the loan number, branch, amount of loans of greater than or equal to 100 amount.

{≺l, b, a≻ | ≺l, b, a≻ ∈ loan ∧ (a ≥ 100)}

### Resulting relation:

**AMOUNT**

**BRANCH NAME**

**LOAN NUMBER**

L01 Main 200

L03 Main 150

**Query-2:** Find the loan number for each loan of an amount greater or equal to 150.

{≺l≻ | ∃ b, a (≺l, b, a≻ ∈ loan ∧ (a ≥ 150)}

### Resulting relation:

**LOAN NUMBER**

L01

L03

**Query-3:** Find the names of all customers having a loan at the “Main” branch and find the loan amount .

{≺c, a≻ | ∃ l (≺c, l≻ ∈ borrower ∧ ∃ b (≺l, b, a≻ ∈ loan ∧ (b = “Main”)))}

### Resulting relation:

**AMOUNT**

**CUSTOMER NAME**

Ritu 200

Debomit 60

Soumya 150

**Note:** The domain variables those will be in resulting relation must appear before | within ≺ and

≻ and all the domain variables must appear in which order they are in original relation or table.

## Difference between Relational Algebra and Relational Calculus:

**RELATIONAL CALCULUS**

**RELATIONAL ALGEBRA**

**S.NO**

1. It is a Procedural language.

While Relational Calculus is Declarative language.

Relational Algebra means how to obtain

2.

the result.

While Relational Calculus means what result we have to obtain.

In Relational Algebra, The order is

3. specified in which the operations have to be performed.

While in Relational Calculus, The order is not specified.

Relational Algebra is independent on

4.

domain.

While Relation Calculus can be a domain dependent.

Relational Algebra is nearer to a

5.

programming language.

While Relational Calculus is not nearer to programming language.

# Data Definition language

## Introduction to DDL

* DDL stands for **Data Definition Language.**
* It is a language used for defining and modifying the data and its structure.
* It is used to build and modify the structure of your tables and other objects in the database.

### DDL commands are as follows:

1. CREATE
2. DROP
3. ALTER
4. RENAME
5. TRUNCATE
   * These commands can be used to add, remove or modify tables within a database.
   * DDL has pre-defined syntax for describing the data.

## CREATE COMMAND

* + **CREATE command** is used for creating objects in the database.
  + It creates a new table.

### Syntax:

CREATE TABLE <table\_name> ( column\_name1 datatype,

column\_name2 datatype,

.

.

.

column\_name\_n datatype

);

**Example:** Create command CREATE TABLE employee

(

empid INT, ename CHAR, age INT,

city CHAR(25), phone\_no VARCHAR(20)

);

## DROP COMMAND

* + **DROP command** allows to remove entire database objects from the database.
  + It removes entire data structure from the database.
  + It deletes a table, index or view.

### Syntax:

DROP TABLE <table\_name>; OR

DROP DATABASE <database\_name>;

**Example:** DROP Command DROP TABLE employee; OR

DROP DATABASE employees;

If you want to remove individual records, then use DELETE command of the DML statement.

## ALTER COMMAND

* + An **ALTER command** allows to alter or modify the structure of the database.
  + It modifies an existing database object.
  + Using this command, you can add additional column, drop existing column and even change the data type of columns.

### Syntax:

ALTER TABLE <table\_name> ADD <column\_name datatype>;

OR

ALTER TABLE <table\_name>

CHANGE <old\_column\_name> <new\_column\_name>; OR

ALTER TABLE <table\_name> DROP COLUMN <column\_name>;

Example: Alter Command ALTER TABLE employee

ADD (address varchar2(50)); OR

ALTER TABLE employee CHANGE (phone\_no) (contact\_no); OR

ALTER TABLE employee

DROP COLUMN age;

To view the changed structure of table, use 'DESCRIBE' command.

### For example:

DESCRIBE TABLE employee;

## RENAME COMMAND

* + **RENAME command** is used to rename an object.
  + It renames a database table.

### Syntax:

RENAME TABLE <old\_name> TO <new\_name>;

### Example:

RENAME TABLE emp TO employee;

## TRUNCATE COMMAND

* + **TRUNCATE command** is used to delete all the rows from the table permanently.
  + It removes all the records from a table, including all spaces allocated for the records.
  + This command is same as DELETE command, but TRUNCATE command does not generate any rollback data.

### Syntax:

TRUNCATE TABLE <table\_name>;

### Example:

TRUNCATE TABLE employee;

# Operators: Select, Project, Join, Rename etc

## SELECT (σ)

The SELECT operation is used for selecting a subset of the tuples according to a given selection condition. Sigma(σ)Symbol denotes it. It is used as an expression to choose tuples which meet the selection condition. Select operation selects tuples that satisfy a given predicate.

σp(r)

σ is the predicate

r stands for relation which is the name of the table p is prepositional logic

### Example 1

σ topic = "Database" (Tutorials)

Output - Selects tuples from Tutorials where topic = 'Database'.

### Example 2

σ topic = "Database" and author = "Edunet"( Tutorials)

Output - Selects tuples from Tutorials where the topic is 'Database' and 'author' is Edunet.

### Example 3

σ sales > 50000 (Customers)

Output - Selects tuples from Customers where sales is greater than 50000

## Projection(π)

The projection eliminates all attributes of the input relation but those mentioned in the projection list. The projection method defines a relation that contains a vertical subset of Relation.

This helps to extract the values of specified attributes to eliminate duplicate values. (pi) The symbol used to choose attributes from a relation. This operation helps you to keep specific columns from a relation and discards the other columns.

### Example of Projection:

Consider the following table

|  |  |  |
| --- | --- | --- |
| **CustomerID** | **CustomerName** | **Status** |
| 1 | Google | Active |
| 2 | Amazon | Active |
| 3 | Apple | Inactive |
| 4 | Alibaba | Active |

Here, the projection of CustomerName and status will give Π CustomerName, Status (Customers)

**CustomerName Status**

Google Active

Amazon Active

Apple Inactive

Alibaba Active

## JOIN

Join operation is essentially a cartesian product followed by a selection criterion. Join operation denoted by ⋈.

JOIN operation also allows joining variously related tuples from different relations.

### Types of JOIN

Various forms of join operation are:

1. Inner Joins:
   * Theta join
   * EQUI join
   * Natural join
2. Outer join:
   * Left Outer Join
   * Right Outer Join
   * Full Outer Join

### Inner Join:

In an inner join, only those tuples that satisfy the matching criteria are included, while the rest are excluded. Let's study various types of Inner Joins:

### Theta Join:

The general case of JOIN operation is called a Theta join. It is denoted by symbol **θ Example:** A ⋈θ B

Theta join can use any conditions in the selection criteria.

### For example:

A ⋈ A.column 2 > B.column 2 (B)



**A** ⋈ **A.column 2 > B.column 2 (B)**

### column 1 column 2

1 2

**EQUI join:**

When a theta join uses only equivalence condition, it becomes a equi join.

### For example:

A ⋈ A.column 2 = B.column 2 (B)



**A** ⋈ **A.column 2 = B.column 2 (B)**

### column 1 column 2

1 1

EQUI join is the most difficult operations to implement efficiently in an RDBMS and one reason why RDBMS have essential performance problems.

### NATURAL JOIN (⋈)

Natural join can only be performed if there is a common attribute (column) between the relations. The name and type of the attribute must be same.

Example

Consider the following two tables

**C**

### Num Square

2 4

3 9

**D**

### Num Cube

2 8

3 27

C ⋈ D



|  |  |  |
| --- | --- | --- |
| **C** ⋈ **D** |  | |
| **Num** | **Square** | **Cube** |
| 2 | 4 | 4 |
| 3 | 9 | 27 |

### OUTER JOIN

In an outer join, along with tuples that satisfy the matching criteria, we also include some or all tuples that do not match the criteria.

### Left Outer Join(A B)

In the left outer join, operation allows keeping all tuple in the left relation. However, if there is no matching tuple is found in right relation, then the attributes of right relation in the join result are filled with null values.



Image 26- Left Outer Join

Reference- https://[www.guru99.com/relational-algebra-dbms.html](http://www.guru99.com/relational-algebra-dbms.html)

Consider the following 2 Tables

|  |  |
| --- | --- |
| **A** |  |
| **Num** | **Square** |
| 2 | 4 |
| 3 | 9 |
| 4 | 16 |
|  |  |
| **B** |  |
| **Num** | **Cube** |
| 2 | 8 |
| 3 | 18 |
| 5 | 75 |

A  B



|  |  |  |
| --- | --- | --- |
| **A** ⋈ **B** |  | |
| **Num** | **Square** | **Cube** |
| 2 | 4 | 4 |
| 3 | 9 | 9 |
| 4 | 16 | - |

### Right Outer Join: ( A B )

In the right outer join, operation allows keeping all tuple in the right relation. However, if there is no matching tuple is found in the left relation, then the attributes of the left relation in the join result are filled with null values.

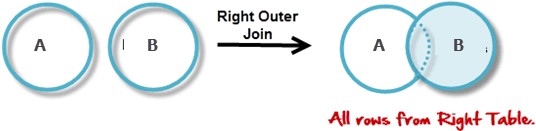


Image 27- Right Outer Join

Reference- https://[www.guru99.com/relational-algebra-dbms.html](http://www.guru99.com/relational-algebra-dbms.html)

A  B



**A** ⋈ **B**

### Num Cube Square

2

8

4

|  |  |  |
| --- | --- | --- |
| 3 | 18 | 9 |
| 5 | 75 | - |

**Full Outer Join: ( A B)**

In a full outer join, all tuples from both relations are included in the result, irrespective of the matching condition.

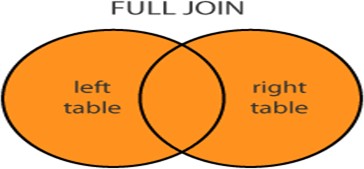


Image 28- Full Outer Join

A B



|  |  |  |
| --- | --- | --- |
| **A** ⋈ **B** |  | |
| **Num** | **Cube** | **Square** |
| 2 | 4 | 8 |
| 3 | 9 | 18 |

|  |  |  |
| --- | --- | --- |
| 4 | 16 | - |
| 5 | - | 75 |

## Rename(ρ)

The results of relational algebra are also relations but without any name. The rename operation allows us to rename the output relation. 'rename' operation is denoted with small Greek letter rho *ρ*.

Notation − *ρ* x (E)

Where the result of expression E is saved with the name of x.

**Example:** ρ(RelationNew, RelationOld)