**II BSC**

**SEMESTER - III**

**DATABASE MANAGEMENT SYSTEMS**

**UNIT I**

**Overview of Database Management System:** Introduction to data, information, database, database management systems, file-based system, Drawbacks of file-Based System, database approach, Classification of Database Management Systems, advantages of database approach, Various Data Models, Components of Database Management System, three schema architecture of data base, costs and risks of database approach.

**UNIT II**

**Entity-Relationship Model:** Introduction, the building blocks of an entity relationship diagram, classification of entity sets, attribute classification, relationship degree, relationship classification, reducing ER diagram to tables, enhanced entity-relationship model (EER model), generalization and specialization.

**UNIT III**

**Relational Model:** Introduction, CODD Rules, relational data model, concept of key, relational integrity, relational algebra, relational algebra operations, advantages of relational algebra, limitations of relational algebra, Functional dependencies and normal forms upto 3rd normal form.

**UNIT IV**

**Structured Query Language:** Introduction, History of SQL Standard, Commands in SQL, Data Types in SQL, Data Definition Language, Selection Operation, Projection Operation, Aggregate functions, Data Manipulation Language, Table Modification Commands, Join Operation, Set Operations, View, Sub Query.

**UNIT V**

**PL/SQL:** Introduction, Shortcomings of SQL, Structure of PL/SQL, PL/SQL Language Elements, Data Types, Operators Precedence, Control Structure, Steps to Create a PL/SQL, Program, Iterative Control, Procedure, Function, Database Triggers, Types of Triggers.

**BOOKS:**

1. Database System Concepts by Abraham Silberschatz, Henry Korth, and S. Sudarshan, McGrawhill
2. Database Management Systems by Raghu Ramakrishnan, McGrawhill
3. Principles of Database Systems by J. D. Ullman
4. Fundamentals of Database Systems by R. Elmasri and S. Navathe
5. SQL: The Ultimate Beginners Guide by Steve Tale.

**UNIT – I**

**Data:**

Data are raw facts. The word raw indicates that the facts have not yet been processed to reveal their meaning.

**Information:**

Information is the result of processing raw data to reveal its meaning. Data processing can be as simple as organizing data to reveal patterns or as complex as making forecasts or drawing inferences using statistical modeling.

**Database**

A database is a shared integrated computer structure that stores a collection of

* End-user data that is raw facts of interest to the end-user.
* Metadata or data about data, through which the end-user data are integrated and managed.

The metadata provide a description of the data characteristics and the set of relationships that link the data found within the database.

**DBMS**

A database management system is a collection of programs that manages the database structure and controls access to the data stored in the database. In a sense, a database resembles a very well-organized electronic filing cabinet in which powerful software, known as a database management system, helps manage the cabinet’s contents.

**Advantages of DBMS:**

1. **Improved data security:**

The more users access the data, the greater the risks of data security breaks/breaches. Corporations invest considerable amounts of time, effort and money to ensure that corporate data are used properly. A DBMS provides a framework for better enforcement of data privacy and security policies.

1. **Minimized data inconsistency:**

Data inconsistency exists when different versions of the same data appear in different places.

1. **Better data integration**

Wider access to well-managed data promotes an integrated view of the organization’s operations and a clearer view of the big picture. It becomes much easier to see how actions in one segment of the company affect other segments.

1. **Improved data sharing**

The DBMS helps create an environment in which end users have better access to more data and better-managed data. Such access makes it possible for end users to respond quickly to changes in their environment.

1. **Improved data access**

The DBMS makes it possible to produce quick answers to ad hoc queries. From a database perspective, a query is a specific request issued to the DBMS for data manipulation.

1. **Improved decision making**

Better-managed data and improved data access make it possible to generate better quality information, on which better decisions are based.

1. **Increased end-user productivity**

The availability of data, combined with the tools that transform data into usable information, empowers end users to make quick, informed decisions that can make the difference between success and failure in the global economy.

**File-based System**

File-based system was an early attempt to computerize the manual filing system. It is basically a collection of application programs that performed services for the end users. Each program within a file based system defines and manages its own data. In this system a large number of files are needed to perform various tasks.

**For example:** In sales department of an enterprise, one user will be maintaining the details of how many sales personnel are there in the sales department and their grades, these details will be stored and maintained in a separate file. Another user will be maintaining the salesperson salary details working in the concern, the detailed salary report will be stored and maintained in a separate file. Although both of the users are interested in the data’s of the salesperson they will be having their details in separate files and they need different programs to manipulate their files. This will lead to wastage of space and redundancy or replication of data’s, which may lead to confusion, sharing of data among various users is not possible, data inconsistency may occur.

The approach to data management, in which separate data files are created and stored for each application program, is called the Traditional File approach.

**Characteristics of File-based system**

1. It is a group of files storing data of an organization
2. Each file is independent from one another.
3. Each file is called a flat file.
4. Each file contained and processed information for one specific task, such as accounting or inventory.
5. File are designed by using programs written in programming languages such as C,C++
6. Each file must have its own file management system.

**Drawbacks of File-Based System**

In the early days, database applications were built on top of file systems. The drawbacks of using the file systems to store data are

1. **Uncontrolled redundancy of data:**

Each subsystem of an organization maintains a set of files without data sharing; the same data will be available in different files. This will result in increased disk space requirement, increased time of data entry and inconsistency of data.

1. **Inconsistency of data:**

The uncontrolled redundancy of data will permit the system to have the same data in different files. As a result, there is a possibility of a particular data element will be entered differently in different files which is nothing but inconsistency of particular data. While performing basic data processing operations, this inconsistency will give misleading results.

1. **Inflexibility:**

The conventional file processing system will not be flexible enough to provide the desired results because of its insensitiveness to dynamics of the real world.

1. **Limited Data Sharing:**

In the conventional file processing system, the data is stored in decentralized manner on different standalone systems, there is a remote probability of sharing of data. The limited data sharing is due to the nesting of files within each subsystem of the organization.

1. **Poor enforcements of standards:**

While designing files under conventional file processing system, each group follows its own standards of defining field name, field width, field type and so. This will create serious difficulty while modifying programs and data structures of the files by different groups of users will lead to low programming productivity.

1. **Excessive program maintenance:**

The maintenance group will find difficulty to relate names of fields of different files for the same attribute. Many of the program variables may be defined differently in different programs. While modifying the file structures of some applications it will take more time to implement the modifications correctly. All these difficulties will lead to excessive maintenance effort.

1. **Low programmer productivity:**

Programmer productivity is a measure of time taken to develop an application. In conventional file processing system, there will be inflexibility, poor enforcement of standards and excessive maintenance effort. Hence the programmer productivity will become considerably low in conventional processing systems.

**Classification Of DBMS**

A DBMS can support many different types of database. DBMS can be classified according to the number of users, the DB location and the expected type and extent of use.

1. **Based on Data Models**

Depending on the data model they use, the DBMSs can be classified as hierarchical, network and relational.

* 1. **Hierarchical DBMS:**

Hierarchical DBMS organizes the data records in a tree structure i.e hierarchy of parent-child relationships. In a hierarchical database, a parent record may have more than one child, but a child always has only one parent. This is called a one-to-many relationship.

* 1. **Network DBMS**

It organizes the data records linked to one another through pointers, which is an association between two records. A network database is similar to a hierarchical database except that each child can have more than one parent record. This is called as man-to-many relationship.

* 1. **Relational DBMS**

It organizes the data records in the form of table and relationships among the tables are set using common fields. It is simple in nature because data is simply represented in tabular format

1. **Based on number of users**

The database is classified based on the number of users are

* 1. **Single user**

A single user database supports only one user at a time. In other words, if user A is using the database, users B and C must wait until user A is done. A single user database that runs on a personal computer is called a desktop database.

* 1. **Multi user.**

A Multiuser database supports multiple users at the same time. When the multiuser database supports a relatively small number of users (usually fewer than 50) or a specific department within an organization, it is called as workgroup database. When the database is used by the entire organization and supports many users (more than 50, usually hundreds) across many departments, the database is known as an enterprise database.

1. **Based on Database Distribution**

There are four main distribution systems for database systems and these in turn can be used to classify the DBMS as

* 1. **Centralized database**

A database that supports data located at a single site is called as centralized database. This database can be used by several other systems too.

* 1. **Distributed Database**

A database that supports data distributed across several different sites is called as distributed database. The extent to which a database can be distributed and the way in which such distribution is managed

* 1. **Homogenous Distributed Database**

Homogenous distributed database systems use the same DBMS software from multiple sites. Data exchange between these various sites can be handled easily.

* 1. **Heterogeneous Distributed Database**

In a heterogeneous distributed database system, different sites might use different DBMS software, but there is additional common software to support data exchange between these sites.

1. **Based on the purpose**

Depending on the purpose the DBMS serves, it can be classified as **general purpose or special purpose.**

* 1. **Operational Database**

A database that is designed primarily to support a company’s day-to-day operations is classified as an operational database.

* 1. **Data warehouse**

A Data warehouse focuses primarily on storing data used to generate information required to make tactical or strategic decisions. Such decisions typically require extensive “data manipulation” to extract information to formulate pricing decisions, sales forecasts, market positioning and so on. Most decision support data are based on historical data obtained from operational databases. Additionally, the data warehouse can store data derived from any many sources. To make it easier to retrieve such data, the data warehouse structure is quite different from that of an operational or transactional database.

**DBMS APPROACH**

In order to remove all the limitations of a file-based system, a new approach was required that must be more effective. So, the concept of database was introduced. This approach is known as database approach / DBMS approach

A database is a computer based record keeping system whose over all purpose is to record and maintains information. The database is a single, large repository of data, which can be used simultaneous by many departments and users. With the database approach, instead of disconnected files with redundant data, all data items are integrated with minimum amount of duplication and maintain only at one location. In this approach, related data is shared by multiple application programs.

**Characteristics:**

DBMS approach possesses the following characteristics.

1. It is central repository of shared data. It allows several users to access the database concurrently.
2. A primary feature of the database approach is a standardized, uniform approach to database access. This means that the same overall procedures are used by all application programs to retrieve data and information.
3. Data should be correct with respect to the real world entity that they represent.
4. Data should be protected from unauthorized users.
5. Duplication of data is minimized

**Data Model**

**A data model is a relatively simple representation, usually graphical of more complex real-world data structures**. In general terms, a model is an abstraction of a more complex real-world object or event. A model’s main function is to help understand the complexities of the real world environment. Within the database environment, a data model represents data structures and their characteristics, relations, constraints, transformations and other constructs with the purpose of supporting a specific problem domain.

Data models can facilitate interaction among the designer, the applications programmer and the end user. In short, data models are communication tool.

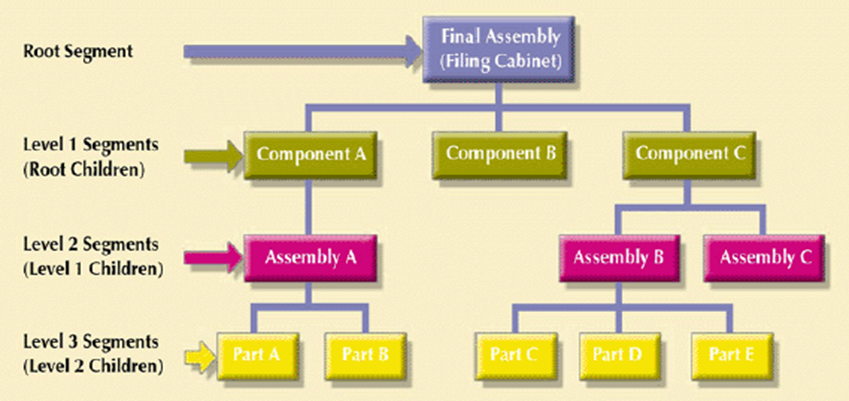
**Evolution of Data Models**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Generation** | **Time** | **Model** | **Examples** | **Comments** |
| First | 1960s to 1970s | File System | VMS/ VSAM | Used mainly on IBM mainframe  Managed records, not relationships |
| Second | 1970s | Hierarchical and Network Data Model | IMS  ADABAS  IDS-II | Early database systems |
| Third | Mid-1970’s to present | Relational Data Model | DB2  Oracle  Ms-SQL-Server | Conceptual simplicity  ER-modeling and support for relational data modeling |
| Fourth | Mid-1980s to present | Object Oriented | Versant FastObject.net | Support complex data  Extended relational products |
| Next Generation | Present to future | XML | DbXML  DB2 UDB  Oracle10g | Organization and management of unstructured data  Relational and object models add support for XML documents |

**Hierarchical Data Model**

The hierarchical model was developed in the 1960s to manage large amounts of data for complex manufacturing projects such as the **APOLLO rocket** that landed on the moon in 1969. Its basic logical structure is represented by an upside-down tree. The hierarchical structure contains levels or segments. A segment is the equivalent of a file system’s record type. Within the hierarchy the top layer (root) is perceived as the parent of the segment directly beneath it. The segment below other segments is the children of the segments above.

In short hierarchical model represents a set of one-to-many relationships between a parent and its children segment.



**Advantages:**

1. It promotes data sharing
2. Parent child relationship promotes conceptual simplicity.
3. Database security is provided and enforced by DBMS
4. Parent/child relationship promotes data integrity
5. It is efficient with 1:M relationships

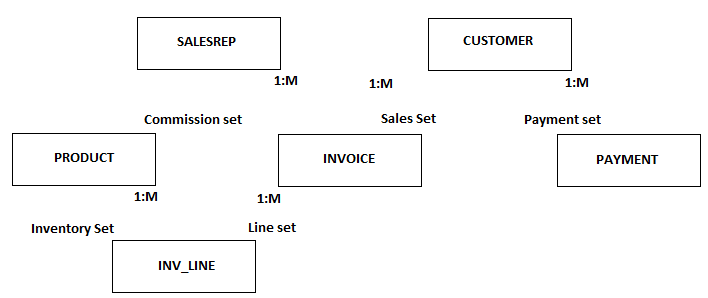
**Disadvantages:**

1. Complex implementation requires knowledge of physical data storage characteristics.
2. Navigational system yields complex application development, management and use requires knowledge of hierarchical paths.
3. Changes in structure require changes in all application programs.
4. There is no data definition or data manipulation language in the DBMS.
5. There is a lack of standards.

**Network Data Model**

The network model was created to represent complex data relationships more effectively than hierarchical model, **to improve database performance and to impose a database standard**. The lack of database standards was troublesome to programmers and application designers because it made database designs and applications less portable.

In the network model, the user perceives the network database as a collection of records in 1:M relationships. However, unlike the hierarchical model, the network model allows a record to have more than one parent. In network database terminology, a relationship is called a set. Each set is composed of at least two record types; an owner record and a member record. A set represents a 1:M relationship between the owner and the member.

****

**Advantages:**

1. It handles more relationship types such as M:N AND multi-parent.
2. Data access is more flexible.
3. Data owner membership promotes data integrity
4. It includes DDL and DML in DBMS

**Disadvantages:**

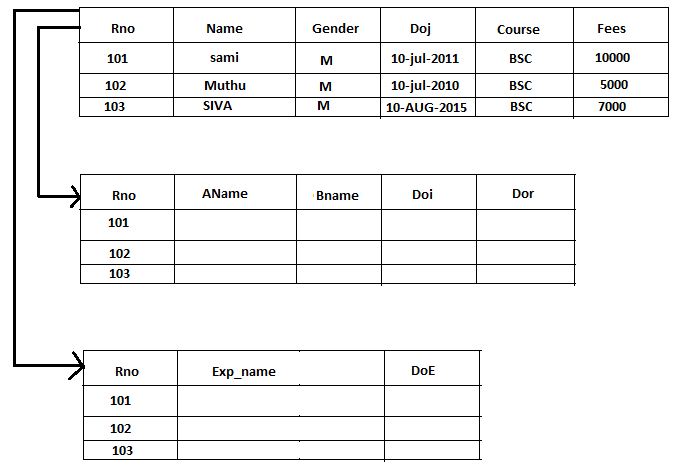
1. Navigations system yields complex implementation, application development and management.
2. Structural changes require changes in all application programs.

**Relational Model**

The relation model was introduced in **1970 by E.F. Codd** in his land mark paper **“A Relational Model of Data for Large Shared Databanks”.** The relational model represented a major breakthrough for both users and designers.

The relational model foundation is a mathematical concept known as relation. To avoid complexity of abstract mathematical theory, you can think of a relation a matrix composed of intersecting rows and columns. Each row in a relation is called a **tuple**. Each column represents an attribute. The relational model also describes a precise set of data manipulation constructs based on advanced mathematical concepts.

Today even micro-computers can run sophisticated relational database software such as ORACLE, DB2, MYSQL and other mainframe relational software.

****

**Advantages:**

1. Structural independence is promoted.
2. Tabular view substantially improves conceptual simplicity.
3. ADHOC query capability is based on SQL
4. Powerful RDBMS isolates the end user from physical level details.

**Disadvantages**

1. The RDBMS requires substantial hardware and system software overhead
2. Conceptual simplicity gives relatively untrained people, the tools to use a good system poorly.

**Object Oriented Data Model:**

Increasingly complex real world problems demonstrated a need for a data model that more closely represented the real world. In the object oriented data model both data and their relationships are contained in a single structure known as an object. In turn, the OODM is the basis for the object oriented data base management system

An OODM reflects a very different way to define and use entities. Like the relational model’s entity, an object is described by its factual content. But quite unlike an entity, an object includes information about relationships between the facts within the object, as well as information about its relationships with other objects. Therefore, the facts within the object are giving greater meaning. The OODM is said to be a semantic data model because semantic indicates meaning.

The OO data model is based on the following components:

1. An object is an abstraction of a real world entity. In general terms, an object may be considered equivalent to an ER model’s entity. More precisely, an object represents only one occurrence of an entity.
2. Attributes describe the properties of an object.
3. Objects that share similar characteristics are grouped in classes. A class is a collection of similar objects with shared structure and behavior. In a general sense, a class resembles the ER model’s entity set. However a class is different from an entity set in that it contains a set of procedures known as methods. A class’s method represents a real world action such as finding a selected person’s name, changing a person’s name or printing a person’s address. In other words, methods are the equivalent of procedures in traditional programming languages. In OO terms, methods define an object’s behavior.
4. Classes are organized in a class hierarchy. The class hierarchy resembles a upside down tree in which each class has only one parent.
5. Inheritance is the ability of an object within the class hierarchy to inherit the attributes and methods of the classes above it.

**Components of DBMS OR Database Environment**

The database system refers to an organization of components that define and regulate the collection, storage, management and use of data within a database environment. From a general management point of view, the database system is composed of **five major parts**.

1. **Hardware:** Hardware refers to all the system’s physical devices
2. **Software**

Although the most readily identified software is the DBMS itself, to make the database system function fully, three types of software are needed.

1. **Operating system:** It manages all hardware components and makes it possible for other software to run on the computers.
2. **DBMS Software:** It manages the database within the database system
3. **Application programs and utility software:** It is used to access and manipulate data in the DBMS and manages the computer environment in which data access and manipulation take place. Application programs are most commonly used to access data found within the database to generate reports, tabulations and other information to facilitate decision making.
4. **People:**

This component includes all users of the database system. On the basis of primary job functions five types of users can be identified in a database system. **System administrators, Database administrators, database designers, system analysts and programmers and end users.**

1. **System Administrators:** These people are responsible to oversee the database systems general operations.
2. **Database Administrators:** DBMS manages the DBMS and ensure that the database is functioning properly.
3. **Database Designers:** These people design the database structure.
4. **System Analysts and Programmers:** They design and implement the application programs
5. **Procedure**

Procedures are the instructions and rules that govern the design and use of the database system. Procedures are a critical, although occasionally forgotten, component of the system. Procedures play an important role in a company because they enforce the standards by which business is conducted within the organization and with customers. Procedures also are used to ensure that there is an organized way to monitor and audit both the data that enter the database and the information that is generated through the use of the data.

1. **Data**

The word data covers the collection of facts stored in the database. Because data are the raw material from which information is generated, the determination of what data are to be entered into the database and how that data are to be organized is a vital part of the database designers’ job.

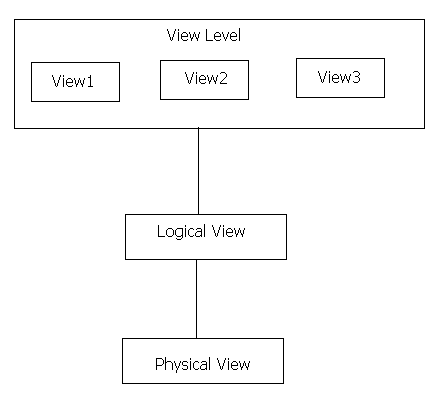
**Three Schema Architecture:**

The major purpose of a database system is to provide users with an abstract view of the system. The system hides certain complex details from the user like-how data is stored and shows only those database records to the user which he/she demands. **This is called Data Abstraction.**

In DBMS, the database can have different aspects to reveal if seen from different levels of abstraction. The term abstraction means the amount of detail you want to hide. Any entity can be seen from different perspectives and levels of complexity to make it a reveal its current amount of abstraction.

The database can be viewed from different levels of abstraction to reveal different levels of details. From a bottom-up manner, we may find that there are three levels of abstraction in the database. They are

1. External Level / View Level
2. Conceptual Level / Logical View
3. Internal Level / Physical View



1. **View level / External level:**

The highest level of abstraction describes only part of the entire database. Many users of the database system will not be concerned with all this information. Instead such users need to access only a part of the database. So that their interaction with the system is simplified the view level of abstraction is defined. The system may provide many views for the same database.

1. **Conceptual Level / Logical View**

This level describes what data are stored in the database, and what relationships exist among those data. The entire database is thus described in terms of a small number of relatively simple structures. The logical level of abstraction is used by database administrators, who must decide what information is to be kept in the database.

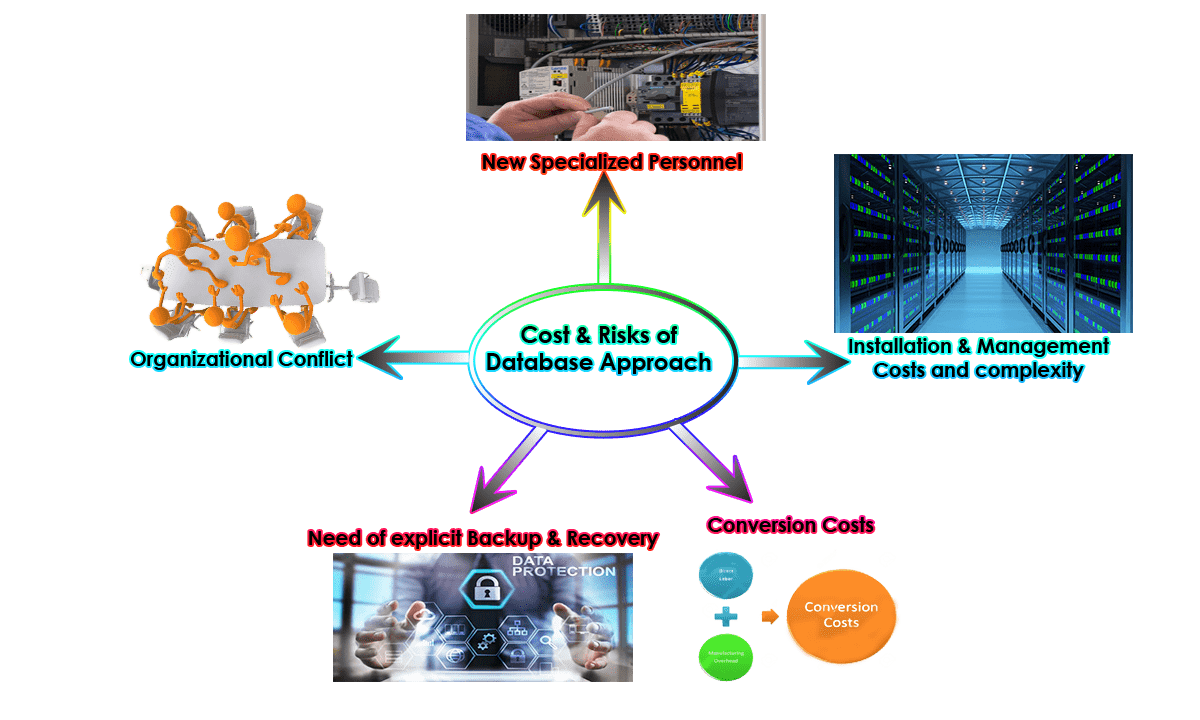
1. **Internal Level / Physical View**

This is the lowest level of data abstraction. It describes how data is actually stored in database. You can get the complex data structure details at this level.

Although, both internal level and physical level is considered as a single level, but there is slight difference between them. Actually, physical level is one that is managed by the operating system under the direction of DBMS, while the internal level is managed by DBMS.

**Costs and risks of database approach.**

The database approach emphasizes data integration and sharing across organizations. As with any business decision, the database approach entails some additional costs and risks that must be recognized and managed when implementing this approach. Mentionable some costs & risks of database approach are as follows;



* 1. **New Specialized Personnel:**

Frequently, organizations that adopt the database approach need to hire or train individuals to design & implement databases, provide database administration services and manage a staff of new people, further, because of the rapid changes in technology these new people will have to be retrained or upgraded on a regular basis.

* 1. **Installation & Management Costs and complexity:**

A multi-user database management system is a large and complex suite of software that has a high initial cost, requires a staff of trained personnel to install and operate, and also has substantial annual maintenance & support costs. Installing such a system may also require upgrades to the hardware and data communications system in the organization.

* 1. **Conversion Costs:**

The cost of converting the traditional file processing systems to modern database technology: measured in terms of money, time, and organizational commitment.

* 1. **Need for explicit Backup & Recovery:**

A shared corporate database must be accurate and available at all times. These require that comprehensive procedures be developed and used for providing backup copies of data and for restoring a database when damage occurs.

* 1. **Organizational Conflict:**

A shared database requires a consensus on data definitions and ownership as well as responsibilities for accurate data maintenance. Experience has shown that conflicts on data definitions, data formats, and coding, rights to update shared data are frequent and often difficult to resolve.

**UNIT – II**

**Basic Building Blocks of an ER-Model**

The basic building blocks of Entity Relationship diagram are **Entity, Attribute, Relationship and constraints.**

1. **ENTITY:**

An entity is a person, place, object, event or concept in the user environment about which the organization where wishes to maintain data. Entities can be represented by means of **rectangle and named** with the entities they represent.

**Eg:**

**Person:** EMPLOYEE, STUDENT, PATIENT

**Place:** STORE, WAREHOUSE, STATE

**Object:** MACHINE, BUILDING, AUTOMOBILE

**Event:** SAL, REGISTRATION, RENEWAL

**Concept:** ACCOUNT, COURSE, WORK CENTER

**Types of Entity Sets**

Entities are of different types. They are

1. Weak Entity
2. Strong Entity

|  |  |  |
| --- | --- | --- |
| **Entities** | **Description** | **Symbolic Representation** |
| Weak Entity | Weak entity is one whose existence depends on other entity. An entity that does not has an attribute which may act as a primary key or key attribute is called weak entity. ***A weak entity in ER diagram can be represented as*double outlined box** |  |
| Strong Entity | It is one whose existence does not depend on other entity. An entity that has an attribute which may act as a primary key or key attribute is called strong entity. **A strong entity in ER diagram is represented by Single Outlined box** |  |

1. **ATTRIBUTES:**

It is a property or characteristic of an entity that is of interest to the organization. The attributes can be represented by means of **ellipses**. Every ellipse represents one attribute and is directly connected to its entity. Attributes are of different types. They are

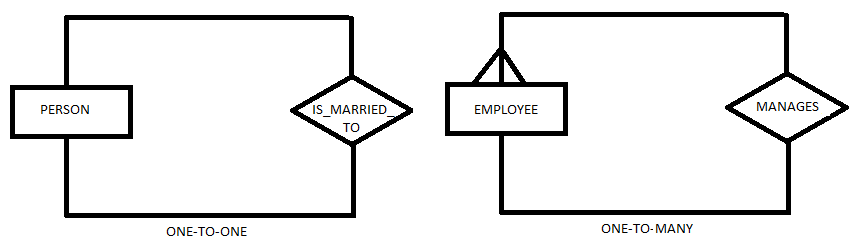
|  |  |  |
| --- | --- | --- |
| **Attributes** | **Description** | **Symbolic Representation** |
| Simple Attribute | An attribute that consist of a single atomic value. It cannot be broken down into smaller components.  **Eg: Age, Salary** |  |
| Derived Attribute | An attribute that’s value is derived from a stored attribute. **For eg,“age”** its value is derived from the stored attribute “**Date\_of\_Birth”** |  |
| **Composite Attribute** | An attribute that can be broken down into small components.  **Eg:** Address (Hno, street, city, pin code) |  |
| **Multi-valued Attribute** | An attribute that may take on more than one value for a given entity instance.  **Eg:** Multiple Phones Numbers, Multiple email-ids, etc. |  |
| **Single Valued Attribute** | A single-valued attribute can have only a single value. For **eg,** A person can have only one **“Date\_of\_Birth”, ”Age”,** etc. It can be simple or composite attribute. That is **Date\_of\_Birth** is a composite attribute, where **“Age”** is a simple attribute. But both are single valued attributes. | Simple Attribute  As  Single-Valued Attribute  Composite Attribute  As  Single Value Attribute |

1. **RELATIONSHIPS:**

A Relationship is an association among the instances of one or more entity types. Relationships are represented by diamond-shaped box. Name of the relationship is written inside the diamond box. There are three kinds of relationships are available, they are

1. **Unary Relationship:**

A unary relationship is a relationship between the instances of single entity types.



1. **Binary Relationship**

A binary relationship is a relationship between the instances of two entity types and is the most common type of relationship encountered in data modeling. The following three figures shows the binary relationship

**Employee**

**Parking\_Place**

Is\_assign

**One-to-One**

**Product\_line**

**Product**

Contains

**One-to-Many**

**Student**

**course**

Registers

**Many-to-Many**

The first indicates that an employee is assigned one parking place and each parking place is assigned to one employee.

The second indicates that a product line may contain several products, and each product belongs to only one product line.

The third shows that a student may register for more than one course, and that each course may have many student registrants.

1. **Ternary Relationship:**

A ternary relationship is a simultaneous relationship among the instances of three entity types. A typical business situation that leads to a ternary relationship is as follows.

PART

VENDOR

Supplies

Warehouse

**Advantages of ER Model:**

1. **Conceptual Simplicity**

ER-Model represents the concepts of a database along with its entities and relationships in an easy way. It becomes even easier to create and manage the complex database designs by using ER-model

1. **Visual Representation**

ER-Model provides a visual representation of data and the relationships among data. It helps the database designers, programmers and end users to understand the structure of data in easy way. It also minimizes the data redundancy.

1. **Effective Communication Tool**

The data base designer can use ER-model to get different views of data as seen by programmers, managers and end users etc. ER-model works as an effective communication tool.

1. **Integrated with Relational Database Model**

ER-model is well integrated with the relational database model. This integration makes relational database design a very structured process.

**Disadvantages of ER Model:**

1. **Popular for High level Design**

The ER-Model is especially popular for high level database design

1. **No Industry Standard**

There is no industry standard notation for developing an ER-diagram

1. **Limited Relationship Representation**

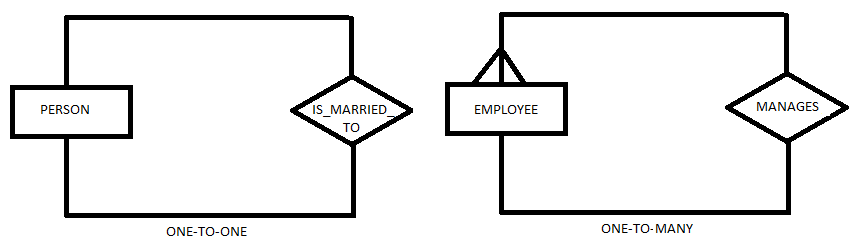
ER-model represents limited relationships among the entities.

**Degree of Relationships**

The degree of a relationship is the number of entity types that participate in that relationship. Thus, the relationship completes is of degree 2, since there are two entity types **EMPLOYEE** and **COURSE**. The three most common relationship degrees in E-R models are **unary, binary** and **ternary**. **Higher degree relationships** are possible, but they are rarely encountered in practice. **The degree of a relationship is also called cardinality.**

1. **Unary Relationship:**

A unary relationship is a relationship between the instances of single entity types.



1. **Binary Relationship**

A binary relationship is a relationship between the instances of two entity types and is the most common type of relationship encountered in data modeling. The following three figures shows the binary relationship

**Employee**

**Parking\_Place**

Is\_assign

**One-to-One**

**Product\_line**

**Product**

Contains

**One-to-Many**

**Student**

**course**

Registers

**Many-to-Many**

The first indicates that an employee is assigned one parking place and each parking place is assigned to one employee.

The second indicates that a product line may contain several products, and each product belongs to only one product line.

The third shows that a student may register for more than one course, and that each course may have many student registrants.

1. **Ternary Relationship:**

A ternary relationship is a simultaneous relationship among the instances of three entity types. A typical business situation that leads to a ternary relationship is as follows.

PART

VENDOR

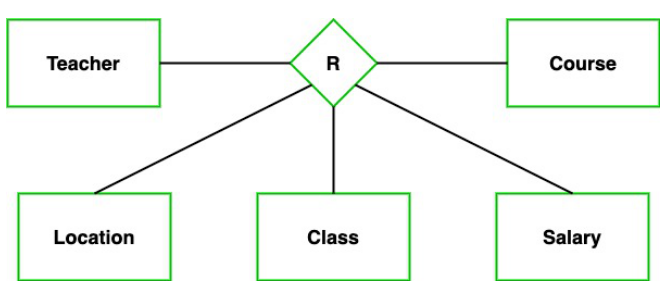
Suppliess

Warehouse

**N-ary Relationship**

In the N-ary relationship, there are n types of entity that associates. So, we can say that an N-ary relationship exists when there are n types of entities. There is one limitation of the N-ary relationship, as there are many entities so it is very hard to convert into an entity, rational table. So, this is very uncommon, unlike binary which is very much popular.

**Example:**We have 5 entities Teacher, Class, Location, Salary, Course. So, here five entity types are associating we can say an n-ary relationship is 5.



**Reducing ER Diagram to Tables**

To implement the database, it is necessary to use the relational model. There is a simple way of mapping from ER model to the relational model.

**Mapping Algorithm:**

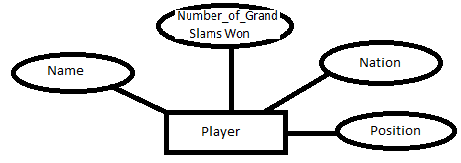
The mapping algorithm gives the procedure to map ER diagram to tables. The rules in mapping algorithm are given as:

1. For each strong entity type say **E**, create a new table. The columns of the table are the attribute of the entity type **E**.
2. For each weak entity **W** that is associated with only one 1-1 identifying owner relationships, identify the table **T** of the owner entity type. Include as columns of **T**, all the simple attributes and simple components of the composite attribute of **W**.

**Mapping Regular Entities:**

* Each regular entity type in the ER diagram is transformed into a relation. The name given to the relation is generally the same as the entity type.
* Each simple attribute of the entity type becomes an attribute of the relation.
* The identifier of the entity type becomes the primary key of the corresponding relation.

**Example:** Mapping regular entity type tennis player

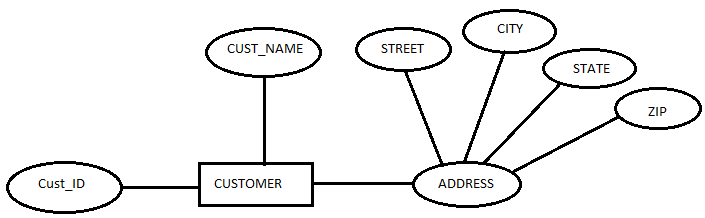
****

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Nation** | **Position** | **No\_Of\_Grand\_Slams\_Won** |
| Roger | Switzerland | 1 | 5 |
| Nadal | USA | 2 | 4 |

**Converting Composite Attribute in an ER diagram to Tables**

When a regular entity type has a composite attribute, only the simple component attributes of the composite attribute are included in the relation.

**Example**

****

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Cust\_ID** | **Cust\_Name** | **Street** | **City** | **State** | **Zip** |
|  |  |  |  |  |  |

When the regular entity type contains a multivalued attributes, two new relations are created.

1. The first relation contains all of the attributes of the entity type except the multivalued attribute
2. The second relation contains two attributes that form the primary key of the second relation. The first of these attributes is the primary key from the first relation, which becomes a foreign key in the second relation. The second is the multivalued attribute.

**Extended entity relationship model (EERM):**

EER is a high-level data model that incorporates the extensions to the original ER model. Enhanced ERD are high level models that represent the requirements and complexities of complex database.

In addition to ER model concepts EE-R includes −

1. Subclasses and Super classes.
2. Specialization and Generalization.
3. Category or union type.
4. Aggregation.

These concepts are used to create EE-R diagrams.

1. **Sub Class and Super Class**

* Sub class and Super class relationship leads the concept of Inheritance.
* The relationship between sub class and super class is denoted with https://www.tutorialride.com/images/dbms/sub-super-class.jpg symbol.

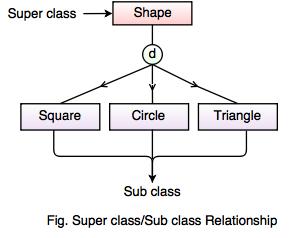
1. **Super Class**

* Super class is an entity type that has a relationship with one or more subtypes.
* An entity cannot exist in database merely by being member of any super class.

**For example:** Shape super class is having sub groups as Square, Circle, Triangle.

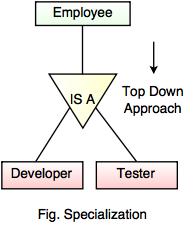
1. **Sub Class**

* Sub class is a group of entities with unique attributes.
* Sub class inherits properties and attributes from its super class.  
  **For example:** Square, Circle, Triangle are the sub class of Shape super class.



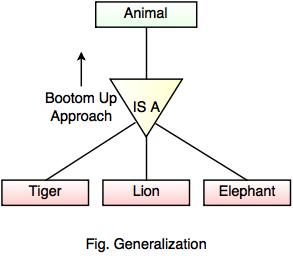
1. **Specialization and Generalization**
   1. **Specialization**
   * Specialization is a process that defines a group entities which is divided into sub groups based on their characteristic.
   * It is a top down approach, in which one higher entity can be broken down into two lower level entity.
   * It maximizes the difference between the members of an entity by identifying the unique characteristic or attributes of each member.
   * It defines one or more sub class for the super class and also forms the superclass/subclass relationship.

**For example**



In the above example, Employee can be specialized as Developer or Tester, based on what role they play in an Organization.

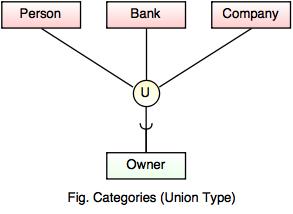
* 1. **Generalization**
  + Generalization is the process of generalizing the entities which contain the properties of all the generalized entities.
  + **It is a bottom approach,** in which two lower level entities combine to form a higher level entity.
  + Generalization is the reverse process of Specialization.
  + It defines a general entity type from a set of specialized entity type.
  + It minimizes the difference between the entities by identifying the common features.  
    **For example:**



In the above example, Tiger, Lion, Elephant can all be generalized as Animals.

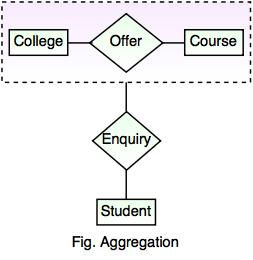
1. **Category or Union**

* Category represents a single super class or sub class relationship with more than one super class.
* It can be a total or partial participation.
* **For example** Car booking, Car owner can be a person, a bank (holds a possession on a Car) or a company. Category (sub class) → Owner is a subset of the union of the three super classes → Company, Bank, and Person. A Category member must exist in at least one of its super classes.



1. **Aggregation**

* Aggregation is a process that represent a relationship between a whole object and its component parts.
* It abstracts a relationship between objects and viewing the relationship as an object.
* It is a process when two entity is treated as a single entity.



* In the above example, the relation between College and Course is acting as an Entity in Relation with Student.

**UNIT III**

**RELATIONAL DATA MODEL:**

The relational data model is the conceptual basis of relational databases **proposed by Dr. E.F. Codd in 1970.** The relational model is considers as one of the most popular developments in the database technology because it can be used for representing most of the real world objects and the relationships between them.

The main construct for representing data in the relational model is a relation, where each relation is a table with rows and columns. A row is also called a record. A column is also called a field. A database table is similar to a spreadsheet. In simplest terms, a relational database is one that presents information in tables with rows and columns. This simple tabular representation enables even novice users to understand the contents of a database, and it permits the use of simple, high-level languages to query the data. The major advantages of the relational model over the older data models are its simple data representation and the ease with which even complex queries can be expressed.

Relational database establish a well-defined relationship between database tables. However, the relationships that can be created among the tables enable a relational database to efficiently store huge amount of data and effectively retrieve selected data.

Relational database design provides guidelines to define tables, columns, and establish relationships between tables. A well-designed relational database should not be stored in more than one place. This is because duplicate data not only waste storage spaces but also easily lead to inconsistencies. It should also ensure data integrity and accuracy.

**Concepts of Relational Model**

The relational data model uses formal terms like **attribute, tuple, relation, domain, schema and keys to define its concepts**

1. **Attributes:**

An attribute can be defined as a characteristic of data. A real world data feature, modeled in the database will be represented by an attribute.

1. **Tuple:**

A tuple represents a collection of information that describes a person or thing. Informal terms used for tuples are: row in a table or records in a data file.

1. **Relation:**

A relation is the core of the relational data model. In relational data model, data is organized in terms of rows and columns in a table known as relation.

1. **Domain:**

A domain describes the set of possible values that are all of the same type for a given attribute. A value is the smallest unit of data in the relational model.

1. **Schema:**

A database schema is a formal description of all the database relations and all the relationships existing between them.

1. **Key**

The relational data model uses keys to define identifiers for a relation’s tuples. A key is an attribute of a table which helps to uniquely identify a row.

**CODD RULES**

1. **The Information rule:**

All information in relational database must be logically represented as column values in rows within tables.

1. **The rule of Guaranteed Access:**

Every value in a table is guaranteed to be accessible through a combination of table name, primary key value and column name.

1. **Systematic treatment of null values**

Nulls must be represented and treated in a systematic way, independent of data type.

1. **Dynamic on-line catalog**

The metadata must be stored and managed as ordinary data that is in tables.

1. **Comprehensive sub-language rule**

The system must support for data definition view definition, data manipulation

1. **The View update rule**

Any view that is theoretical updatable must be up datable through the system.

1. **The insert, update rule**

The database must support set level inserts, updates and deletes.

1. **Physical data independence rule**

Application programs are logical unaffected when physical access methods or storage structure are changed.

1. **The logical data independence rule**

Application programs are logically unaffected when changes are made to the table structure that, preserve the original table values.

1. **The data integrity rule**

All relational integrity constraints must be definable in the relational language.

1. **The distribution rule**

The end users and application programmers are unaffected by the data location.

1. **Non-subversion rule**

If the system supports low level access to the data, there must not be a way to bypass the integrity rules of the database.

**Key:**

A key is an attribute of a table which helps to identify a row. Keys are of different types. They are

1. **Candidate Key**

It is such an attribute of a table that can uniquely identify a row in a table. Generally they contain unique values and can never contain null values. There can be more than one super key or candidate key in a table.

**Eg:** Rno, Phone\_No.

1. **Primary Key:**

It is one of the candidate keys that are chosen to be the identifying key for the entire table.

1. **Composite Key:**

Sometimes one key is not enough to uniquely identify a row.

**Eg:** Rno,Course

1. **Alternate Key:**

This is the candidate key which is not chosen as the primary key of the table. They are named so because although not the primary key, but they can still identify a row.

1. **Foreign Key:**

Sometimes we may have to work with an attribute that does not have a primary key of its own. To identify its rows, we have to use the primary attribute of a related table. Such a copy of another related table’s primary key is called foreign key.

In relational database, a foreign key is used to link tables together and create a relationship between them. It acts as a cross-reference between tables because it references the primary key of another table, thereby establishing a link between them.

In simple terms, a foreign key is one table points to a primary key in another table.

**Relational Integrity**

Database Integrity means the **completeness, correctness** and **consistency of data**. It ensures that data entered into the database must be complete, accurate, valid and consistent. It is another form of database protection.

In a relational data model, data integrity or data accuracy can be achieved using integrity rules or constraints. An integrity constraint is a condition that is specified on a database schema, and restricts the data that can be stored in an instance of the database. If a database instance satisfies all the integrity constraints specified on the database schema, it is a legal instance.

Integrity constraints maybe divided into three main categories

1. Domain Integrity Constraints
2. Entity Integrity Constraints
3. Referential Integrity Constrains
4. **Domain Integrity Constraints**

This constraint allows the user **not to accept null values** and allows the user to accept the correct data based on the conditions defined. The domain integrity constraints are

1. **Not Null:**

This constraint allows the user not to accept null values.

1. **Check**

This constraint allows the user to accept the correct data based on the conditions defined.

**Syntax: Constraint <cons\_name> check(condition)**

1. **Entity Integrity Constraints**

This constraint allows the user not to accept duplicate values and null values. The Entity Integrity constraints are

1. **Primary Key**
2. **Unique**
3. **Primary Key:**

This constraint allows the user not to accept duplicate values and null values. The primary key constraint can be set for only one column at column level and if we want to set primary key on more than one column then it can be set at table level. At a time we can set primary key constraint for 16 columns, such constraint can be called as composite key.

1. **Unique:**

This constraint allows the user not to accept duplicate values but accept null values.

1. **Referential Integrity Constraints:**

The Referential Integrity Constraint enforces relationship between tables. It designates a column or combination of columns as a foreign key. The foreign key establishes a relationship with a specified primary or unique key in another table, called the referenced key. In this relationship, the table containing the foreign key is called the child table and the table containing the referenced key is called the parent table. The referential integrity constraints are of two types. They are

1. **References**

This constraint allows the user to provide relation or reference from one table to another.

1. **Foreign Key**

Foreign key constraint is the one to which the child column values refer the values of parent table primary or unique key. Foreign key or reference key constraint can be at table level or at column level. If you define the foreign key at table level then it is called a composite-foreign key.

All the above constraints can be written in two levels. They are –

1. **Column Level Constraints**

A constraint which can be defined immediately after defining a column, such constraint can be called as column level constraint.

1. **Table Level Constraints**

A constraint which can be defined after defining all the columns of a table, such constraint can be called as table level constraint. The table level constraint can be defined under the following situations-

* 1. A Constraint which depends on other columns of a table.
  2. A Primary key constraint can be set on more than one column

**Relational Algebra**

The relational algebra is a procedural query language. It consists of a set of operations that take one or two relations as input and produce a new relation as their result. **The fundamental operations in the relational algebra are select, project, union, set difference, Cartesian product** and **rename**. In addition to the fundamental operations, there are several other operations namely, set, intersection, natural join, division and assignment.

Relational algebra is one of the two formal query languages associated with the relational model. Queries in algebra are composed using a collection of operators. A fundamental property is that every operator in the algebra accepts relation instances as arguments and returns a relation instance as the result. This property makes it easy to compose operators to form a complex query – a relational algebra expression is recursively defined to be a relation, a unary algebra operator applied to a single expression, or a binary algebra operator applied to two expressions

**Operations of Relational Algebra**

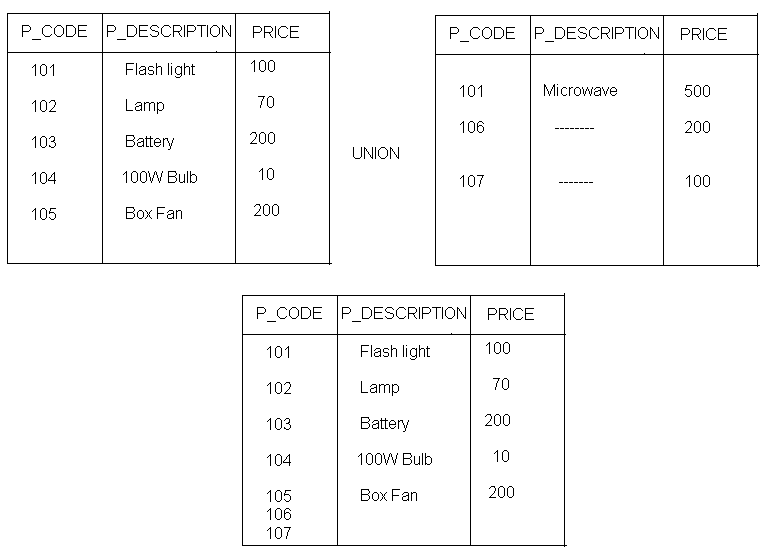
The data in relational tables are of limited value unless the data can be manipulated to generate useful information. Relational algebra defines the theoretical way of manipulating table contents using the eight relational operators.

1. Union
2. Intersection
3. Difference
4. Product
5. Select
6. Project
7. Join
8. Divide

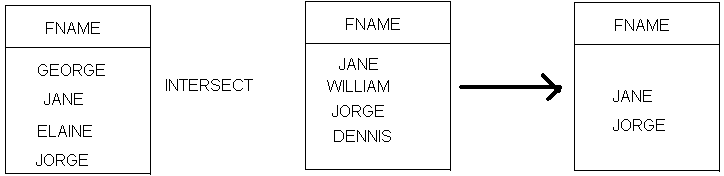
The relational operators have the property of closure, that is, the use of relational algebra operators on existing tables produces new relations. There is no need to examine the mathematical definitions, properties and characteristics of those relational algebra operators.

1. **UNION:**

It combines all rows from two tables, excluding duplicate rows. The tables must have the same attribute characteristics to be used in the UNION. When two or more tables share the same number of columns, when the columns have the same names, and when they share the same domains, they are said to be union-compatible.

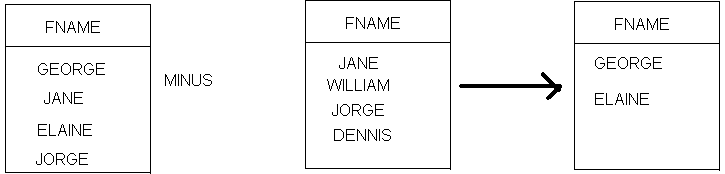


1. **INTERSECT:** This operator yields only the rows that appear in both tables.



1. **DIFFERENCE/MINUS:**

This operator yields all rows in one table that are not found in the other table; that is it subtracts one table from the other.



1. **PRODUCT:**

This operator yields all possible pairs of rows from two tables also known as **Cartesian product**. Therefore, if one table has six rows and the other table has three rows, the product yields a list composed of 6x3=18 rows.

1. **SELECT:**

It is also known as RESTRICT, yields values for all rows found in a table that satisfy a given condition. Select can be used to list all of the row values or it can yield only those rows values that match a specified criterion.

**Eg:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Rno** | **Name** | **Course** | **Fees** |
| 101 | Sami | Bsc | 9000 |
| 102 | Arun | Bcom | 8000 |
| 103 | Siva | Bsc | 8000 |
| 104 | Arvind | Bcom | 7000 |
| 105 | Rahul | Bsc | 8000 |
| 106 | Tony | Bsc | 9000 |

**SQL>Select \* from student where course=’Bsc’;**

|  |  |  |  |
| --- | --- | --- | --- |
| **Rno** | **Name** | **Course** | **Fees** |
| 101 | Sami | Bsc | 9000 |
| 103 | Siva | Bsc | 8000 |
| 105 | Rahul | Bsc | 8000 |
| 106 | Tony | Bsc | 9000 |

**SQL>Select \* from student where course=’Bsc’ and fees>8000;**

|  |  |  |  |
| --- | --- | --- | --- |
| **Rno** | **Name** | **Course** | **Fees** |
| 101 | Sami | Bsc | 9000 |
| 106 | Tony | Bsc | 9000 |

1. **PROJECTION:**

It yields all values for selected attributes. In other words, project yields a vertical subset of a table. The formal notation for a project operation is:

**π <attribute list> (<Relation>)**

**Eg: R**

|  |  |  |  |
| --- | --- | --- | --- |
| **Rno** | **Name** | **Course** | **Fees** |
| 101 | Sami | Bsc | 9000 |
| 102 | Arun | Bcom | 8000 |
| 103 | Siva | Bsc | 8000 |
| 104 | Arvind | Bcom | 7000 |
| 105 | Rahul | Bsc | 8000 |
| 106 | Tony | Bsc | 9000 |

**R1= (Name,Course) (R)**

|  |  |
| --- | --- |
| **Name** | **Course** |
| Sami | Bsc |
| Arun | Bcom |
| Siva | Bsc |
| Arvind | Bcom |
| Rahul | Bsc |
| Tony | Bsc |

1. **JOIN:**

The join operation combines two relations based on a joining condition. The relations must have at least one common attribute with the same underlying domain, and on such attributes a joining condition can be specified. The formal notation for a join operation is:

R <join condition>S

Where join condition is <attribute from R><comparison operator><attribute from S>

The comparison operator can be >, <, >=, <=, =, <> and it depends on attributes domain. If relation R has attributes A1,A2,… An and relations S has attributes B1,B2,…Bm and attribute Ai and attribute Bj have the same underlying domain, then we can define a join operation between relation R and relation S on a join condition between attribute Ai and Bj. The result is another relation T that contains all the tuples t such that t is the concatenation of a tuple r belonging to R and a tuple s belonging to S if the join condition is true. This type of join operation is also called **Theta-join**.

1. **DIVIDE:**

The divide operation uses one single column table(i.e. column “a”) as the divisor and one 2-column table (i.e columns “a” and “b”) as the dividend. The output of the divide operation is a single column with the values of column “a” from the dividend table rows where the value of the common column in both tables match.

**Advantages of Relational Algebra:**

1. The relational algebra has solid mathematical background. The mathematical background of relational algebra is the basis of many interesting developments and theorems. If we have two expressions for the same operation and if the expressions are proved to be equivalent, then a query optimizer can automatically substitute the more efficient form.
2. Relational algebra is a high level language which talks in terms of properties of sets of tuples and not in terms of for..loops.

**Limitations of Relational Algebra:**

1. The relational algebra cannot do arithmetic.
2. The relational algebra cannot sort or print results in various formats.
3. The relational algebra cannot perform aggregates.
4. The relational algebra cannot modify the database.

**Functional dependencies**

A functional dependency is a constraint that specifies the relationship between two sets of attributes where one set can accurately determine the value of other sets.

It is denoted as **X → Y**, where X is a set of attributes that is capable of determining the value of Y. The attribute set on the left side of the arrow, **X**is called **Determinant**, while on the right side, **Y**is called the **Dependent**.

**Example:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Roll\_no** | **Name** | **Dept\_name** | **Dept\_building** |
| 42 | Abc | CO | A4 |
| 43 | Pqr | IT | A3 |
| 44 | Xyz | CO | A4 |
| 45 | Xyz | IT | A3 |
| 46 | Mno | EC | B2 |
| 47 | Jkl | ME | B2 |

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

* roll\_no → { name, dept\_name, dept\_building },→  Here, roll\_no can determine values of fields name, dept\_name and dept\_building, hence a valid Functional dependency
* roll\_no → dept\_name , Since, roll\_no can determine whole set of {name, dept\_name, dept\_building}, it can determine its subset dept\_name also.
* dept\_name → dept\_building ,  Dept\_name can identify the dept\_building accurately, since departments with different dept\_name will also have a different dept\_building
* More valid functional dependencies: roll\_no → name, {roll\_no, name} ⇢ {dept\_name, dept\_building}, etc.

**Here are some invalid functional dependencies:**

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

**Armstrong’s axioms/properties of functional dependencies:**

1. **Reflexivity:**If Y is a subset of X, then X→Y holds by reflexivity rule  
   **For example,** {roll\_no, name} → name is valid.
2. **Augmentation:** If X → Y is a valid dependency, then XZ → YZ is also valid by the augmentation rule.

**For example,** If {roll\_no, name} → dept\_building is valid, hence {roll\_no, name, dept\_name} → {dept\_building, dept\_name} is also valid.→

1. **Transitivity**: If X → Y and Y → Z are both valid dependencies, then X→Z is also valid by the Transitivity rule.

**For example,** roll\_no → dept\_name & dept\_name → dept\_building, then roll\_no → dept\_building is also valid.

**Types of Functional dependencies in DBMS:**

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

**Trivial Functional Dependency**

* In **Trivial Functional Dependency**, a dependent is always a subset of the determinant.
* i.e. If **X → Y** and **Y is the subset of X**, then it is called trivial functional dependency

**For example,**

|  |  |  |
| --- | --- | --- |
| **Roll\_no** | **Name** | **Age** |
| **42** | **Abc** | **17** |
| **43** | **Pqr** | **18** |
| **44** | **Xyz** | **18** |

Here, **{roll\_no, name} → name** is a trivial functional dependency, since the dependent **name** is a subset of determinant set **{roll\_no, name}**  
Similarly, **roll\_no → roll\_no**is also an example of trivial functional dependency.

#### Non-trivial Functional Dependency

* In **Non-trivial functional dependency**, the dependent is strictly not a subset of the determinant.
* i.e. If **X → Y**and **Y** **is not a subset of X**, then it is called Non-trivial functional dependency.

**For example,**

|  |  |  |
| --- | --- | --- |
| **Roll\_no** | **Name** | **Age** |
| **42** | **Abc** | **17** |
| **43** | **Pqr** | **18** |
| **44** | **Xyz** | **18** |

Here, **roll\_no → name** is a non-trivial functional dependency, since the dependent **name** is **not a subset of**determinant**roll\_no**  
Similarly, **{roll\_no, name} → age** is also a non-trivial functional dependency, since **age** is**not a subset of {roll\_no, name}**

#### Multivalued Functional Dependency

* In **Multivalued functional dependency**, entities of the dependent set are **not dependent** **on each other.**
* i.e. If **a → {b, c}** and there exists **no functional dependency** between **b and c**, then it is called a **multivalued functional dependency.**

**For example,**

|  |  |  |
| --- | --- | --- |
| **Roll\_no** | **Name** | **Age** |
| 42 | Abc | 17 |
| 43 | Pqr | 18 |
| 44 | Xyz | 18 |
| 45 | Abc | 19 |

Here, **roll\_no → {name, age}**is a multivalued functional dependency, since the dependents **name** & **age** are **not dependent** on each other(i.e. **name → age**or**age → name doesn’t exist !**)

#### Transitive Functional Dependency

* In transitive functional dependency, dependent is indirectly dependent on determinant
* i.e. If **a → b** & **b → c**, then according to axiom of transitivity, **a → c**. This is a **transitive functional dependency**

**For example,**

|  |  |  |  |
| --- | --- | --- | --- |
| **Enrol\_no** | **Name** | **Dept** | **Building\_no** |
| **42** | **ABC** | **CO** | **4** |
| **43** | **PQR** | **EC** | **2** |
| **44** | **XYZ** | **IT** | **1** |
| **45** | **ABC** | **EC** | **2** |

Here, **enrol\_no → dept** and **dept → building\_no**,   
Hence, according to the axiom of transitivity, **enrol\_no → building\_no** is a valid functional dependency. This is an indirect functional dependency, hence called Transitive functional dependency.

**Normalization**

* **Normalization** is a database design technique that reduces data redundancy and eliminates undesirable characteristics like Insertion, Update and Deletion Anomalies.
* Normalization rules divide larger tables into smaller tables and links them using relationships.
* The purpose of Normalization in SQL is to eliminate redundant (repetitive) data and ensure data is stored logically.
* The inventor of the [relational model](https://www.guru99.com/relational-data-model-dbms.html) **Edgar Codd** proposed the theory of normalization of data with the introduction of the First Normal Form, and he continued to extend theory with Second and Third Normal Form. Later he joined Raymond F. Boyce to develop the theory of Boyce-Codd Normal Form.

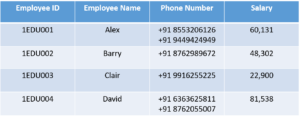
## Types of Normal Forms

There are the four types of normal forms:

* 1. First Normal Form
  2. Second Normal Form
  3. Third Normal Form
  4. Boyce Codd Normal Formal

# First Normal Form (1NF)

* In this Normal Form, we tackle the problem of atomicity. Here atomicity means values in the table should not be further divided.
* In simple terms, a single cell cannot hold multiple values. If a table contains a composite or multi-valued attribute, it violates the First Normal Form.



* In the above table, we can clearly see that the Phone Number column has two values. Thus it violated the 1st NF. Now if we apply the 1st NF to the above table we get the below table as the result.

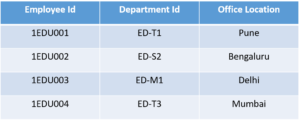


* By this, we have achieved atomicity and also each and every column have unique values.

### ****2nd Normal Form (2NF)****

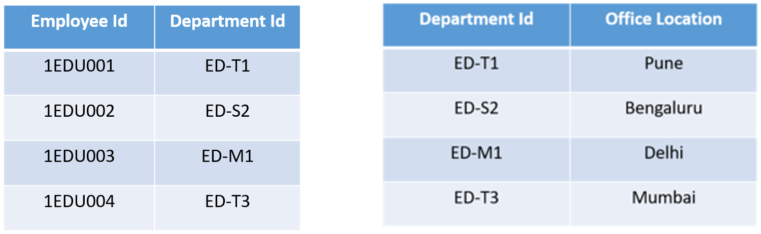
The first condition in the 2nd NF is that the table has to be in 1st NF. The table also should not contain partial dependency. Here partial dependency means the proper subset of candidate key determines a non-prime attribute.

**Eg:**



This table has a composite [primary key](https://www.edureka.co/blog/primary-key-in-sql/) **Emplyoee ID**, **Department ID**. The non-key attribute is **Office Location**. In this case, **Office Location** only depends on **Department ID**, which is only part of the primary key. Therefore, this table does not satisfy the second Normal Form.

To bring this table to Second Normal Form, we need to break the table into two parts. Which will give us the below tables:



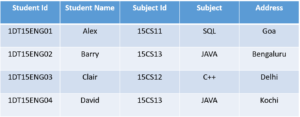
As you can see we have removed the partial functional dependency that we initially had. Now, in the table, the column **Office Location** is fully dependent on the primary key of that table, which is **Department ID**.

Now that we have learnt 1st and 2nd normal forms lets head to the next part of this Normalization in SQL article.

### ****3rd Normal Form (3NF)****

The same rule applies as before i.e, the table has to be in 2NF before proceeding to 3NF. The other condition is there should be no transitive dependency for non-prime attributes. That means non-prime attributes (which doesn’t form a candidate key) should not be dependent on other non-prime attributes in a given table. So a transitive dependency is a functional dependency in which X → Z (X determines Z) indirectly, by virtue of X → Y and Y → Z (where it is not the case that Y → X)

Let’s understand this more clearly with the help of an example:



In the above table, **Student ID** determines **Subject ID**, and **Subject ID** determines **Subject**. Therefore, **Student ID** determines **Subject** via **Subject ID.**This implies that we have a transitive functional dependency, and this structure does not satisfy the third normal form.

Now in order to achieve third normal form, we need to divide the table as shown below:

As you can see from the above tables all the non-key attributes are now fully functional dependent only on the primary key. In the first table, columns **Student Name, Subject ID** and **Address** are only dependent on **Student ID**. In the second table, **Subject** is only dependent on **Subject ID**.

**Unit – IV**

**History of SQL Standard**

* **SQL** stands for **“Structured Query Language”.**
* SQL can be pronounced as “SEQUEL”.
* SEQUEL means Simple/Structured English Query language.
* SQL is the international standard language for relational database management systems.
* SQL is considered a **fourth-generation language.**
* It is a special-purpose language used to define, access and manipulate data in RDBMS.
* SQL provides a set of statements for storing and retrieving data to and from a relational database.
* The original version called **SEQUEL** (Simple English Query Language) was designed by **IBM in mid-1970s.**
* SQL was introduced as a commercial database system in 1979 by Oracle Corporation.

**SQL commands**

SQL commands are instructions used to communicate with the database to perform specific task that work with data. SQL commands can be used not only for searching the database but also to perform various other functions. SQL commands are group into different categories depending on their functionality.

1. **Data Definition Language Commands**

These commands are used to create, modify and remove the database objects. The DDL commands are **create, alter, drop, rename** and **truncate.**

1. **Data Manipulation Language commands**

These commands are used for inserting, modifying and deleting data from the table. The DML commands are **insert, update** and **delete**.

1. **Data Retrieval Language commands/Data Query Language Commands**

This command is used for retrieving data from the database.The DQL command is **Select.**

1. **Data Control Language commands**

These commands are used for providing security. It means these commands are used to grant privileges to access the data one user to another. It is also used to revoke the privileges which are granted. The DCL commands are **Grant** and **Revoke**

1. **Transaction Control Language commands**

These commands are used to make the transaction either permanent or temporary. The TCL commands are **commit, rollback** and **savepoint**.

**Datatypes**

A datatype can be defined as the type of values to be stored. Oracle supports different types of data types.

|  |  |  |  |
| --- | --- | --- | --- |
| **S.No.** | **Name** | **Types** | **Description** |
| **1** | **Scalar** | **Number** | The number data type can store positive, negative numbers, zeros, fixed point numbers and float point numbers with a precision of 38. The decimal points which varies between -84 to 127. |
| **Char** | It is used when a fixed length character string is required. It can store alpha-numeric values. The column length of such a data type can vary between 1 to 2000 bytes. By default it is one byte. |
| **Varchar2** | It supports a variable length character string. It is also used to alpha-numeric values. The size of this data type ranges from 1 to 4000 bytes. While defining this data type we should specify the size. Using varchar2 saves disk space when compared to char. |
| **Date** | This data type is used to store data and time in a table. Oracle database makes its own format to store data in a fixed length of 7 bytes each for century, month, day, year, hour, minute and second. The default date format is “dd-mon-yy”. The valid date is from jan 1, 4712 BC to Dec 31, 4712 AD. |
| **Long Raw** | It is used to store images in an unstructured format i.e binary data of variable length. It holds a maximum of 2 GB. This data type should be used only once in a table if required. |
| **2** | **Composite** | **LOB’s** | It is known as Large Object data type. This can store unstructured information such as sound clips, video files, images, graphics, documents, etc, upto 4 GB in size. They allow efficient, random piece-wise access to the data. The LOB’s are BLOB, CLOB and BFILE. |

**SQL Functions**

SQL support different types of functions. They are

1. Numeric Functions
2. String Functions
3. Aggregate Functions
4. Date Functions
5. Conversion Functions

**NUMERIC FUNCTIONS:**

**1. abs():** This function is used to convert a negative value to positive value.

**Syntax:** abs(No.)

sql> select abs(100-120) from dual;

20

**2. Sqrt():** This function returns the square root of a given number.

**Syntax** – SQRT(No.)

**SQL>** select SQRT (81) from dual;

9

**3. Mod():** This function returns the modulus(remainder).

**Syntax**: mod(divident, divisor)

**SQL>** select mod(15,2) from dual;

1

**4. Power():** This function returns the power of given number as per the raised number

specified.

**Syntax:** power(number, raised number)

**SQL**> select power(4,3) from dual;

64

**5. round():** This function is used to round off the required number of decimals.

**Syntax** - round(No, No. of decimals)

**SQL**> select round(23.6789,2) from dual;

23.68

**SQL**> select round(23.6789,1) from dual;

23.7

**STRING FUNCTIONS**

**1. upper():** This function converts a string into upper case.

**Syntax:** upper('string')

**sql>** select upper('hello') from dual;

HELLO

**2. lower():** This function converts a string into lower case

**Syntax:** lower('string')

**sql>** select lower('HELLO') from dual;

hello

**3. initcap():** This function converts a string into initial caps.

**Syntax:** initcap('string')

**sql>** select initcap('computer education') from dual;

Computer Education

**4. ltrim():** This function is used to remove the unnecessary spaces or characters available to a string.

**Syntax:** ltrim('string','unnecessary char’)

**sql>** select ltrim('xyxycomputer','xy') from dual;

computer

**sql>** select ltrim('xyzxycomputer','xy') from dual;

zxycomputer

**5. rtrim:** This function is used to remove the unnecessary characters or spaces available on the right side of a string.

**Syntax:** rtrim('string','unnecessary char')

**sql>** select rtrim('computerxyxy','xy') from dual;

computer

**sql>** select rtrim('computerxyzxy','xy') from dual;

computerxyz

**AGGREGATE FUNCTIONS**

1. **Sum():** This function calculates sum of group of values

**Syn: sum(col\_name)**

**Eg: SQL>**select sum(sal) from emp;

**34999**

1. **Average():** This function returns the average of group of values

**Syn: average(col\_name)**

**Eg: SQL>**select average(sal) from dual;

**6999**

1. **Max():** This function returns the highest value from group of values

**Syn: max(col\_name)**

**Eg: SQL>**select max(sal) from emp;

**10000**

1. **Min():** This function returns the least value from group of values

**Syn: min(col\_name)**

**Eg: SQL>**select min(sal) from emp;

**1000**

1. **Count():** This function counts the number of values in a group

**Syn: count(col\_name)**

**eg: SQL>**select count(sal) from emp;

**5**

**DATE FUNCTIONS:**

**1. Sysdate:** This returns the system date in the format of dd-mon-yy.

sql> select sysdate from dual;

08-may-07

**2. last\_day():** This function returns the last date of a given date.

**Syntax:** last\_day(date)

sql> select last\_day(sysdate) from dual;

31-may-07

**3. next\_day():** This function returns the next date of a given date.

**Syntax:** next\_day(date,'dy')

sql> select next\_day(sysdate,'thu') from dual;

10-may-07

**4. add\_months():** This function is used to add/subtract the number of months to a given date.

**Syntax:** add\_months(date,no of months)

sql> select add\_months(sysdate,2) from dual;

08-jul-07

sql> select add\_months(sysdate,-2) from dual;

08-mar-07

**5. Months\_between():** This function returns the number of months present b/w two given dates.

**Syntax:** months\_between(date1,date2)

sql> select months\_between(sydate,to\_date('01-jan-07')) from dual;

4

**CONVERSION FUNCTIONS**

**1. To\_char():** This function converts a string/value into characters or in a specified format.

**Syntax:** to\_char(value,'format')

**Eg:**

**SQL>**select to\_char(sysdate,'dd-mon-yyyy') from dual;

08-may-2007

**SQL>** select to\_char(sysdate,'w-dy') from dual;

2-tue

**SQL>** select to\_char(sysdate,'q') from dual;

2

**2. To\_date():** This function is used to convert a string value into date format

**Syntax:** to\_date('value','format')

**Eg:**

**SQL>** select to\_date('10-mar-2007','dd-mon-yyyy') from dual;

10-mar-07

**3. To\_number():**  This function converts a string value into numeric.

**Syntax:** to\_number('value');

**Eg:**

**SQL>** select to\_number('10') + 10 from dual;

20

**5Q. Write about DDL commands with examples**

**Ans:** The DDL commands are **create, alter, drop, rename and truncate**

1. **CREATE:**

It is a Data Definition language. This command is used to create database objects.

**Syntax:**

Create table tname

(col1 datatype(width) [default] [constraint ],

col2 datatype(width) [default][ constraint ],

col3 datatype(width) [default][ constraint ],

col4 datatype(width) [default][ constraint ],

col5 datatype(width) [default] [constraint ],…..);

**Eg: SQL>** create table student

( rno number(4),

name varchar2(15),

gender char, doj date,

course varchar2(10), fees number(8));

**Table created.**

1. **ALTER:**

It is a Data Definition language command. This command is used to perform the following tasks. They are –

1. Used to add new columns in the existed table.
2. Used to add constraints to the columns of a table.
3. Used to increase or decrease the column width of table.
4. Used to modify the type of a column.
5. Used to remove the constraints
6. Used to enable or disable the constraints
7. Used to remove the columns of a table.

**Syntax:**

Alter table tname [add/modify/drop] [Enable/Disable] (col / constraint)

**SQL>** create table student(rno number(3), name varchar2(15));

Table created

**Eg:**

1. **SQL>** Alter table student add(sex char, doj date, course varchar2(15),

fees number(8));

**Table altered**

1. **SQL>** Alter table student modify(rno number(5));

**Table altered.**

1. **SQL>** Alter table student add(constraint t1 primary key(rno))

**Table altered**

1. **SQL>** Alter table student add(constraint t2 check(rno>=1 and rno<=100));

**Table altered.**

1. **SQL>** Alter table student disable constraint t1;

**Table altered**

1. **SQL>** Alter table student enable constraint t1;

**Table altered**

1. **SQL>** Alter table student drop constraint t1;

**Table altered**

1. **SQL>** Alter table student drop column sex;

**Table altered.**

1. **DROP:**

It is a Data Definition language command. This command is used to remove the database objects permanently from the database.

**Syntax:** drop table <tname>

**eg:** **SQL>** drop table student;

**Table Dropped**

1. **Rename:** This command is used to rename the database objects

**Syntax:** Rename <oldTname> to <NewTname>

**Eg: SQL>**Rename emp to employee;

**Table Renamed**

1. **Truncate:**

It is a data definition language command. This command is used to remove all the records of a table permanently.

**Syntax**: Truncate table <tname>

**Eg: SQL>** Truncate table student;

**Table Truncated.**

**DML commands**

The Data manipulation language commands are used for inserting, modifying and deleting records in a table. The data manipulation language commands are

1. Insert
2. Delete
3. update
4. **Insert:**

It is a Data Manipulation language command. This command is used to insert values into a table.

**Syntax-**1: ( Inserting values in all the columns of a table)

insert into tname values (&col1,'&col2','&col3','&col4',.....)

**Syntax-2:** (inserting in selected columns of a table )

insert into tname ( col1,col2,col3) values (&col1,'&col2','&col3)

**Eg:**

SQL> insert into student values(&rno,'&name','&sex','&doj','&course',&fees)

Enter value for rno = 101

Enter value for name = amjad

Enter value for sex = m

Enter value for doj = 10-jan-06

Enter value for course = bsc

Enter value for fees =6000

**1 row created**

**Inserting values only in selected columns of a table:**

**SQL>** insert into student (rno,name,course) values (&rno,'&name','&course')

Enter value for rno =103

Enter value for name = rahul

Enter value for course =bcom

**1 row created**

1. **DELETE**

It is a Data Manipulation Language command. This command is used to delete the rows/records of a table. The deleting of rows/records will be temporary, if we want to make them permanent deletion then commit command has to be implemented after deleting.

**Syntax: Delete Tname [where]**

**Eg:**

**SQL>** delete emp;

**SQL>** commit;

Records of emp table will be deleted permanently.

**SQL>** delete emp where sal>=2000;

This statement deletes the rows of emp table whose salary is >=2000

**SQL>** delete emp where sal>=2500 and sal<=5000;

This statement deletes the rows of emp table whose salary is >=2500

and <=5000

1. **UPDATE**

It is a Data Manipulation Language command. This command is used to update/ modify the existed data of a table.

**Syntax:**

Update tname set col1=value, [<col2>=value , <col3>= value... ] [where]

**Eg: Create a table and insert 5 to 10 records without tot, avgs and grade**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Rno** | **Name** | **M1** | **M2** | **M3** | **Tot** | **Avgs** | **Grade** |
| **101** | **Anand** | **54** | **55** | **55** |  |  |  |
| **102** | **Sunil** | **78** | **77** | **80** |  |  |  |
| **103** | **Sami** | **45** | **44** | **56** |  |  |  |
| **104** | **Tahseen** | **55** | **52** | **57** |  |  |  |
| **105** | **Mohsin** | **45** | **46** | **47** |  |  |  |
| **106** | **Srikanth** | **55** | **08** | **59** |  |  |  |
| **107** | **Arun** | **36** | **37** | **39** |  |  |  |

**Eg:**

**SQL>**update student set tot=m1+m2+m3, avgs=(m1+m2+m3)/3;

**7 rows updated**

**SQL>** update student grade='Distinction' where m1>=35 and m2>=35 and

m3>=35 and avgs>=75

**SQL>** update student grade='First' where m1>=35 and m2>=35 and m3>=35

And avgs>=60 and avgs<75

**SQL>** update student grade='Second' where m1>=35 and m2>=35 and

m3>=35 and avgs>=50 and avgs<60

**SQL>** update student grade='Third' where m1>=35 and m2>=35 and m3>=35

and avgs>=35 and avgs<50

**SQL>** update student grade='Fail' where m1<35 or m2<35 or m3<35;

**SQL>** Select \* from student;

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Rno** | **Name** | **M1** | **M2** | **M3** | **Tot** | **Avgs** | **Grade** |
| 101 | Anand | 54 | 55 | 55 | **164** | **54.67** | **Second** |
| 102 | Sunil | 78 | 77 | 80 | **235** | **78.33** | **Distinction** |
| 103 | Sami | 45 | 44 | 56 | **145** | **48.33** | **Third** |
| 104 | Tahseen | 55 | 52 | 57 | **164** | **54.67** | **Second** |
| 105 | Mohsin | 45 | 46 | 47 | **138** | **46** | **Third** |
| 106 | Srikanth | 55 | 08 | 59 | **122** | **40.67** | **Fail** |
| 107 | Arun | 36 | 37 | 39 | **112** | **37.33** | **Third** |

**Queries OR Select Statement**

Queries are one of the things that make databases so powerful. A **“Query”** is nothing but extracting either the entire data/records or required data from the database. A query can be performed by using Data Query language command called **“Select”.**

The **Select statement** is used to query or retrieve data from a table in the database. A query may retrieve information from specified columns or from all of the columns in the table. The Query can be performed either on a single table or on multiple tables. The select statement is used to perform either simple or complex queries.

**Syntax:**

Select [distinct] [\*/<col1,col2,....>] [alias names] from <tname1> [,<tname2>,.]

[where] [order by] [group by] [having] [set operators] [joins] [sub-queries]

**Eg:**

* 1. **SQL>** select \* from tab;

This statement displays list of objects available under a current user.

* 1. **SQL>**select \* from emp;

This statement displays the rows of a emp table

* 1. **SQL>** select empno,ename,job,sal from emp;

This statement displays the empno,ename,job and salary details

**Distinct:**

The distinct command in select statement allows the user to display unique data of the table

**Eg:**

1. **SQL>** select distinct empno from emp;
2. **SQL>** select distinct empno,ename from emp;

**Alias names:**

Alias names are nothing but the temporary names which can be set for the columns of a table while performing query.

**Eg:**

**SQL>**select empno as Employee\_No, ename as Emp\_Name, sal as Salary from emp;

**Employee\_No Emp\_Name Salary**

101 kiran 5000

102 anand 5600

**SQL>**select empno,ename,sal, sal\*5/100 as bonus from emp;

**empno ename sal bonus**

101 anand 5000 250

**SELECT... WHERE CLAUSE**

The **where clause** in select statement allows the user to retrieve the data from the table as per the condition specified

**Eg:**

1. SQL> select \* from emp;
2. SQL> select \* from emp where sal>=5000;
3. SQL> select \* from emp where sal>=2500 and sal<=5000;
4. SQL> select \* from emp where sal between 2500 and 5000

**SELECT .. ORDER BY**

The order by clause in select statement allows the user to retrieve the data of a database either in ascending order or in descending order.

**Eg:**

**SQL>** select \* from emp order by empno ;

**SQL>** select \* from emp order by empno desc;

**SQL>** select \* from emp where dept='computers' order by empno;

**SQL>** select \* from emp where dept='computers' order by empno desc;

**SELECT .. GROUP BY**

**The group by clause** is used to divide the rows of a table into smaller groups based on specific column. Once the groups made, then group functions can be used to return summary information for each group. **The group by clause** in select statement is used to combine a group of rows based on the values of a particular column or expression.

**Eg**

**Empno ename job sal**

101 anand medical 5000

102 rahul computers 6500

103 ravi medical 2500

104 sumith medical 3000

105 rao executive 2500

106 arvind steno 3000

**Eg:**

**SQL>**select job from emp group by job

**job**

medical

computers

executive

steno

**SQL>**select job,count(job) from emp group by job;

**job count(job)**

medical 3

computers 1

executive 1

steno 1

SQL>select job,count(job),sum(sal) from emp group by job;

**job count(job) sum(sal)**

medical 3 10500

computers 1 6500

executive 1 2500

steno 1 3000

SQL>select job,count(job),sum(sal),avg(sal) from emp group by job;

**job count(job) sum(sal) avg(sal)**

medical 3 10500

computers 1 6500

executive 1 2500

steno 1 3000

SQL>select job,count(job),sum(sal),max(sal) from emp group by job;

**Job count(job) sum(sal) max(sal)**

medical 3 10500 5000

computers 1 6500 6500

executive 1 2500 2500

steno 1 3000 3000

SQL>select job,count(job),sum(sal),max(sal),min(sal) from emp group by job;

**job count(job) sum(sal) max(sal) min(sal)**

medical 3 10,500 5000 2500

computers 1 6,500 6500 6500

executive 1 2,500 2500 2500

steno 1 3,000 3000 3000

**SELECT .. GROUP BY .. HAVING Clause**

The **HAVING Clause** is very similar to **where clause** but the **Having clause** should always be used with Group By clause. **The Having clause** is also used to extract the required information from the table based on the condition specified.

**Eg:**

**SQL>**select job,count(job),sum(sal),max(sal),min(sal) from emp group by job;

**job count(job) sum(sal) max(sal) min(sal)**

medical 3 10,500 5000 2500

**SET OPERATORS**

The set operators are used to combine information of similar type from one or more than one table. The set operators look similar to SQL joins although there is a big difference. SQL joins tends to combine columns from different tables, whereas SQL set operators combine rows from different queries. There are different types of set operators. They are -

1. Union
2. Union All
3. Intersect
4. Minus
5. **UNION:**

This set operator is used to join the outputs of two or more queries into a single set of rows and columns having distinct records.

**Eg:**

**SQL>** select deptno from emp union select deptno from dept;

1. **UNION ALL:**

This set operator is used to join the outputs of two or more queries into a single set of rows and columns without the removal of any duplicates.

**Eg:**

**SQL>** select deptno from emp union all select deptno from dept;

1. **INTERSECT:**

This set operator is used to retrieve the data which is common in both tables.

**Eg:**

**SQL>** select deptno from emp intersect select deptno from dept;

1. **MINUS:**

This set operator is used to retrieve the data one table which is not available in other table.

**Eg:**

**SQL>** select deptno from emp minus select deptno from dept;

**SQL>** select deptno from dept minus select deptno from emp;

**SQL JOINS**

The joins are used to retrieve the data which is scattered in different tables. The joins are of different types. They are -

1. Simple join
2. Self join
3. Outer join
4. Cartesian Join
5. **Simple joins:**

In simple joins the rows of different table can be retrieved based on its respective conditions. The simple joins are of two types. They are -

1. Equi Join
2. Non-Equi Join
3. **Equi-join:**

In Equi-Join the rows of different tables can be retrieved based on equalities. The Equi-joins can be performed by using "=" operator

**Eg:**

**SQL>** select x.\* , y.\* from emp x, dept y where x.deptno=y.deptno;

**SQL>** select x.deptno,ename,sal,y.deptno,dname,loc from emp x, dept y

where x.deptno = y.deptno;

1. **Non Equi joins:**

In non equi-joins the rows of different tables can be retrieved based on non-equalities. The Non-equi joins can be performed by using relational operators like >,< and <>

**Eg:**

**SQL>** select x.\* , y.\* from emp x, dept y where x.deptno<>y.deptno;

**SQL>** select x.deptno,ename,sal,y.deptno,dname,loc from emp x, dept y

where x.deptno > y.deptno;

1. **Self Joins:**

The Self join is mainly preferred to join a table with itself. This means that each row of the table is combined with itself and with every other row of the table. The Self join can be viewed as a join of two copies of the same table. The table is not actually copied, but SQL performs the command as though it were.

**Eg:**

**SQL>**Select x.ename || 'Worker of '|| y.ename from emp x, emp y where

x.mgr= y.empno;

1. **Outer Joins:**

This SQL Join condition returns all rows from both tables which satisfy the join condition along with rows which do not satisfy the join condition from one of the tables. In outer joins the rows of different tables can be retrieved. This is similar to the combination of Equi and non Equi joins. It means the rows of different tables can be retrieved either it equalities or non-equalities. The outer joins can be performed by using "(+)"

**Eg:**

**SQL>** select x.deptno,ename,job,sal,y.deptno,dname,loc from emp x, dept y where x.deptno (+) = y.deptno;

**SQL>** select x.deptno,ename,job,sal,y.deptno,dname,loc from emp x, dept y where x.deptno = y.deptno (+);

**SUB-QUERIES AND CORRELATED QUERIES**

1. Sub-query means placing an inner query within a WHERE or HAVING clause of outer query.
2. The inner query provides values for the search condition of the outer query. Such queries are referred to as sub-queries or nested sub-queries, and may be nested multiple times.
3. A statement containing sub-query is called a parent statement.
4. A sub-query is normally expressed inside parentheses.
5. The first query in the SQL statement is known as the outer query.
6. The query inside the SQL statement is known as the inner query.
7. The inner query is executed first.

**Advantages:**

1. Sub queries allow a developer to build powerful commands out of simple ones
2. The nested sub query is very useful when you need to select rows from a table with a condition that depends on the data in the table itself.

**Eg:**

**SQL>**Select ename from emp where deptno =

(select deptno from emp where ename=’MILLER’)

**SQL>**Select ENAME from emp where sal = (select max(sal) from emp);

**CORRELATED SUBQUERIES:**

A query is called correlated sub query when both the inner query and the outer query are interdependent. For every row processed by the inner query, the outer query is processed as well. The inner query depends on the outer query before it can be processed.

A correlated sub-query is nested sub-query which is executed once for each ‘candidate row’ considered by the main query and which on execution uses a value from a column in the outer query. This causes the correlated sub-query to be processed in a different way from the ordinary nested sub-query.

**Eg:SQL>**Select \* from emp x where sal > (select avg(sal) from emp where

deptno=x.deptno);

**Views or Virtual table**

1. View is a database object.
2. View is a **virtual table**,
3. A View is a private SQL area where query results can be stored.
4. A view contains no data of its own but it is like a window through which data from tables can be viewed or changed.
5. The tables on which a view is based are called **base tables**.
6. Base tables might in turn be actual tables or might be views themselves.
7. All operations performed on a view actually affect the base table of the view. DML operations on a view like insert, update, delete affects the data in the original table upon which the view is based.

**Syntax: Create or replace view <view\_name> as [select]**

**Advantages of Views**

1. A view restricts access to the database because the view can display selective columns from the table.
2. As a view does not store any data, the redundancy problem does not arise
3. Critical data in the base table is protected as access to such data can be controlled using views by not selecting those columns during the creation of a view.
4. Views allow users to make simple queries to retrieve the results from complicated queries.

**Eg:**

**SQL>**Create view v1 as select \* from emp where sal>=3000;

**SQL>**Create view v2 as select deptno, ename, job, sal, dname, loc from emp x,

dept y where x.deptno = y.deptno;

**SQL>**Update v1 set sal=sal+1000;

**SQL>**Drop view v1;

**Unit – V**

**PL/SQL**

* + - * **PL/SQL** stands for **Procedural structured query language**.
      * A procedure can be defined as block of statements which performs a particular task.
      * It combines the data manipulation power of SQL and procedural power of standard programming languages.
      * It was developed by Oracle Corporation in the early 1990s to enhance the capabilities of SQL. So, it integrates well with SQL\*Plus and other application development products of Oracle.
      * PL/SQL is the superset of SQL. So, it fully supports SQL DML, TCL commands and SQL data types.
      * PL/SQL also has Boolean and composite data types to handle complex data.
      * It supports standard programming features such as control statements, procedures and functions.
      * PL/SQL is not a case-sensitive language. i.e, you can use both lowercase and uppercase letters in a PL/SQL code.

**PL/SQL Elements**

Like every other programming language, PL/SQL has a set of elements. They are

1. Identifiers
2. Reserve Words
3. Delimiters
4. character set
5. Literals
6. Lexical units
7. **Identifiers:**

Identifiers are used to name PL/SQL program items and units, which include constants, variables, exceptions, cursors, cursor variables, sub-programs and packages. To define identifiers the following rules to be followed.

1. The first character of identifier name should always begins with alphabet.
2. In between identifier name there should not any special symbol except underscore
3. Duplication identifier names cannot be defined
4. Reserved words should not be defined as identifiers
5. Identifier name should not exceed 30 characters
6. **Reserve Words:**

The words which are already existed in PL/SQL are known as reserved words and these words will have special syntactic meaning and so it cannot be redefined.

1. **Character set:**

PL/SQL programs are written as lines of text using a specific set of characters. The PL/SQL character set includes:

1. The upper and lower case letters
2. The numerals 0..9
3. Tabs, spaces and carriage returns
4. The special symbols
5. **Delimiters:**

Delimiters are simple or compound symbols that have special meaning in PL/SQL.

|  |  |
| --- | --- |
| **Symbol** | **Meaning** |
| + | Addition |
| - | Subtraction |
| \* | Multiplication |
| / | Division |
| = | Comparison operator |
| @ | Remote access indicator |
| ; | Semi-colon |
| <> | Not Equals to |
| || | Concatenation |
| - - | Single line comment indicator |
| := | Assignment operator |

1. **Literals**

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

1. Character Literals
2. Numeric Literals
3. **Character Literals:**

Character literals include all the printable characters in the PL/SQL character set: letters, numerals, spaces and special symbols. Character and Date literals must be enclosed in string quotes.

1. **Numeric literals:**

Numeric literals are of two types: Integer and Real. An integer literal is an optionally signed whole number without a decimal point.

**Eg:** 23,4545,-343,0,-23

**A real literal** is an optional signed whole or fractional number with a decimal point.

**Eg:** 6.453,34.23,-23.34

1. **Lexical Unit:**

A line of PL/SQL text contains groups of characters known as lexical units, which can be classified as follows:

1. Delimiters
2. Identifiers
3. Literals
4. Comments

**Structure of PL/SQL Block**

PL/SQL is a block-structured language. This means that programs can be divided into logical blocks. A PL/SQL block consists of up to four sections. They are

1. **Documentation Section:**

The documentation section comprises a set of comment lines giving **the name of program, author name, date on which the program is created and other details, which the programmer would like to refer at a later stage.** The comment can begin with **double minus symbols.**

1. **Declarative Section**

The declarative section of a PL/SQL block starts with the **keyword DECLARE**. This section is used to declare any place holders like **variables, constants, records and cursors,** which are used to manipulate data in the executable section.

1. **Executable Section**

The executable section of a PL/SQL block **starts with thekeyword BEGIN** and **ends with END**. This is the section where the program logic is written to perform any task. The programmatic constructs like loops, conditional statement and SQL statements from the part of the executable section

1. **Exception Handling Section**

The exception section of a PL/SQL block starts with the **keyword EXCEPTION.** Any errors in the program can be handled in this section, so that the PL/SQL blocks terminates gracefully. If the PL/SQL block contains exceptions that cannot be handled, the block terminates immediately with errors.

**Syntax:**

**DECLARE**

**Variable Declarations;**

**[Exception Declaration];**

**[Cursor Declarations];**

**BEGIN**

**Executable statements**

**EXCEPTION**

**Exception Handlers**

**END;**

**Control Structures**

The control structures or statements are of two types. They are

1. Conditional statements
2. Iterative Statements
3. **Conditional Statements**

The conditional statements are used to control the flow of execution of statements of a program. The conditional statements are –

1. simple if
2. if.. then else
3. if.. then elsif
4. Case Expression
5. **Simple if**

This conditional statement is used to execute the statements only when the condition is true otherwise it skips the if statement

**Syntax:**if (expression) then

statement1;

statement2;

end if;

1. **if.. then else**

This conditional statement is used to execute the true statements only when the condition is true otherwise it executes the false statements.

**Syntax:**if (expression) then

statement1;

statement2;

else

statement1;

statement2;

end if;

1. **if.. then elsif:**

This conditional statement is also called as branching statement or ladder statement. This conditional statement is used to execute the statements based on its respective condition

**Syntax:** if(expression1) then

statement1;

statement2;

elsif(expression2) then

statement1;

statement2;

elsif(expression3) then

statement1;

statement2;

end if;

1. **Case Expressions**

A case expression selects a result and returns it. To select the result, the **CASE** expression uses a selector, an expression whose value is used to select one of several alternatives. The selector is followed by one or more WHEN clauses, which are checked sequentially. The value of the selector determines which clause is executed. If the value of the selector equals the value of a WHEN-clause expression, that WHEN clause is executed.

**Syn:**CASE <expression>

WHEN search\_condition1 then result1

WHEN search\_condition2 then result2

….

WHEN search\_condition\_N then result

Else

resultN+1

END;

**Loops**

Executing a statement or group of statements for a repeated number of times, it is called as a loop. Oracle contains different types of loops. They are -

1. For .. loop
2. while .. loop
3. Loop .. end loop
4. **for .. loop:**

In this loop a statement or group of statements can be executed for a repeated number of times only when the condition is true otherwise it terminates the loop.

**Syntax:** for <index> in [reverse] startingvalue .. stopingvalue loop

statement1;

statement2;

end loop ;

1. **While ..loop:**

It is also called as a conditional loop. In this loop a statement or group of statements can be executed for a repeated number of times only when the condition is true otherwise the loop will be terminated.

**Syntax**: initial value;

while(expression) loop

statement1;

statement2;

changingvalue;

end loop;

1. **Loop .. End loop**

It is also called as a conditional loop. In this loop a statement or group of statements can be executed first and then the condition will be verified. In this loop the statements can be executed for repeated number of times only when the condition is false otherwise the loop will be terminated.

**Syntax:** initialvalue;

loop

statement1;

statement2;

changing value;

exit when <expression>

end loop;

**Unconditional Statements**

1. **Goto:** The PL/SQL goto statement is a sequential control structure available in Oracle. The goto statement immediately transfers program control unconditionally to a named statement label or block label. The statement or label name must be unique in the block.

**Syn:** Goto <<label\_name>>;

Where label\_name is the name of the label identifying the target statement

1. **Null Statement:**

Generally when you write a statement in the program, you want it to do something but in some cases you want to tell PL/SQL to do nothing and in such cases, NULL statement can be used. The NULL statement does nothing other than pass control to the next statement.

**CURSORS**

Oracle allocates an area of memory known as context area for the processing of SQL statements. The context area contains information necessary to complete the processing, including the number of rows processed by the statement, a pointer to the parsed representation of the statement.

A cursor is a handle or pointer to the context area. Through the cursor, a PL/SQL program can control the context the context area and what happens to it as the statement is processed. The cursors are classified into 2 types. They are –

* 1. Implicit cursors
  2. Explicit cursors

**Implicit Cursors:**

* PL/SQL implicitly declares cursors for all SQL data manipulation statements, including queries that return one rows. For queries that return more than one row, we should use explicit cursors to access the rows individually.
* Implicit cursor attributes can be used to access information about the most recently executed SQL statement. The most recently executed SQL statement is referred as SQLCURSOR.
* An implicit cursor need not be created. An implicit cursor should not be created, opened or closed. Here everything will be look after by the SELECT statement (Query) itself.

**Explicit Cursors:**

An explicit cursor is one in which the cursor name is explicitly assigned to the select statement. An explicit cursor allows the user to perform transactions on multiple rows at a time. Here the user has to create the cursor and it should be controlled by the user itself.

**CURSOR:** This procedure is used to create a cursor.

**Syntax:** Cursor <cur\_name> is [select]

Once a cursor is created, it has to be controlled by using the following commands. They are -

1. Open
2. Fetch
3. Close

**Open:** This command is used to open the cursor which was created.

**Syn: Open <cursor\_name>**

**Fetch:** This command is used to fetch the data to a variable from a cursor.

**Syn: Fetch cursor\_name into var**

**Close:** This command is used to close the cursor which was opened.

**Syn: close <cur\_name>**

**Attributes:**

The cursor can be controlled by using the following attributes. They are -

1. %notfound
2. %found
3. %rowcount
4. %isopen

**%NOT FOUND:**

This attribute allows the user to find whether a cursor contains rows or not. if a cursor contains rows then %notfound attribute returns true otherwise it return false

**%FOUND:**

This attribute allows the user to find whether a cursor contains rows or not if a cursor contains rows then it returns true otherwise it returns false.

**%ROWCOUNT:**

This attribute finds to how many rows a transaction is performed.

**%ISOPEN:**

This attribute allows the user to find whether a cursor is opened or not. If a cursor is open it returns true otherwise it returns false.

**TRIGGERS**

A database trigger is a stored procedure that is fired when an insert, update or delete statement is issued against the associated table. Database triggers can be used for the following purposes.

1. To generate data automatically
2. To enforce complex integrity constraints
3. To customize complex security authorizations
4. To maintain replicate tables
5. To audit data modifications

**Parts of trigger:**

A database trigger has three parts namely a trigger statement, a trigger body and a trigger restriction.

**Syntax:**

Create or replace trigger trig\_name [before/after] [insert/update/delete] on <tname>[for each row/ for each statement] [when clause]

DECLARE

variable declarations;

BEGIN

statement1;

statement2;

END;

**Trigger statement:**

The trigger statement specifies the DML statements like update, delete and insert and it fires the trigger body. It also specifies the table to which the trigger is associated.

**Trigger body:**

It is a PL/SQL block that is executed when a triggering statement is issued.

**Trigger restriction:**

Restrictions on a trigger can be achieved using the WHEN clause. When clause is evaluated for each row that is affected by the trigger

**Types of triggers**

1. before

2. after

3. for each row

4. for each statement

**Before / after option**

The before/after options can be used to specify when the trigger body should be fired with respect to the triggering statement. If the user includes a before option, then, oracle fires the trigger before executing the triggering statement. On the other hand, if after is used, then oracle fires the trigger after executing the triggering statement.

**For each row/ statement**

The for each row/ statement option when included in the **'create trigger'** syntax specifies that the trigger fires once per row. By default, a database trigger fires for each statement.

Using a combination of the above options, we can assign 12 triggers to a database table. Only one trigger of each type can be assigned to a table.

1. Before update row/ statement

2. Before delete row/ statement

3. Before insert row/ statement

4. After insert row / statement

5. After update row / statement

6. After delete row / statement

To raise uses defined error message we can use the command called 'raise\_application\_error'.

**Syntax**: raise\_application\_error(error\_no,'err\_msg')

The error\_number ranges from -20000 to -20999 and the error\_msg can be a string.

**Procedures**

A procedure is a sub-program that performs a specific action it is called as a procedure. A procedure will never return a value to a calling procedure.

A procedure has **two parts**, namely, **specifications** and **body**. The procedure specification begins with the keyword **procedure** and ends with the **procedure name** or **parameter list**. The procedure body begins with the keyword **“is”** and ends with the keyword **end**. It can also include **declarative, executable** and **exceptional parts** within the keywords are and end.

**Syntax:**

Create or replace procedure proc\_name [parameter list] is

Local declarations;

Begin

Executable statements;

[Exception]

End;

**The parameter list** can hold any of the following modes, **namely, in out and inout.** These parameter modes can be used within any sub program.

1. **In Parameter:**

The **in parameter mode** is used to pass values to the subprogram when invoked. It acts like a constant and therefore it cannot be assigned a value.

1. **Out Parameter:**

The **out parameter mode** is used to return values to the caller of a subprogram. Since the initial value for an out parameter is undefined, its value can be assigned to another variable.

1. **In out Parameter:**

The **in out parameter** is used to pass initial values to the sub program when invoked and it also returns updated values to the caller. An **in out parameter** acts like an initialized variable and therefore can be assigned to other variables or to itself.

**Note:**

Once the procedure is created it can be called or executed by using the keyword **“Exec” followed by procedure name**

**Syn:** Exec proc\_Name(arg1,arg2)

**Functions**

A function is a sub program that computes a value. **A block of statements which performs a particular task is called as a function.** A function will always returns a value to a calling function.

**Syntax:**

Create or replace function fun\_name(arguments) return datatype is

Local declarations

Begin

Executable statements

[Exception]

Exception Handlers  
End;