**Unit-1**

**2ND YEAR – 4TH SEM B.COM(CA)**

**Course 4F: Database Management System**

**Overview of Database Management System**

Introduction, Data and Information, Database, Database Management System, Objectives of DBMS, Evolution of Database Management System, Classification of Database Management System.

**Unit-2**

**File-Based System**

File Based System. Drawbacks of File-Based System, DBMS Approach, Advantage of DBMS, Data Models, Components of Database System, Database Architecture, DBMS Vendors and their products.

**Unit-3**

**Entity-Relationship Model:**

Introduction, The Building Blocks of an Entity-Relationship, Classification of Entity Set, Attribute Classification, Relationship Degree, Relationship Classification, Generalization and Specialization, Aggregation and Composition, CODD's Rules, Relational Data Model. Concept of Relational Integrity.

**Unit-4**

**Structured Query Language**

Introduction, History of SQL Standards, Commands in SQL, Data types in SQL, Data Definition Language (DDL),Selection Operation Projection Operation, Aggregate Functions, Data Manipulation Language, Table Modification, Table Truncation, Imposition of Constraints, Set Operations.

**Unit-5**

**PL/SQL:**

Introduction, Structure of PL/SQL, PL/SQL Language Elements, Data Types, Control Structure, Steps to Create a PL/SQL Program, Iterative Control Cursors, Steps to Create a Cursor, Procedure, Functions, Packages, Exceptions Handling, Database Triggers, Types of triggers.

## Unit-1

**Overview of Database Management System**

Introduction, Data and Information, Database, Database Management System, Objectives of DBMS, Evolution of Database Management System, Classification of Database Management System.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

## Data:-

Data is nothing but a collection of some raw facts. the raw indicates that the data is not yet been processed to reveal its meaning. Data is represented with the help of characters such as alphabets ( A- Z , a – z ), digits ( 0 -9

) or special characters ( + , - , \* , / , > , < = ) etc.

**Information :-** The processed data is called an Information . Which has some meaningful values for the receiver. Information is the Processed data on which decisions and actions are based.

**Characteristics of information: -**

1. **Timely: -** Information should be available when required. 2.. **Accuracy: -** Information should be accurate

1. **Completeness: -** Information should be complete

## Data Vs Information:

|  |  |
| --- | --- |
| **DATA** | **INFORMATION** |
| 1. Collection of raw facts 2. in the form of digits, characters, symbols, audio, video etc. 3. data can be represented in the form of table and tree etc. 4. data is not having any meaning. 5)data is not depend on information. 6)data is not help in decision making. 7)it is difficult to understand.   8)data must be processed to understand. | 1. Processed data (or) the meaning full data 2. in the form of ideas. 3. information can be represented in language, ideas & thoughts based on given data. 4. information have meaning. 5)information is always depend on data.   6) information is helped in decision making. 7)it is easy to understand.  8) it already is in understandable form. |

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# Database: -

An organized collection of logically related data without having any redundancy (duplication) stored in one place and accessed by multiple users is called a database.

Databases are used for storing, maintaining and accessing any sort of data. They collect information on people, places or things. That information is gathered in one place so that it can be observed and analysed. Databases can be thought of as an organized collection of information.

. The data are stored in such a process that they are independent of the programs of people using the data. A Common and controlled approach is used in adding new data and modifying and retrieving existing data within the database.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# Database Management System:-

* Database management system is a commercial software which is used to manage the database. For example: [MySQL,](https://www.javatpoint.com/mysql-tutorial) [Oracle,](https://www.javatpoint.com/oracle-tutorial) etc are a very popular commercial database which is used in different applications.
* DBMS provides an interface to perform various operations like database creation, storing data in it, updating data, creating a table in the database and a lot more.
* It provides protection and security to the database. In the case of multiple users, it also maintains data consistency.

## Advantages of DBMS

* **Controls database redundancy:** It can control data redundancy because it stores all the data in one single database file and that recorded data is placed in the database.
* **Data sharing:** In DBMS, the authorized users of an organization can share the data among multiple users.
* **Easily Maintenance:** It can be easily maintainable due to the centralized nature of the database system.
* **Reduce time:** It reduces development time and maintenance need.
* **Backup:** It provides backup and recovery subsystems which create automatic backup of data from [hardware](https://www.javatpoint.com/hardware) and [software](https://www.javatpoint.com/software) failures and restores the data if required.
* **multiple user interface:** It provides different types of user interfaces like graphical user interfaces, application program interfaces

## Disadvantages of DBMS

* **Cost of Hardware and Software:** It requires a high speed of data processor and large memory size to run DBMS software.
* **Size:** It occupies a large space of disks and large memory to run them efficiently.
* **Complexity:** Database system creates additional complexity and requirements.
* **Higher impact of failure:** Failure is highly impacted the database because in most of the organization, all the data stored in a single database and if the database is damaged due to electric failure or database corruption then the data may be lost forever.

**Objectives of DBMS**

The main objectives of database management system are

* 1. Data availability,
  2. Data integrity,
  3. Data security, and
  4. Data independence
  5. To Reduce Duplication

1. **Data Availability**: Data availability refers to the fact. That the data are creates available to the wide variety of users can access the date easily in a meaningful format
2. **Data Integrity:** Data integrity refers to the correctness of the data in the database. In other words, the data available in the database is a reliable data.
3. **Data Security**: Data security refers to the fact that only authorized users can access the data. Data security can be enforced by passwords. If two separate users are accessing a particular at the same time. The DBMS must not allow them to make conflicting changes.
4. **Data Independence:** DBMS allows the user to store, update, and retrieve data in efficient manner. DBMS provides an "abstract view of how the data is stored in the database.
5. **To Reduce Duplication**

You see that when we fill the data, there is the chance of duplicacy. It should the objective of DBMS to reduce this duplicacy. Recent DBMS will automatically check the duplicate record and give warning to you for not recording. **6. Mass Storage**

DBMS can store a lot of data in it. So for all the big firms, DBMS is really ideal technology to use. It can store thousands of records in it and one can fetch all that data whenever it is needed.

**7. Multiple Users Access**

No one handles the whole database alone. There are lots of users who are able to access database. So this situation may happen that two or more users are accessing database. They can change whatever they want, at that time DBMS makes it sure that they can work concurrently.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# Evolution of Database Management System

The chronological order of the development of DBMS is as follows 1. Flat Files (1960s-1980s)

2. Hierarchical (1970s-1990s) 3. Network (1970s- 1990s) 4. Relational (1980s-present)

1. Object-Oriented (1990s- present)
2. Object-Relational (1990s-present)
3. Web enabled (1990s-present)
4. **Flat Files (1970s-1990s) :** Flat files database is a database that stores information in a single file or table. In text file, every line contains one record where fields either have fixed length or they are separated by commas, whitespaces, tabs or any records and they cannot contain multiple tables as well.
5. **Hierarchical (1970s-1990s) :** As the name indicates, hierarchical database contains data in a hierarchically-arranged data. More Perceptively it can parent can have many children but one child can only have one parent i. e.,; one-to-many relationship. Its hierarchical structure contains levels or segments which are equivalent to the file system’s record type. All attributes of a specific record are listed under the entity type.
6. **Network database (1970s -1990s) :** The inventor of network model is Charles Bachmann. Unlike hierarchical database model, network database allows multiple parent and child relationships i. e., it maintains many-to many relationship.
7. **Realationl database (1980s-present) :** Relationship database model was proposed by E. F. Codd. After the hierarchical and network model the birth of this model was huge step ahead. It allows the entities to be related through a common attribute. In the table there are alternative keys. This property makes this model extremely flexible.
8. **Object – oriented database (1990s -present) :** Object-Oriented database management system is that database system in which the data or information is presented in the form of objects, much like in object-Oriented programming language like JAVA or C++.
9. **Object- relationship database (1990s-present) : A**n object relationship database management system displays a modified object- oriented. user display over the already implemented relationship database management system. The basic working of this database management system is that is translated the useful data into organized tables distributed in rows and columns, and from then onwards, it manages data the same way done in s relational database system. Similarly, when the data is to be accessed by the user, it is again translated from processed to complex for m.
10. **Web enabled (1990s-present)** : In web enabled Data Base the data access through the internet with different places. Different web applications designed for data retrieve from the internet. Web enabled data base is a internet based system.

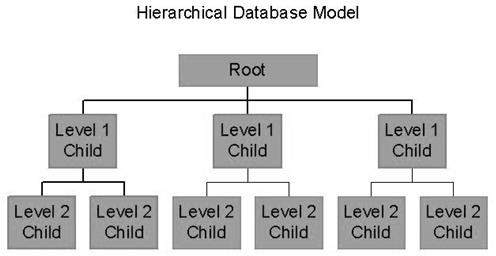
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

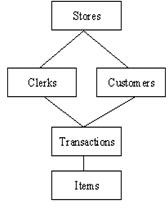
# Classifications of DBMS:-

## Based on the data model:

**Relational database** – This is the most popular data model used in industries. It is based on the SQL. They are table oriented which means data is stored in different access control tables, each has the key field whose task is to identify each row. The tables or the files with the data are called as relations that help in designating the row or record, and columns are referred to attributes or fields. Few examples are MYSQL (Oracle, open source), Oracle database (Oracle), Microsoft SQL server (Microsoft) and DB2(IBM).

**Hierarchical database** – In this, the information about the groups of parent or child relationships is present in the records which is similar to the structure of a tree. Here the data follows a series of records, set of values attached to it. They are used in industry on mainframe platforms. Examples are IMS(IBM), Windows registry (Microsoft).



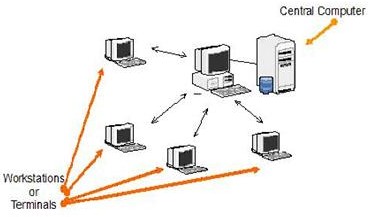
**Network database** – Mainly used on a large digital computers. If there are more connections, then this database is efficient. They are similar to hierarchical database, they look like a cobweb or interconnected network of records.

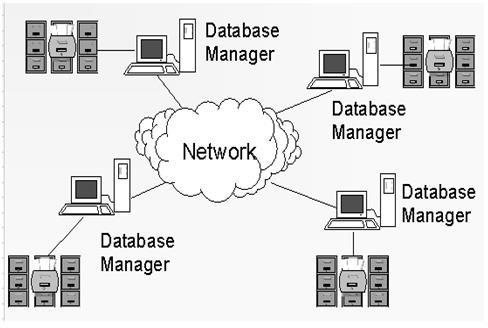
## Based on the number of users

**Single user** – As the name itself indicates it can support only one user at a time. It is mostly used with the personal computer on which the data resides accessible to a single person. The user may design, maintain and write the database programs.

**Multiple users** – It supports multiple users concurrently. Data can be both integrated and shared,a database should be integrated when the same information is not need be recorded in two places. For example a student in the college should have the database containing his information. It must be accessible to all the departments related to him. For example the library department and the fee section department should have information about student’s database. So in such case, we can integrate and even though database resides in only one place both the departments will have the access to it.

# Based on the sites over which network is distributed

**Centralized database system** – The DBMS and database are stored at the single site that is used by several other systems too. We can simply say that data here is maintained on the centralized server.

**Distributed database system** – In this data and the DBMS software are distributed over several sites but connected to the single computer.

Further they are classified as

* 1. **Homogeneous DBMS** – They use same software but from the multiple sites. Data exchange between the sites can be handled easily. For example, library information systems by the same vendor ,such as Geac Computer corporation, use the same DBMS software that allows the exchanges between various Geac library sites.

2.**heterogeneous DBMS** – They use different DBMS software for different sites but there is a additional software that helps the exchange of the data between the sites.

# Based on the cost

**Low cost DBMS** – The cost of these systems vary from $100 to $3000.

**Medium cost DBMS** – Cost varies from $10000 to $100000.

**High cost DBMS** – Cost pf these systems are usually more than $100000.

# Based on the access

This classification simply based on the access to data in the database systems

**Sequential access** – One after the other.

**Direct access**

**Unit-2**

**File-Based System**

File Based System. Drawbacks of File-Based System, DBMS Approach, Advantage of DBMS, Data Models, Components of Database System, Database Architecture, DBMS Vendors and their products.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**File based system**

The systems that are used to organize and maintain data files are known as file based data systems. These file systems are used to handle a single or multiple files and are not very efficient.

**Functionalities**

The functionalities of a File-based Data Management System are as follows −

* A file based system helps in basic data management for any user.
* The data stored in the file based system should remain consistent. Any transactions done in the file based system should not alter the consistency property.
* The file based system should not allow any illegal or potentially hazardous operations to occur on the data.
* The file based system should allow concurrent access by different processes and this should be carefully coordinated.
* The file based system should make sure that the data is uniformly structured and stored so it is easier to access it.

**Advantages of File Based System**

* The file Based system is not complicated and is simpler to use.
* Because of the above point, this system is quite inexpensive.
* Because the file based system is simple and cheap, it is normally suitable for home users and owners of small businesses.
* Since the file based system is used by smaller organisations or individual users, it stores comparatively lesser amount of data. Hence, the data can be accessed faster and more easily.

## Disadvantage of File-oriented system:(Drawbacks)

### 1. Data Redundancy:

* It is possible that the same information may be duplicated in different files.this leads to data redundancy results in memory wastage.

### 2. Data Inconsistency:

* Because of data redundancy,it is possible that data may not be in consistent state.

### 3. Difficulty in Accessing Data:

* Accessing data is not convenient and efficient in file processing system.

### 4. Limited Data Sharing:

* Data are scattered in various files.also different files may have different formats and these files may be stored in different folders may be of different departments.
* So, due to this data isolation, it is difficult to share data among different applications.

### 5. Integrity Problems:

* Data integrity means that the data contained in the database in both correct and consistent.for this purpose the data stored in database must satisfy correct and constraints.

### 6. Atomicity Problems:

* Any operation on database must be atomic.
* this means, it must happen in its **entirely** or not at all.

### 7. Concurrent Access Anomalies:

* Multiple users are allowed to access data simultaneously.this is for the sake of better performance and faster response.

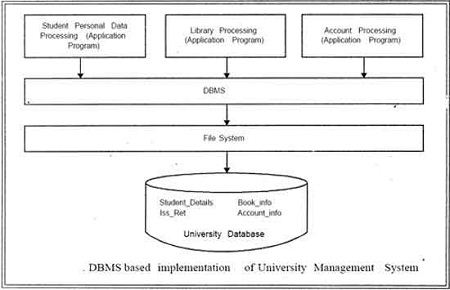
### 8. Security Problems:

* Database should be accessible to users in limited way.
* Each user should be allowed to access data concerning his requirements only.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# Database Approach

In order to remove all limitations of the File Based Approach, a new approach was required that must be more effective known as [Database](https://ecomputernotes.com/fundamental/what-is-a-database/advantages-and-disadvantages-of-dbms) approach.Database approach is a single centralized database is used for all departments of oraganization.



**ADVANTAGES OF DATABASE APPROACH:**

The database approach emphasizes the integration & sharing of data throughout the organization. The database approach offers a number of potential advantages compared to the traditional file processing system. Some advantages of the database approach are as follows:

1. **Minimal Database Redundancy:** In traditional no database systems each department maintains its own files for handling its data processing applications. For examples: The university database might have two groups of users e.g. the personnel department & the accounts department. Most of the data is stored twice in the files of each department. This problem can be avoided by having a centralized database. For consistency, a database must store each logical data item: such as students’ names – in only one place in the database.
2. **Improved Data Consistency**: By eliminating controlling data redundancy, we greatly reduce the opportunities for inconsistency. Updating data values in greatly simplified when each value is stored in one place only. Finally, we avoid the wastage storage space that results from redundant data storage.
3. **Improved Data Sharing:** The data stored at a centralized location can easily be shared by existing applications. The same stored data can be used with the new applications also.
4. **Enforcement of standards:** The interchange of data between systems necessitates, standardization of the data representation, with the central control of the database, the database administrator can enforce standards in the representation of data. These standards will include data quality standards and uniform procedures for accessing, updating, and protecting data.
5. **Improved Data Integrity:** The database approach provides a number of tools and processes to improved data quality & integrity. Integrity implies the correctness and accuracy of data. Centralized control of the data enables the database administrator to define integrity constraints to ensure the accuracy and correctness of data stored in the database.
6. **Improved Data Accessibility:** With a relational database, end-users without programming experience can often retrieve and display data.
7. **Improved Data Security:** The Database Administrator (DBA) can be defined as security rules to check unauthorized access to data. Some users may be given the right to only retrieve data. Whereas others may be permitted to retrieve & edit the data. The Database Administrator can formulate different rules for each type of access (retrieve, modify, delete) to each piece of information in the database.
8. **Increased Productivity:** A major advantage of the database approach is that it greatly reduces the cost & time for developing new business applications.
9. **Reduce Program Maintenance:** Stored data must be changed frequently for a variety of reasons: new data item, types are added, data formats are changed and so or in a database, within limits, we can change the data without necessitating a change in other factors, as a result, program maintenance can be significantly reduced in a modern database environment.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

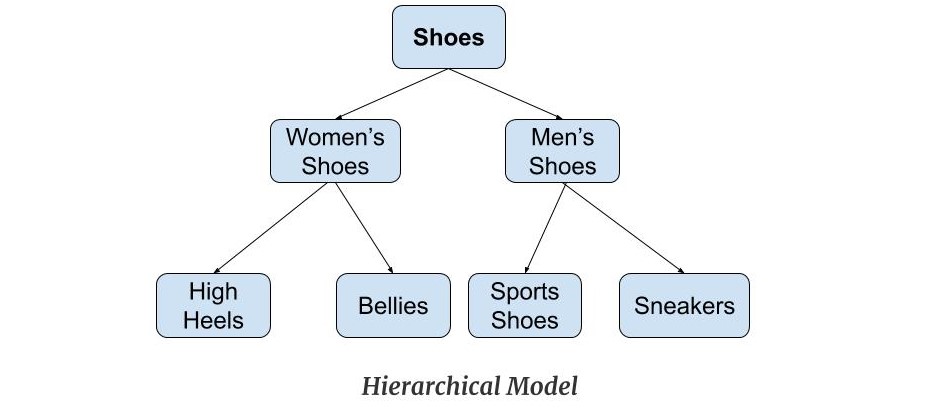
**What is Data Model in DBMS and what are its types? (or) various data models**

Data Model gives us an idea that how the final system will look like after its complete implementation. It defines the data elements and the relationships between the data elements. Data Models are used to show how data is stored, connected, accessed and updated in the database management system. Here, we use a set of symbols and text to represent the information so that members of the organisation can communicate and understand it. Though there are many data models being used nowadays but the Relational model is the most widely used model. Apart from the Relational model, there are many other types of data models about which we will study in details in this blog. Some of the Data Models in DBMS are:

1. Hierarchical Model
2. Network Model
3. Entity-Relationship Model
4. Relational Model
5. Object-Oriented Data Model etc

#### Hierarchical Model

Hierarchical Model was the first DBMS model. This model organises the data in the hierarchical tree structure. The hierarchy starts from the root which has root data and then it expands in the form of a tree adding child node to the parent node. This model easily represents some of the real-world relationships like food recipes, sitemap of a website etc. **Example:** We can represent the relationship between the shoes present on a shopping website in the following way:



**Advantages of Hierarchical Model**

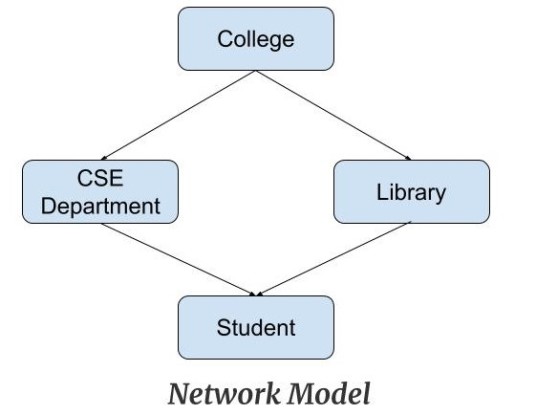
* It is very simple and fast to traverse through a tree-like structure.
* Any change in the parent node is automatically reflected in the child node so, the integrity of data is maintained.

**Disadvantages of Hierarchical Model**

* Complex relationships are not supported.
* As it does not support more than one parent of the child node so if we have some complex relationship where a child node needs to have two parent node then that can't be represented using this model.
* If a parent node is deleted then the child node is automatically deleted.

#### Network Model

This model is an extension of the hierarchical model. It was the most popular model before the relational model. This model is the same as the hierarchical model, the only difference is that a record can have more than one parent. It replaces the hierarchical tree with a graph. **Example:** In the example below we can see that node student has two parents i.e. CSE Department and Library. This was earlier not possible in the hierarchical model.

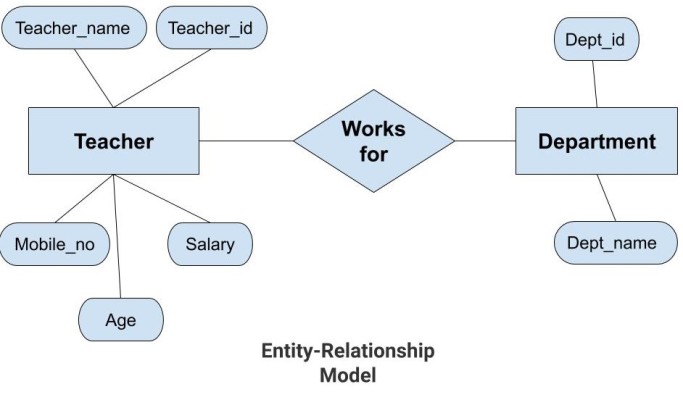


#### Entity-Relationship Model

Entity-Relationship Model or simply ER Model is a high-level data model diagram. In this model, we represent the real-world problem in the pictorial form to make it easy for the stakeholders to understand. It is also very easy for the developers to understand the system by just looking at the ER diagram. We use the ER diagram as a visual tool to represent an ER Model. ER diagram has the following three components:

* **Entities:** Entity is a real-world thing. It can be a person, place, or even a concept. Example: Teachers, Students, Course, Building, Department, etc are some of the entities of a School Management System.
* **Attributes:** An entity contains a real-world property called attribute. This is the characteristics of that attribute. Example: The entity teacher has the property like teacher id, salary, age, etc.
* **Relationship:** Relationship tells how two attributes are related. Example: Teacher works for a department.

**Example:**



In the above diagram, the entities are Teacher and Department. The attributes of **Teacher**entity are Teacher\_Name, Teacher\_id, Age, Salary, Mobile\_Number. The attributes of entity **Department**entity are Dept\_id, Dept\_name. The two entities are connected using the relationship. Here, each teacher works for a department.

**Advantages of ER Model**

* **Simple:** Conceptually ER Model is very easy to build. If we know the relationship between the attributes and the entities we can easily build the ER Diagram for the model.
* **Effective Communication Tool**: This model is used widely by the database designers for communicating their ideas.
* **Easy Conversion to any Model**: This model maps well to the relational model and can be easily converted relational model by converting the ER model to the table. This model can also be converted to any other model like network model, hierarchical model etc.

**Disadvatages of ER Model**

* **No industry standard for notation:** There is no industry standard for developing an ER model. So one developer might use notations which are not understood by other developers.
* **Hidden information:** Some information might be lost or hidden in the ER model. As it is a high-level view so there are chances that some details of information might be hidden.

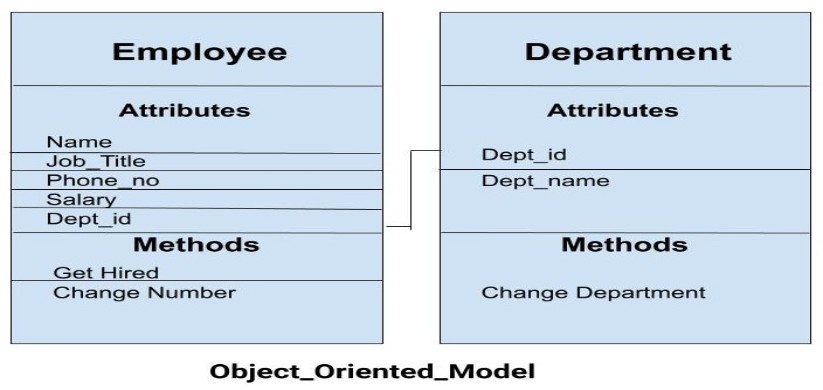
#### Relational Model

Relational Model is the most widely used model. In this model, the data is maintained in the form of a two-dimensional table. All the information is stored in the form of row and columns. The basic structure of a relational model is tables. So, the tables are also called relations in the relational model. **Example:**In this example, we have an Employee table.



#### Object-Oriented Data Model

The real-world problems are more closely represented through the object-oriented data model. In this model, both the data and relationship are present in a single structure known as an object. We can store audio, video, images, etc in the database which was not possible in the relational model(although you can store audio and video in relational database, it is adviced not to store in the relational database). In this model, two are more objects are connected through links. We use this link to relate one object to other objects. This can be understood by the example given below.



In the above example, we have two objects Employee and Department. All the data and relationships of each object are contained as a single unit. The attributes like Name, Job\_title of the employee and the methods which will be performed by that object are stored as a single object. The two objects are connected through a common attribute i.e the Department\_id and the communication between these two will be done with the help of this common id.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Components of a DBMS:**

Hardware, Software, Data, Database Access Language, Procedures and Users all together form the components of a DBMS. Let us discuss the components one by one clearly.

**Hardware**

The hardware is the actual computer system used for keeping and accessing the database. The conventional DBMS hardware consists of secondary storage devices such as hard disks. Databases run on the range of machines from micro computers to mainframes.

**Software**

Software is the actual DBMS between the physical database and the users of the system. All the requests from the user for accessing the database are handled by DBMS.

**Data** It is an important component of the database management system. The main task of DBMS is to process the data. Databases are used to store the data, retrieved, and updated to and from the databases.there are two types of data : 1. user data 2. meta data(data about data)

### Procedures

Procedures refer to general instructions to use a database management system. This includes procedures to setup and install a DBMS, To login and logout of DBMS software, to manage databases, to take backups, generating reports etc

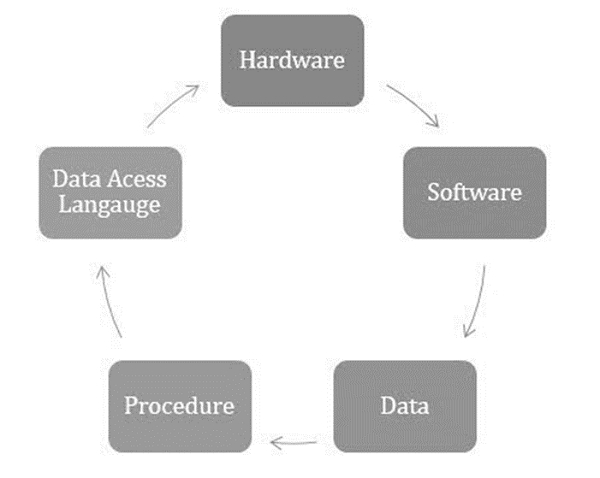
**Users(people)**

There are a number of users who can access or retrieve the data on demand using the application and the interfaces provided by the DBMS.

The users of the database can be classified into different groups −

* system administrator
* database administrator
* database designer
* system analyst & programmer
* end users

The components of DBMS are given below in pictorial form −



\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Database Architecture**

A **Database Architecture** is a representation of DBMS design. It helps to design, develop, implement, and maintain the database management system. A DBMS architecture allows dividing the database system into individual components that can be independently modified, changed, replaced, and altered. It also helps to understand the components of a database.

A [Database](https://www.guru99.com/introduction-to-database-sql.html) stores critical information and helps access data quickly and securely. Therefore, selecting the correct Architecture of DBMS helps in easy and efficient data management.

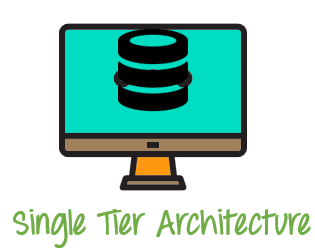
Types of DBMS Architecture

There are mainly three types of DBMS architecture:

* One Tier Architecture (Single Tier Architecture)
* Two Tier Architecture
* Three Tier Architecture

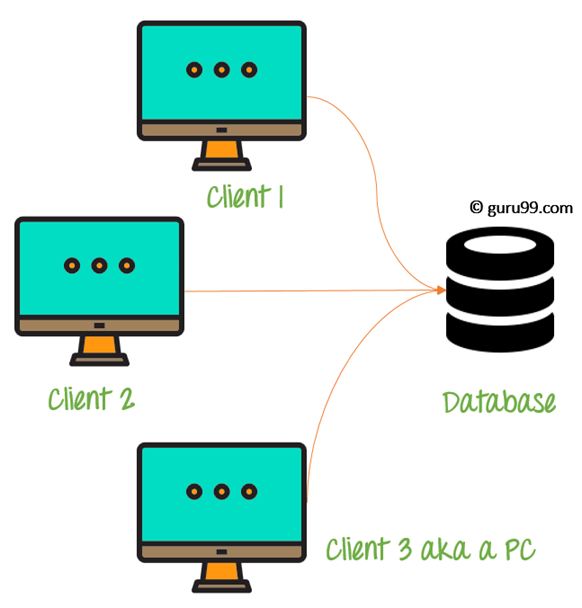
## 1-Tier Architecture

**1 Tier Architecture** in DBMS is the simplest architecture of Database in which the client, server, and Database all reside on the same machine. A simple one tier architecture example would be anytime you install a Database in your system and access it to practice SQL queries. But such architecture is rarely used in production.



## 2-Tier Architecture

A **2 Tier Architecture** in DBMS is a Database architecture where the presentation layer runs on a client (PC, Mobile, Tablet, etc.), and data is stored on a server called the second tier. Two tier architecture provides added security to the DBMS as it is not exposed to the end-user directly. It also provides direct and faster communication.



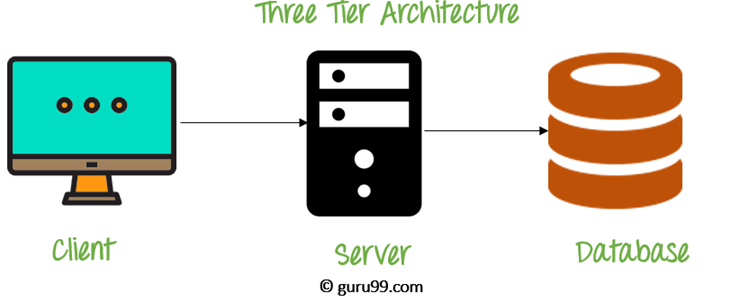
In the above 2 Tier client-server architecture of database management system, we can see that one server is connected with clients 1, 2, and 3.

## 3-Tier Architecture

A **3 Tier Architecture** in DBMS is the most popular client server architecture in DBMS in which the development and maintenance of functional processes, logic, data access, data storage, and user interface is done independently as separate modules. Three Tier architecture contains a presentation layer, an application layer, and a database server.

3-Tier database Architecture design is an extension of the 2-tier client-server architecture. A 3-tier architecture has the following layers:

1. Presentation layer (your PC, Tablet, Mobile, etc.)
2. Application layer (server)
3. Database Server



The Application layer resides between the user and the DBMS, which is responsible for communicating the user’s request to the DBMS system and send the response from the DBMS to the user. The application layer(business logic layer) also processes functional logic, constraint, and rules before passing data to the user or down to the DBMS.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**DBMS Vendors and their Products**

Some of the popular DBMS Vendors and their products are given below:

|  |  |
| --- | --- |
| **Vendors** | **Products** |
| Oracle Corporation | Oracle RDBMS, Sql Developer , MY SQL |
| IBM | SQL, DB2, Informix, & Informix Dynamic Servers |
| MicroSoft | Sql Server , Access |
| Altibase | Altibase |
| Amazon | Amazon RDS |

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**UNIT-III**

**Entity-Relationship Model**: Introduction, The Building Blocks of an Entity-Relationship, Classification of Entity Set, Attribute Classification, Relationship Degree, Relationship Classification, Generalization and Specialization, Aggregation and Composition, CODD’s Rules, Relational Data Model, Concept of Relational Integrity.

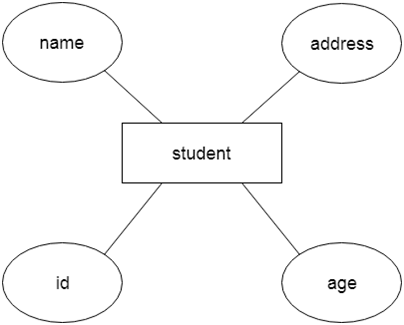
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Entity-Relationship Model**

# ER model

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

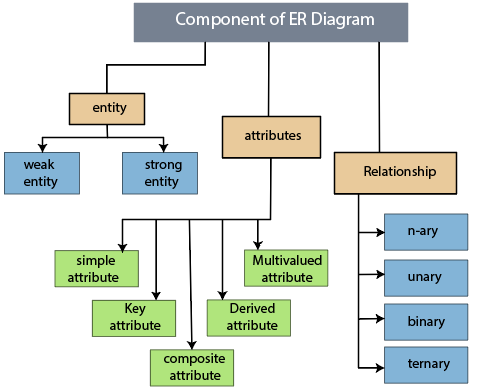
**For example,** Suppose we design a school database. In this database, the student will be an entity with attributes like address, name, id, age, etc. The address can be another entity with attributes like city, street name, pin code, etc and there will be a relationship between them.

  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

## Component (or) Building Blocks of ER Diagram

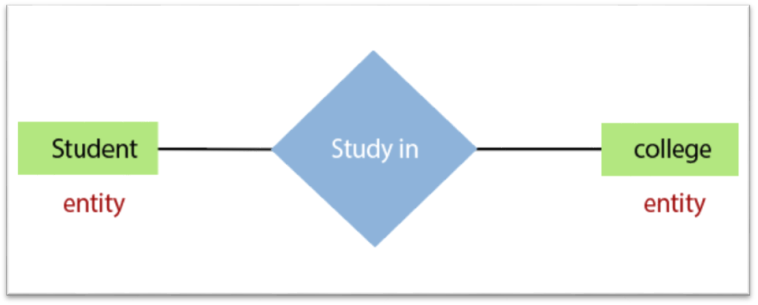
An ER Diagram consists of the following components:

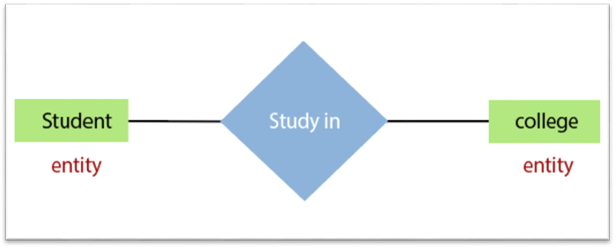
1. Entity
2. Attributes
3. Relationships



**1. Entity**

An entity may be an object, place, person, or an event which stores data in the database. In an entity-relationship diagram, an entity is represented by a rectangle.**Student, course, manager, employee, patient,** etc. are examples of an entity.

  
**Entity type:** An entity type is a collection or a set of entities having some common attributes. In a database, each entity type is described by a name and list of attributes.



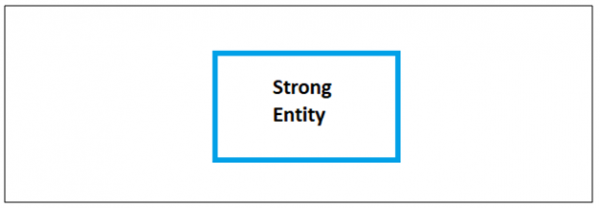
**Entity set:** It is a set (or collection) of entities of the same type which share the similar properties, or attributes.

**For example,**the group of people who are lecturers in a university can be de?ned as an entity set**lecturer**. Similarly, the entity set of**students** might represent the group of all students in the university.

### 6.png

**An entity can be characterized into two types:**

1. **Strong entity:** This type of entity has a primary key attribute which uniquely identifies each record in a table. In the ER diagram, a strong entity is usually represented by a single rectangle.

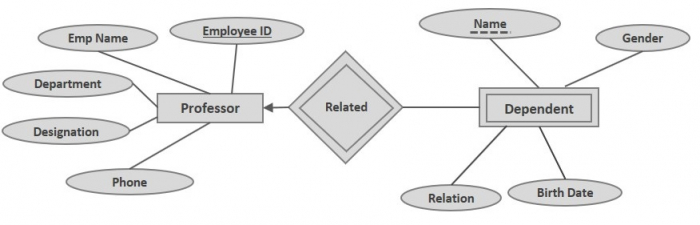


1. **Weak entity:**An entity does not have a primary key attribute and depends on another strong entity via foreign key attribute. In the ER diagram, a weak entity is usually represented by a double rectangle.

### weak_entity.png

**Example of Strong and Weak Entity**

The example of a strong and weak entity can be understood by the below figure



The Strong Entity is **Professor**, whereas **Dependent**is a Weak Entity.

**ID**is the primary key (represented with a line) and the Name in **Dependent**entity is called **Partial Key** (represented with a dotted line).

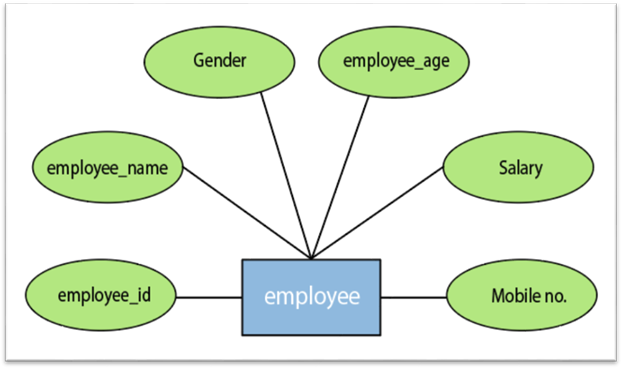
**2. Attributes:**An attribute in an Entity-Relationship Model describes the properties or characteristics of an entity. It is represented by an **oval or ellipse** shape in the ER diagram. Every oval shape represents one attribute and is directly connected to its entity which is in the rectangle in shape.

**For example**, **employee\_id, employe\_name, Gender, employee\_age, Salary, and Mobile no.**are the attributes which define entity type **Employee**.

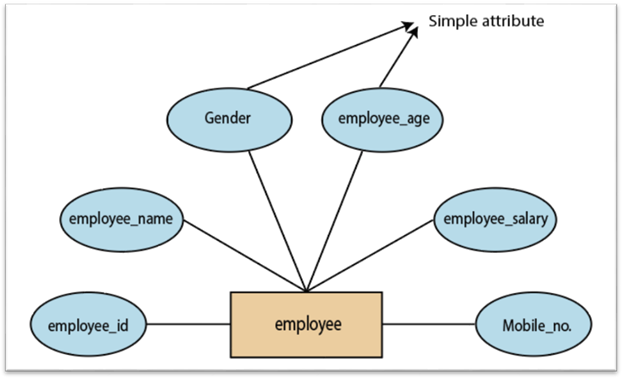
### 7.png

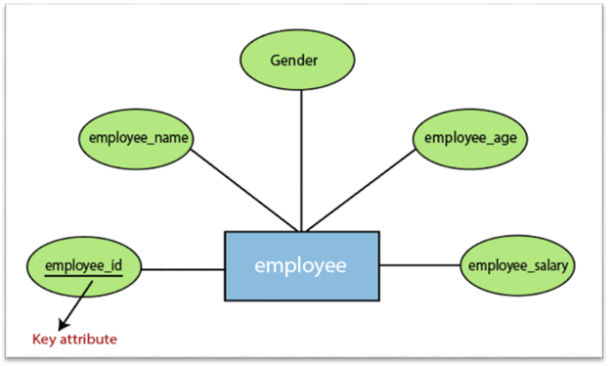
**In the ER model, an attribute can be characterized into the following types(TYPES OF ATTRIBUTES)**

1. **Simple attribute:** An attribute which contains an atomic value and cannot be divided further is called a simple attribute.**For example, Gender and Salary**of a person. It is also represented by an oval.

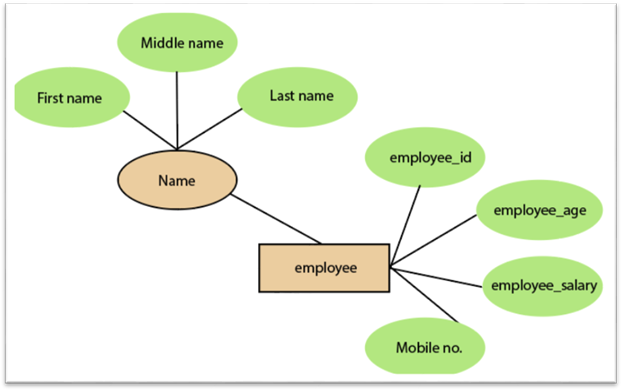


* **Key attribute:**An attribute which can uniquely identify an entity in an entity set is called a key attribute. It represents a [primary key](https://www.tutorialandexample.com/primary-key-in-dbms/) in the ER diagram. In an Entity-Relationship diagram, the key attribute is denoted by an oval with an underlying line**. For example**, **employee\_id** will be unique for each employee.

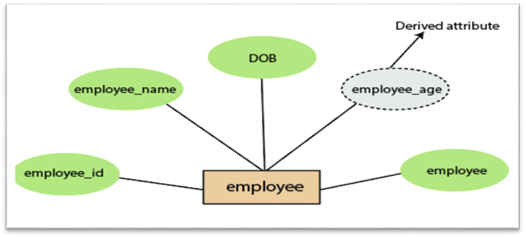




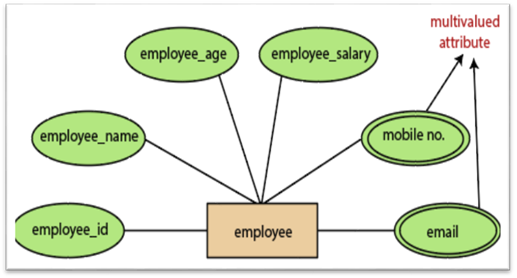
* **Composite attribute:**An attribute which is a combination of two or more simple attributes is called a composite attribute. In an Entity-Relationship diagram, it is represented by an ellipse, and that ellipse comprises of other ellipses.**For example, the** Name attribute of an employee entity type consists of **First name, Second name, and Last name.**



* **Derived attribute:**An attribute which can be derived from other attributes is called a derived attribute. In an entity-relationship diagram, these attributes are represented by a **dashed oval** shape. **For example,**employee\_age is a derived attribute as it changes over time and can be derived from another attribute **DOB** (Date of birth).



* **Multi-valued attribute:**An attribute which contains more than one value for a given entity. **For example,** an employee can have more than one mobile number and email address.



**3. Relationship**

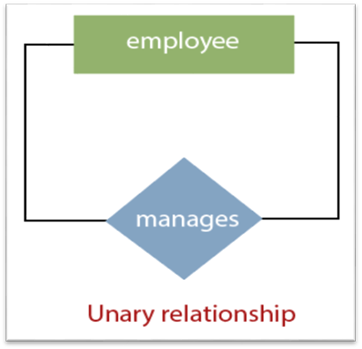
A relationship in Entity-Relationship Model is used to describe the relation between two or more entities. It is represented by a diamond shape in the ER diagram. **For example,** student **study in** college, employee **works in** a department. Here, ‘study in’ and ‘works in’ are the relationships.

**Degree of Relationship**

A relationship where a number of different entity set participate is called as degree of a relationship.

**Degree of relationship can be categorized into the following types:**

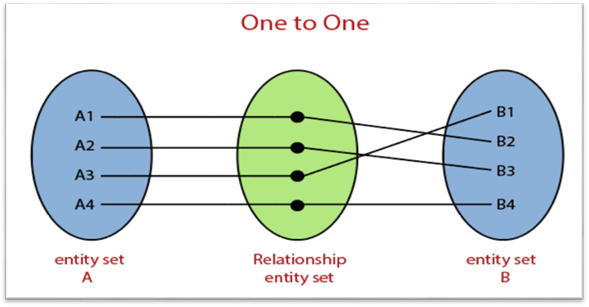
1. Unary Relationship
2. Binary Relationship
3. Ternary Relationship
4. n-ary Relationship
5. **Unary Relationship:** A relationship where a single entity set participates is called as a unary relationship. **For example,** In a company, an employee manages or supervises another employee.

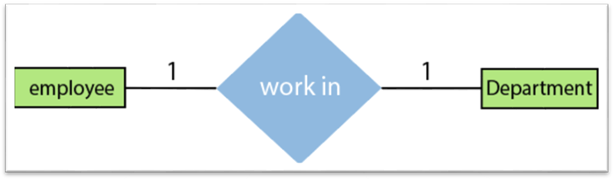


1. **Binary Relationship:** When two entity set participates in a relationship is called a binary relationship.

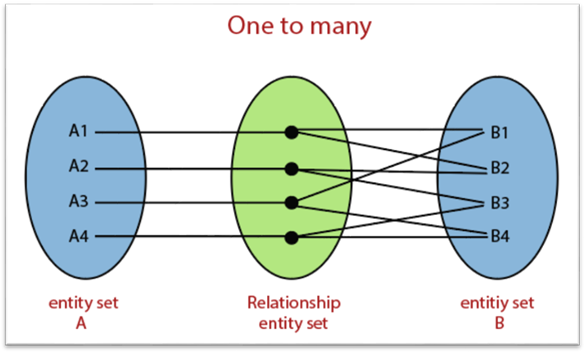
**It is further categorized into four types:**

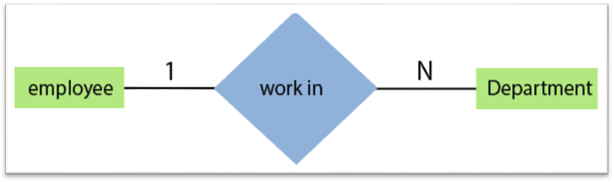
1. **One-to-One Relationship:** When one entity from an entity set A is associated with another entity of entity set B and vice versa. Such type of relationship is called one to one relationship.



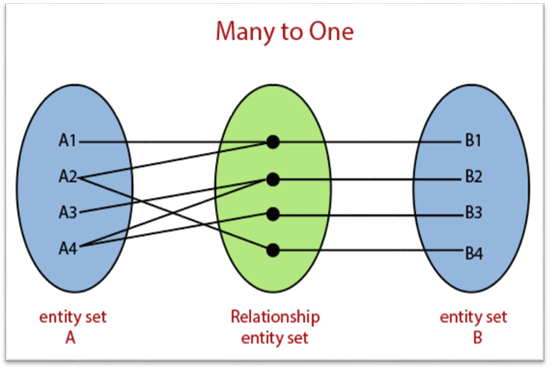


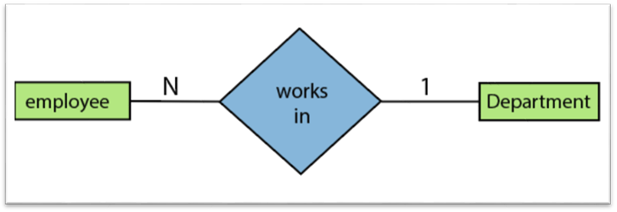
**2. One-to-Many Relationship:**When one entity from an entity set A is associated or linked with multiple entities of entity set B, then it is called a one-to-many relationship.



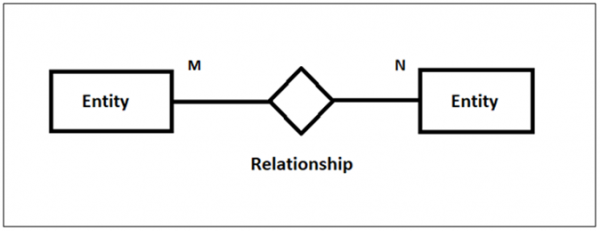


3. **Many-to-One Relationship:** When more than one entities from an entity set A is associated with one entity of entity set B, then it is called a many-to-one relationship.





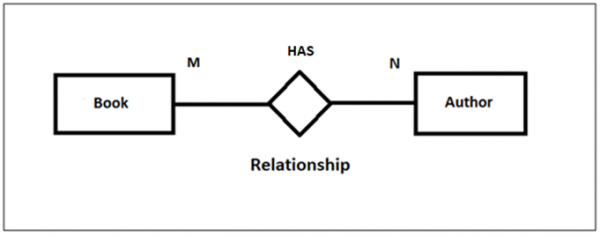
4. **Many-to-Many Relationship**: When more than one entity from an entity set A is associated with many entity of an entity set B. Such types of relationship are called a many-to-many relationship.



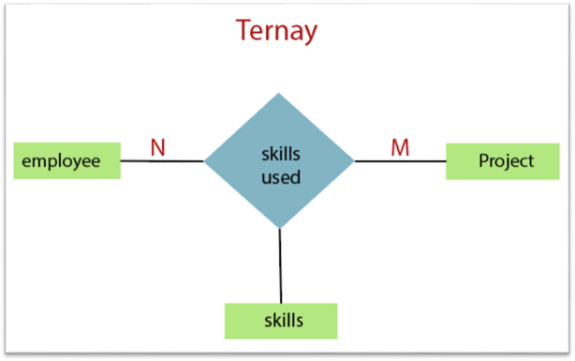
Let us see an example −

Many Authors can write a Book, whereas an Author has written more than one book.

Here, **Book**and **Author**are entities.



**c) Ternary Relationship:**When three entity set participates in a relationship, is called a ternary relationship.



**d) n-ary Relationship:**When more than three entity set involves in a relationship is called an n-ary relationship.

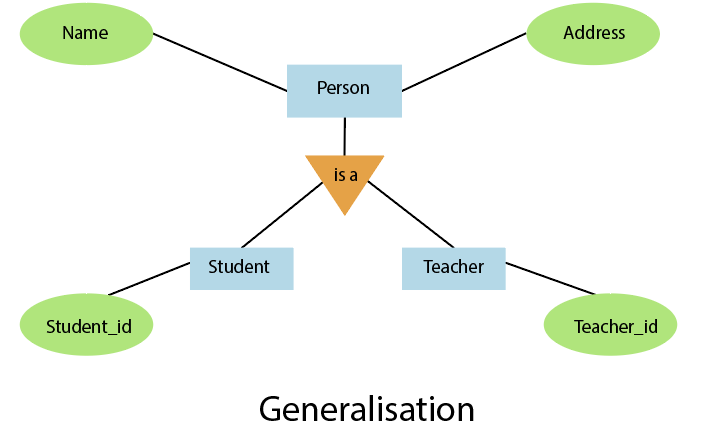
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# DBMS Generalization and Specialization

### DBMS Generalization

**Generalization** is a bottom-up approach in which the common attributes of two or more lower-level entities combines to form a new higher-level entity. In generalization, the generalized entity of higher level can also combine with entities of the lower-level to make further higher-level entity.  
It is like a superclass and subclass system, but the only difference is that it uses the bottom-up approach. It helps in reducing the schema size. It is always applied to the group of entities and result in the formation of a single entity.

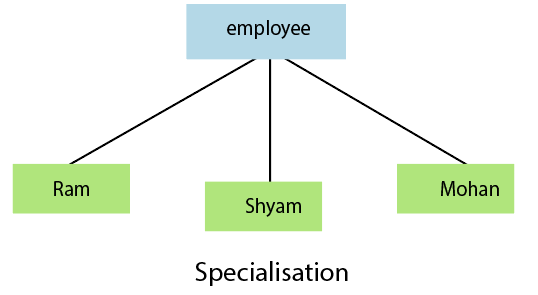
According to the below diagram, there are two entities, namely **teacher** and **student**. The teacher entity contains attributes such as **teacher\_id, name, and address** and student entity include **student\_id, name, and address**. Both entities can be combined to create a higher-level entity**person**. The address and name are common to both the entities. The teacher entity has its attribute teacher\_id, and the student has its attribute student\_id. The entities teacher and student are generalized further into the person entity.  
A lower-level entity is called a subclass, and the higher-level entity is called a superclass. So, the**person** entity is the superclass of two subclasses**teacher**and**student**.



### DBMS Specialization

It is opposite or inverse of generalization. A specialization is a top-down approach in which an entity of higher-level entity is broken down into two or more entities of lower level. In specialization, a higher-level entity set may not have any lower-level entity set. It is always applied to a single entity and results in the formation of multiple new entities. It increases the size of schema due to the increase in the number of entities.

In the below example, it can be seen that the **employee** is a high-level entity which is divided into three sub-entities (Ram, Shyam, and Mohan). The sub-entities are the names of the employees (relating to the high-level entity). Therefore, splitting a high-level entity into a low-level entity is called **specialization**.



\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# Aggregation and Composition

**Aggregation**

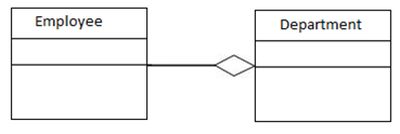
Aggregation is a special type of Association. Aggregation is "\*the\*" relationship among objects. We can say it is a direct association among the objects. In Aggregation, the direction specifies which object contains the other object. There are mutual dependencies among objects.

For example, departments and employees, a department has many employees but a single employee is not associated with multiple departments.

UML Representation of Aggregation (white diamond):

OOP2.jpg

The UML representation of the example above (relation between employee and department):



Here, the lives of both objects are independent of each other. That means that in this Association (Aggregation) the object has their own life cycle. Employees may exist without a department. Here, department can be called an owner object and the employee can be called a child object. The owner and child objects cannot belong to a different parent object.

**Composition**

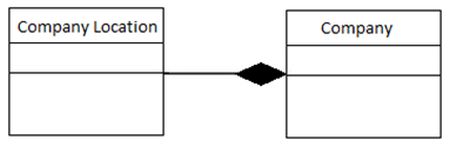
Composition is special type of Aggregation. It is a strong type of Aggregation. In this type of Aggregation the child object does not have their own life cycle. The child object's life depends on the parent's life cycle. Only the parent object has an independent life cycle. If we delete the parent object then the child object(s) will also be deleted. We can define the Composition as a "Part of" relationship.

For example, the company and company location, a single company has multiple locations. If we delete the company then all the company locations are automatically deleted. The company location does not have their independent life cycle, it depends on the company object's life (parent object).

UML Representation of Composition (black diamond):

OOP4.jpg

UML representation of the example above (relation between Company and Company Location):



Here, the lives of both objects are not independent. The life of the company location object can be determined by the life of the company object. The company object is responsible for creating and destroying company location objects.

# Relational Model(OR)Relational Data Model in DBMS

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 RDBMS 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 |  | 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.

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**CODD’S RULES:-**

Dr Edgar F. Codd, after his extensive research on the Relational Model of database systems, came up with twelve rules of his own, which according to him, a database must obey in order to be regarded as a true relational database.

These rules can be applied on any database system that manages stored data using only its relational capabilities. This is a foundation rule, which acts as a base for all the other rules.

**Rule 0:** Fundamental Rule

**Rule 1:** Information Rule

The data stored in a database, may it be user data or metadata, must be a value of some table cell. Everything in a database must be stored in a table format.

**Rule 2:** Guaranteed Access Rule

Every single data element (value) is guaranteed to be accessible logically with a combination of table-name, primary-key (row value), and attribute-name (column value). No other means, such as pointers, can be used to access data.

**Rule 3:** Systematic Treatment of NULL Values

The NULL values in a database must be given a systematic and uniform treatment. This is a very important rule because a NULL can be interpreted as one the following − data is missing, data is not known, or data is not applicable.

**Rule 4:** Active Online Catalog

The structure description of the entire database must be stored in an online catalog, known as **data dictionary**, which can be accessed by authorized users. Users can use the same query language to access the catalog which they use to access the database itself.

**Rule 5:** Comprehensive Data Sub-Language Rule

A database can only be accessed using a language having linear syntax that supports data definition, data manipulation, and transaction management operations. This language can be used directly or by means of some application. If the database allows access to data without any help of this language, then it is considered as a violation.

**Rule 6:** View Updating Rule

All the views of a database, which can theoretically be updated, must also be updatable by the system.

**Rule 7:** High-Level Insert, Update, and Delete Rule

A database must support high-level insertion, updation, and deletion. This must not be limited to a single row, that is, it must also support union, intersection and minus operations to yield sets of data records.

**Rule 8:** Physical Data Independence

The data stored in a database must be independent of the applications that access the database. Any change in the physical structure of a database must not have any impact on how the data is being accessed by external applications.

**Rule 9:** Logical Data Independence

The logical data in a database must be independent of its user’s view (application). Any change in logical data must not affect the applications using it. For example, if two tables are merged or one is split into two different tables, there should be no impact or change on the user application. This is one of the most difficult rule to apply.

**Rule 10:** Integrity Independence

A database must be independent of the application that uses it. All its integrity constraints can be independently modified without the need of any change in the application. This rule makes a database independent of the front-end application and its interface.

**Rule 11:** Distribution Independence

The end-user must not be able to see that the data is distributed over various locations. Users should always get the impression that the data is located at one site only. This rule has been regarded as the foundation of distributed database systems.

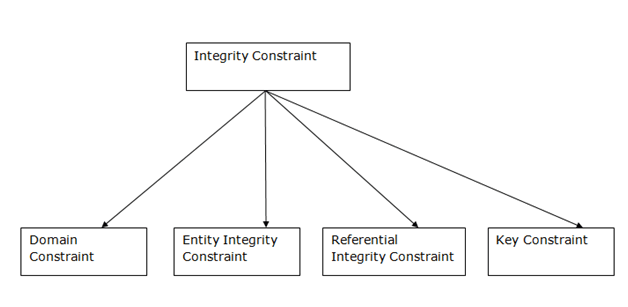
**Rule 12:** Non-Subversion Rule

If a system has an interface that provides access to low-level records, then the interface must not be able to subvert the system and bypass security and integrity constraints.

# Integrity Constraints(or) Relational Integrity

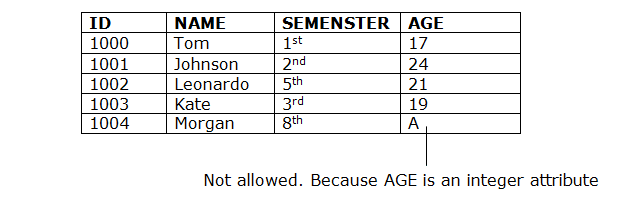
* Integrity constraints are a set of rules. It is used to maintain the quality of information.
* Integrity constraints ensure that the data insertion, updating, and other processes have to be performed in such a way that data integrity is not affected.
* Thus, integrity constraint is used to guard against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

## Types of Integrity Constraint



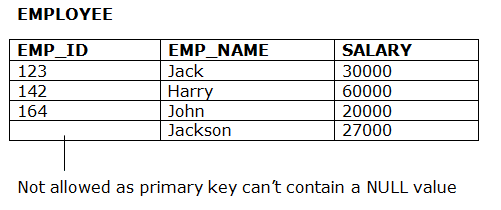
### 1. Domain constraints

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

**Example:**  
  
2. Entity integrity constraints

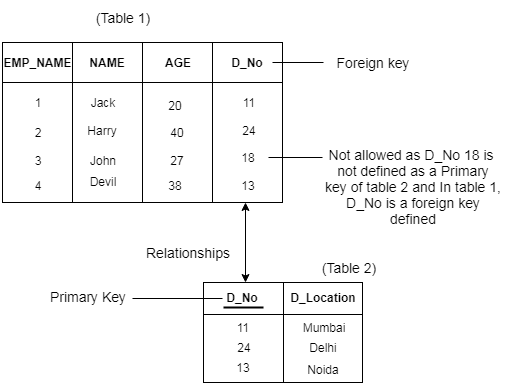
* The entity integrity constraint states that primary key value can't be null.
* This is because the primary key value is used to identify individual rows in relation and if the primary key has a null value, then we can't identify those rows.
* A table can contain a null value other than the primary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

**Example:**



### 3. Referential Integrity Constraints

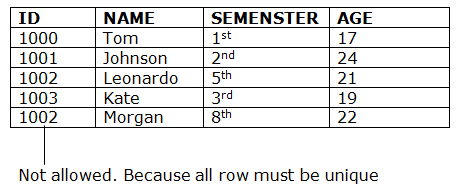
* A referential integrity constraint is specified between two tables.
* In the Referential integrity constraints, if a foreign key in Table 1 refers to the Primary Key of Table 2, then every value of the Foreign Key in Table 1 must be null or be available in Table 2.

**Example:**  


### 4. Key constraints

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

**Example:**



**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**UNIT-IV**

**Structured Query Language** Introduction, History of SQL Standards, Commands in SQL, Data types in SQL, Data Definition Language (DDL),Selection Operation Projection Operation, Aggregate Functions, Data Manipulation Language, Table Modification, Table Truncation, Imposition of Constraints, Set Operations.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Structured Query Language**

SQL is a language to operate databases; it includes database creation, deletion, fetching rows, modifying rows, etc. SQL is an **ANSI** (American National Standards Institute) standard language, but there are many different versions of the SQL language.

## What is SQL?

SQL is Structured Query Language, which is a computer language for storing, manipulating and retrieving data stored in a relational database.

SQL is the standard language for Relational Database System. All the Relational Database Management Systems (RDBMS) like MySQL, MS Access, Oracle, Sybase, Informix, Postgres and SQL Server use SQL as their standard database language.

Also, they are using different dialects, such as −

* MS SQL Server using T-SQL,
* Oracle using PL/SQL,
* MS Access version of SQL is called JET SQL (native format) etc.

## Why SQL?

SQL is widely popular because it offers the following advantages −

* Allows users to access data in the relational database management systems.
* Allows users to describe the data.
* Allows users to define the data in a database and manipulate that data.
* Allows to embed within other languages using SQL modules, libraries & pre-compilers.
* Allows users to create and drop databases and tables.
* Allows users to create view, stored procedure, functions in a database.
* Allows users to set permissions on tables, procedures and views.

## A Brief History of SQL

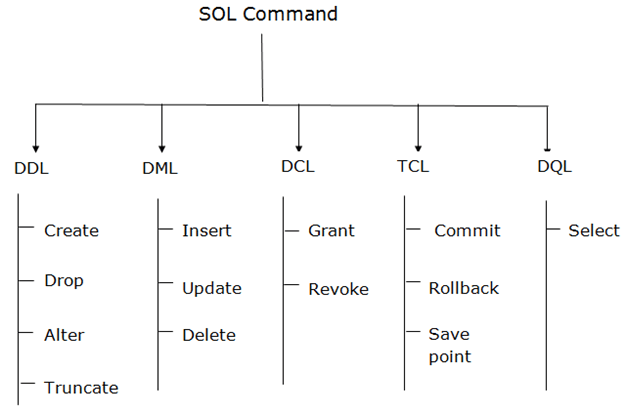
* **1970** − Dr. Edgar F. "Ted" Codd of IBM is known as the father of relational databases. He described a relational model for databases.
* **1974** − Structured Query Language appeared.
* **1978** − IBM worked to develop Codd's ideas and released a product named System/R.
* **1986** − IBM developed the first prototype of relational database and standardized by ANSI. The first relational database was released by Relational Software which later came to be known as Oracle.
* **1987** – SQL was approved by ISO.

# SQL Commands

* SQL commands are instructions. It is used to communicate with the database. It is also used to perform specific tasks, functions, and queries of data.
* SQL can perform various tasks like create a table, add data to tables, drop the table, modify the table, set permission for users.

## Types of SQL Commands

There are five types of SQL commands: DDL, DML, DCL, TCL, and DQL.



### 1. Data Definition Language (DDL)

* DDL changes the structure of the table like creating a table, deleting a table, altering a table, etc.
* All the command of DDL are auto-committed that means it permanently save all the changes in the database.

Here are some commands that come under DDL:

* CREATE
* ALTER
* DROP
* TRUNCATE

**a. CREATE** It is used to create a new table in the database.

**Syntax:**rime Ministers of India | List of Prime Minister of India (1947-2020)

1. CREATE TABLE TABLE\_NAME (COLUMN\_NAME DATATYPES[,....]);

**Example:**

1. CREATE TABLE EMPLOYEE(Name VARCHAR2(20), Email VARCHAR2(100), DOB DATE);

**b. DROP:** It is used to delete both the structure and record stored in the table.

**Syntax**

1. DROP TABLE table\_name;

**Example**

1. DROP TABLE EMPLOYEE;

**c. ALTER:** It is used to alter the structure of the database. This change could be either to modify the characteristics of an existing attribute or probably to add a new attribute.

**Syntax:**

To add a new column in the table

1. ALTER TABLE table\_name ADD column\_name COLUMN-definition;

To modify existing column in the table:

1. ALTER TABLE table\_name MODIFY(column\_definitions....);

**EXAMPLE**

1. ALTER TABLE STU\_DETAILS ADD(ADDRESS VARCHAR2(20));
2. ALTER TABLE STU\_DETAILS MODIFY (NAME VARCHAR2(20));

**d. TRUNCATE:** It is used to delete all the rows from the table and free the space containing the table.

**Syntax:**

1. TRUNCATE TABLE table\_name;

**Example:**

1. TRUNCATE TABLE EMPLOYEE;

### 2. Data Manipulation Language

* DML commands are used to modify the database. It is responsible for all form of changes in the database.
* The command of DML is not auto-committed that means it can't permanently save all the changes in the database. They can be rollback.

Here are some commands that come under DML:

* INSERT
* UPDATE
* DELETE

**a. INSERT:** The INSERT statement is a SQL query. It is used to insert data into the row of a table.

**Syntax:**

1. INSERT INTO TABLE\_NAME
2. (col1, col2, col3,.... col N)
3. VALUES (value1, value2, value3, .... valueN);

Or

1. INSERT INTO TABLE\_NAME
2. VALUES (value1, value2, value3, .... valueN);

**For example:**

1. INSERT INTO javatpoint (Author, Subject) VALUES ("Sonoo", "DBMS");

**b. UPDATE:** This command is used to update or modify the value of a column in the table.

**Syntax:**

1. UPDATE table\_name SET [column\_name1= value1,...column\_nameN = valueN] [WHERE CONDITION]

**For example:**

1. UPDATE students
2. SET User\_Name = 'Sonoo'
3. WHERE Student\_Id = '3'

**c. DELETE:** It is used to remove one or more row from a table.

**Syntax:**

1. DELETE FROM table\_name [WHERE condition];

**For example:**

1. DELETE FROM javatpoint
2. WHERE Author="Sonoo";

### 3. Data Control Language

DCL commands are used to grant and take back authority from any database user.

Here are some commands that come under DCL:

* Grant
* Revoke

**a. Grant:** It is used to give user access privileges to a database.

**Example**

1. GRANT SELECT, UPDATE ON MY\_TABLE TO SOME\_USER, ANOTHER\_USER;

**b. Revoke:** It is used to take back permissions from the user.

**Example**

1. REVOKE SELECT, UPDATE ON MY\_TABLE FROM USER1, USER2;

### 4. Transaction Control Language

TCL commands can only use with DML commands like INSERT, DELETE and UPDATE only.

These operations are automatically committed in the database that's why they cannot be used while creating tables or dropping them.

Here are some commands that come under TCL:

* COMMIT
* ROLLBACK
* SAVEPOINT

**a. Commit:** Commit command is used to save all the transactions to the database.

**Syntax:**

1. COMMIT;

**Example:**

1. DELETE FROM CUSTOMERS
2. WHERE AGE = 25;
3. COMMIT;

**b. Rollback:** Rollback command is used to undo transactions that have not already been saved to the database.

**Syntax:**

1. ROLLBACK;

**Example:**

1. DELETE FROM CUSTOMERS
2. WHERE AGE = 25;
3. ROLLBACK;

**c. SAVEPOINT:** It is used to roll the transaction back to a certain point without rolling back the entire transaction.

**Syntax:**

1. SAVEPOINT SAVEPOINT\_NAME;

### 5. Data Query Language

DQL is used to fetch the data from the database.

It uses only one command:

* SELECT

**a. SELECT:** This is the same as the projection operation of relational algebra. It is used to select the attribute based on the condition described by WHERE clause.

**Syntax:**

1. SELECT expressions
2. FROM TABLES
3. WHERE conditions;

**For example:**

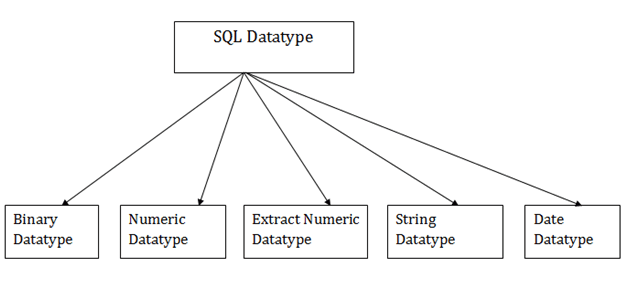
1. SELECT emp\_name
2. FROM employee
3. WHERE age > 20;

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# SQL Datatype

* SQL Datatype is used to define the values that a column can contain.
* Every column is required to have a name and data type in the database table.

## Datatype of SQL:



**1. Binary Datatypes**

There are Three types of binary Datatypes which are given below:

|  |  |
| --- | --- |
| **Data Type** | **Description** |
| binary | It has a maximum length of 8000 bytes. It contains fixed-length binary data. |
| varbinary | It has a maximum length of 8000 bytes. It contains variable-length binary data. |
| image | It has a maximum length of 2,147,483,647 bytes. It contains variable-length binary data. |

### 2. Approximate Numeric Datatype :

The subtypes are given below:

|  |  |  |  |
| --- | --- | --- | --- |
| **Data type** | **From** | **To** | **Description** |
| float | -1.79E + 308 | 1.79E + 308 | It is used to specify a floating-point value e.g. 6.2, 2.9 etc. |
| real | -3.40e + 38 | 3.40E + 38 | It specifies a single precision floating point number |

### 3. Exact Numeric Datatype

The subtypes are given below:

|  |  |
| --- | --- |
| **Data type** | **Description** |
| Int | It is used to specify an integer value. |
| Smallint | It is used to specify small integer value. |
| Bit | It has the number of bits to store. |
| Decimal | It specifies a numeric value that can have a decimal number. |
| Numeric | It is used to specify a numeric value. |

### 4. Character String Datatype

The subtypes are given below:

|  |  |
| --- | --- |
| **Data type** | **Description** |
| char | It has a maximum length of 8000 characters. It contains Fixed-length non-unicode characters. |
| varchar | It has a maximum length of 8000 characters. It contains variable-length non-unicode characters. |
| text | It has a maximum length of 2,147,483,647 characters. It contains variable-length non-unicode characters. |

### 5. Date and time Datatypes

The subtypes are given below:

|  |  |
| --- | --- |
| **Datatype** | **Description** |
| date | It is used to store the year, month, and days value. |
| time | It is used to store the hour, minute, and second values. |
| timestamp | It stores the year, month, day, hour, minute, and the second value. |

**Selection Operator-**

 Selection Operator (σ) is a unary operator in relational algebra that performs a selection operation.

It selects those rows or tuples from the relation that satisfies the selection condition.

**Syntax-**

|  |
| --- |
| **σ<selection\_condition>(R)** |

**Examples-**

* 1. Select tuples from a relation “Books” where subject is “database”

σsubject = “database” (Books)

* 1. Select tuples from a relation “Books” where subject is “database” and price is “450”

σsubject = “database” ∧ price = “450” (Books)

* 1. Select tuples from a relation “Books” where subject is “database” and price is “450” or have a publication year after 2010

σsubject = “database” ∧ price = “450” ∨ year >”2010″ (Books)

  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

## ****Projection Operator-****

 Projection Operator (π) is a unary operator in relational algebra that performs a projection operation.

It displays the columns of a relation or table based on the specified attributes.

## ****Syntax-****

|  |
| --- |
| **π<attribute list>(R)** |

**Example-**

 Consider the following Student relation-

|  |  |  |  |
| --- | --- | --- | --- |
| **ID** | **Name** | **Subject** | **Age** |
| 100 | Ashish | Maths | 19 |
| 200 | Rahul | Science | 20 |
| 300 | Naina | Physics | 20 |
| 400 | Sameer | Chemistry | 21 |

#### ****Student****

 Then, we have-

### ****Result for Query πName, Age(Student)-****

|  |  |
| --- | --- |
| **Name** | **Age** |
| Ashish | 19 |
| Rahul | 20 |
| Naina | 20 |
| Sameer | 21 |

### ****Result for Query πID , Name(Student)-****

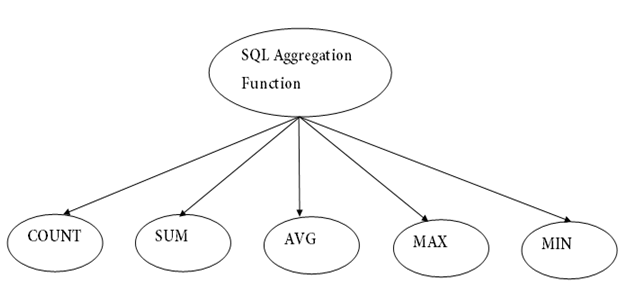
|  |  |
| --- | --- |
| **ID** | **Name** |
| 100 | Ashish |
| 200 | Rahul |
| 300 | Naina |
| 400 | Sameer |

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# Aggregate Functions

* SQL aggregation function is used to perform the calculations on multiple rows of a single column of a table. It returns a single value.
* It is also used to summarize the data.

## Types of SQL Aggregation Function



### 1. COUNT FUNCTION

* COUNT function is used to Count the number of rows in a database table. It can work on both numeric and non-numeric data types.
* COUNT function uses the COUNT(\*) that returns the count of all the rows in a specified table. COUNT(\*) considers duplicate and Null.

**Syntax**

1. COUNT(\*)

**Sample table:**

**PRODUCT\_MAST**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **PRODUCT** | **COMPANY** | **QTY** | **RATE** | **COST** |
| Item1 | Com1 | 2 | 10 | 20 |
| Item2 | Com2 | 3 | 25 | 75 |
| Item3 | Com1 | 2 | 30 | 60 |
| Item4 | Com3 | 5 | 10 | 50 |
| Item5 | Com2 | 2 | 20 | 40 |
| Item6 | Cpm1 | 3 | 25 | 75 |
| Item7 | Com1 | 5 | 30 | 150 |
| Item8 | Com1 | 3 | 10 | 30 |
| Item9 | Com2 | 2 | 25 | 50 |
| Item10 | Com3 | 4 | 30 | 120 |

**Example: COUNT()**

1. SELECT COUNT(\*)
2. FROM PRODUCT\_MAST;

**Output: 10**10

**Example: COUNT with WHERE**

1. SELECT COUNT(\*)
2. FROM PRODUCT\_MAST;
3. WHERE RATE>=20;

**Output: 7**

### 2. SUM Function

Sum function is used to calculate the sum of all selected columns. It works on numeric fields only.

**Syntax**

1. SUM()

**Example: SUM()**

1. SELECT SUM(COST)
2. FROM PRODUCT\_MAST;

**Output: 670**670

**Example: SUM() with WHERE**

1. SELECT SUM(COST)
2. FROM PRODUCT\_MAST
3. WHERE QTY>3;

**Output: 320**

### 3. AVG function

The AVG function is used to calculate the average value of the numeric type. AVG function returns the average of all non-Null values.

**Syntax**

1. AVG()

**Example:**

1. SELECT AVG(COST)
2. FROM PRODUCT\_MAST;

**Output: 67.00**67.00

### 4. MAX Function

MAX function is used to find the maximum value of a certain column. This function determines the largest value of all selected values of a column.

**Syntax**

1. MAX()

**Example:**

1. SELECT MAX(RATE)
2. FROM PRODUCT\_MAST;

**Output : 3030**

### 5. MIN Function

MIN function is used to find the minimum value of a certain column. This function determines the smallest value of all selected values of a column.

**Syntax**

1. MIN()

**Example:**

1. SELECT MIN(RATE)
2. FROM PRODUCT\_MAST;

**Output: 10**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Table Modification Commands**

# (SQL | ALTER (ADD, DROP, MODIFY))

ALTER TABLE is used to add, delete/drop or modify columns in the existing table. It is also used to add and drop various constraints on the existing table.

**ALTER TABLE – ADD**

ADD is used to add columns into the existing table. Sometimes we may require to add additional information, in that case we do not require to create the whole database again, **ADD** comes to our rescue.

**Syntax:**

**ALTER TABLE table\_name**

**ADD (Columnname\_1 datatype,**

**Columnname\_2 datatype,**

**…**

**Columnname\_n datatype);**

**ALTER TABLE – DROP**

DROP COLUMN is used to drop column in a table. Deleting the unwanted columns from the table.

**Syntax:**

**ALTER TABLE table\_name**

**DROP COLUMN column\_name;**

**ALTER TABLE-MODIFY**

It is used to modify the existing columns in a table. Multiple columns can also be modified at once.

**Syntax:**

**ALTER TABLE table\_name**

**MODIFY column\_name column\_type;**

**Sample Table:**

**Student**

| **ROLL\_NO** | **NAME** |
| --- | --- |
| 1 | Ram |
| 2 | Abhi |
| 3 | Rahul |
| 4 | Tanu |

**QUERY:**

* To ADD 2 columns AGE and COURSE to table Student.

**ALTER TABLE Student ADD (AGE number(3),COURSE varchar(40));**

**OUTPUT:**

| **ROLL\_NO** | **NAME** | **AGE** | **COURSE** |
| --- | --- | --- | --- |
| 1 | Ram |  |  |
| 2 | Abhi |  |  |
| 3 | Rahul |  |  |
| 4 | Tanu |  |  |

* MODIFY column COURSE in table Student

**ALTER TABLE Student MODIFY COURSE varchar(20);**

After running the above query maximum size of Course Column is reduced to 20 from 40.

* DROP column COURSE in table Student.

**ALTER TABLE Student DROP COLUMN COURSE;**

**OUTPUT:**

| **ROLL\_NO** | **NAME** | **AGE** |
| --- | --- | --- |
| 1 | Ram |  |
| 2 | Abhi |  |
| 3 | Rahul |  |
| `4 | Tanu |  |

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# SQL - TRUNCATE TABLE Command

The SQL **TRUNCATE TABLE** command is used to delete complete data from an existing table.

You can also use DROP TABLE command to delete complete table but it would remove complete table structure form the database and you would need to re-create this table once again if you wish you store some data.

Syntax

The basic syntax of a **TRUNCATE TABLE** command is as follows.

TRUNCATE TABLE table\_name;

Example

Consider a CUSTOMERS table having the following records −

+----+----------+-----+-----------+----------+

| ID | NAME | AGE | ADDRESS | SALARY |

+----+----------+-----+-----------+----------+

| 1 | Ramesh | 32 | Ahmedabad | 2000.00 |

| 2 | Khilan | 25 | Delhi | 1500.00 |

| 3 | kaushik | 23 | Kota | 2000.00 |

| 4 | Chaitali | 25 | Mumbai | 6500.00 |

| 5 | Hardik | 27 | Bhopal | 8500.00 |

| 6 | Komal | 22 | MP | 4500.00 |

| 7 | Muffy | 24 | Indore | 10000.00 |

+----+----------+-----+-----------+----------+

Following is the example of a Truncate command.

SQL > TRUNCATE TABLE CUSTOMERS;

Now, the CUSTOMERS table is truncated and the output from SELECT statement will be as shown in the code block below −

SQL> SELECT \* FROM CUSTOMERS;

Empty set (0.00 sec)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Imposition of Constraints**

SQL Constraints are rules used to limit the type of data that can go into a table, to maintain the accuracy and integrity of the data inside table.

Constraints can be divided into the following two types,

1. **Column level constraints:** Limits only column data.
2. **Table level constraints:** Limits whole table data.

Constraints are used to make sure that the integrity of data is maintained in the database. Following are the most used constraints that can be applied to a table.

* NOT NULL
* UNIQUE
* PRIMARY KEY
* FOREIGN KEY
* CHECK
* DEFAULT

## NOT NULL Constraint

By default, a [column](https://www.studytonight.com/dbms/rdbms-concept.php) can hold NULL values. If you do not want a column to have a NULL value, use the NOT NULL constraint.

* It restricts a column from having a NULL value.
* We use [ALTER](https://www.studytonight.com/dbms/alter-query.php) statement and [MODIFY](https://www.studytonight.com/dbms/alter-query.php) statement to specify this constraint.

One important point to note about this constraint is that it cannot be defined at table level.

### Example using NOT NULL constraint:

CREATE TABLE Student

( s\_id int NOT NULL,

name varchar(60),

age int

);

The above query will declare that the **s\_id** field of **Student** table will not take NULL value.

If you wish to alter the table after it has been created, then we can use the ALTER command for it:

ALTER TABLE Student

MODIFY s\_id int NOT NULL;

## UNIQUE Constraint

It ensures that a column will only have unique values. A UNIQUE constraint field cannot have any duplicate data.

* It prevents two records from having identical values in a column
* We use [ALTER](https://www.studytonight.com/dbms/alter-query.php) statement and [MODIFY](https://www.studytonight.com/dbms/alter-query.php) statement to specify this constraint.

### Example of UNIQUE Constraint:

Here we have a simple CREATE query to create a table, which will have a column **s\_id** with unique values.

CREATE TABLE Student

( s\_id int NOT NULL,

name varchar(60),

age int NOT NULL UNIQUE

);

The above query will declare that the **s\_id** field of **Student** table will only have unique values and wont take NULL value.

If you wish to alter the table after it has been created, then we can use the ALTER command for it:

ALTER TABLE Student

MODIFY age INT NOT NULL UNIQUE;

The above query specifies that **s\_id** field of **Student** table will only have unique value.

## Primary Key Constraint

Primary key constraint uniquely identifies each record in a database. A Primary Key must contain unique value and it must not contain null value. Usually Primary Key is used to index the data inside the table.

### PRIMARY KEY constraint at Table Level

CREATE table Student

( s\_id int PRIMARY KEY,

Name varchar(60) NOT NULL,

Age int);

The above command will creates a PRIMARY KEY on the s\_id.

## Foreign Key Constraint

[Foreign Key](https://www.studytonight.com/dbms/database-key.php) is used to relate two tables. The relationship between the two tables matches the Primary Key in one of the tables with a Foreign Key in the second table.

* This is also called a referencing key.
* We use [ALTER](https://www.studytonight.com/dbms/alter-query.php) statement and [ADD](https://www.studytonight.com/dbms/alter-query.php) statement to specify this constraint.

To understand FOREIGN KEY, let's see its use, with help of the below tables:

**Customer\_Detail** Table

|  |  |  |
| --- | --- | --- |
| **c\_id** | **Customer\_Name** | **address** |
| 101 | Adam | Noida |
| 102 | Alex | Delhi |
| 103 | Stuart | Rohtak |

**Order\_Detail** Table

|  |  |  |
| --- | --- | --- |
| **Order\_id** | **Order\_Name** | **c\_id** |
| 10 | Order1 | 101 |
| 11 | Order2 | 103 |
| 12 | Order3 | 102 |

In **Customer\_Detail** table, **c\_id** is the primary key which is set as foreign key in **Order\_Detail** table. The value that is entered in **c\_id** which is set as foreign key in **Order\_Detail** table must be present in **Customer\_Detail** table where it is set as primary key. This prevents invalid data to be inserted into **c\_id** column of **Order\_Detail** table.

If you try to insert any incorrect data, DBMS will return error and will not allow you to insert the data.

### FOREIGN KEY constraint at Table Level

CREATE table Order\_Detail(

order\_id int PRIMARY KEY,

order\_name varchar(60) NOT NULL,

c\_id int FOREIGN KEY REFERENCES Customer\_Detail(c\_id)

);

In this query, **c\_id** in table Order\_Detail is made as foriegn key, which is a reference of **c\_id** column in Customer\_Detail table.

## CHECK Constraint

**CHECK** constraint is used to restrict the value of a column between a range. It performs check on the values, before storing them into the database. Its like condition checking before saving data into a column.

### Using CHECK constraint at Table Level

CREATE table Student(

s\_id int NOT NULL CHECK(s\_id > 0),

Name varchar(60) NOT NULL,

Age int

);

The above query will restrict the **s\_id** value to be greater than zero.

# SQL Set Operation

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



### 1. 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**ncepts in Java

|  |  |
| --- | --- |
| **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 result set table will look like:

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

### 2. 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 |

### 3. 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 |

### 4. 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 |

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**UNIT - V**

**PL/SQL:** Introduction, Structure of PL/SQL, PL/SQL Language Elements, Data Types, Control Structure, Steps to Create a PL/SQL Program, Iterative Control Cursors, Steps to Create a Cursor, Procedure, Functions, Packages, Exceptions Handling, Database Triggers, Types of triggers.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**PL/SQL - INTRODUCTION**

* + PL/SQL stands for "Procedural Language extensions to SQL"
  + PL/SQL is a combination of SQL along with the procedural features of programming languages.
  + PL/SQL was developed by Oracle Corporation in the early 90’s to increase the capabilities of SQL.
  + PL/SQL allows the users and designers to develop complex database applications using control structures and procedural elements such as procedures, functions, and modules.
  + PL/SQL also offers a mechanism to process query results in a tuple-oriented way, that is, one tuple at a time. It is implemented by “cursors”

**ADVANTAGE OF USING PL/SQL**

These are the Advantages of PL/SQL

1. **Support for SQL *:*** PL/SQL allows the use of all SQL commands as well as SQL functions,operators and datatypes.
2. **Block Structure:** PL/SQL is a block-structured language. Each program written in PL/SQL is written as a block. Blovks also nested within each other. Each block forms a unit of a task. PL/SQL Blocks can be stored in the database and reused.
3. **Control structure:** PL/SQL contains procedural language constructs such as conditional statements (if else statements) and loops like (FOR loops).
4. **Better Performance:** PL/SQL engine processes multiple SQL statements simultaneously as a single block, thereby reducing network traffic.
5. **Modularity *:*** PL/SQL allows process to be divied into different modules such as procedures and functions, called as *subprograms*.
6. **Portability *:*** The programs written in PL/SQL are portable to any platform
7. **Error Handling:** PL/SQL handles errors or exceptions effectively during the execution of a PL/SQL program. Once an exception is found, specific actions can be taken depending upon the type of the exception or it can be displayed to the user with a message.

## Shortcomings of SQL (OR) Disadvantages of SQL

The disadvantages of SQL is:

### 1. Poor Interface

SQL has a poor interface as it makes look everything very complex even when it's not! Due to its difficult interfacing, users find it difficult to deal with the databases.

### 2. Cost Inefficient

SQL Server Standard costs around $1,418/year. The high cost makes it difficult for some programmers to use it.

### 3. Partial Control

SQL doesn't grant the complete control over databases to its users. This is due to some hidden business rules.

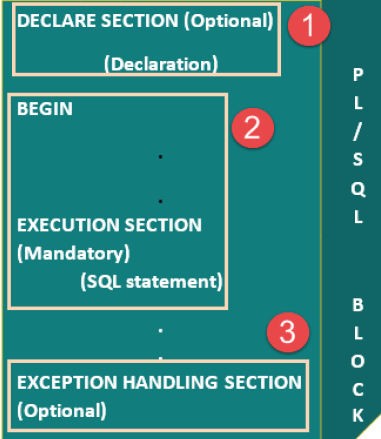
### 4. Security

Regardless of the SQL version, databases in SQL is constantly under threat as it holds huge amounts of sensitive data.

**STRUCTURE OF PL/SQL**

PL/SQL is a block-strucutred language. This means, programs are divided into logical blocks, called ***blocks***. The named blocks are called as ***subprograms*** and unnamed blocks(i.e. a block without a name) are called as ***anonymous blocks***. The anonymous block is the simplest unit in PL/SQL. It is called anonymous block because it is not saved in the Oracle database.

Subprograms can be referred as either ***functions*** or ***procedures.*** Subprograms can be nested within one another and can be grouped in larger units called ***packages***.

A PL/SQL block contains three sections –

1. Declaration section
2. Execution section
3. Exception handling section

**Syntax / Structure :**

*DECLARE*

*Declaration statements;*

*BEGIN*

*Execution statements;*

*EXCEPTION*

*Exception handling statements;*

*END;*

**/**

1. **Declaration Section**
   * It is the first section of the PL/SQL blocks.This section is an optional part.
   * This section starts with the keyword 'DECLARE'.
   * In this section, we define variables, constants, cursors, exceptions, subprograms which are used to manipulate data in the execution section.
   * This section followed by execution section. **2.Execution Section**
   * It is the main and mandatory part that used to executes the code.
   * This section starts with 'BEGIN' and ends ‘END’ keyword.
   * This section contains both PL/SQL code and SQL code and also flow control statements, loops and assignments.
   * This section followed by Exception-Handling section. **3.Exception-Handling Section:**
   * It is used for handling runtime errors and warnings. It also contains PL/SQL statements.
   * It is the last part and optional section of the PL/SQL blocks.
   * This section starts with the keyword 'EXCEPTION'.

**Rules for writting a PL/SQL program:**

While writing a PL/SQL program

* + Place a semi-colon at the end of every pl/sql statement.
  + Section keywords DECLARE,BEGIN and EXECPTION are not followed by semi-colon.
  + The END keyword always followed by semi-colon
  + When the block is executed successfully,thn the message otput is as follows:

*PL/SQL procedure successfully completed*.

**EXECUTION OF PL/SQL PROGRAM:**

To display database’s output on the screen, follow these steps:

1. First, use the **SET SERVEROUTPUT ON** command to instruct SQL\*Plus to echo database’s output after executing the PL/SQL block.

2.Second, use the **DBMS\_OUTPUT.PUT\_LINE** procedure to output a string on the screen.

**Example 1:**

1. Set serveroutput on
2. BEGIN
3. dbms\_output.put\_line (‘Hello World..');
4. END:
5. /

**Example 2:**

1. Set serveroutput on
2. DECLARE
3. text VARCHAR2(25);
4. BEGIN
5. text:= ‘Hello World’;
6. dbms\_output.put\_line(text);
7. END:
8. /

**PL/SQL LANGUAGE ELEMENTS**

PL/SQL contains a set of elements like character set,reserved words,identifiers,literals etc.

1. **Character set:**

PL/SQL programs are written as lines of text using a specific set of characters.The PL/SQL character set contains – upper and lower case letters(A…Z,a…z), numbers(0…..9), Tabs,spaces and carriage returns, Special symbols like ( , ) , +, -, /, <, >, ! , : , &, { , } , [ , ] etc.

1. **Reserved words**

Reserved words has special meaning and it cannot be redefined. Reserved words are generally written in uppercase,but it can be written in lower or mixed cases.

Example: BEGIN,DECLARE,END

1. **Delimiters**

Delimiters are simple or compound symbols that have special meaning to PL/SQL. Simple symbols are  *+, - , \* , / , = , @(remote access indicator) , ;(statement terminator)*

Compound symbols are  *<, >, !=,* || *( concatenation) , :=(assignment) , -- (single line comment), /\* ….*

*\*/ (multi line comment)*

1. **Identifiers**

Identifiers are used to name PL/SQL program items and units like constant,variable, exception, cursors, procedures,function, package, trigger,label etc.

**Properties of Identifiers:**

* + Must start with a letter
  + Maximum size is upto 30 letters
  + Cannot contain whitespace,hyphens,slashes
  + Can contain dollar sign ('$'), underscore ('\_') and hash sign ('#')
  + It is case-insensitive

1. **Literals**

A literal is an explicit numeric, character, string, or Boolean value not represented by an identifier.

PL/SQL, literals are case-sensitive. PL/SQL supports the following kinds of literals:

* + Numeric Literals
  + Character Literals
  + String Literals
  + BOOLEAN Literals
  + Date and Time Literals

**DATA TYPES IN PL/SQL**

PL/SQL provides built-in data types which are used for declaring variables. Every variable has a data type which specifies the storage format, set of specified values.

**Types/Categories:** The Data Types used in PL/SQL are as follows:

1. **Scalar datatype** – contains single value with no internal components. Ex:number,integer,int,real,float,char,character,long,varchar,varchar2,raw,rowed,date,Boolean
2. **Large Object (LOB) datatype** – indicates to large objects that are stored separately from other data itmes like text,graphic images,video clips and sound forms.
3. **Scalar Data Types a.NUMERIC DataType:**

This datatype is used to work with only number values to perfor arithmetic operations. It stores fixed or floating point numbers up to 38 digits of precision. It contains number, decimal, real, float, etc.

**Ex:** A number(5,2); B number(5); C number;

**b.Character DataType:**

This datatype stores alphanumeric values in string format. The values of Character datatype are always enclosed in single quotes. It contains char, varchar, varchar2, nvarchar2, etc.

The character data type is further classified as follows:

**Ex:** grade CHAR;

manager CHAR(10):= 'reddy'; manager VARCHAR2(10) := ‘reddy';

**c.BOOLEAN DataType:**

This datatype stores the logical values. It represents either TRUE, FALSE or NULL. It is mainly used in conditional/logical statements.

**Ex:** var1 BOOLEAN;

**d.DATE DataType:**

This data type is used to store fixed data type that displays and saves time and date values. The default data format is ‘DD-MM-YY’. Valid dates range from January 1, 4712 BC to December 31,9999 AD.

**Ex:** newyear DATE:='01-JAN-2019';

current\_date DATE:=SYSDATE;

1. **LOB DataType:**

The LOB datatype is used to store large objects like images, multimedia files, etc. LOB is used instead of the LONG datatype.

* + **BFILE:** It is used to store large binary objects in to operating system file. Cannot exceed 4GB. BFILE data type read only.
  + **BLOB:** It is used to store large unstructured binary objects in to OS file. Memory Capacity: 8 to 128 TB.
  + **CLOB:** It is used to store large blocks of character data in the database. Memory Capacity: 8 to 128 TB.
  + **NCLOB:** Used to store large blocks of NCHAR data in the database. Memory Capacity: 8 to 128 TB.

**Control Structure**

PL/SQL Control Structures are used to control flow of execution.  PL/SQL provides different kinds of statements to provide such type of procedural capabilities.These statements are almost same as that of provided by other languages.  
 The flow of control statements can be classified into the following categories:

* Conditional Control
* Iterative Control
* Sequential Control

#### ****Conditional Control :****

PL/SQL allows the use of an IF statement to control the execution of a block of code.   
 In PL/SQL, the IF -THEN - ELSIF - ELSE - END IF construct in code blocks allow specifying certain conditions under which a specific block of code should be executed.

### *Syntax:*

IF < Condition > THEN

  < Action >

ELSIF <Condition> THEN

< Action >

ELSE < Action >

END IF;

### *Example:*

create file named "condi.sql"

DECLARE

   a Number := 30;       b Number;

BEGIN

       IF a > 40 THEN

      b := a - 40;

    DBMS\_OUTPUT.PUT\_LINE('b=' || b);

  elseif a = 30 then

  b := a + 40;

  DBMS\_OUTPUT.PUT\_LINE('b=' || b);

    ELSE

    b := 0;

  DBMS\_OUTPUT.PUT\_LINE('b=' || b);

 END IF;

END;

/

### *Output:*

Run SQL Command Line

SQL>set serveroutput on  
  
SQL>start d://condi.sql  
b=70  
  
PL/SQL successfully completed.

#### ****Iterative Control :****

 Iterative control indicates the ability to repeat or skip sections of a code block.   
 A **loop** marks a sequence of statements that has to be repeated. The keyword loop has to be placed before the first statement in the sequence of statements to be repeated, while the keyword end loop is placed immediately after the last statement in the sequence.   
Once a loop begins to execute, it will go on forever. Hence a conditional statement that controls the number of times a loop is executed always accompanies loops.  
 PL/SQL supports the following structures for iterative control:

* 1. Basic or Simple loop
  2. WHILE loop
  3. FOR loop

**Basic (or) Simple loop**

It is the simplest loop structure in PL/SQL. The sequence of statements are placed inside the LOOP and END LOOP. The **EXIT** or **EXIT WHEN** is used to break the loop.

**Syntax:**

LOOP

Sequence of statements;

END LOOP;

**WHILE LOOP Statement**

A **WHILE LOOP** statement is used to repeatedly executes a statement until given condition is true.

First, the condition is evaluated. If the condition is TRUE, the sequence of statements is executed, otherwise loop is by passed and control passes to the next statement.

**Syntax:**

WHILE condition LOOP

Statements;

END LOOP;

**Example:**

SQL>set serveroutput on

SQL>DECLARE

N NUMBER := 0;

BEGIN

WHILE n < 10 LOOP

n := n + 1;

END LOOP;

DBMS\_OUTPUT.PUT\_LINE('Sum :' || n);

END;

/

**FOR LOOP Statement**

A **FOR LOOP** is a is used to execute the code for a no. of times repeatedly.

**Syntax**

FOR counter IN [REVERSE] initial\_value .. final\_value LOOP

statements;

END LOOP;

**Example**

SQL> Set serveroutput on

SQL> DECLARE

a number(2);

BEGIN

FOR a in 10 .. 20 LOOP

dbms\_output.put\_line(a);

END LOOP;

END;

/

**Example**

SQL> SET SERVEROUTPUT ON;

SQL> DECLARE

n NUMBER := 10; BEGIN

FOR i IN REVERSE 1..n LOOP DBMS\_OUTPUT.PUT\_LINE(i); END LOOP;

END;

/

#### ****Sequential Control :****

##### **The GOTO Statement**

The GOTO statement changes the flow of control within a PL/SQL block. This statement allows execution of a section of code, which is not in the normal flow of control. The entry point into such a block of code is marked using the tags «userdefined name». The GOTO statement can then make use of this user-defined name to jump into that block of code for execution.

### *Syntax :*

GOTO jump;

....

<<jump>>

### *Example :*

**DECLARE**

**BEGIN**

**dbms\_output.put\_line ('code starts');**

**dbms\_output.put\_line ('before GOTO statement');**

**GOTO down;**

**dbms\_output.put\_line ('statement will not get executed..');**

**<<down>>**

**dbms\_output.put\_line ('flow of execution jumped here.');**

**END;**

**/**

### *Output :*

Run SQL Command Line

SQL>set serveroutput on  
  
SQL>start d://a.sql  
code starts  
before gotostatements  
flow of execution jumped here.  
  
PL/SQL successfully completed.

# PL/SQL - Cursors

Oracle creates a memory area, known as the context area, for processing an SQL statement, which contains all the information needed for processing the statement; for example, the number of rows processed, etc.

A **cursor** is a pointer to this context area. PL/SQL controls the context area through a cursor. A cursor holds the rows (one or more) returned by a SQL statement. The set of rows the cursor holds is referred to as the **active set**.

You can name a cursor so that it could be referred to in a program to fetch and process the rows returned by the SQL statement, one at a time. There are two types of cursors −

* Implicit cursors
* Explicit cursors

**Implicit Cursors**

Implicit cursors are automatically created by Oracle whenever an SQL statement is executed, when there is no explicit cursor for the statement. Programmers cannot control the implicit cursors and the information in it.

Whenever a DML statement (INSERT, UPDATE and DELETE) is issued, an implicit cursor is associated with this statement. For INSERT operations, the cursor holds the data that needs to be inserted. For UPDATE and DELETE operations, the cursor identifies the rows that would be affected.

In PL/SQL, you can refer to the most recent implicit cursor as the **SQL cursor**, which always has attributes such as **%FOUND, %ISOPEN, %NOTFOUND**, and **%ROWCOUNT**. The following table provides the description of the most used attributes –

|  |  |
| --- | --- |
| **S.No** | **Attribute & Description** |
| 1 | **%FOUND**  Returns TRUE if an INSERT, UPDATE, or DELETE statement affected one or more rows or a SELECT INTO statement returned one or more rows. Otherwise, it returns FALSE. |
| 2 | **%NOTFOUND**  The logical opposite of %FOUND. It returns TRUE if an INSERT, UPDATE, or DELETE statement affected no rows, or a SELECT INTO statement returned no rows. Otherwise, it returns FALSE. |
| 3 | **%ISOPEN**  Always returns FALSE for implicit cursors, because Oracle closes the SQL cursor automatically after executing its associated SQL statement. |
| 4 | **%ROWCOUNT**  Returns the number of rows affected by an INSERT, UPDATE, or DELETE statement, or returned by a SELECT INTO statement. |

**Explicit Cursors**

Explicit cursors are programmer-defined cursors for gaining more control over the **context area**. An explicit cursor should be defined in the declaration section of the PL/SQL Block. It is created on a SELECT Statement which returns more than one row.

The syntax for creating an explicit cursor is −

CURSOR cursor\_name IS select\_statement;

Working with an explicit cursor includes the following steps −

* Declaring the cursor for initializing the memory
* Opening the cursor for allocating the memory
* Fetching the cursor for retrieving the data
* Closing the cursor to release the allocated memory

Declaring the Cursor

Declaring the cursor defines the cursor with a name and the associated SELECT statement. For example −

CURSOR c\_customers IS

SELECT id, name, address FROM customers;

Opening the Cursor

Opening the cursor allocates the memory for the cursor and makes it ready for fetching the rows returned by the SQL statement into it. For example, we will open the above defined cursor as follows −

OPEN c\_customers;

Fetching the Cursor

Fetching the cursor involves accessing one row at a time. For example, we will fetch rows from the above-opened cursor as follows −

FETCH c\_customers INTO c\_id, c\_name, c\_addr;

Closing the Cursor

Closing the cursor means releasing the allocated memory. For example, we will close the above-opened cursor as follows −

CLOSE c\_customers;

**PROCEDURE (or) STORED PROCEDURE**

A subprogram/Procedure is a module or a small unit/part of a program that performs a particular task. A procedure contains header,declaration section,executable section and optional exception handling section.

**Creating a Procedure:**

A procedure is created using CREATE OR REPLACE PROCEDURE statement with a list of parameters and must define the actions to be performed by PL/SQL blocks. When creating a procedure, define IN/OUT/INOUT parameters:

**Syntax:**

CREATE [OR REPLACE] PROCEDURE Procedure\_Name [Parameter\_Name [IN | OUT | IN OUT ] datatype ] [IS | AS]

DECLARE

//declaration\_section

BEGIN

//executable\_section PL/SQL body

[EXCEPTION]

//exception\_section END [procedure\_name];

/Where,

* + **CREATE** keyword is used to create a new procedure
  + **OR REPLACE** option is used to modify an existing procedure.
  + **Procedure\_Name** is used to specify the name of the procedure
  + **Parameter\_Name** specifies name of the PL/SQL variable
  + **Datatype** specifies datatype of the argument or procedure.

**Example**

SQL> set serveroutput on

SQL> create or replace procedure prog1(x number,y number) is begin

sum:=x+y;

dbms\_output.put\_line('Sum is:’ || sum);

end prog1;

/

**Calling/Execute a Procedure:**

A procedure is called by using the EXECUTE keyword or called from PL/SQL by giving its names followed by parameters.

**Syntax:** SQL>Execute Procedure-Name;

**Example:** SQL> EXECUTE prog1(10,20);

**Alter a Procedure:**

The ALTER statement is used to recompile or re-design a existing procedure.

**Syntax:** SQL>Alter Procedure [Procedure-Name];

**Delete or Drop a Procedure:**

The DROP keyword is used to delete a existing procedure. **Syntax:** SQL>Drop Procedure [Procedure-Name]; Exaample: SQL> drop procedure prog1;

**FUNCTIONS**

A function is a named PL/SQL block that takes one or more parameter and returns one value. Like procedure, a function contains header, declaration section, executable section and optional exception handling section.A function contains RETURN clause in the header section and atleast one RETURN statement in the execution section.

**Creating a Function:**

A function is created using CREATE OR REPLACE FUNCTION statement with a list of parameters, must return one value and define the actions to be performed by PL/SQL blocks. When creating a procedure, define IN/OUT/INOUT parameters.

**Syntax:**

CREATE [OR REPLACE] FUNCTION function\_name [Parameter\_Name [IN | OUT | IN OUT ] datatype ]

RETURN datatype [IS | AS]

DECLARE

//declaration\_section

BEGIN

//executable\_section

[EXCEPTION]

//exception\_section END function\_name;

/Where,

* + **CREATE FUNCTION** keyword is used to create a new function
  + **OR REPLACE** option is used to modify an existing function.
  + **function\_Name** is used to specify the name of the function.
  + **Parameter\_Name** specifies name of the variable whose value is passed to the fumction.
  + **RETURN** specifies datatype of return value
  + **Datatype** specifies datatype of the argument or procedure.

**Example:**

SQL> set serveroutput on

SQL> create or replace function fun1(x number,y number) return number

is begin

sum:=x+y;

return(sum);

end fun1;

/

**Execute a Function:**

A function accepts one or maore parameters but returns only one value.

**Example:**

SQL> set serveroutput on SQL> declare

n1 umber:100; n2 umber:=200; sum number; begin sum:=fun1(n1,n2);

dbms\_output.put\_line(‘result is:’ || sum); end;

/

**Delete or Drop a Function:**

The DROP keyword is used to delete a function.

**Syntax:** SQL>DROP FUNCTION [Function\_Name];

# PL/SQL - Packages

Packages are schema objects that groups logically related PL/SQL types, variables, and subprograms.

A package will have two mandatory parts −

* Package specification
* Package body or definition

## Package Specification

The specification is the interface to the package. It just **DECLARES** the types, variables, constants, exceptions, cursors, and subprograms that can be referenced from outside the package. In other words, it contains all information about the content of the package, but excludes the code for the subprograms.

All objects placed in the specification are called **public** objects. Any subprogram not in the package specification but coded in the package body is called a **private** object.

The following code snippet shows a package specification having a single procedure. You can have many global variables defined and multiple procedures or functions inside a package.

CREATE PACKAGE cust\_sal AS

PROCEDURE find\_sal(c\_id customers.id%type);

END cust\_sal;

/

When the above code is executed at the SQL prompt, it produces the following result −

Package created.

## Package Body

The package body has the codes for various methods declared in the package specification and other private declarations, which are hidden from the code outside the package.

The **CREATE PACKAGE BODY** Statement is used for creating the package body. The following code snippet shows the package body declaration for the ***cust\_sal*** package created above. I assumed that we already have CUSTOMERS table created in our database as mentioned in the [PL/SQL - Variables](https://www.tutorialspoint.com/plsql/plsql_variable_types.htm) chapter.

CREATE OR REPLACE PACKAGE BODY cust\_sal AS

PROCEDURE find\_sal(c\_id customers.id%TYPE) IS

c\_sal customers.salary%TYPE;

BEGIN

SELECT salary INTO c\_sal

FROM customers

WHERE id = c\_id;

dbms\_output.put\_line('Salary: '|| c\_sal);

END find\_sal;

END cust\_sal;

/

When the above code is executed at the SQL prompt, it produces the following result −

Package body created.

## Using the Package Elements

The package elements (variables, procedures or functions) are accessed with the following syntax −

package\_name.element\_name;

Consider, we already have created the above package in our database schema, the following program uses the ***find\_sal*** method of the ***cust\_sal*** package −

DECLARE

code customers.id%type := &cc\_id;

BEGIN

cust\_sal.find\_sal(code);

END;

/

When the above code is executed at the SQL prompt, it prompts to enter the customer ID and when you enter an ID, it displays the corresponding salary as follows −

Enter value for cc\_id: 1

Salary: 3000

PL/SQL procedure successfully completed.

### Example

The following program provides a more complete package. We will use the CUSTOMERS table stored in our database with the following records −

Select \* from customers;

+----+----------+-----+-----------+----------+

| ID | NAME | AGE | ADDRESS | SALARY |

+----+----------+-----+-----------+----------+

| 1 | Ramesh | 32 | Ahmedabad | 3000.00 |

| 2 | Khilan | 25 | Delhi | 3000.00 |

| 3 | kaushik | 23 | Kota | 3000.00 |

| 4 | Chaitali | 25 | Mumbai | 7500.00 |

| 5 | Hardik | 27 | Bhopal | 9500.00 |

| 6 | Komal | 22 | MP | 5500.00 |

+----+----------+-----+-----------+----------+

### The Package Specification

CREATE OR REPLACE PACKAGE c\_package AS

-- Adds a customer

PROCEDURE addCustomer(c\_id customers.id%type,

c\_name customers.Name%type,

c\_age customers.age%type,

c\_addr customers.address%type,

c\_sal customers.salary%type);

-- Removes a customer

PROCEDURE delCustomer(c\_id customers.id%TYPE);

--Lists all customers

PROCEDURE listCustomer;

END c\_package;

/

When the above code is executed at the SQL prompt, it creates the above package and displays the following result −

Package created.

### Creating the Package Body

CREATE OR REPLACE PACKAGE BODY c\_package AS

PROCEDURE addCustomer(c\_id customers.id%type,

c\_name customers.Name%type,

c\_age customers.age%type,

c\_addr customers.address%type,

c\_sal customers.salary%type)

IS

BEGIN

INSERT INTO customers (id,name,age,address,salary)

VALUES(c\_id, c\_name, c\_age, c\_addr, c\_sal);

END addCustomer;

PROCEDURE delCustomer(c\_id customers.id%type) IS

BEGIN

DELETE FROM customers

WHERE id = c\_id;

END delCustomer;

PROCEDURE listCustomer IS

CURSOR c\_customers is

SELECT name FROM customers;

TYPE c\_list is TABLE OF customers.Name%type;

name\_list c\_list := c\_list();

counter integer :=0;

BEGIN

FOR n IN c\_customers LOOP

counter := counter +1;

name\_list.extend;

name\_list(counter) := n.name;

dbms\_output.put\_line('Customer(' ||counter|| ')'||name\_list(counter));

END LOOP;

END listCustomer;

END c\_package;

/

The above example makes use of the **nested table**. We will discuss the concept of nested table in the next chapter.

When the above code is executed at the SQL prompt, it produces the following result −

Package body created.

### Using The Package

The following program uses the methods declared and defined in the package *c\_package*.

DECLARE

code customers.id%type:= 8;

BEGIN

c\_package.addcustomer(7, 'Rajnish', 25, 'Chennai', 3500);

c\_package.addcustomer(8, 'Subham', 32, 'Delhi', 7500);

c\_package.listcustomer;

c\_package.delcustomer(code);

c\_package.listcustomer;

END;

/

When the above code is executed at the SQL prompt, it produces the following result −

Customer(1): Ramesh

Customer(2): Khilan

Customer(3): kaushik

Customer(4): Chaitali

Customer(5): Hardik

Customer(6): Komal

Customer(7): Rajnish

Customer(8): Subham

Customer(1): Ramesh

Customer(2): Khilan

Customer(3): kaushik

Customer(4): Chaitali

Customer(5): Hardik

Customer(6): Komal

Customer(7): Rajnish

PL/SQL procedure successfully completed

# PL/SQL - Exceptions

An exception is an error condition during a program execution. PL/SQL supports programmers to catch such conditions using **EXCEPTION** block in the program and an appropriate action is taken against the error condition. There are two types of exceptions −

* System-defined exceptions
* User-defined exceptions

## Syntax for Exception Handling

The general syntax for exception handling is as follows. Here you can list down as many exceptions as you can handle. The default exception will be handled using ***WHEN others THEN*** −

DECLARE

<declarations section>

BEGIN

<executable command(s)>

EXCEPTION

<exception handling goes here >

WHEN exception1 THEN

exception1-handling-statements

WHEN exception2 THEN

exception2-handling-statements

WHEN exception3 THEN

exception3-handling-statements

........

WHEN others THEN

exception3-handling-statements

END;

### Example

Let us write a code to illustrate the concept. We will be using the CUSTOMERS table we had created and used in the previous chapters −

DECLARE

c\_id customers.id%type := 8;

c\_name customerS.Name%type;

c\_addr customers.address%type;

BEGIN

SELECT name, address INTO c\_name, c\_addr

FROM customers

WHERE id = c\_id;

DBMS\_OUTPUT.PUT\_LINE ('Name: '|| c\_name);

DBMS\_OUTPUT.PUT\_LINE ('Address: ' || c\_addr);

EXCEPTION

WHEN no\_data\_found THEN

dbms\_output.put\_line('No such customer!');

WHEN others THEN

dbms\_output.put\_line('Error!');

END;

/

When the above code is executed at the SQL prompt, it produces the following result −

No such customer!

PL/SQL procedure successfully completed.

The above program displays the name and address of a customer whose ID is given. Since there is no customer with ID value 8 in our database, the program raises the run-time exception **NO\_DATA\_FOUND**, which is captured in the **EXCEPTION block**.

## Raising Exceptions

Exceptions are raised by the database server automatically whenever there is any internal database error, but exceptions can be raised explicitly by the programmer by using the command **RAISE**. Following is the simple syntax for raising an exception −

DECLARE

exception\_name EXCEPTION;

BEGIN

IF condition THEN

RAISE exception\_name;

END IF;

EXCEPTION

WHEN exception\_name THEN

statement;

END;

You can use the above syntax in raising the Oracle standard exception or any user-defined exception. In the next section, we will give you an example on raising a user-defined exception. You can raise the Oracle standard exceptions in a similar way.

## User-defined Exceptions

PL/SQL allows you to define your own exceptions according to the need of your program. A user-defined exception must be declared and then raised explicitly, using either a RAISE statement or the procedure **DBMS\_STANDARD.RAISE\_APPLICATION\_ERROR**.

The syntax for declaring an exception is −

DECLARE

my-exception EXCEPTION;

### Example

The following example illustrates the concept. This program asks for a customer ID, when the user enters an invalid ID, the exception **invalid\_id** is raised.

DECLARE

c\_id customers.id%type := &cc\_id;

c\_name customerS.Name%type;

c\_addr customers.address%type;

-- user defined exception

ex\_invalid\_id EXCEPTION;

BEGIN

IF c\_id <= 0 THEN

RAISE ex\_invalid\_id;

ELSE

SELECT name, address INTO c\_name, c\_addr

FROM customers

WHERE id = c\_id;

DBMS\_OUTPUT.PUT\_LINE ('Name: '|| c\_name);

DBMS\_OUTPUT.PUT\_LINE ('Address: ' || c\_addr);

END IF;

EXCEPTION

WHEN ex\_invalid\_id THEN

dbms\_output.put\_line('ID must be greater than zero!');

WHEN no\_data\_found THEN

dbms\_output.put\_line('No such customer!');

WHEN others THEN

dbms\_output.put\_line('Error!');

END;

/

When the above code is executed at the SQL prompt, it produces the following result −

Enter value for cc\_id: -6 (let's enter a value -6)

old 2: c\_id customers.id%type := &cc\_id;

new 2: c\_id customers.id%type := -6;

ID must be greater than zero!

PL/SQL procedure successfully completed.

## Pre-defined Exceptions

PL/SQL provides many pre-defined exceptions, which are executed when any database rule is violated by a program. For example, the predefined exception NO\_DATA\_FOUND is raised when a SELECT INTO statement returns no rows. The following table lists few of the important pre-defined exceptions −

|  |  |  |  |
| --- | --- | --- | --- |
| **Exception** | **Oracle Error** | **SQLCODE** | **Description** |
| ACCESS\_INTO\_NULL | 06530 | -6530 | It is raised when a null object is automatically assigned a value. |
| CASE\_NOT\_FOUND | 06592 | -6592 | It is raised when none of the choices in the WHEN clause of a CASE statement is selected, and there is no ELSE clause. |
| COLLECTION\_IS\_NULL | 06531 | -6531 | It is raised when a program attempts to apply collection methods other than EXISTS to an uninitialized nested table or varray, or the program attempts to assign values to the elements of an uninitialized nested table or varray. |
| DUP\_VAL\_ON\_INDEX | 00001 | -1 | It is raised when duplicate values are attempted to be stored in a column with unique index. |
| INVALID\_CURSOR | 01001 | -1001 | It is raised when attempts are made to make a cursor operation that is not allowed, such as closing an unopened cursor. |
| INVALID\_NUMBER | 01722 | -1722 | It is raised when the conversion of a character string into a number fails because the string does not represent a valid number. |
| LOGIN\_DENIED | 01017 | -1017 | It is raised when a program attempts to log on to the database with an invalid username or password. |
| NO\_DATA\_FOUND | 01403 | +100 | It is raised when a SELECT INTO statement returns no rows. |
| NOT\_LOGGED\_ON | 01012 | -1012 | It is raised when a database call is issued without being connected to the database. |
| PROGRAM\_ERROR | 06501 | -6501 | It is raised when PL/SQL has an internal problem. |
| ROWTYPE\_MISMATCH | 06504 | -6504 | It is raised when a cursor fetches value in a variable having incompatible data type. |
| SELF\_IS\_NULL | 30625 | -30625 | It is raised when a member method is invoked, but the instance of the object type was not initialized. |
| STORAGE\_ERROR | 06500 | -6500 | It is raised when PL/SQL ran out of memory or memory was corrupted. |
| TOO\_MANY\_ROWS | 01422 | -1422 | It is raised when a SELECT INTO statement returns more than one row. |
| VALUE\_ERROR | 06502 | -6502 | It is raised when an arithmetic, conversion, truncation, or sizeconstraint error occurs. |
| ZERO\_DIVIDE | 01476 | 1476 | It is raised when an attempt is made to divide a number by zero. |

**Database Triggers, Types of Triggers**

Triggers are stored programs, which are automatically executed or fired when some events occur. Triggers are, in fact, written to be executed in response to any of the following events −

* A **database manipulation (DML)** statement (DELETE, INSERT, or UPDATE)
* A **database definition (DDL)** statement (CREATE, ALTER, or DROP).
* A **database operation** (SERVERERROR, LOGON, LOGOFF, STARTUP, or SHUTDOWN).

Triggers can be defined on the table, view, schema, or database with which the event is associated.

### Benefits of Triggers

Triggers can be written for the following purposes −

* Generating some derived column values automatically
* Enforcing referential integrity
* Event logging and storing information on table access
* Auditing
* Synchronous replication of tables
* Imposing security authorizations
* Preventing invalid transactions

## Creating Triggers

The syntax for creating a trigger is −

CREATE [OR REPLACE ] TRIGGER trigger\_name

{BEFORE | AFTER | INSTEAD OF }

{INSERT [OR] | UPDATE [OR] | DELETE}

[OF col\_name]

ON table\_name

[REFERENCING OLD AS o NEW AS n]

[FOR EACH ROW]

WHEN (condition)

DECLARE

Declaration-statements

BEGIN

Executable-statements

EXCEPTION

Exception-handling-statements

END;

Where,

* CREATE [OR REPLACE] TRIGGER trigger\_name − Creates or replaces an existing trigger with the *trigger\_name*.
* {BEFORE | AFTER | INSTEAD OF} − This specifies when the trigger will be executed. The INSTEAD OF clause is used for creating trigger on a view.
* {INSERT [OR] | UPDATE [OR] | DELETE} − This specifies the DML operation.
* [OF col\_name] − This specifies the column name that will be updated.
* [ON table\_name] − This specifies the name of the table associated with the trigger.
* [REFERENCING OLD AS o NEW AS n] − This allows you to refer new and old values for various DML statements, such as INSERT, UPDATE, and DELETE.
* [FOR EACH ROW] − This specifies a row-level trigger, i.e., the trigger will be executed for each row being affected. Otherwise the trigger will execute just once when the SQL statement is executed, which is called a table level trigger.
* WHEN (condition) − This provides a condition for rows for which the trigger would fire. This clause is valid only for row-level triggers.

### Example

To start with, we will be using the CUSTOMERS table we had created and used in the previous chapters −

Select \* from customers;

+----+----------+-----+-----------+----------+

| ID | NAME | AGE | ADDRESS | SALARY |

+----+----------+-----+-----------+----------+

| 1 | Ramesh | 32 | Ahmedabad | 2000.00 |

| 2 | Khilan | 25 | Delhi | 1500.00 |

| 3 | kaushik | 23 | Kota | 2000.00 |

| 4 | Chaitali | 25 | Mumbai | 6500.00 |

| 5 | Hardik | 27 | Bhopal | 8500.00 |

| 6 | Komal | 22 | MP | 4500.00 |

+----+----------+-----+-----------+----------+

The following program creates a **row-level** trigger for the customers table that would fire for INSERT or UPDATE or DELETE operations performed on the CUSTOMERS table. This trigger will display the salary difference between the old values and new values −

CREATE OR REPLACE TRIGGER display\_salary\_changes

BEFORE DELETE OR INSERT OR UPDATE ON customers

FOR EACH ROW

WHEN (NEW.ID > 0)

DECLARE

sal\_diff number;

BEGIN

sal\_diff := :NEW.salary - :OLD.salary;

dbms\_output.put\_line('Old salary: ' || :OLD.salary);

dbms\_output.put\_line('New salary: ' || :NEW.salary);

dbms\_output.put\_line('Salary difference: ' || sal\_diff);

END;

/

When the above code is executed at the SQL prompt, it produces the following result −

Trigger created.

The following points need to be considered here −

* OLD and NEW references are not available for table-level triggers, rather you can use them for record-level triggers.
* If you want to query the table in the same trigger, then you should use the AFTER keyword, because triggers can query the table or change it again only after the initial changes are applied and the table is back in a consistent state.
* The above trigger has been written in such a way that it will fire before any DELETE or INSERT or UPDATE operation on the table, but you can write your trigger on a single or multiple operations, for example BEFORE DELETE, which will fire whenever a record will be deleted using the DELETE operation on the table.

## Triggering a Trigger

Let us perform some DML operations on the CUSTOMERS table. Here is one INSERT statement, which will create a new record in the table −

INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)

VALUES (7, 'Kriti', 22, 'HP', 7500.00 );

When a record is created in the CUSTOMERS table, the above create trigger, **display\_salary\_changes** will be fired and it will display the following result −

Old salary:

New salary: 7500

Salary difference:

Because this is a new record, old salary is not available and the above result comes as null. Let us now perform one more DML operation on the CUSTOMERS table. The UPDATE statement will update an existing record in the table −

UPDATE customers

SET salary = salary + 500

WHERE id = 2;

When a record is updated in the CUSTOMERS table, the above create trigger, **display\_salary\_changes** will be fired and it will display the following result −

Old salary: 1500

New salary: 2000

Salary difference: 500

**Types of Triggers**

SQL Server has three types of triggers:

* DML (Data Manipulation Language) Triggers
* DDL (Data Definition Language) Triggers
* Logon Triggers

DML Triggers allow us to execute code in response to data modification. In other words, they allow us to run additional code in response of the execution of an insert, update or delete statement.

DDL Triggers give us the possibility to execute code in response of changes in the structure of a database, like dropping or creating a table; or a server event like when someone logs in. DDL Triggers can be split into two different types according to where they are scoped.

* Database Scoped DDL Triggers
* Server Scoped DDL Triggers

Logon Triggers are a particular case of Server Scoped DDL triggers that fire in response to the LOGON event that's raised when a user's session is being established.