



STUDY MATERIAL



**Detartment of
Computer Science**



STUDENT NAME: _____

SUBJECT _____ CLASS: _____

VILLAGE: _____

PH NO: _____

COLLEGE: _____

Unit-1

Database Management System

1. Explain the concepts of DBMS.

A. Data: Data is the collection of raw facts and figures.

B. Information: The meaningful form of data or processed data is called as information.

C. Field: A field is the lowest level of data item of an entity which is alternatively called as attribute of that entity. A field is a single item of data within a database or software program.

For ex, a field may be a customer name, address, or phone number.

D. Record: Record is the collection of related fields or data.

Ex: S.no Sname Marks

1001 Raj 950

E. Database File: File is a collection of records having the same set of fields arranged in the same sequence.

Ex: Student_Profile, Student_marks, etc.

F. Database: Database is a collection of inter-related data which helps in efficient retrieval, insertion, and deletion of data from database and organizes the data in the form of tables, views, schemas, reports etc.

For Example, university database organizes the data about students, faculty, and admin staff etc.

G. DBMS: The software which is used to manage database is called Database Management System (DBMS). It acts as the interface between users and the database itself. It is a record keeping system. It allows the data to be stored, maintained, manipulated, and retrieved.

For Example, MySQL, Oracle etc. are popular commercial DBMS used in different applications.

It consists of DBMS utilities, database, and data dictionary. Its responsibilities are:

- Multiuser Access Control
- Security Management
- Backup and Recovery Management

2. What is File-based system? Explain the Drawbacks of file systems?

A File system is used to store, manage, and retrieve data. A file system can be either manual or computerized. File system allows storing data of different departments in different data files.

A. Manual File System: In this system data can be stored and maintained in books, ledgers, and journals etc.

B. Computerized File System: In this system data can be stored and maintained in computers. Data can be entered and modified with high speed. Data can be shared partially.

In File System-

- ❖ Files are stored in different places.
- ❖ There is no relation between the files.
- ❖ Files are developed by different persons on different locations.

Drawbacks of file systems:

- A. Uncontrolled data redundancy
- B. Inconsistency of data
- C. Difficulty in Accessing Data
- D. Limited data sharing
- E. Poor enforcement of standards
- F. Concurrent access
- G. Low programmer productivity
- H. Security Problems
- I. No Backup & Recovery facilities

A. Uncontrolled redundancy of data: Redundancy means repeating data. It is possible that the same information may be duplicated in different files.

This leads to data redundancy results in memory wastage.

B. Inconsistency of data: Because of data redundancy there is a possibility of entering same data differently in different sub-systems. This leads to data inconsistency.

C. Difficulty in Accessing Data: Accessing data is not convenient and efficient in file processing system.

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

E. Poor enforcement of standards: Different applications are developed by different persons, each person will follow its own standards of defining field name, field width, and field type. This will create a serious difficulty while modifying programs, sometimes there will be serious errors due to mismatch of fields.

F. Concurrent Access: Multiple users are allowed to access data simultaneously. This is for the sake of better performance and faster response. But File system does not support multiple users to access the data at a time.

G. Low programmer productivity:- Programmer productivity is a measure of time taken to develop an application. It is inversely proportional to developing time. Because of File system like data redundancy, inflexibility, and poor enforcement standards the programmer productivity will become lower.

H. Security Problems: File System has no password protection to the data. So unauthorized persons can access the data. So it has limited security.

I. No Backup & Recovery Facilities:- Backup means the original data will be stored in different device. If the original data was removed then the backup

3. **Various Objectives of Database Management**

System: A. Mass Storage:

DBMS can store a lot of data in it. It can store thousands of records in it and one can fetch all that data whenever it is needed.

B. Removes Duplicity:

If you have lots of data then data duplicity will occur for sure at any instance. DBMS guarantee it that there will be no data duplicity among all the records. While storing new records, DBMS makes sure that same data was not inserted before.

C. Multiple Users Access:

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

D. Data Protection:

DBMS gives a master level security to their data. No one can alter or modify the information without the privilege of using that data. Information such as bank details, employee's salary details and sale purchase details should always be kept secured. Also all the companies need their data secured from unauthorized use.

E. Data Back up and recovery:

Sometimes database failure occurs so there is no option like one can say that all the data has been lost. There should be a backup of database so that on database failure it can be recovered. DBMS has the ability to backup and recover all the data in database.

F. Everyone can work on DBMS:

There is no need to be a master of programming language if you want to work on DBMS. Any accountant who is having less technical knowledge can work on DBMS.

G. Integrity:

Integrity means your data is authentic and consistent. DBMS has various validity checks that make your data completely accurate and consistence.

H. Platform Independent

One can run dbms at any platform. No particular platform is required to work on database management system.

4. **Evaluations of DBMS:**

The following are some of the major evaluations of DBMS

- ❖ 1960 - First DBMS designed by Charles Bachman at general electronics known as integrated data source(IDS).
- ❖ 1960 - IBM developed the information management system called IBM.
- ❖ 1970 - Edgar Codd from IBM create a relational data model
- ❖ 1976 - Peter Chen presented the Entity relationship model
- ❖ 1980 - SQL developed by IBM became the standard query language for databases becomes a widely accepted database component

- ❖ 1985- Object-oriented DBMS develops.
- ❖ 1990s- Incorporation of object-orientation in relational DBMS.
- ❖ 1991- Microsoft ships MS access, a personal DBMS and that displaces all other personal DBMS products.
- ❖ 1995: First Internet database applications
- ❖ 1997: XML applied to database processing. Many vendors begin to integrate XML into DBMS products.

5. **Classification of Database Management System:**

A. **Based on the number of users**

Single user: As the name itself indicates it can support only one user at a time. The user may design, maintain and write the database programs.

Multiple users: It supports multiple users concurrently.

For example a student in the college should have the database containing his information. It must be accessible to all the departments related to him.

B. **Based on the cost**

Low cost DBMS : The cost of these systems vary from \$100 to \$3000.

Medium cost DBMS : Cost varies from \$10000 to \$100000.

High cost DBMS : Cost pf these systems are usually more than \$100000..

C. **Based on the access**

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

Sequential access – One after the other.

Direct access- all at once

D. **Based on the usage**

Online transaction processing (OLTP) DBMS – They manage the operational data. Database server must be able to process lots of simple transactions per unit of time. Transactions are initiated in real time, in simultaneous by lots of user and applications hence it must have high volume of short, simple queries.

Online analytical processing (OLAP) DBMS – They use the operational data for tactical and strategic decision making. They have limited users deal with huge amount of data, complex queries.

Big data and analytics DBMS – To cope with big data new database technologies have been introduced. One such is NoSQL (not only SQL).

Multimedia DBMS – Stores data such as text, images, audio, video and 3D games which are usually stored in binary large object.

GIS DBMS – Stores and queries the spatial data.

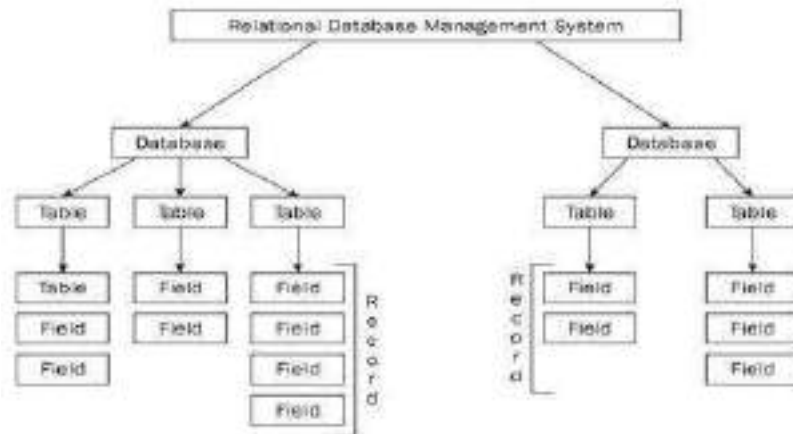
Sensor DBMS – Allows to manage sensor data, bio-metric and telematics data.

Mobile DBMS – Runs on the smartphones, tablets. It Handles the local

E. Based on the data model

Relational database – This is the most popular data model used in industries. It is based on the SQL. They are table oriented which means data is stored in different access control tables, each has the key field whose task is to identify each row.

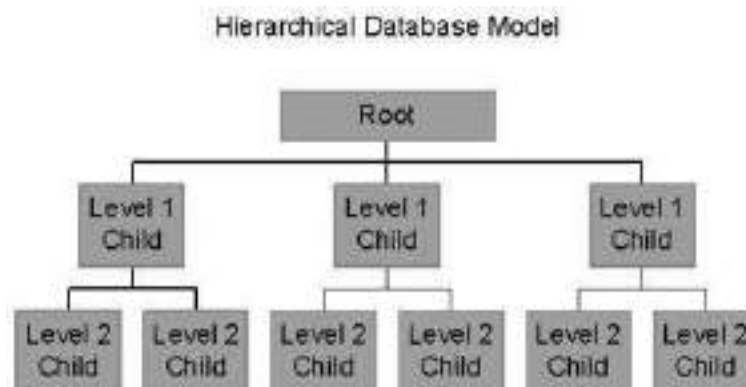
Examples are MYSQL(Oracle, open source), Oracle database (Oracle), Microsoft SQL server(Microsoft) and DB2(IBM).



Object oriented database – The information here is in the form of the object as used in object oriented programming. It adds the database functionality to object programming languages. It requires less code, use more natural data and also code bases are easy to maintain.

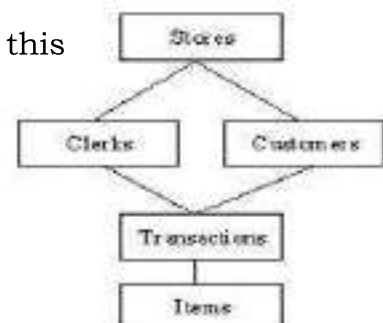
Examples are ObjectDB (ObjectDB software).

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



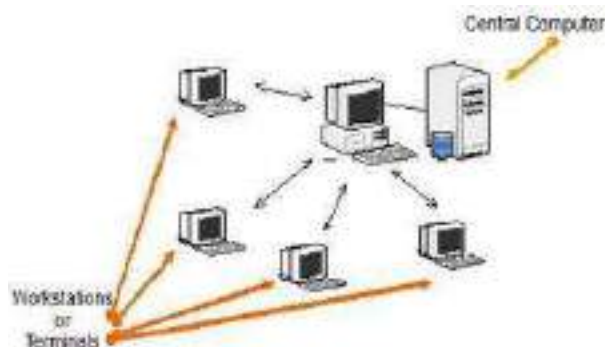
Examples are IMS(IBM), Windows registry (Microsoft). **Network database** – Mainly used on large digital computers. If there are more connections, then this database is efficient. They are similar to hierarchical database they look like interconnected network of records.

Examples are CA-IDMS (COMPUTER associates), IMAGE(HP).



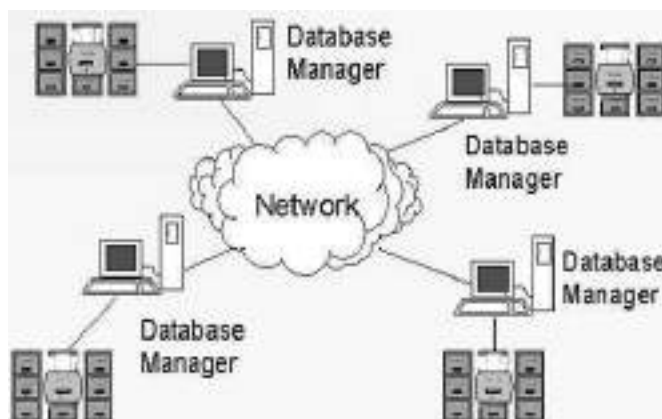
F. Based on the sites over which network is distributed

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



Parallel network database system – This system has the advantage of improving processing input and output speeds. Majorly used in the applications that have query to larger database. It holds the multiple central processing units and data storage disks in parallel.

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



Client-server database system:

Clients are generally the personal computers or workstations whereas servers are the large workstations, mini range computers or a main frame computer system. The applications and tools of the DBMS run on the client platforms and the DBMS software on the server. Both server and client computers are connected over the network.

6. What is data model? Explain various types of database models?

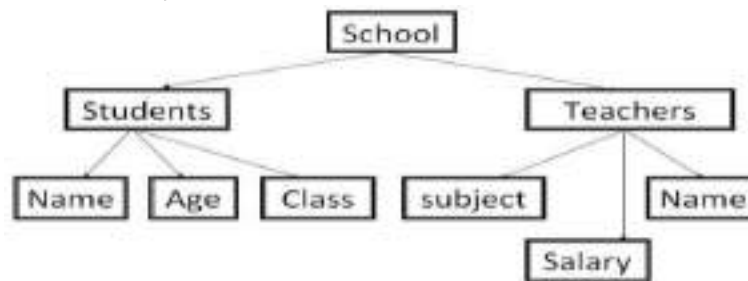
Model is an abstraction of reality. The purpose of the data model is to convey the details of the system for better understanding of the organization of data. There are five types of data models. They are:

- A. Hierarchical data model
- B. Network data model
- C. Relational Data model
- D. Object oriented data model
- E. ER model
- F. EER model

A. Hierarchical Data Model:

This model was developed in the end of 1960 to manage large amounts of data for Complex projects. This type of data model is based on the hierarchical relation presented in the form of a tree structure in which the root segment is kept at the top and further branches come downwards from the “root” segment. In this model 1:1 and 1:M associations are allowed and M:1 association is not permitted. While mapping conceptual model M:1 association is converted into 1:1 association.

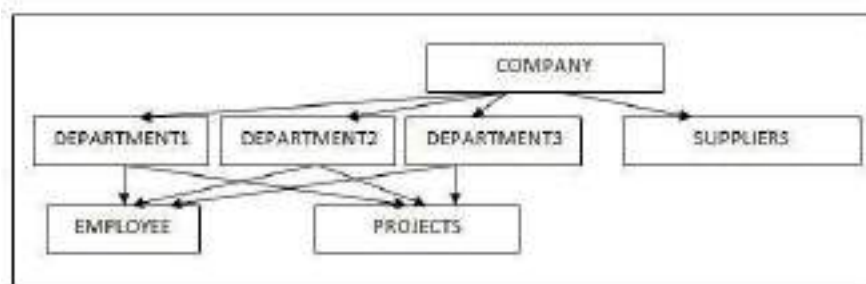
The hierarchical structure is used as the physical order of records in storage. One can access the records by navigating down through the data structure using pointer. The "root" in the structure is a single table in the database and other tables act as the branches flowing from the root. The diagram below shows a typical hierarchical database structure.



B. Network Data Model:

The network model was implemented in the early 1978. It was created to represent Complex data relationships more efficiently than the hierarchical data model. To organize data it uses “directed graphs” instead of tree-structure. In this child can have more than one parent it allows more connections between nodes.

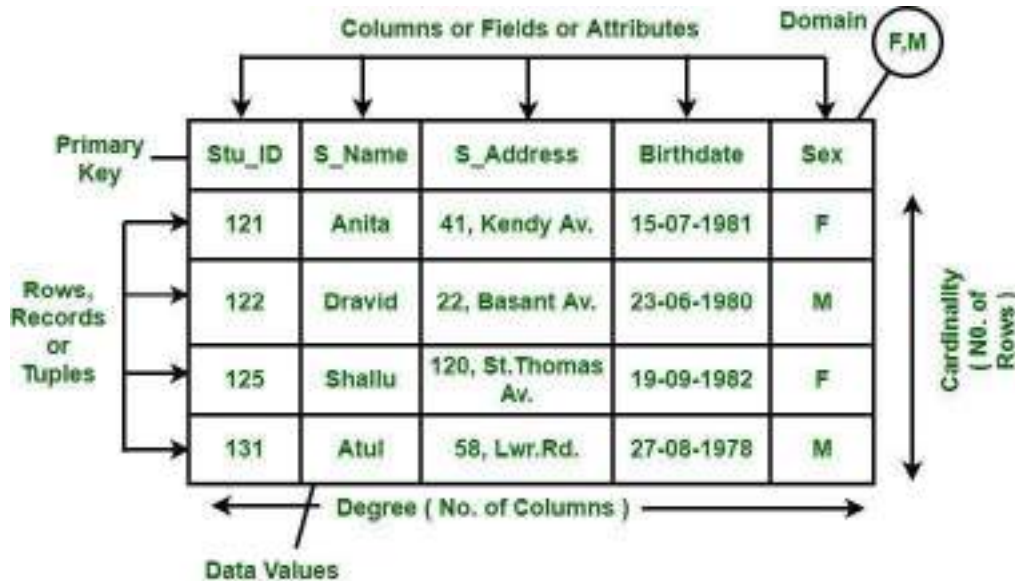
Ex: An employ work for two departments is not possible in hierarchal model, but here it is possible



Every conceptual data model is almost like a network data model. It is classified into three types. They are:

- 1. Simple Network Data Model:** It consists of a set of files with 1:1 and 1:M pair wise associations. May to many association is not permitted in the simple network data model.
- 2. Complex Network Data Model:** If a network data model has at least one many to many associations, then it is called complex network data model.
- 3. Limited Network Data Model:** In this model each and every file of the simple network data model will be divided into master file and transaction file.

C. Relational Data Model: The relational data model was developed by E.F. Codd in 1970. There are no physical links as they are in hierarchical data model. It is the primary data model, which is used widely around the world for data storage and processing. It represents the database as a collection of relations. A relation is nothing but a table of values. Every row in the table represents a collection of related data values. It organizes records in form of table and relationships between tables are using common fields.



It is the most popular data model of the present day because of the following reasons:

- It is simple to implement.
- It has simple terminology.
- It uses the simple concept of primary key and secondary key to connect any two files.
- In reality almost all the databases are developed based on relational data model. Some of them are Dbase, FoxPro, Ms-Access and Oracle etc.

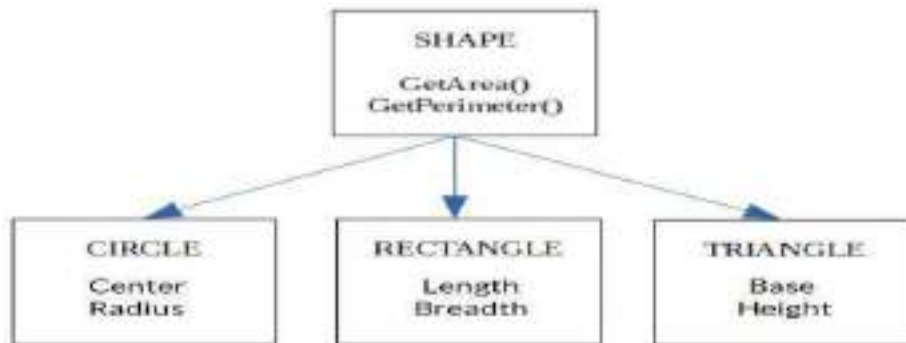
D. Object oriented data model:

Object oriented data model is based upon real world situations. These situations are represented as objects, with different attributes. These entire objects have multiple relationships between them. Elements of Object oriented data model are:

- ✓ **Objects:** The real world entities and situations are represented as objects
- ✓ **Attributes and Method:** Every object has certain characteristics. These are represented using Attributes. The behavior of the objects is represented using Methods.
- ✓ **Class:** Similar attributes and methods are grouped together using a class. An object can be called as an instance of the class.
- ✓ **Inheritance:** A new class can be derived from the original class. The derived class contains attributes and methods of the original class as well as its own.

Example In this Object Oriented data model-

- ✓ Shape is a class. Circle, Rectangle and Triangle are all objects in this model.
 - ✓ Circle has the attributes Center and Radius.
 - ✓ Rectangle has the attributes Length and Breadth
 - ✓ Triangle has the attributes Base and Height.
- The objects Circle, Rectangle and Triangle inherit from the class Shape.



E. Entity Relationship Data Model (ER Data Model):

An **Entity-relationship model (ER model)** describes the structure of a database with the help of a diagram, which is known as **Entity Relationship Diagram (ER Diagram)**. An ER model is a design or blueprint of a database that can later be implemented as a database.

An ER diagram shows the relationship among entity sets. An entity set is a group of similar entities and these entities can have attributes.

In this model

- **Rectangle** :Represents Entity sets.
- **Ellipses** :Attributes
- **Diamonds** : Relationship Set
- **Lines**: They link attributes to Entity Sets and Entity sets to Relationship Set



F. Extended Entity-Relationship (EE-R) Model:

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

In addition to ER model concepts EE-R includes -

- ❖ Subclasses and Super classes.
- ❖ Specialization and Generalization.
- ❖ Category or union type.
- ❖ Aggregation.

7. Advantages of Database Management System:

A Database Management System (DBMS) is defined as the software system that allows users to create, read, update and delete data in database. It is a layer between programs and data.

Compared to the File Based Data Management System, Database Management System has many advantages. Some of these advantages are given below.

A. Reducing Data Redundancy

The file based data management systems contained multiple files that were stored in many different locations in a system or even across multiple systems. Because of this, there were sometimes multiple copies of the same file which lead to data redundancy.

This is prevented in a database as there is a single database and any change in it is reflected immediately.

B. Consistency of data:-

In Database the repeated data will be minimized then it always gives consistent results.

C. Sharing of Data:

In a database, the users of the database can share the data among themselves. There are various levels of authorisation to access the data, and consequently the data can only be shared based on the correct authorisation.

Many remote users can also access the database simultaneously and share the data between themselves.

D. Concurrent Access:-

Database supports multiple users to access the data. Multiple users can access the data at a time.

E. Data Security

Data Security is vital concept in a database. Only authorised users should be allowed to access the database and their identity should be authenticated using a username and password. Unauthorised users should not be allowed to access the database under any circumstances as it violates the integrity constraints.

F. Backup and Recovery

Database Management System automatically takes care of backup and recovery. The users don't need to backup data periodically because this is taken care of by the DBMS. Moreover, it also restores the database after a crash or system failure.

G. Better enforcement of standards:-

Database is a collection of related files. These files are developed by different persons under the guidance of Data Base Administrator. They will give the same the field name, field type and field size in the files.

H. Increased programmer productivity:-

Programmer productivity is a measure of time taken to develop an application. Database has so many advantages like controlled data redundancy, consistent of data etc. So programming productivity is always high

8. ANSI-SPARC Architecture

In 1975, ANSI-SPARC realized the need for a three-level approach with the three levels of abstraction comprises of an external, a conceptual, and an internal level.

The three-level architecture aims to separate each user's view of the database from the way the database is physically represented.

1. Internal level:

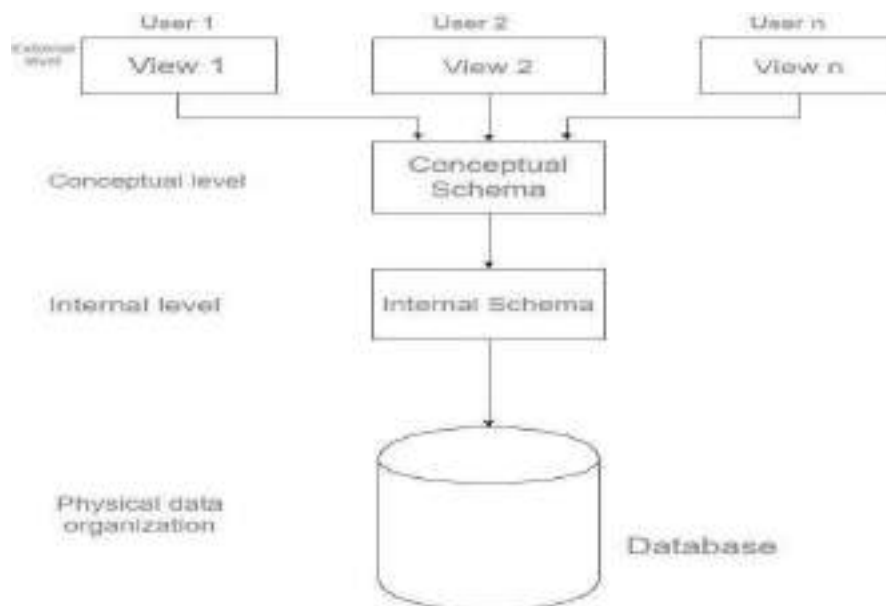
- ❖ This is the lowest level of data abstraction. It is also known as physical level.
- ❖ It describes how the data are actually stored on storage devices.
- ❖ It provides internal view of physical storage of data.
- ❖ It deals with Data Compression and Encryption techniques if used.

2. Conceptual level:

- ❖ This is the next higher level than internal level of data abstraction.
- ❖ It describes what data are stored in the database and what relationships exist among those data.
- ❖ It is also known as logical level.
- ❖ It hides low level complexities of physical storage.
- ❖ Database administrator, application developers and designers work at this level to determine what data to keep in database.

3. External Level:

- ❖ This is the highest level of data abstraction. It is also known as a view level.
- ❖ It describes only part of the entire database that a end user concern.
- ❖ End users need to access only part of the database rather than entire database.
- ❖ Different user needs different views of database and so, there can be many view level abstractions of the same database.



What are Interfaces in DBMS?

A database management system (DBMS) interface is a user interface which allows for the ability to input queries to a database without using the query language itself.

User-friendly interfaces provide by DBMS may include the following:

A. Menu-Based Interfaces for Web Clients or Browsing

These interfaces present the user with lists of options (called menus) that lead the user through the formation of a request.

B. Forms-Based Interfaces:

A forms-based interface displays a form to each user. Users can fill out all of the form entries to insert a new data, or they can fill out only certain entries, in which case the DBMS will redeem same type of data for other remaining entries.

C. Graphical User Interface:

A GUI typically displays a schema to the user in diagrammatic form. The user can specify a query by manipulating the diagram. In many cases, GUI's utilize both menus and forms.

D. Natural language Interface:

These interfaces accept request written in English or some other language and attempt to understand them.

E. Speech Input and Output:

There is an limited use of speech say it for a query or an answer to a question or being a result of a request it is becoming common place Applications with limited vocabularies

Ex: inquiries for telephone directory, flight arrival/departure, and bank account information are allowed speech for input and output to enable ordinary folks to access this information.

F. Interfaces for DBA:

Most database system contains privileged commands that can be used only by the DBA's staff. These include commands for creating accounts, setting system parameters, granting account authorization, changing a schema, reorganizing the storage structures of a database.

9. Components of Database Management System:

DBMS consists of database, DBMS utilities, data dictionary, application developers, users, and Database Administrator.

A. DBMS:

The software which is used to manage database is called Database Management System (DBMS). It acts as the interface between users and the database itself. It is a record keeping system. It allows the data to be stored, maintained, manipulated, and retrieved. These are very expensive than file systems.

For Example, MySQL, Oracle etc. are popular commercial DBMS used in different applications.

B. Database:

Database is a collection of inter-related data which helps in efficient retrieval, insertion, and deletion of data from database and organizes the data in the form of tables, views, schemas, reports etc.

For Example, university database organizes the data about students, faculty, and admin staff etc.

C. Data dictionary:

Data Dictionary consists of database metadata i.e, Data about Data. Data Dictionary consists of the following information –

- ❖ Name of the tables in the database
- ❖ Constraints of a table i.e. keys, relationships, etc.
- ❖ Columns of the tables that related to each other which specifies field name, field type and field size.
- ❖ Owner of the table
- ❖ Last accessed information of the object
- ❖ Last updated information of the object

Field Name	Datatype	Field Length	Constraint	Description
Student_ID	Number	5	Primary Key	Student id
Student_Name	Varchar	20	Not Null	Name of the student
Student_Address	Varchar	30	Not Null	Address of the student
Student_City	Varchar	20	Not Null	City of the student

There are the two types of data dictionary –

Active Data Dictionary

The DBMS software manages the active data dictionary automatically. It is also known as integrated data dictionary.

Passive Data Dictionary

Managed by the users and is modified manually when the database structure change. Also known as non-integrated data dictionary.

D. Application developers:

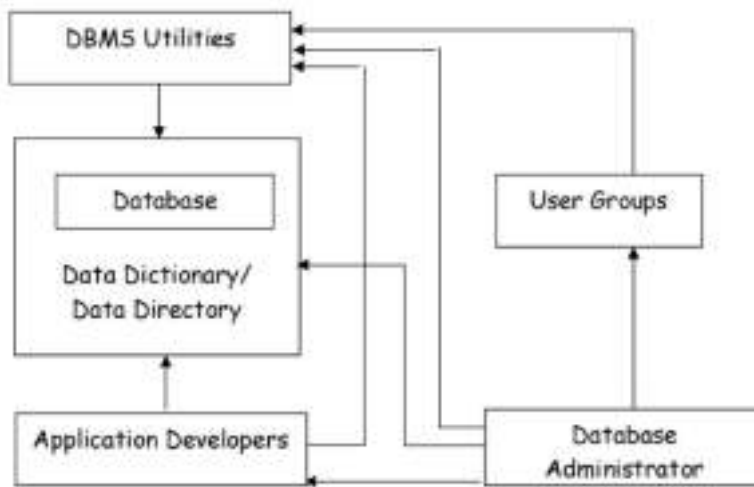
These are the qualified programmers/analysts. They can write and develop programs. These programs are developed under the guidance of DBA. These programs are used by user groups.

E. User groups:

These are group of end users. They can use the programs developed by the Application developers.

F. Database Administrator:

- ❖ It is the apex body.
- ❖ It controls and coordinates all the software entities, database entities, application developers and user groups.



10. **DBMS – Architecture:**

The design of a DBMS depends on its architecture. It can be centralized or decentralized or hierarchical.

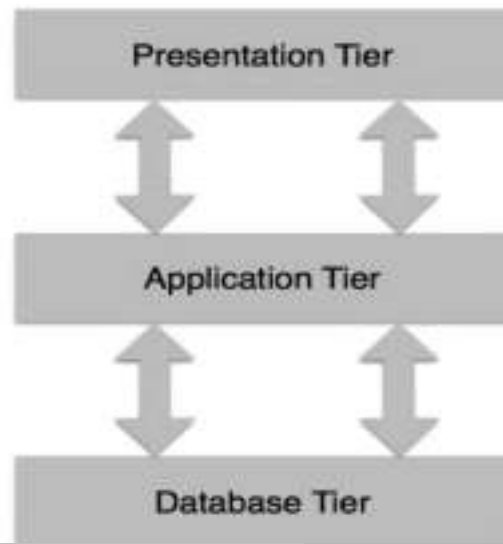
The architecture of a DBMS can be seen as either single tier or multi-tier. An n-tier architecture divides the whole system into related but independent n modules, which can be independently modified, altered, changed, or replaced.

In **1-tier architecture**, the DBMS is the only entity where the user directly sits on the DBMS and uses it. Any changes done here will directly be done on the DBMS itself. Database designers and programmers normally prefer to use single-tier architecture.

In **2-tier architecture**, it must have an application through which the DBMS can be accessed. Programmers use 2-tier architecture where they access the DBMS by means of an application. Here the application tier is entirely independent of the database in terms of operation, design, and programming.

In 3-tier Architecture, it separates its tiers from each other based on the complexity of the users and how they use the data present in the database. It is the most widely used architecture to design a DBMS. The tiers are

- **Database (Data) Tier** – At this tier, the database resides along with its query processing languages. We also have the relations that define the data and their constraints at this level.
- **Application (Middle) Tier** – At this tier reside the application server and the programs that access the database. The application layer sits in the middle and acts as a mediator between the end-user and the database.
- **User (Presentation) Tier** – End-users operate on this tier and they know nothing about any existence of the database beyond this layer. At this layer, multiple views of the database can be provided by the application. All views are generated by applications that reside in the application tier.



11. **DBMS Vendors and Their Products:**

There are many different vendors that currently produce relational database management systems (RDBMS).

The leading vendors of RDBMS are listed below:

**** RDBMS Vendors ****	**** RDBMS Product ****
Computer Associates	INGRES
IBM	DB2
INFORMIX Software	INFORMIX
Oracle Corporation	Oracle
Microsoft Corporation	MS Access
Microsoft Corporation	SQL Server
MySQL AB	MySQL
NCR Teradata	
PostgreSQL Dvlp Grp	PostgreSQL
Sybase	Sybase 11

UNIT-II

1. Entity-relationship model:

An Entity-relationship model (ER model) describes the structure of a database with the help of a diagram, which is known as Entity Relationship Diagram (ER Diagram). It is a high-level data model. An ER model is a design or blueprint of a database that can later be implemented as a database.

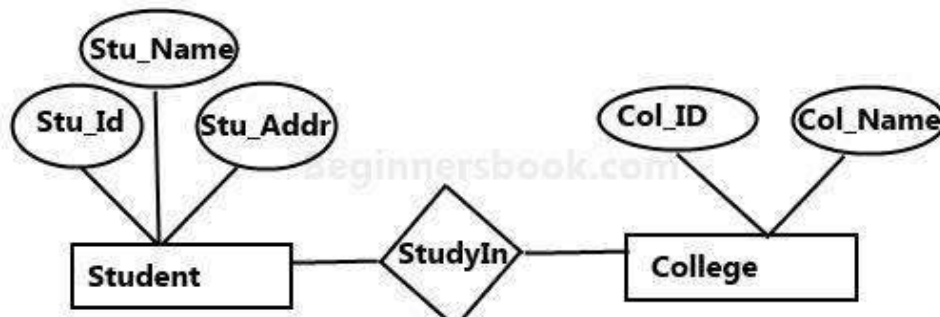
The main components of E-R model are Entity set, Attributes and Relationships.

2. ER Diagram:

An ER diagram shows the relationship among entity sets. An entity set is a group of similar entities and these entities can have attributes.

In terms of DBMS, an entity is a table in database, so by showing relationship among tables and their attributes, ER diagram shows the complete logical structure of a database.

In the following diagram we have two entities Student and College and their relationship. Student entity has attributes such as Stu_Id, Stu_Name&Stu_Addr and College entity has attributes such as Col_ID&Col_Name.



3. Building blocks of Entity Relationship Diagram:

The components of the entity-relationship model are the building block which helps in the generation of an ER diagram, which finally results in the design of the logical structure of a database. There are three basic components of the Entity-Relationship Model.

All these components have definite diagrammatical representations that are used for the generation of ER Diagram.

A. Entity:

An entity can be called as the basic object which represents the ER Model. Entities are Real world objects or things or articles or pieces that have an existence without any dependence on any other object.

For example: Bank account, Student, employ all these are entities have their own properties which are known as attributes.

B. Attribute:

Attributes describe the properties and characteristics of the entity.

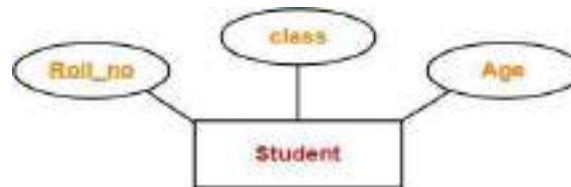
For example: Attributes of Bank account can be "Account No, Account Holder Name, Balance" etc.

There exist seven types of attributes for specific types of entities.



a) Simple Attribute: Simple attribute are those attributes which cannot be divided further because of their atomic nature.

For Example: The roll number of a student.



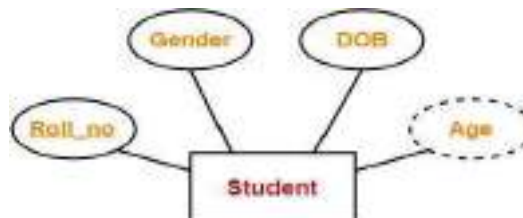
b) Composite attribute: An attribute which can be divided into components is a composite attribute.

For Example: The address can be further divided into house number, street number, city, state, country and pin code, the name can also be divided into first name middle name and last name.



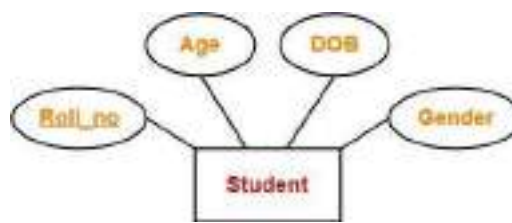
c) Derived Attribute: Derived Attributes are those attributes whose values are derived from the already stored attributes in the database.

For example: Age of a student calculated with current date and date of birth.



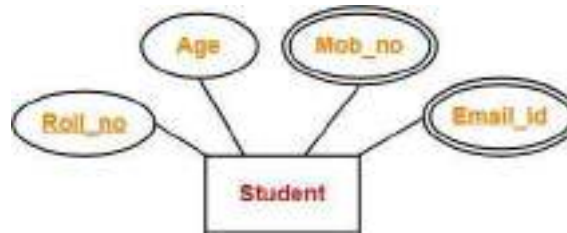
d) Single Valued Attribute: Single valued attribute are those which contains only a single specified value or a single unique value.

For example: Roll no, Aadhar Number, SSN No, etc.



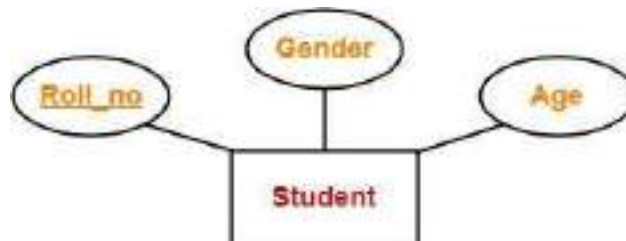
- e) **Multi-Valued Attribute:** Multi-valued attributes are those which can have multiple values against them.

For example: Address(permanent, present), Email_id, Phone number of a student (Landline and mobile).



- f) **Key Attributes:** Key attributes are those attributes which can identify an entity uniquely in an entity set.

For example: Here, the attribute "Roll_no" is a key attribute as it can identify any student uniquely.



- g) **Null Value Attribute:** Null value attributes are those which can be left blank without passing any value.

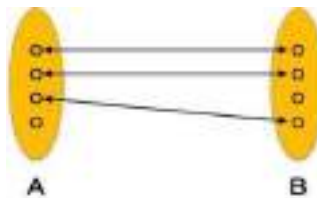
For example: A person can have a name without a middle name and hence middle name field can be left blank or null.

First Name + _____ + Last Name

c. Relationship:

The association among entities is called a relationship.

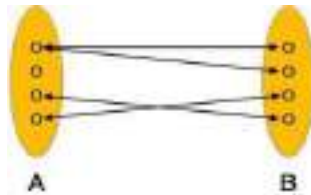
- a) **One-to-one** – One entity from entity set A can be associated with at most one entity of entity set B and vice versa.



For example, If there are two entities „Person“ (Id, Name, Age, Address) and „Passport“ (Passport_id, Passport_no). So, each person can have only one passport and each passport belongs to only one person.



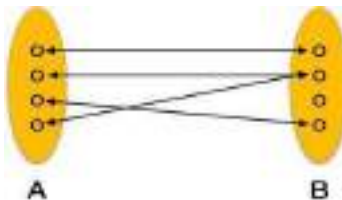
- b) One-to-many** – One entity from entity set A can be associated with more than one entities of entity set B however an entity from entity set B, can be associated with at most one entity.



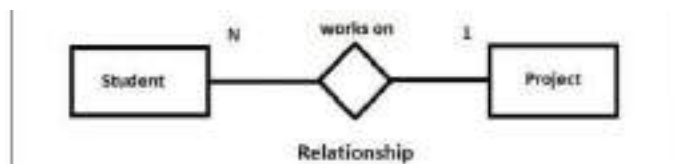
For example, If there are two entity type „Customer“ and „Account“ then each „Customer“ can have more than one „Account“ but each „Account“ is held by only one „Customer“. In this example, we can say that each Customer is associated with many Account. So, it is a one-to-many relationship. But, if we see it the other way i.e many Account is associated with one Customer then we can say that it is a many-to-one relationship.



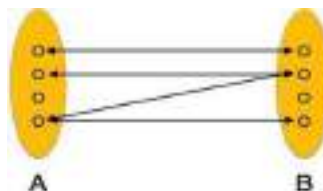
- c) Many-to-one** – More than one entities from entity set A can be associated with at most one entity of entity set B, however an entity from entity set B can be associated with more than one entity from entity set A.



A project can have more than one student working on it. A team of five students in a college is assigned a project that they need to complete in let us say one month. This states a relationship between two entities Student and Project.



- d) Many-to-many** – One entity from A can be associated with more than one entity from B and vice versa.



Example: If there are two entity type „Customer“ and „Product“ then each customer can buy more than one product and a product can be bought by many different customers.



Cardinality:

It defines the number of entities in one entity set, which can be associated with the number of entities of other set via relationship set.

Degree of Relationship:

The degree of a relationship is the number of entity types that participate (associate) in a relationship. By seeing an E-R diagram, we can simply tell the degree of a relationship i.e the number of an entity type that is connected to a relationship is the degree of that relationship.

For example: if we have two entity type „Customer“ and „Account“ and they are linked using the primary key and foreign key. We can say that the degree of relationship is 2 because here two entities are taking part in the relationship.



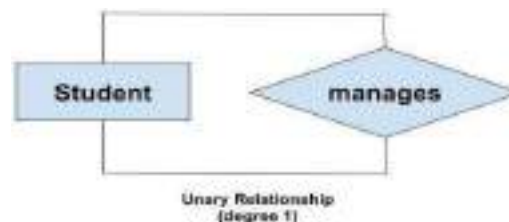
Based on the number of entity types that are connected we have the following degree of relationships:

- A. Unary
- B. Binary
- C. Ternary
- D. N-ary

A. Unary (degree 1):

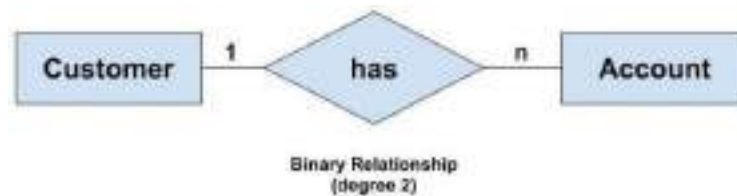
A unary relationship exists when both the participating entity type are the same. When such a relationship is present we say that the deg of relationship is 1.

For example: Suppose in a classroom, we have many students who belong to a particular club-like dance club, basketball club etc. and some of them are club leads. So, a particular group of student is managed by their respective club lead. Here, the group is formed from students and also, the club leads are chosen from students. So, the „Student“ is the only entity participating here. We can represent this relationship using the E-R diagram as follows:

**B. Binary (degree 2):**

A binary relationship exists when exactly two entity types participates. When such a relationship is present we say that the degree is 2. This is the most common degree of relationship. It is easy to deal with such relationship as these can be easily converted into relational tables.

For example: We have two entity type „Customer“ and „Account“ where each „Customer“ has an „Account“ which stores the account details of the „Customer“. Since we have two entity types participating we call it a binary relationship. Also, one „Customer“ can have many „Account“ but each „Account“ should belong to only one „Customer“. We can say that it is a one-to-many binary relationship.

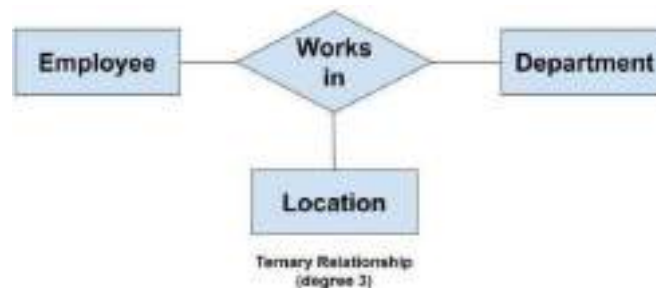


C. Ternary (degree 3):

A ternary relationship exists when exactly three entity type participates. When such a relationship is present we say that the degree is 3.

As the number of entity increases in the relationship, it becomes complex to convert them into relational tables.

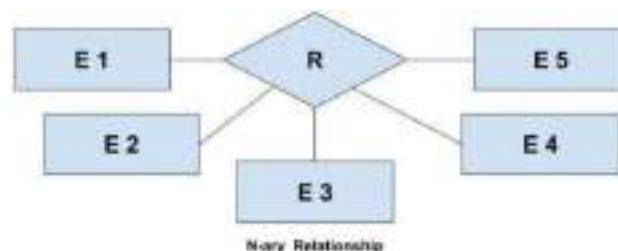
For example: We have three entity type „Employee“, „Department“ and „Location“. The relationship between these entities is defined as an employee works in a department, an employee works at a particular location. So, we can see we have three entities participating in a relationship so it is a ternary relationship. The degree of this relation is 3.



D. N-ary (n degree):

An N-ary relationship exists when „n“ number of entities are participating. So, any number of entities can participate in a relationship. There is no limitation to the maximum number of entities that can participate.

We represent an N-ary relationship as follows:



In the above example, E1 denotes the first entity type, E2 denotes the second entity type and so on. R represents the relationship. So, here we have a total of 5 entity type which participates in the relationship. Therefore, the degree of the above n-ary relationship is 5.

4. Classification of Entity Set in DBMS:

An entity set is a group of entities that possess the same set of attributes. Each entity in an entity set has its own set of values for the attributes which make it distinct from other entities in a table. No two entities in an entity set will have the same values for the attributes.

Types of Entity Sets-

An entity set may be of the following two types-



a) Strong Entity Set:

An entity set that has a primary key using which, entities in the table can be uniquely identified. This kind of entity set is termed as a strong entity set. It is also known as a regular entity set.

A strong entity set is an entity set that contains sufficient attributes to uniquely identify all its entities. In other words, a primary key exists for a strong entity set.

Symbols Used:

- ✓ Primary key of a strong entity set is represented by underlining it.
- ✓ A single rectangle is used for representing a strong entity set.
- ✓ A diamond symbol is used for representing the relationship that exists between two strong entity sets.
- ✓ A single line is used for representing the connection of the strong entity set with the relationship set.
- ✓ A double line is used for representing the total participation of an entity set with the relationship set.

Example: Consider the following ER diagram-



Two strong entity sets **“Student”** and **“Course”** are related to each other.

Student ID and Student name are the attributes of entity set “Student”. Student ID is the primary key using which any student can be identified uniquely.

Course ID and Course name are the attributes of entity set “Course”. Course ID is the primary key using which any course can be identified uniquely.

Double line between Student and relationship set signifies total participation.

Single line between Course and relationship set signifies partial participation.

It suggests that there might exist some courses for which no enrollments are made.

b) Weak Entity Set-

A weak entity set does not have any primary key which can identify each entity in a set distinctly.

In other words, a primary key does not exist for a weak entity set. However it contains a partial key called as a “discriminator”. Discriminator can identify a group of entities from the entity set. Discriminator is represented by underlining with a dashed line.

Note:- The combination of discriminator and primary key of the strong entity set makes it possible to uniquely identify all entities of the weak entity set. Thus, this combination serves as a primary key for the weak entity set.

Clearly, this primary key is not formed by the weak entity set completely.

Primary key of weak entity set

= Its own discriminator + Primary key of strong entity set

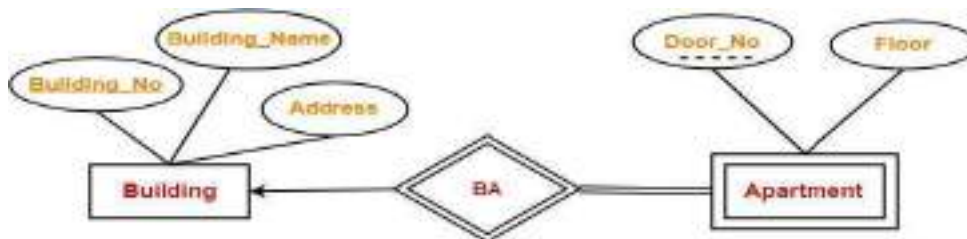
Symbols Used:

A double rectangle is used for representing a weak entity set.

A double diamond symbol is used for representing the relationship that exists between the strong and weak entity sets and this relationship is known as identifying relationship.

A double line is used for representing the connection of the weak entity set with the relationship set.

Example:



In this ER diagram,

One strong entity set “**Building**” and one weak entity set “**Apartment**” are related to each other.

Strong entity set “Building” has building number as its primary key.

Door number is the discriminator of the weak entity set “Apartment”.

This is because door number alone cannot identify an apartment uniquely as there may be several other buildings having the same door number.

Double line between Apartment and relationship set signifies total participation.

It suggests that each apartment must be present in at least one building.

Single line between Building and relationship set signifies partial participation.

It suggests that there might exist some buildings which has no apartment.

5. Enhanced entity-relationship model:

EER is a high-level data model that incorporates the extensions to the original ER model. Enhanced entity-relationship diagrams are advanced database diagrams very similar to regular ER diagrams which represents requirements and complexities of complex databases.


It is a diagrammatic technique for displaying the following concepts.

- A. Sub Class and Super Class
- B. Specialization and Generalization
- C. Union or Category
- D. Aggregation

These concepts are used when they come in EER schema and the resulting schema diagrams called as EER Diagrams.

It is a diagrammatic technique for displaying the Sub Class and Super Class; Specialization and Generalization; Union or Category; Aggregation etc.

A. Sub Class and Super Class

- Sub class and Super class relationship leads the concept of Inheritance.
- Relationship between subclass and superclass is denoted with  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.

2. 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.

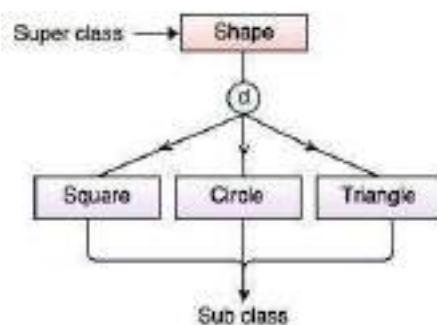


Fig. Super class/Sub class Relationship

B. Specialization and Generalization

Generalization:

- The process of defining a general entity type from a collection of specialized entity types.
- It is a bottom-up 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.

For example: Tiger, Lion, Elephant can all be generalized as Animals.

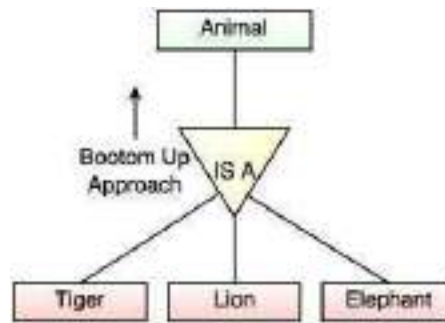


Fig. Generalization

Specialization:

- It is the opposite of generalization:
- Specialization is a process that defines a group entity which is divided into subgroups 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 defines one or more sub class for the super class and also forms the superclass/subclass relationship.

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

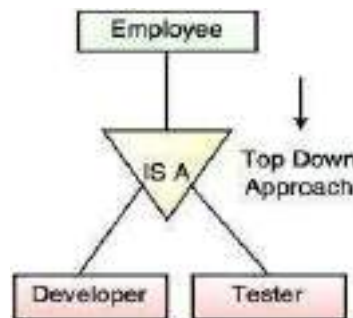


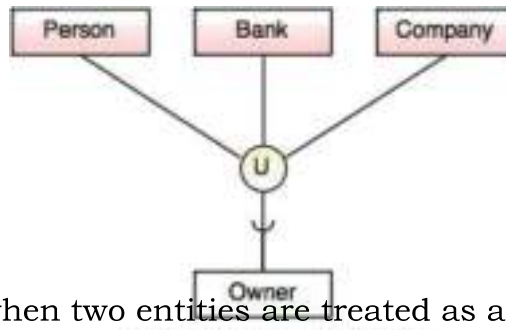
Fig. Specialization

C. Category or Union

- Category represents a Relationship of one super or sub class with more than one super class.

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.



D. Aggregation:

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

For Example: The relation between College and Course is acting as an Entity in Relation with Student.

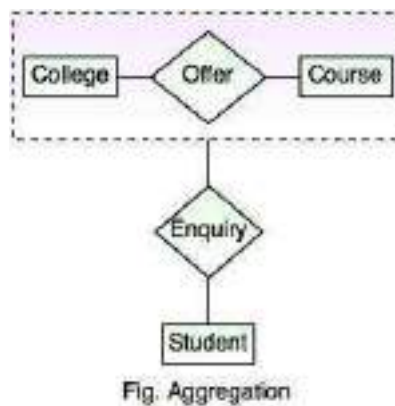


Fig. Aggregation

6. Advantages of ER Model:

The ER diagram is highly popular as it carries multiple benefits which include:

➤ It is very simple:

ER model is very simple because if we know relationship between entities and attributes, then we can easily draw an ER diagram.

➤ Blueprint to work:

It provides a blueprint to work with when you do start creating the actual database.

➤ Highly Flexible:

The ER diagram can be effectively utilized for establishing and deriving relationships from the existing ones.

➤ Better visual representation:

ER model is a diagrammatic representation of any logical structure of database. By seeing ER diagram, we can easily understand relationship among entities and relationship.

➤ Effective communication tool:

It is an effective communication tool for database designer.



Highly integrated with relational model:

ER model can be easily converted into relational model by simply converting ER model into tables.

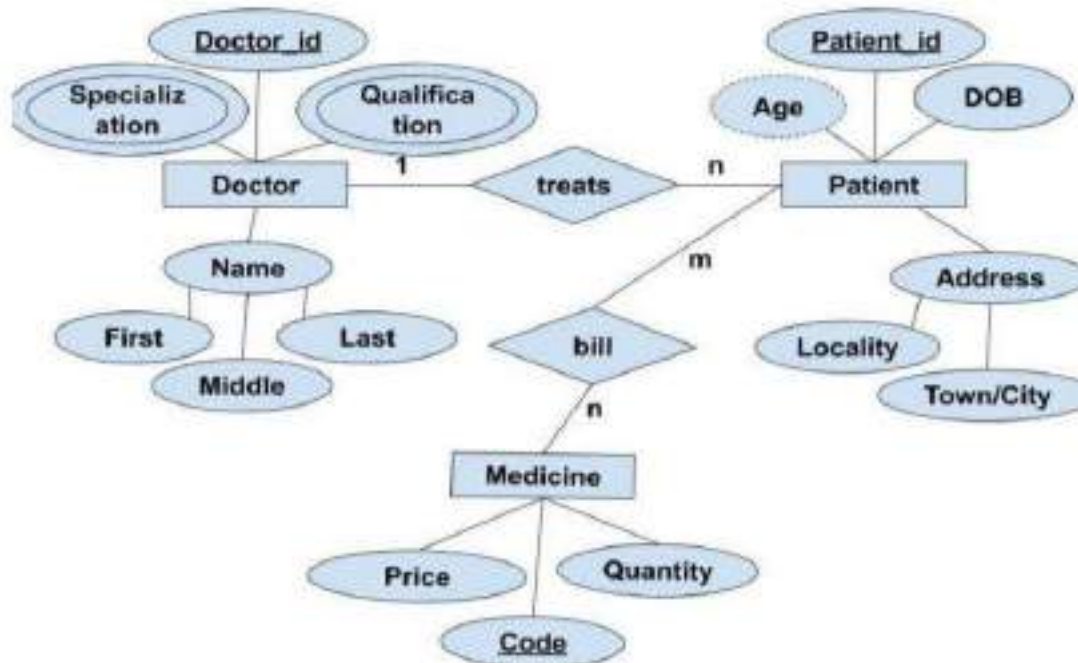


Easy conversion to any data model:

ER model can be easily converted into another data model like hierarchical data model, network data model and so on.

7. ER-diagram for Hospital Management System:

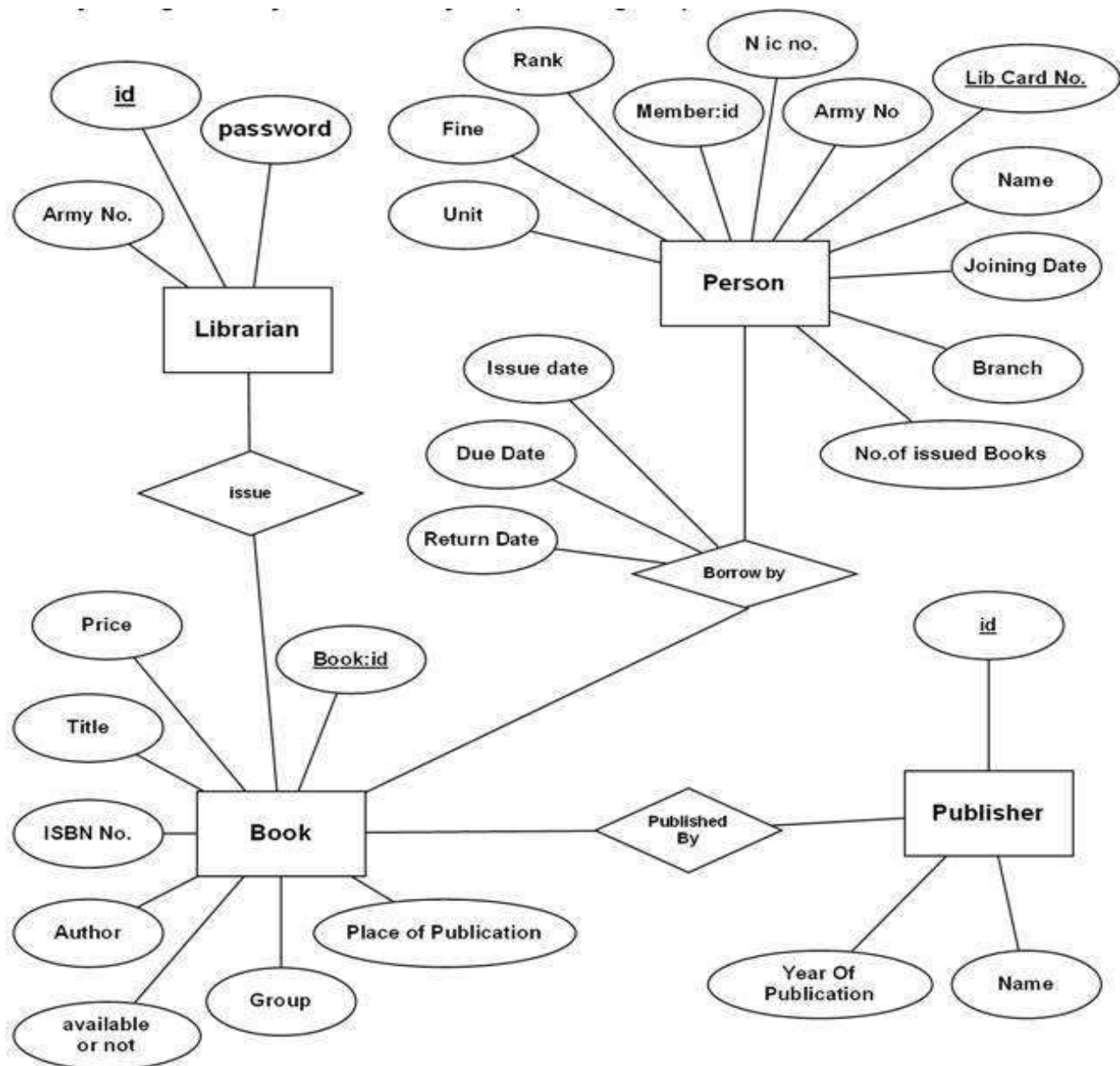
ER diagram for a hospital management system have three entities i.e Doctor, Patient, Medicine.



Hospital Management System

In the above diagram, Entity Doctor has key attribute 'doctor_id' which will be used to identify the doctors. It also has two multivalued attributes as 'specialization' and 'qualification' as a doctor may have more than one qualification and may be specialized in more than one fields. The Doctor and Patient entity have a one-to-many relationship as a Doctor may treat more than one patient. Similarly, Patient and Medicine have a many-to-many relationship as a patient may buy more than one medicine and vice-versa. 'Code' is the key attribute for Medicine which is unique for every medicine. The Patient has many attributes Patient_id, DOB, Age, etc. 'Age' is the derived attribute here. Also, it has a composite attribute 'Address' which can further be divided into two attributes 'Locality' and 'Town'.

8. ER-Diagram for Library Management System:



UNIT-III

What is Relational Model?

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

Some popular Relational Database management systems are:

DB2 and Informix Dynamic Server - IBM

Oracle and RDB – Oracle

SQL Server and Access - Microsoft

Relational Model Concepts

a. Attribute: Each column in a Table. Attributes are the properties which define a relation.

E.g: Student_Rollno, NAME, etc.

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

c. Tuple: It is nothing but a single row of a table, which contains a single record.

d. Relation Schema: A relation schema represents the name of the relation with its attributes.

Ex: Employee(Emp number, Name, Designation, Age, Salary)

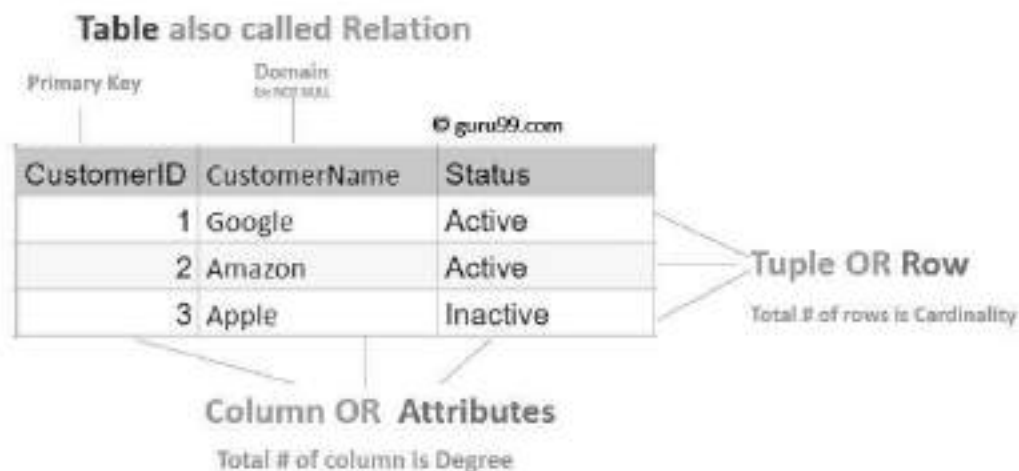
e. Degree: The total number of attributes which in the relation is called the degree of the relation.

f. Cardinality: Total number of rows present in the Table.

g. Column: The column represents the set of values for a specific attribute.

h. Relation key: The key for a table is the attribute that can uniquely identify all the tuples. In the Employee table, the key is Emp Number as it is unique for every single employee,.

i. Attribute domain: Every attribute has some pre-defined value and scope which is known as attribute domain



Constraints:

Every relation has some constraints that must hold for it to be called a relational model. These are as followed -

- a) Key constraints - There must be at least one set of attributes that can identify a tuple in a unique manner. This set is known as a key.
- b) Domain constraints - There are some domain specific constraints that must be followed in a database.
 - i. Example - The salary of an employee cannot be negative so the salary field has only positive values.
- c) Referential Integrity constraints - These constraints are used to describe the behaviour of foreign keys. A foreign key is a key of a relation that can be referred in another relation.

Codd's 12 Rules:

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

- Rule 1: Information Rule
 - The data stored in a database, may it be user data or metadata, must be a value of some table cell. Everything in a database must be stored in a table format.
- Rule 2: Guaranteed Access Rule
 - Every single data element (value) is guaranteed to be accessible logically with a combination of table-name, primary-key (row value), and attribute-name (column value).
- Rule 3: Systematic Treatment of NULL Values
 - The NULL values in a database must be given a systematic and uniform treatment. This is a very important rule because a NULL can be interpreted as one the following - data is missing, data is not known, or data is not applicable.
- Rule 4: Active Online Catalog
 - The structure description of the entire database must be stored in an online catalog, known as data dictionary, which can be accessed by authorized users.
- Rule 5: Comprehensive Data Sub-Language Rule
 - A database can only be accessed using a languages that supports data definition, data manipulation, and transaction management operations.
- Rule 6: View Updating Rule
 - All the views of a database, which can theoretically be updated, must also be updatable by the system.
- Rule 7: High-Level Insert, Update, and Delete Rule
 - A database must support high-level insertion, updation, and deletion. This must not be limited to a single row, that is, it must also support union, intersection and minus operations to yield sets of data records.

- Rule 8: Physical Data Independence
 - The data stored in a database must be independent of the applications that access the database. Any change in the physical structure of a database must not have any impact on how the data is being accessed by external applications.
- Rule 9: Logical Data Independence
 - The logical data in a database must be independent of its user's view (application). Any change in logical data must not affect the applications using it.
 - For example, if two tables are merged or one is split into two different tables, there should be no impact or change on the user application. This is one of the most difficult rule to apply.
- Rule 10: Integrity Independence
 - A database must be independent of the application that uses it. All its integrity constraints can be independently modified without the need of any change in the application. This rule makes a database independent of the front-end application and its interface.
- Rule 11: Distribution Independence
 - The end-user must not be able to see that the data is distributed over various locations. Users should always get the impression that the data is located at one site only.
 - This rule has been regarded as the foundation of distributed database systems.
- Rule 12: Non-Subversion Rule
 - If a system has an interface that provides access to low-level records, then the interface must not be able to subvert the system and bypass security and integrity constraints.

Types of Keys in Relational Model:

STUDENT

STUD_NO	STUD_NAME	STUD_PHONE	STUD_STATE	STUD_COUNT RY	STUD_AG E
1	RAM	9716271721	Haryana	India	20
2	RAM	9898291281	Punjab	India	19
3	SUJIT	7898291981	Rajsthan	India	18
4	SURESH		Punjab	India	21

Table 1

STUDENT_COURSE

STUD_NO	COURSE_NO	COURSE_NAME
1	C1	DBMS
2	C2	Computer Networks
1	C2	Computer Networks

Table 2

A. Candidate Key:

- Candidate Key The minimal set of attribute which can uniquely identify a tuple is known as candidate key. For Example, STUD_NO in STUDENT relation.
- The value of Candidate Key is unique and non-null for every tuple.
- There can be more than one candidate key in a relation.
- The candidate key can be simple (having only one attribute) or composite as well.
For Example, STUD_NO is candidate key for relation STUDENT. For Example, {STUD_NO, COURSE_NO} is a composite candidate key for relation STUDENT_COURSE.

B. Super Key:

- The set of attributes which can uniquely identify a tuple is known as Super Key.
- Adding zero or more attributes to candidate key generates super key.
- A candidate key is a super key but vice versa is not true.
For Example, STUD_NO, (STUD_NO, STUD_NAME) etc.

C. Primary Key:

- The primary key is a column, or set of columns, whose values uniquely identify each row in the table.
- Each table in a relational database must be assigned a primary key.
- There can be more than one candidate key in relation out of which one can be chosen as the primary key.
For Example, STUD_NO, as well as STUD_PHONE both, are candidate keys for relation STUDENT but STUD_NO can be chosen as the primary key (only one out of many candidate keys).

D. Alternate Key:

The candidate key other than the primary key is called an alternate key.

For Example, STUD_NO, as well as STUD_PHONE both, are candidate keys for relation STUDENT but STUD_PHONE will be alternate key (only one out of many candidate keys).

E. Foreign Key:

If an attribute can only take the values which are present as values of some other attribute, it will be a foreign key to the attribute to which it refers.

The relation which is being referenced is called referenced relation and the corresponding attribute is called referenced attribute and the relation which refers to the referenced relation is called referencing relation and the corresponding attribute is called referencing attribute. The referenced attribute of the referenced relation should be the primary key for it.

For Example, STUD_NO in STUDENT_COURSE is a foreign key to STUD_NO in STUDENT relation.

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

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

Relational Algebra:

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

Basic SQL Relational Algebra Operations

A. Unary Relational Operations

SELECT (symbol: σ)

PROJECT (symbol: π)

RENAME (symbol: ρ)

B. Relational Algebra Operations from Set Theory

UNION (\cup)

INTERSECTION (\cap),

DIFFERENCE ($-$)

CARTESIAN PRODUCT (\times)

C. Binary Relational Operations

JOIN

DIVISION

SELECT (σ):

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

$\sigma_p(r)$ σ is the predicate

r stands for relation which is the name of the

table p is propositional logic

Example 1 $\sigma_{\text{topic} = \text{"Database"}}(\text{Tutorials})$

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

Example 2 $\sigma_{\text{topic} = \text{"Database"} \text{ and } \text{author} = \text{"giri"}}(\text{Tutorials})$

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

Example 3 $\sigma_{\text{sales} > 50000}(\text{Customers})$

Output - Selects tuples from Customers where sales is greater than

50000

Projection(π):

The projection method defines a relation that contains a vertical subset of Relation.

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

Example of Projection:

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

Here, the projection of CustomerName and status will give

π CustomerName, Status (Customers)

CustomerName	Status
Google	Active
Amazon	Active
Apple	Inactive
Alibaba	Active

Rename (ρ)

Rename is a unary operation used for renaming attributes of a relation. ρ (a/b)R will rename the attribute 'b' of relation by 'a'. „rename“ operation is denoted with small Greek letter rho ρ

Union operation (\cup)

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

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

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

Example: π author (Books) \cup π author (Articles)

Output: Projects the names of the authors who have either written a book or an article or both.

Example: Consider the following tables.

Table A		Table B	
column 1	column 2	column 1	column 2
1	1	1	1
1	2	1	3

A \cup B gives
Table A \cup B

column 1	column 2
1	1
1	2
1	3

Set Difference (-)

- Symbol denotes it. The result of $A - B$, is a relation which includes all tuples that are in A but not in B.

- ✓ The attribute name of A has to match with the attribute name in B.
- ✓ The two-operand relations A and B should be either compatible or Union compatible.
- ✓ It should be defined relation consisting of the tuples that are in relation A, but not in B.

Example: $\Pi_{\text{author}}(\text{Books}) - \Pi_{\text{author}}(\text{Articles})$

Output: Provides the name of authors who have written books but not articles.

Example $A - B$

Table A - B	
column 1	column 2
1	2

Intersection

An intersection is defined by the symbol \cap

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

Example: $A \cap B$

Table $A \cap B$	
column 1	column 2
1	1



Cartesian Product(X) in DBMS:

Cartesian Product in DBMS is an operation used to merge columns from two relations. Generally, a cartesian product is never a meaningful operation when it performs alone. However, it becomes meaningful when it is followed by other operations. It is also called Cross Product or Cross Join.

Example: $\sigma_{\text{author} = \text{'tutorialspoint'}}(\text{Books} \times \text{Articles})$

Output: Yields a relation, which shows all the books and articles written by tutorialspoint.

Example:

$\sigma_{\text{column 2} = \text{'1'}}(A \times B)$

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

$\sigma_{\text{column 2} = \text{'1'}}(A \times B)$	
column 1	column 2
1	1
1	1

Join Operations:

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

Join operation denoted by \bowtie .

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

Types of JOIN: Various forms of join operation are:

Inner Joins:

- ✓ Theta join
- ✓ EQUI join
- ✓ Natural

join Outer join:

- ✓ Left Outer Join
- ✓ Right Outer Join
- ✓ Full Outer Join

Inner Join:

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

Theta Join:

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

Example

$A \bowtie_{\theta} B$

Theta join can use any conditions in the selection criteria.

For example:

$A \bowtie_{A.column\ 2 > B.column\ 2} B$

<small>$A \bowtie_{A.column\ 2 > B.column\ 2} B$</small>	
column 1	column 2
1	2

EQUI join:

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

For example:

$A \bowtie_{A.column\ 2 = B.column\ 2} B$

<small>$A \bowtie_{A.column\ 2 = B.column\ 2} B$</small>	
column 1	column 2
1	1

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

NATURAL JOIN (\natural)

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

Example: Consider the following two tables

C	
Num	Square
2	4
3	9

D	
Num	Cube
2	8
3	27

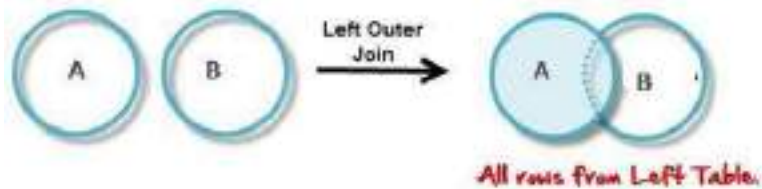
<small>$C \natural D$</small>		
Num	Square	Cube
2	4	4
3	9	27

OUTER JOIN

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

Left Outer Join($A \bowtie B$)

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



Consider the following 2 Tables

A	
Num	Square
2	4
3	9
4	16

B	
Num	Cube
2	8
3	18
5	75

$A \bowtie B$		
Num	Square	Cube
2	4	8
3	9	18
4	16	-

Right Outer Join: ($A \bowtie B$)

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



$A \bowtie B$		
Num	Cube	Square
2	8	4
3	18	9
5	75	-

Full Outer Join: ($A \bowtie B$)

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

$A \bowtie B$		
Num	Cube	Square
2	8	4
3	18	9
4	16	-
5	-	75

Summary:

Operation(Symbols)	Purpose
Select(σ)	The SELECT operation is used for selecting a subset of the tuples according to a given selection condition
Projection(π)	The projection eliminates all attributes of the input relation but those mentioned in the projection list.
Union Operation(\cup)	UNION is symbolized by symbol. It includes all tuples that are in tables A or in B.
Set Difference($-$)	- Symbol denotes it. The result of $A - B$, is a relation which includes all tuples that are in A but not in B.
Intersection(\cap)	Intersection defines a relation consisting of a set of all tuple that are in both A and B.
Cartesian Product(\times)	Cartesian operation is helpful to merge columns from two relations.
Inner Join	Inner join, includes only those tuples that satisfy the matching criteria.
Theta Join(θ)	The general case of JOIN operation is called a Theta join. It is denoted by symbol θ .
EQUI Join	When a theta join uses only equivalence condition, it becomes a equi join.
Natural Join(\bowtie)	Natural join can only be performed if there is a common attribute (column) between the relations.
Outer Join	In an outer join, along with tuples that satisfy the matching criteria.
Left Outer Join(\ltimes)	In the left outer join, operation allows keeping all tuple in the left relation.
Right Outer join(\rtimes)	In the right outer join, operation allows keeping all tuple in the right relation.
Full Outer Join(\Join)	In a full outer join, all tuples from both relations are included in the result irrespective of the matching condition.

Relational Calculus:

In contrast to Relational Algebra, Relational Calculus is a non-procedural query language, that is, it tells what to do but never explains how to do it.

Relational calculus exists in two forms –

- A.** Tuple Relational Calculus (TRC)
- B.** Domain Relational Calculus (DRC)

Tuple Relational Calculus (TRC):

Tuple relational calculus works on filtering the tuples based on the specified conditions. TRC can be quantified. We can use Existential (\exists) and Universal Quantifiers (\forall).

Syntax of TRC: $\{T \mid \text{Conditions}\}$

Returns all tuples T that satisfies a condition.

For instance, if the data need to be represented for the particular product id of value 10, it can be denoted as T.product_id=10, where T is the tuple variable that represents the row of the table.

Let us assume the Product table in the database as follows:

Product_id	Product Category	Product Name	Product Unit Price
8	New	TV Unit 1	\$100
10	New	TV Unit 2	\$120
12	Existing	TV Cabinet	\$77

Now to represent the relational calculus to return the product name that has the product id value as 10 from the product table, it can be denoted as with the tuple variable T.

T.Product Name | Product(T) AND T.Product_id = 10

This relational calculus predicate describes what to do for getting the resultant tuple from the database. The result of the tuple relational calculus for the Product table will be:

Product_id	Product Name
10	TV Unit 2

Domain Relational Calculus (DRC):

The domain relational calculus works based on the filtering of the domain and the related attributes. DRC is the variable range over the domain elements or the filed values.

In DRC, the filtering variable uses the domain of attributes instead of entire tuple values (as done in TRC, mentioned above).

Syntax of DRC in DBMS: {c1, c2,...,cn | F(c1, c2,... ,cn)}

The domain attributes in DRC can be represented as C1, C2,..., Cn and the condition related to the attributes can be denoted as the formula defining the condition for fetching the F(C1, C2, ...Cn)

Let us assume the same Product table in the database as follows:

Product_id	Product Category	Product Name	Product Unit Price
8	New	TV Unit 1	\$100
10	New	TV Unit 2	\$120
12	Existing	TV Cabinet	\$77

DRC for the product name attribute from the Product table needs where the product id is 10, It will be demoted as:

{< Product Name, Product_id> | ∈ Product ∧ Product_id = 10}

The result of the domain relational calculus for the Product table will be

Product_id	Product Name
10	TV Unit 2

Some of the commonly used logical operator notations for DRC are \wedge for

AND, \vee for OR, and \neg for NOT. imilarly, the mathematical symbol \in refers to the relation “is an

element of” or known as the set membership

Advantages and disadvantages of relational algebra:

Advantages:

- 1.Flexibility: Different tables from which information has to be linked and extracted can be easily manipulated by operators such as project and join to give information in the form in which it is desired.
2. Precision: The usage of relational algebra and relational calculus in the manipulation of the relations between the tables ensures that there is no ambiguity, which may otherwise arise in establishing the linkages in a complicated network type database.

Disadvantages

- 1.Performance: If the number of tables between which relationships to be established are large and the tables themselves effect the performance in responding to the sql queries.
2. Physical Storage Consumption: With an interactive system an operation like join would depend upon the physical storage also.

Entity Cluster:

Entity cluster is temporary entity type which is not real because it is not used in final Entity Relationship Diagram. It is virtual entity which gives benefit to present the multiple entities and relationship in the ERD.

An Entity cluster is used to combine the multiple entities and relationship which relate each other into a single entity.

Explain the Steps to Create an ERD:

An Entity Relationship Diagram (ERD) is a visual representation of different entities within a system and how they relate to each other.

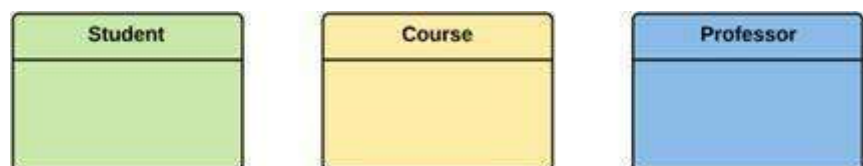
To create the ERD follow the given steps with example.



Example: In a university, a Student enrolls in Courses. A student must be assigned to at least one or more Courses. Each course is taught by a single Professor. To maintain instruction quality, a Professor can deliver only one course.

Step 1) Entity Identification

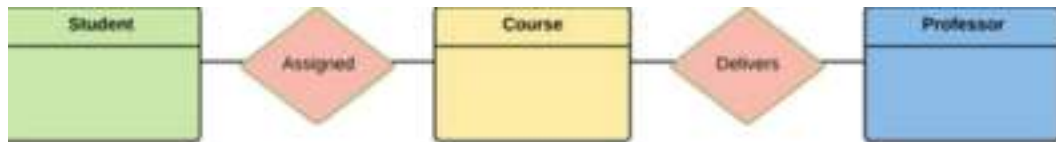
We have three entities Student, Course, Professor



Step 2) Relationship Identification

We have the following two relationships

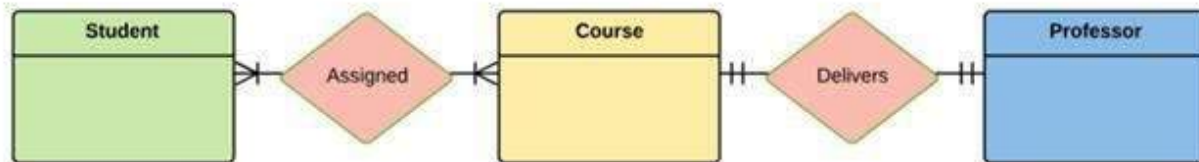
- The student is **assigned** a course
- Professor **delivers** a course



Step 3) Cardinality Identification

For them problem statement we know that,

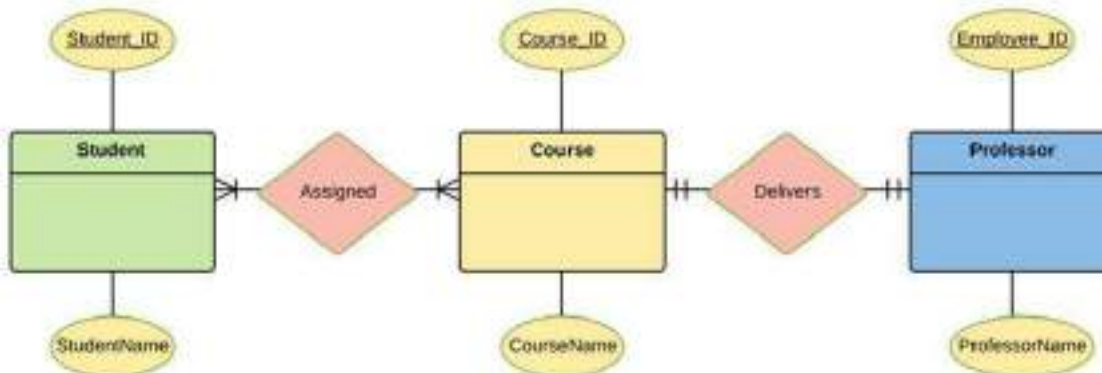
- A student can be assigned **multiple** courses
- A Professor can deliver only **one** course



Step 4) Identify Attributes

Now we need to study the files, forms, reports, data currently maintained by the organization to identify attributes. it's important to identify the attributes without mapping them to a particular entity. Once, you have a list of Attributes, you need to map them to the identified entities. Ensure an attribute is to be paired with exactly one entity.

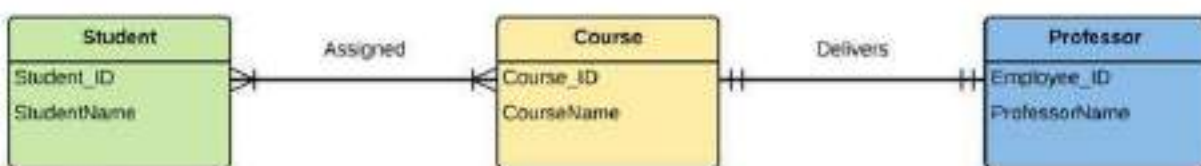
Entity	Primary Key	Attribute
Student	Student_ID	StudentName
Professor	Employee_ID	ProfessorName
Course	Course_ID	CourseName



For Course Entity, attributes could be Duration, Credits, Assignments, etc. For the sake of ease we have considered just one attribute.

Step 5) Create the ERD

A more modern representation of ERD Diagram



Unit-4

Q: What is structured query language?

SQL is the interface language between the user and the oracle database. The American National Standards Institute (ANSI) has accepted SQL as standard access language for Table Database Management Systems.

- ❖ IBM developed SQL in mid of 1970"s.
- ❖ Oracle incorporated in the year 1979.
- ❖ SQL used by IBM/DB2 and DS Database Systems.
- ❖ SQL adopted as standard language for RDBS by ANSI in 1989.

Rules of SQL Statement:

- ❖ An SQL statement starts with a verb. This verb may have additional nouns and adjectives.
- ❖ Each verb is followed by a number of clauses.
- ❖ Each clause has one or more parameters.
- ❖ A space separates clauses within an SQL statement.
- ❖ A comma separates parameters within a clause.
- ❖ A semicolon is used to terminate the SQL statement.

Q: What are various SQL Data Types?

It decides what type of data that will be stored inside each column when creating a table.

1. **NUMBER (P, S):** The NUMBER data type is used to store number (fixed or floating point). The precision (p) determines the number of total digits. Scale(s) is omitted then the default is zero.
Ex: percentage number (4, 2)
Here total digits: 4, digits after decimal point: 2
Ex: 98.56
2. **CHAR (Size):** A FIXED length string (can contain letters, numbers, and special characters). The *size* parameter specifies the column length in characters - can be from 0 to 255. Default is 1
3. **VARCHAR (Size) / VARCHAR2 (Size):** A VARIABLE length string (can contain letters, numbers, and special characters). The *size* parameter specifies the maximum column length in characters - can be from 0 to 65535
4. **DATE:** This data type is used to represent date and time. The standard format is DD-MM-YYYY as in 17-SEP-2009. The supported range is from '01-01-1000' to '31-12-9999'
5. **LONG:** This data type is used to store variable length character strings containing up to 2GB. Long data can be used to store arrays of binary data in ASCII form.

Q: What are Integrity Constraints available in ORACLE?

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

1. NOT NULL: When a column is defined as NOTNULL, then that column becomes a mandatory column. It implies that a value must be entered into the column if the record is to be accepted for storage in the table.

Syn: CREATE TABLE Table_Name (column_name data_type (size) NOT NULL,);

Ex: CREATE TABLE student (sno NUMBER(3) NOT NULL, name CHAR(10));

2. UNIQUE: The purpose of a unique key is to ensure that information in the column(s) is unique i.e. a value entered in column(s) defined in the unique constraint must not be repeated across the column(s). A table may have many unique keys.

Syn: CREATE TABLE Table_Name(column_name data_type(size) UNIQUE, ...);

Ex: CREATE TABLE student (sno NUMBER(3) UNIQUE, name CHAR(10));

3. CHECK: Specifies a condition that each row in the table must satisfy. To satisfy the constraint, each row in the table must make the condition either TRUE or unknown (due to a null).

Syn: CREATE TABLE Table_Name(columnname datatype(size) CHECK(logical expression), ...);

Ex: CREATE TABLE student (sno NUMBER(3), name CHAR(10), class CHAR(5), CHECK (class IN('CSE', 'CAD', 'VLSI')));

4. PRIMARY KEY: A field which is used to identify a record uniquely. A column or combination of columns can be created as primary key, which can be used as a reference from other tables. A table contains primary key is known as Master Table.

- ❖ It must uniquely identify each record in a table.
- ❖ It must contain unique values.
- ❖ It cannot be a null field.
- ❖ It cannot be multi port field.
- ❖ It should contain a minimum no. of fields necessary to be called unique.

Syn: CREATE TABLE Table_Name(column_name data_type(size) PRIMARY KEY,.);

Ex: CREATE TABLE faculty (fcode NUMBER(3) PRIMARY KEY, fname CHAR(10));

5. FOREIGN KEY: It is a table level constraint. We cannot add this at column level. To reference any primary key column from other table this constraint can be used. The table in which the foreign key is defined is called a **detail table**.

The table that defines the primary key and is referenced by the foreign key is called the **master table**.

Syn: CREATE TABLE Table_Name(column_name data_type(size) FOREIGN KEY (column_name) REFERENCES table_name);

6. Default: The default constraint is used to insert a default value into a column.

Ex: CREATE TABLE student (sno NUMBER(3)NOT NULL, name CHAR(10), colz varchar2(20) default „sgcsrc“);

Q: What are various SQL Sub languages and its Commands?

SQL sub Languages used for storing and managing data in relational database management system (RDMS). It is a standard language for Relational Database System. It enables a user to create, read, update and delete relational databases and tables.

There are five types of SQL commands. They are:

1. Data Definition Language (ddl)
2. Data Manipulation Language (DML)
3. Data Retrieval Language (DRL)
4. Transactional Control Language (T.C.L)
5. Data Control Language (D.C.L)

1. Data Definition Language (DDL):

It is used to define structure of a database. Create, Truncate, Drop, Alter, Rename

A. Create: It is used to create a new table.

Syn: Create table table_name(field_1 data_type(size),field_2 data_type(size),...);

Ex: Create table student(sno number(3),sname char(10),class char(5));

B. Truncate: TRUNCATE command removes all rows from a table, but the table structure and its columns, constraints, indexes, and so on remain.

Syn: Truncate table table_name;

Ex: **Truncate** table student;

C. Drop: It is used to delete the structure of a table. it permanently deletes the records of the table.

Syn: drop table table_name;

Ex: drop table student;

D. Rename: It is used to change the name of the existing database table.

Syn: rename table old_table_name to new_table_name;

Ex: rename table student to std;

E. Alter: It is used to modify or add fields to the structure of the table.

a) Alter Table ...Add...:It is used to add some extra fields to the existing table.

Syn: alter table table name add(new field_1 data_type(size), new field_2 data_type(size),...);

Ex: alter table std add(address char(10));

b) Alter Table...Modify...: It is used to change the width as well as data type of the fields of the existing tables.

Syn: alter table table_name **modify** (field_1 newdata_type(size), field_2 new data_type (size), field_new data_type(size));

Ex: alter table student **modify**(sname varchar(10),class varchar(5));

c) Alter table drop...:- It is used to delete a column of the existing tables.

Syn: **ALTER TABLE** table_name **DROP COLUMN** column_name;

Ex: Alter table student drop (sname);

2. Data Manipulation Language (DML):

It is used to manipulate the data of a table. Insert, Update, Delete

A. INSERT INTO: It is used to add records to the table.

i) Inserting a single record:

Syn: insert into table_name values (data_1,data_2,..data_n);

Ex: insert into student values (1,"ravi","b.sc","palakol"); **ii)**

Inserting multiple records:

Syn: insert into table name values(&data_1,&data_2,.....,&data_n);

Ex: insert into student values (&sno,"&sname","&class","&address");

Enter value for sno: 101

Enter value for name: Hemanth

Enter value for class: B.Sc

Enter value for name: Palakol

iii) Inserting all values

Syn:

Ex: insert into relation_name_1 values(value1,valu2,...valuen); insert into student(101,"rashmitha","mca","SKLM");

B. UPDATE-SET-

WHERE: This is used to update the content of a record in a table.

Syn: update table_name set field_name = data where field_name=data;

Ex: update student set sname = „kumar" where sno=1;

C. DELETE-FROM: this is used to delete all the records of a table.

a) delete-from: This is used to delete all the records of table. **Syn:** delete from table_name;

Ex: delete from student;

b) delete -from-where : this is used to delete a selected record from a table. **Syn:** delete from table_name where condition;

Ex: delete from student where sno = 2;

3. Data retrieval language (DRL):

It is used to retrieve the data from the table.

a. select from: It displays all records with all fields.

Syn: select * from table_name;

Ex: select * from dept;

deptno	dname	loc
-----	-----	-----
10	accounting	new york
20	research	dallas
30	sales	chicago
40	operations	boston

b. SELECT FROM: To display a set of fields for all records of table.

Syn: SELECT a set of fields FROM table_name;

Ex: select deptno, dname from dept;

DEPTNO	DNAME
-----	-----
10	accounting
20	research
30	sales
40	operations

c. SELECT - FROM -WHERE: This query is used to display a selected set of fields for a selected set of records of a table.

Syn: SELECT a set of fields FROM table_name WHERE condition;

Ex: select * from dept where deptno<=20;

deptno	dname	loc
-----	-----	-----
10	accounting	new york
20	research	dallas

d. SELECT - FROM -GROUP BY: This query is used to group all the records in a table together for each and every value of a specific key(s) and then display them for a selected set of fields in the table.

Syn: select a set of fields from table_name group by field_name;

ex: select empno, sum (salary) from emp group by empno;

empno	sum (salary)
-----	-----
1	3000
2	4000
3	5000
4	6000

4rows selected.

e. SELECT - FROM -ORDER BY: This query is used to display a selected set of fields from a table in an ordered manner base on some field.

Syn: select a set of fields from table_name order by field_name;

ex: select empno,ename,job from emp order by job;

empno	ename	job
-----	-----	-----
4	ravi	manager
2	aravind	Manager
1	sagar	clerk
3	Laki	clerk

4rows selected.

4. TRANSATIONAL CONTROL LANGUAGE (TCL): It is used to make the transactions permanent.

A. commit: this command is used to end a transaction and made them permanent.

syn: commit;

Ex:commit;

B. save point: save points are like marks to divide a very lengthy transaction to smaller once. They are used to identify a point in a transaction to which we can latter role back.

syn: save point id;

Ex: save point xyz;

C. Roll back: a role back command is used to undo the current transactions.

syn: 1. Rollback(current transaction can be roll back)

2. Rollback to save point id;

ex: Rollback;

Rollback to save point xyz;

5. DATA CONTROL LANGUAGE (D.C.L): DCL provides users with privilege commands

a. grant: the grant command allows granting various privileges to other users. **syn:** grant privileges on table_name to user_name;

ex: grant select, update on emp to hemanth;

b. revoke: it is used to withdraw the privileges that has been granted to the user.

syn: revoke privileges on table_name from user_name;

ex: revoke select, update on emp from hemanth;

6: Explain about views in SQL.

Views in SQL are considered as a virtual table. A view also contains rows and columns. View is a query that is stored as an object. It is similar to a table but it does not be stored in the database.

To create the view, we can select the fields from one or more tables present in the database. A view can either have specific rows based on certain condition or all the rows of a table.

Characteristics of views:

- We can use the name of the view anywhere in a SQL statement.
- Views are dynamically updated.
- Views provide a level of security in the database.
- Views may be used as the basis for reports.

Student_Detail

STU_ID	NAME	ADDRESS
1	Stephan	Delhi
2	Kathrin	Noida
3	David	Ghaziabad
4	Alina	Gurugram

Student_Marks

STU_ID	NAME	MARKS	AGE
1	Stephan	97	19
2	Kathrin	86	21
3	David	74	18
4	Alina	90	20
5	John	96	18

Creating view:

A view can be created using the CREATE VIEW statement. We can create a view from a single table or multiple tables.

Syntax: CREATE VIEW view_name AS SELECT column1, column2..... FROM table_name WHERE condition;

Creating View from a single table:

```
CREATE VIEW DetailsView AS SELECT NAME, ADDRESS FROM  
Student_Details WHERE STU_ID < 4;
```

Just like table query, we can query the view to view the data.

```
SELECT * FROM DetailsView;
```

Output:

NAME	ADDRESS
Stephan	Delhi
Kathrin	Noida
David	Ghaziabad

Creating View from multiple tables:

View from multiple tables can be created by simply include multiple tables in the SELECT statement.

In the given example, a view is created named MarksView from two tables Student_Detail and Student_Marks.

Query:

```
CREATE VIEW MarksView AS  
SELECT Student_Detail.NAME, Student_Detail.  
ADDRESS, Student_Marks.MARKS FROM Student_Detail, Student_Mark  
WHERE Student_Detail.NAME = Student_Marks.NAME;
```

To display data of View MarksView:

```
SELECT * FROM MarksView;
```

NAME	ADDRESS	MARKS
Stephan	Delhi	97
Kathrin	Noida	86
David	Ghaziabad	74
Alina	Gurugram	90

Deleting View:

A view can be deleted using the Drop View statement.

Syntax: DROP VIEW view_name;

Example:

If we want to delete the View **MarksView**, we can do this as:

```
DROP VIEW MarksView;
```

Q: Explain about SQL JOINS?

A JOIN clause is used to combine rows from two or more tables, based on a related column between them by using values common to each.

Here, it is noticeable that the join is performed in the WHERE clause. Several operators can be used to join tables, such as =, <, >, <>, <=, >=, !=, BETWEEN, LIKE, and NOT; they can all be used to join tables. However, the most common operator is the equal to symbol.

There are different types of joins available in SQL –

INNER JOIN – The INNER JOIN keyword selects records that have matching values in both tables.

Syntax: **The basic syntax of the INNER JOIN is as follows.**

```
SELECT table1.column1, table2.column2...
FROM table1
INNER JOIN table2
ON table1.common_field = table2.common_field;
```

Example: **Consider the following two tables.**

Table 1- CUSTOMERS Table is as follows.

OID	DATE	CUSTOMER_ID	AMOUNT
102	2009-10-08 00:00:00	3	3000
100	2009-10-08 00:00:00	3	1500
101	2009-11-20 00:00:00	2	1560
103	2008-05-20 00:00:00	4	2060

Table 2 – ORDERS Table is as follows.

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00

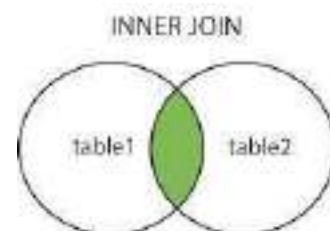
Now, let us join these two tables using the INNER JOIN as follows –

```
SQL> SELECT ID, NAME, AMOUNT, DATE FROM CUSTOMERS
```

INNER JOIN ORDERS ON CUSTOMERS.ID = ORDERS.CUSTOMER_ID;

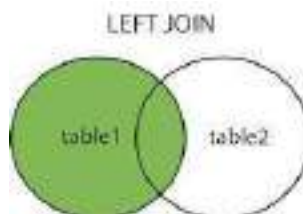
This would produce the following result.

ID	NAME	AMOUNT	DATE
3	kaushik	3000	2009-10-08 00:00:00
3	kaushik	1500	2009-10-08 00:00:00
2	Khilan	1560	2009-11-20 00:00:00
4	Chaitali	2060	2008-05-20 00:00:00



LEFT JOIN – The LEFT JOIN keyword returns all records from the left table (table1), and the matching records from the right table (table2). The result is 0 records from the right side, if there is no match.

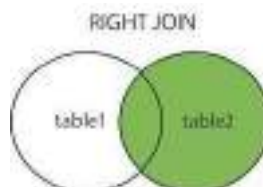
EX: SQL> SELECT ID, NAME, AMOUNT, DATE FROM CUSTOMERS LEFT JOIN ORDERS ON CUSTOMERS.ID = ORDERS.CUSTOMER_ID;



RIGHT JOIN – returns all rows from the right table, even if there are no matches in the left table.

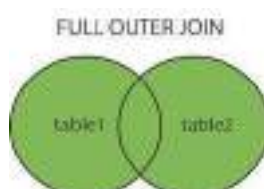
The RIGHT JOIN keyword returns all records from the right table (table2), and the matching records from the left table (table1). The result is 0 records from the left side, if there is no match.

EX:SQL> SELECT ID, NAME, AMOUNT, DATE FROM CUSTOMERS RIGHT JOIN ORDERS ON CUSTOMERS.ID = ORDERS.CUSTOMER_ID;



FULL JOIN – The FULL OUTER JOIN keyword returns all records when there is a match in left (table1) or right (table2) table records.

EX:SQL> SELECT ID, NAME, AMOUNT, DATE FROM CUSTOMERS FULL JOIN ORDERS ON CUSTOMERS.ID = ORDERS.CUSTOMER_ID;



SELF JOIN: A self join is a regular join, but the table is joined with itself.

EX: SQL> SELECT ID, NAME, AMOUNT, DATE FROM CUSTOMERS SELF JOIN ORDERS ON CUSTOMERS.ID = ORDERS.CUSTOMER_ID;

CARTESIAN JOIN: – returns the Cartesian product of the sets of records from the two or more joined tables

Q: What are SQL Set Operations?

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

Types of Set Operation

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



1. Union: The SQL Union operation is used to combine the result of two or more SQL SELECT queries. The union operation eliminates the duplicate rows from its result set.

In the union operation, all the number of data type and columns must be same in both the tables on which UNION operation is being applied.

Syntax:

SELECT column_name FROM table1 UNION SELECT column_name FROM table 2;

The First table

ID	NAME
1	Jack
2	Harry
3	Jackson

The Second table

ID	NAME
3	Jackson
4	Stephan
5	David

Union SQL query will be:

SELECT * FROM First UNION SELECT * FROM

Second; The result set table will look like:

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

2. Union All:

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

Syntax:

```
SELECT column_name FROM table1  
UNION ALL  
SELECT column_name FROM table2;
```

Example:

SELECT * FROM First UNION ALL SELECT * FROM
Second; The result set table will look like:

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

3. Intersect:

It is used to combine two SELECT statements. The Intersect operation returns the common rows from both the SELECT statements.

In the Intersect operation, the number of data type and columns must be the same.

It has no duplicates and it arranges the data in ascending order by default.

Syntax:

```
SELECT column_name FROM table1  
INTERSECT  
SELECT column_name FROM table2;
```

Example:

SELECT * FROM First INTERSECT SELECT * FROM
Second; The result set table will look like:

ID	NAME
3	Jackson

4. Minus:

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

Syntax:

```
SELECT column_name FROM table1  
MINUS  
SELECT column_name FROM table2;
```

Example

SELECT * FROM First MINUS SELECT * FROM
Second; The result set table will look like:

ID	NAME
1	Jack
2	Harry

Q: What is mean by SQL - Sub Query?

A Sub query or Inner query or a Nested query is a query within another SQL query and embedded within the WHERE clause.

A sub query is used to return data that will be used in the main query as a condition.

Sub queries can be used with the SELECT, INSERT, UPDATE, and DELETE statements along with the operators like =, <, >, >=, <=, IN, BETWEEN, etc.

There are a few rules that sub queries must follow:

- ❖ Sub queries must be enclosed within parentheses.
- ❖ A sub query can have only one column in the SELECT clause:
- ❖ Sub queries that return more than one row can only be used with multiple value operators such as the IN operator.
- ❖ The SELECT list cannot include any references to values that evaluate to a BLOB, ARRAY, CLOB, or NCLOB.

Sub queries with the SELECT Statement

Syntax: SELECT column_name [, column_name] FROM table1 [, table2]
 WHERE column_name OPERATOR
 (SELECT column_name [, column_name] FROM table1 [, table2
] [WHERE])

Example:

Consider the CUSTOMERS table having the following records –

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	35	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00

Now, let us check the following sub query with a SELECT statement.

```
SQL> SELECT * FROM CUSTOMERS WHERE ID IN (SELECT ID FROM  
CUSTOMERS WHERE SALARY > 4500) ;
```

This would produce the following result.

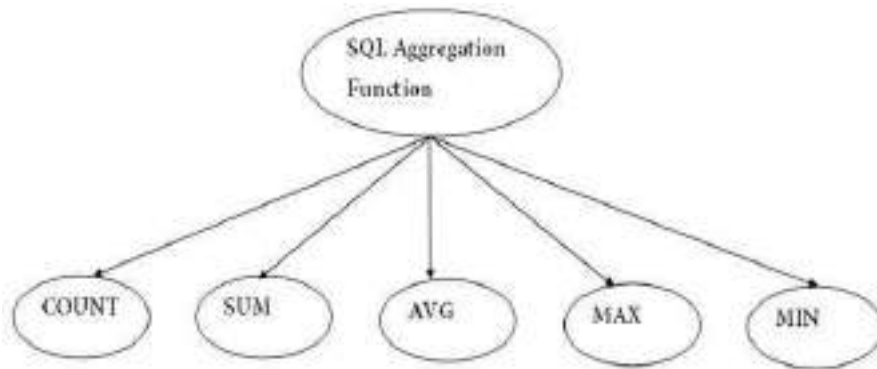
ID	NAME	AGE	ADDRESS	SALARY
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
7	Muffy	24	Indore	10000.00

Q: What are SQL Aggregate Functions?

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

It is also used to summarize the data.

Types of SQL Aggregation Function



1. COUNT FUNCTION

COUNT function is used to Count the number of rows in a database table. It can work on both numeric and non-numeric data types.

COUNT function uses the COUNT (*) that returns the count of all the rows in a specified table. COUNT (*) considers duplicate and Null.

Syntax: COUNT(*) or COUNT([ALL|DISTINCT] expression)

Example: SELECT COUNT(*) FROM STUDENT;

2. SUM Function:

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

Syntax: SUM() or SUM([ALL|DISTINCT] expression)

Example: SELECT AVG(COST) FROM PRODUCTS;

3. AVG function

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

Syntax: AVG() or AVG([ALL|DISTINCT] expression)

Example: SELECT AVG(COST) FROM PRODUCT_MAST;

4. MAX Function:

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

Syntax: MAX() or MAX([ALL|DISTINCT] expression)

Example: SELECT MAX(RATE) FROM PRODUCT_MAST;

5. MIN Function:

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

Syntax: MIN() or MIN([ALL|DISTINCT] expression)

Example: SELECT MIN(RATE) FROM PRODUCT_MAST;

Q: What is mean by Embed SQL?

Embedded SQL is a method of inserting inline SQL statements or queries into the code of a programming language, which is known as a host language. Because the host language cannot parse SQL, the inserted SQL is parsed by an embedded SQL preprocessor.

Embedded SQL is not supported by all relational database management systems (RDBMS). Oracle DB and PostgreSQL provide embedded SQL support.

UNIT V

PL/SQL

Syllabus: PL/SQL: Introduction, Shortcoming in SQL, Structure of PL/SQL, PL/SQL Language Elements, Data Types, Operators Precedence, Control Structure, Steps to Create a PL/SQL, Program, Iterative Control, Cursors, Steps to create a Cursors, Procedure, Function, Packages, Exceptions Handling, Database Triggers, Types of Triggers.

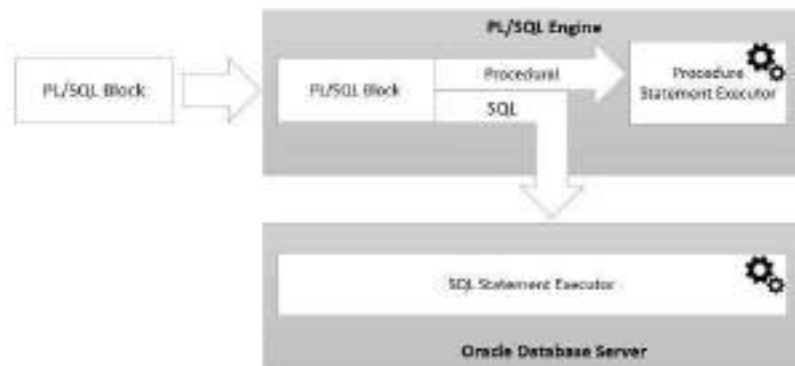
Q: What is PL/SQL?

PL/SQL stands for Procedural Language extension of SQL. PL/SQL is a combination of SQL along with the procedural features of programming languages. It was developed by Oracle Corporation in the early 90"s to enhance the capabilities of SQL.

PL/SQL adds many procedural constructs to SQL language to overcome some limitations of SQL. It is a standard and portable language for Oracle Database development. It is an embedded language and only executes in an Oracle Database.

Q: PL/SQL architecture

The following picture illustrates the PL/SQL architecture:



PL/SQL

engine is in charge of compiling PL/SQL code into byte-code and executes the executable code. The PL/SQL engine can only be installed in an Oracle Database server. Once you submit a PL/SQL block to the Oracle Database server, the PL/SQL engine collaborates with the SQL engine to compile and execute the code.

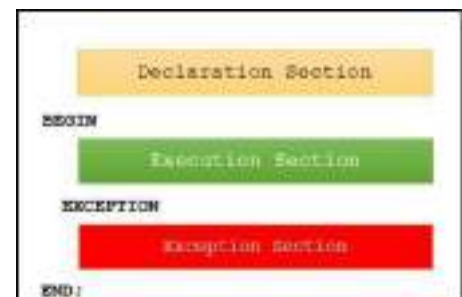
Q: Structure of PL/SQL:

PL/SQL is a block-structured language whose code is organized into blocks. Block is a smallest piece of PL/SQL code having logically related statements and declarations. A PL/SQL block consists of three sections: declaration, executable, and exception-handling sections. In a block, the executable section is mandatory while the declaration and

exception-handling sections are optional. Each PL/SQL program consists of SQL and PL/SQL statements which form a PL/SQL block.

A PL/SQL Block consists of three sections:

- The Declaration section (optional).
- The Execution section (mandatory).



c. The Exception (or Error) Handling section (optional).

a. Declaration Section: The Declaration section of a PL/SQL Block starts with the reserved keyword DECLARE. This section is optional and is used to declare any placeholders like any of Variables, Constants and Records, which stores data temporarily. Cursors are also declared in this section. Which are used to manipulate data in the execution section.

b. Execution Section: The Execution section of a PL/SQL Block starts with the reserved keyword BEGIN and ends with END. This is a mandatory section and is the section where the program logic is written to perform any task. The executable section must have at least one executable statement, even if it is the NULL statement which does nothing.

c. Exception Section: The Exception section of a PL/SQL Block starts with the reserved keyword EXCEPTION. This section is optional. 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 abruptly with errors.

Every statement in the above three sections must end with a semicolon .

PL/SQL blocks can be nested within other PL/SQL blocks.

Comments can be used to document code.

This is how a sample PL/SQL Block looks.

```
DECLARE
    Variable declaration
BEGIN
    Program Execution
EXCEPTION
    Exception handling
END;
```

Ex: **PL/SQL anonymous block**

The following example shows a simple PL/SQL anonymous block with one executable section.

```
BEGIN
    DBMS_OUTPUT.put_line ('Hello World!');
END;
```

In The above program executable section calls the DMBS_OUTPUT. PUT_LINE procedure to display the "Hello World" message on the screen.

Execute a PL/SQL anonymous block using SQL*Plus

Once you have the code of an anonymous block, you can execute it using SQL*Plus, which is a command-line interface.

```
SQL> SET SERVEROUTPUT ON;
SQL> BEGIN
2     DBMS_OUTPUT.PUT_LINE('Hello World');
3 END;
4 /
Hello World

PL/SQL procedure successfully completed.
```

First, connect to the Oracle Database server using a username and password. Second, turn on the server output using the SET SERVEROUTPUT ON command so that the DBMS_OUTPUT.PUT_LINE procedure will display text on the screen.

Q: PL/SQL Language Elements:

Like other programming languages PL/SQL also have specific character sets, operators, indicators, punctuations, identifiers, comments, etc. In the following sections we will discuss about various language elements of PL/SQL.

A. Character sets:

A PL/SQL program consists of text having specific set of characters. Character set may include the following characters:

- ❖ Alphabets, both in upper case [A–Z] and lower case [a–z]
- ❖ Numeric digits [0–9]
- ❖ Special characters [] , ' " / < > = < > % _ # & || ? |
- ❖ Blank spaces, tabs, and carriage returns.

Note: PL/SQL is not case sensitive

B. Lexical Units

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

- a) Delimiters
- b) Identifiers
- c) Literals
- d) Comments

A. Delimiters:

A delimiter is a simple or compound symbol that has a special meaning to PL/SQL. Simple symbol consists of one character, while compound symbol consists of more than one character.

For example, to perform the addition and exponentiation operation in PL/SQL, simple symbol delimiter + and compound symbol delimiter ** is used, respectively. PL/SQL supports following

Simple symbol delimiters:

+ - * / = > < ; % _ , () @ : _

Compound symbol delimiters legal in PL/SQL are as follows:

<> || ? |

B. Identifiers:

Identifiers are used in the PL/SQL programs to name the PL/SQL program items as constants, variables, cursors, cursor variables, subprograms, etc.

Identifiers can consists of alphabets, numerals, dollar signs, underscores, and number signs only. Any other characters like hyphens, slashes, blank spaces, etc. are illegal.

An identifier must begin with an alphabetic letter. An identifier cannot contain more than 30 characters.

Ex: A, A1, Share \$price, e_mail, phone#

C. Literals:

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

Ex: integer numeric literals: 100 006 -10 0 +10

A real literal: 0.0 -19.0 3.56219 +43.99 .6 7. -4.56

Character Literals: A character literal is an individual character enclosed by single quotes (apostrophes).

Ex: „A“, „@“, „5“, „?“ „,“ „(“

String Literals: A string literal is enclosed within single quotes and may consist of one or more characters.

Ex: Good Morning!”
“04-MAY-00”

Boolean Literals: Boolean literals are the predefined values TRUE, FALSE, and NULL. Boolean literals are values, not strings.

D. Comments:

Comments are used in the PL/SQL program to improve the readability and understandability of a program. A comment can appear anywhere in the program code. The compiler ignores comments. A PL/SQL comment may be a single-line or multiline.

Single-Line Comments: Single-line comments begin with a double hyphen (- -) anywhere on a line and extend to the end of the line.

Ex: -- start calculations

Multiline Comments: Multiline comments begin with a slash-asterisk (/*) and end with an asterisk slash(*/), and can span multiple lines.

Ex: /* HelloWorld!This is an example of multiline comments in PL/SQL */

Q: Variables and Constants:

Variables and constants can be used within PL/SQL block, in procedural statements and in SQL statements. These are used to store the values. As the program executes, the values of variables can change, but the values of constants cannot. However, it is must to declare the variables and constants, before using these in executable portion of PL/SQL.

Declaration: Variables and constants are declared in the Declaration section of PL/SQL block. These can be any of the SQL data type like CHAR, NUMBER,DATE, etc.

Variables Declaration: The syntax for declaring a variable is as follows:

Syn: identifier datatype;

Ex: DECLARE
Name VARCHAR2(10);
Age NUMBER(2);

Initializing the Variable: By default variables are initialized to NULL at the time of declaration. If we want to initialize the variable by some other value, syntax would be as follows:

Syn: Identifier data type:= value;

Or

Identifier data type DEFAULT value;

Ex: Joining date DATE := 01-JULY-99; (or)

Joining date DATE DEFAULT 01-JULY-99;

Constants Declaring: Declaration of constant is similar to declaration of variable, except the keyword CONSTANT precedes the data type and it must be initialized by some value.

Syn: identifier CONSTANT datatype := value;

Ex: To define the age limit as a constant, having value 30; the declaration statement would be as follows:

Age limit CONSTANT NUMBER := 30;

Q: Operators Precedence:

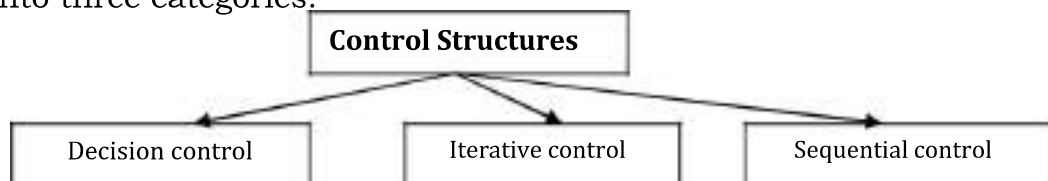
The operations within an expression are done in a particular order depending on their precedence (priority).

Below Table lists the operator's level of precedence from top to bottom. Operators listed in the same row have equal Precedence. Operators with higher precedence are applied first, but if parentheses are used, expression within innermost parenthesis is evaluated first.

Operator	operation
**, NOT	exponentiation, logical negation
+, -	identity, negation
*, /	multiplication, division
+, -,	addition, subtraction, concatenation
=, !=, <, >, <=, >=, IS NULL, LIKE, BETWEEN, IN	comparison
AND	Conjunction
OR	Disjunction

Q: Control Structure

Control Structure controls the flow of process. Control structure is broadly divided into three categories:



A. Decision Control:

A Decision control structure tests a condition to find out whether it is true or false and accordingly executes the different blocks of SQL statements. Conditional control is generally performed by IF statement.

There are three forms of IF statement.

1. IF-THEN,
2. IF-THEN-ELSE,
3. IF-THEN-ELSEIF.

IF-THEN:

It is the simplest form of IF condition. The syntax for this statement is as follows:

Syn: IF condition THEN
 Sequence of statements
 END IF;
Ex: IF A >B THEN
 HIGH := A;
 ENDIF;

IF-THEN-ELSE:

If the condition is TRUE then the control executes the statement 1, otherwise the control executes statement2

Syn: IF condition THEN
 sequence of statements1
 ELSE
 sequence of statements2
 END IF;
Ex: IF A >B THEN
 HIGH := A;
 ELSE
 HIGH := B;
 ENDIF;

IF-THEN-ELSIF: In this structure multiple conditions are involved

Syn: IF condition1 THEN
 sequence of statements1
 ELSIF condition2 THEN
 sequence of statements2
 ELSE
 sequence of statements3
 END IF;

B. Iterative Control:

In iterative control a group of statements are executed repeatedly till certain condition is true, and control exits from loop to next statement when the condition becomes false. There are mainly three types of loop statements:

1. LOOP,
2. WHILE-LOOP,
3. FOR-LOOP.

LOOP: LOOP is the simplest form of iterative control. It encloses a sequence of statements between the keywords LOOP and END LOOP. The general syntax for LOOP control is as follows:

Syn: LOOP
 Sequence of statements
 END LOOP;

With each iteration of the loop, the sequence of statements gets executed, then control reaches at the top of the loop. But a control structure like this gets entrapped into infinite loop. To avoid this it is must to use the key word **EXIT** and **EXIT-WHEN**.

LOOP – EXIT:

An EXIT statement within LOOP forces the loop to terminate unconditionally and passes the control to next statements. The general syntax for this is as follows:

```
Syn: LOOP
      IF condition1 THEN
      Sequence of statements1
      EXIT;
      ELSIF condition2 THEN
      Sequence of statements2
      EXIT
      ELSE
      Sequence of statements3
      EXIT;
      END IF;
    END LOOP;
```

LOOP – EXIT WHEN

The EXIT-WHEN statement terminates a loop conditionally. When the EXIT statement is encountered, the condition in the WHEN clause is evaluated. If the condition is true, the loop terminates and control passes to the next statement after the loop. The syntax for this is as follows:

```
Syn:    LOOP
          EXIT WHEN condition
          Sequence of statements
        END LOOP
```

WHILE-LOOP:

The WHILE statement with LOOP checks the condition. If it is true then only the sequence of statements enclosed within the loop gets executed. Then control resumes at the top of the loop and checks the condition again. The process is repeated till the condition is true. The control passes to the next statement outside the loop for FALSE or NULL condition.

```
Syn:    WHILE condition LOOP
          Sequence of statements
        END LOOP;
```

FOR-LOOP:

FOR loops iterate over a specified range of integers. The range is part of *iteration scheme*, which is enclosed by the keywords FOR and LOOP. A double dot (..) serves as the range operator. The syntax is as follows:

```
Syn: FOR counter IN lower limit .. higher limit LOOP
      sequence of statements
    END LOOP;
```

The sequence of statements is executed once for each integer in the range. After every iteration, the loop counter is incremented.

C. Sequential Control:

The sequential control unconditionally passes the control to specified unique label; it can be in the forward direction or in the backward direction. For sequential control GOTO statement is used.

Syntax: GOTO label;

 <<label>>
 Statement

Q: Steps to Create a PL/SQL Program

1. Open notepad file on oracle as shown in the syntax:
 Syn: SQL>edit filename;
2. Type the required PL/SQL program and save it.
3. Now exit from the notepad.
4. Type the following syntax on sql prompt to execute the
 program Syn: SQL>START filename;

Q: What are Cursors?

A cursor is a temporary work area created in the system memory when a SQL statement is executed. A cursor contains information on a select statement and the rows of data accessed by it. This temporary work area is used to store the data retrieved from the database, and manipulate this data. A cursor can hold more than one row, but can process only one row at a time. The set of rows the cursor holds is called the active set.

There are two types of cursors in PL/SQL:

a) Implicit cursors:

These are created by default when DML statements like, INSERT, UPDATE, and DELETE statements are executed. They are also created when a SELECT statement that returns just one row is executed.

Oracle provides few attributes called as implicit cursor attributes to check the status of DML operations. The cursor attributes available are **%FOUND**, **%NOTFOUND**, **%ROWCOUNT**, and **%ISOPEN**.

Example:

When you execute INSERT, UPDATE, or DELETE statements the cursor attributes tell us whether any rows are affected and how many have been affected.

When a SELECT... INTO statement is executed in a PL/SQL Block, implicit cursor attributes can be used to find out whether any row has been returned by the SELECT statement. PL/SQL returns an error when no data is selected.

Attributes	Return Value	Example
%FOUND	The return value is TRUE, if the DML statements like INSERT, DELETE and UPDATE affect at least one row and if SELECTINTO statement return at least one row.	SQL%FOUND
	The return value is FALSE, if DML statements like INSERT, DELETE and UPDATE do not	

affect row and if SELECT....INTO statement

do not return a row.

%NOTFOUND	The return value is FALSE, if DML statements like INSERT, DELETE and UPDATE at least one row and if SELECTINTO statement return at least one row. The return value is TRUE, if a DML statement like INSERT, DELETE and UPDATE do not affect even one row and if SELECTINTO statement does not return a row.	SQL%NOTFOUND
%ROWCOUNT	Return the number of rows affected by the SQL DML operations INSERT, DELETE, UPDATE, SELECT	SQL%ROWCOUNT

For Example: Consider the PL/SQL Block that uses implicit cursor attributes as shown below:

```
DECLARE var_rows number(5);
BEGIN
    UPDATE employee
    SET salary = salary + 1000;
    IF SQL%NOTFOUND THEN
        dbms_output.put_line('None of the salaries where updated');
    ELSIF SQL%FOUND THEN
        var_rows := SQL%ROWCOUNT;
        dbms_output.put_line('Salaries for ' || var_rows || 'employees are updated');
    END IF;
END;
```

In the above PL/SQL Block, the salaries of all the employees in the „employee“ table are updated. If none of the employee"s salary are updated we get a message 'None of the salaries where updated'. Else we get a message like for example, 'Salaries for 1000 employees are updated' if there are 1000 rows in „employee“ table.

b) Explicit cursors:

They must be created when you are executing a SELECT statement that returns more than one row. Even though the cursor stores multiple records, only one record can be processed at a time, which is called as current row. When you fetch a row the current row position moves to next row.

An **explicit cursor** is defined in the declaration section of the PL/SQL Block. It is created on a SELECT Statement which returns more than one row. We can provide a suitable name for the cursor.

The General Syntax for creating a cursor is as given below:

```
CURSOR cursor_name IS select_statement;
```

Here cursor_name – A suitable name for the cursor.

select_statement – A select query which returns multiple rows. **How to use Explicit Cursor?**

There are four steps in using an Explicit Cursor.

- ❖ **DECLARE** the cursor in the declaration section.
- ❖ **OPEN** the cursor in the Execution Section.
- ❖ **FETCH** the data from cursor into PL/SQL variables or records in the Execution Section.
- ❖ **CLOSE** the cursor in the Execution Section before you end the PL/SQL Block.

Declaring a Cursor in the Declaration Section:

```
DECLARE
CURSOR emp_cur IS
SELECT * FROM emp_tbl WHERE salary > 5000;
```

In the above example we are creating a cursor „emp_cur“ on a query which returns the records of all the employees with salary greater than 5000. Here „emp_tbl“ is the table which contains records of all the employees.

How to access an Explicit Cursor?

These are the three steps in accessing the cursor.

- 1) Open the cursor.
- 2) Fetch the records in the cursor one at a time.
- 3) Close the cursor.

General Syntax to open a cursor is: OPEN cursor_name;

General Syntax to fetch records from a cursor is:

```
FETCH cursor_name INTO record_name;
OR
FETCH cursor_name INTO variable_list;
```

General Syntax to close a cursor is:

```
CLOSE cursor_name;
```

When a cursor is opened, the first row becomes the current row. When the data is fetched it is copied to the record or variables and the logical pointer moves to the next row and it becomes the current row. On every fetch statement, the pointer moves to the next row. If you want to fetch after the last row, the program will throw an error. When there is more than one row in a cursor we can use loops along with explicit cursor attributes to fetch all the records.

Points to remember while fetching a row:

- We can fetch the rows in a cursor to a PL/SQL Record or a list of variables created in the PL/SQL Block.
- If you are fetching a cursor to a PL/SQL Record, the record should have the same structure as the cursor.
- If you are fetching a cursor to a list of variables, the variables should be listed in the same order in the fetch statement as the columns are present in the cursor.

General Form of using an explicit cursor is:

```
DECLARE
```

```

variables;
records;
create a cursor;
BEGIN
  OPEN cursor;
  FETCH cursor;
  process the records;
  CLOSE cursor;
END;

```

Lets Look at the example below **Example 1:**

```

DECLARE
emp_rec emp_tbl%rowtype;
CURSOR emp_cur IS
SELECT *
FROM
WHERE salary > 10;
BEGIN
  OPEN emp_cur;
  FETCH emp_cur INTO emp_rec;
  dbms_output.put_line(emp_rec.first_name || '
' || emp_rec.last_name);
  CLOSE emp_cur;
END;

```

In the above example, first we are creating a record „emp_rec“ of the same structure as of table „emp_tbl“ in line no 2. We can also create a record with a cursor by replacing the table name with the cursor name. Second, we are declaring a cursor „emp_cur“ from a select query in line no 3 - 6. Third, we are opening the cursor in the execution section in line no 8. Fourth, we are fetching the cursor to the record in line no 9. Fifth, we are displaying the first_name and last_name of the employee in the record emp_rec in line no 10. Sixth, we are closing the cursor in line no 11.

What are Explicit Cursor Attributes?

Oracle provides some attributes known as Explicit Cursor Attributes to control the data processing while using cursors. We use these attributes to avoid errors while accessing cursors through OPEN, FETCH and CLOSE Statements.

When does an error occur while accessing an explicit cursor?

- When we try to open a cursor which is not closed in the previous operation.
- When we try to fetch a cursor after the last operation.

These are the attributes available to check the status of an explicit cursor.

Attributes	Return values	Example
%FOUND	TRUE, if fetch statement returns at least one row.	Cursor_name%FOUND
	FALSE, if fetch statement doesn't return a row.	

%NOTFOUND	TRUE, , if fetch statement doesn't return a row.	Cursor_name%NOTFOUND
	FALSE, if fetch statement returns at least one row.	Cursor_name%ROWCOUNT
%ROWCOUNT	The number of rows fetched by the fetch statement	Cursor_name%ISNAME
	If no row is returned, the PL/SQL statement returns an error.	
%ISOPEN	TRUE, if the cursor is already open in the program	
	FALSE, if the cursor is not opened in the program.	

Both implicit and explicit cursors have the same functionality, but they differ in the way they are accessed.

Q: Explain Procedures

A procedure is a subprogram that performs some specific task, and stored in the data dictionary.

A procedure must have a name, so that it can be invoked or called by any PL/SQL program that appears within an application.

Procedures can take parameters from the calling program and perform the specific task.

Benefits of Procedures and Functions

Stored procedures and functions have many benefits

It modifies one routine to affect multiple applications.

It modifies one routine to eliminate duplicate testing.

It ensures that related actions are performed together, or not at all, by doing the activity through a single path.

It avoids PL/SQL parsing at runtime by parsing at compile time.

It reduces the number of calls to the database and database network traffic by bundling the commands.

Defining and Creating Procedures

A procedure consists of two parts:

1. Specification
2. body.

The **specification** starts with keyword PROCEDURE and ends with parameter list or procedure name.

The procedures may accept parameters or may not.

Procedures that do not accept parameters are written parentheses.

The **procedure body** starts with the keyword IS and ends with keyword END.

The procedure body is further subdivided into three parts:

1. Declarative part which consists of local declarations placed between keywords IS and BEGIN.

2. Executable part, which consists of actual logic of the procedure, including statements such as BEGIN and EXCEPTION.

The syntax for creating a procedure is follows:

```
CREATE OR REPLACE PROCEDURE procedure_name
  [(argument {IN, OUT, IN OUT} data type, . . . . .)] {IS,
  AS} [local variable declarations]
BEGIN
  executable statements
EXCEPTION
  exception handlers
END [procedure name];
```

Here

Create: Creates a new procedure, if a procedure of same name already exists, it gives an error.

Replace: Creates a procedure, if a procedure of same name already exists, it replace the older one by the new procedure definition.

Argument: It is the name of the argument to the procedure.

IN: Specifies that a value for the argument must be specified when calling the procedure.

OUT: Specifies that the procedure pass a value for this argument back to its calling environment after execution.

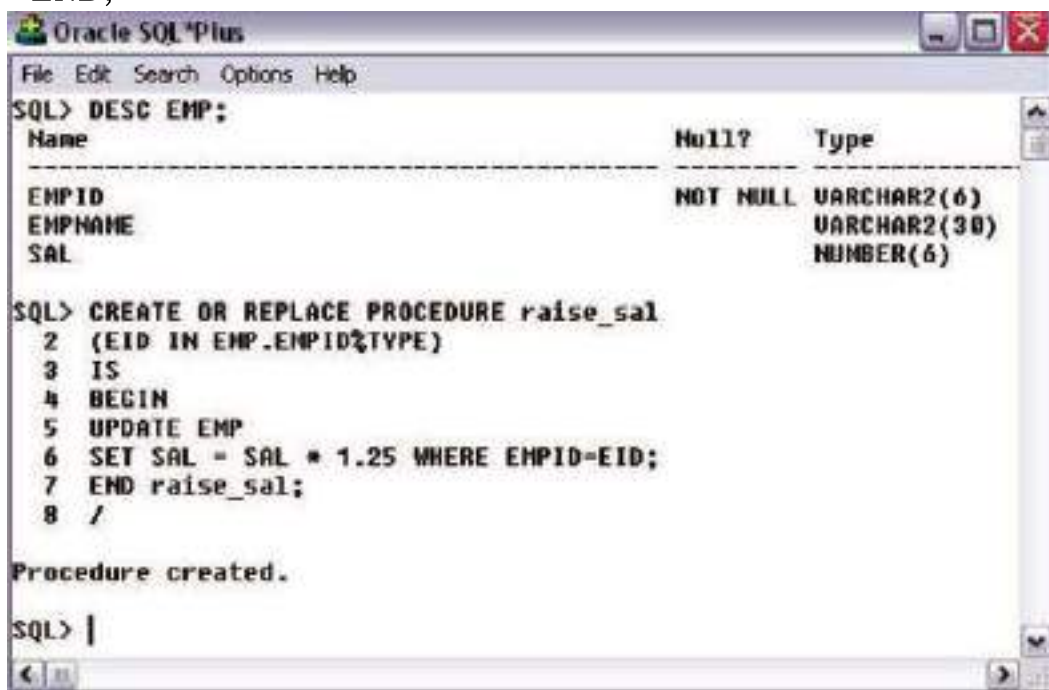
IN OUT: Specifies that a value for the argument must be specified when calling the procedure and that the procedure passes a value for this argument back to its calling environment after execution. If no value is specified then it takes the default value IN.

Datatype: It is the unconstrained datatype of an argument. It supports any data type supported by PL/SQL. No constraints like size constraints or NOT NULL constraints can be imposed on the data type. However, you can put on the size constraint indirectly.

Ex:

```
CREATE OR REPLACE PROCEDURE greetings
AS BEGIN
    dbms_output.put_line('Hello World!');
END;
```

/



The screenshot shows the Oracle SQL*Plus interface. At the top, there's a menu bar with 'File', 'Edit', 'Search', 'Options', and 'Help'. Below the menu bar, the command prompt shows the following sequence of commands and their outputs:

```
SQL> DESC EMP;
Name                               Null?    Type
-----
EMPID                               NOT NULL VARCHAR2(6)
EMPNAME                             VARCHAR2(30)
SAL                                  NUMBER(6)

SQL> CREATE OR REPLACE PROCEDURE raise_sal
2  (EID IN EMP.EMPID%TYPE)
3  IS
4  BEGIN
5  UPDATE EMP
6  SET SAL = SAL * 1.25 WHERE EMPID=EID;
7  END raise_sal;
8  /

Procedure created.

SQL> |
```

Executing/Invoking a Procedure

The syntax used to execute a procedure depends on the environment from which the procedure is being called. From within SQLPLUS, a procedure can be executed by using the EXECUTE command, followed by the procedure name. Any arguments to be passed to the procedure must be enclosed in parentheses following the procedure name.

Syn: EXECUTE procedurename(paramaters);

Removing a Procedure

To remove a procedure completely from the database, following command is used:

Syn: DROP PROCEDURE < procedurename>;

Q: What is a Function:

A Function is similar to procedure except that it must return one and only one value to the calling program.

The exact syntax for defining a function is given below

```
CREATE OR REPLACE FUNCTION [schema.] functionname
    [(argument IN datatype, . . . .)] RETURN datatype
    {IS,AS} [local variable declarations];
BEGIN
    executable statements;
EXCEPTION
    exception handlers;
END [functionname];
```

Thus a function has two parts: function specification and function body.

The function specification begins with keyword FUNCTION and ends with RETURN clause which indicates the data type of the value returned by the function.

Function body is enclosed between the keywords IS and END. Sometimes END is followed by function name, but this is optional. function body also is composed of three parts:

- declarative part,
- executablepart,
- Error/exception handling part (optional).

```
Oracle SQL*Plus
File Edit Search Options Help
SQL> SELECT * FROM EMP;

EMPID  EMPNAME                SAL
-----
E101   KARTHIKEYAN             6250
E102   ANAND                   2500
E103   RAJA                    1900

SQL> SELECT GET_SAL('E102') FROM DUAL;

GET_SAL('E102')
-----
2500

SQL> |
```

```
Oracle SQL*Plus
File Edit Search Options Help
SQL> DESC EMP;
Name                               Null?    Type
-----
EMPID                               NOT NULL VARCHAR2(6)
EMPNAME                             VARCHAR2(30)
SAL                                 NUMBER(6)

SQL> CREATE OR REPLACE FUNCTION get_sal
2 (EID IN EMP.EMPID%TYPE)
3 RETURN NUMBER
4 IS
5 EMPSAL EMP.SAL%TYPE :=0;
6 BEGIN
7 SELECT SAL INTO EMPSAL
8 FROM EMP WHERE EMPID=EID;
9 RETURN(EMPSAL);
10 END;
11 /

Function created.

SQL>
```

Removing a Function

To remove a function, use following command:

Syn: DROP FUNCTION <FUNCTION NAME>;

Parameters

Parameters are the link between a subprogram code and the code calling the subprogram.

Parameter Modes

Parameter modes define the behavior of formal parameters of subprograms.

There are three types of parameter modes: IN, OUT, IN/OUT.

IN Mode

IN mode is used to pass values to the called subprogram. Inside the called subprogram, an IN parameter acts like a constant and hence it cannot be assigned a new value.

OUT Mode

An OUT parameter returns a value back to the caller subprogram.

IN/OUT

An IN/OUT parameter performs the duty of both IN parameter as well as OUT parameter.

Difference between Function and Procedure

1. A procedure never returns a value to the calling portion of code, whereas a function returns exactly *one value* to the calling program.
2. As functions are capable of returning a value, they can be used as elements of SQL expressions, whereas the procedures cannot. However, user-defined functions cannot be used in CHECK or DEFAULT constraints and cannot manipulate database values, to obey function purity rules.
3. It is mandatory for a function to have at least one RETURN statement, whereas for procedures there is no restriction. A procedure may have a RETURN statement or may not.

Q: Explain Packages

PL/SQL Packages is schema object and collection of related data type (variables, constants), cursors, procedures, functions are defining within a single context. Package are device into two part,

1. Package Specification
2. Package Body

Package specification block you can define variables, constants, exceptions and package body you can create procedure, function, and subprogram.

PL/SQL Specification: This contains the list of variables, constants, functions, procedure names which are the part of the package. PL/SQL specification are public declaration and visible to a program.

Syn:

```
CREATE[OR REPLACE] PACKAGE package_name IS|AS [declaration ]
BEGIN
    [PROCEDURE]
    [FUNCTION ]
END[package_name];
```

PL/SQL Body : This contains the actual PL/SQL statement code implementing the logics of functions, procedures which are you already before declare in "Package specification".

Syn:

```
CREATE[OR REPLACE] PACKAGE BODY package_name IS|AS [declaration]
BEGIN
    [initialization_statement]
    [PROCEDURE]
    [FUNCTION ]
    EXCEPTION
        WHEN built-in_exception_name_1 THEN
            User defined statement (action) will be taken;
END;
```

PL/SQL Package Example

PL/SQL Package example step by step explain to you, you are create your own package using this reference example. We have emp1 table having employee information,

Package Specification Code

Create Package specification code for defining procedure, function IN or OUT parameter and execute package specification program.

```
CREATEor REPLACE PACKAGE pkg1 IS|AS PROCEDURE pro1
    (noin number, name out varchar2);
    FUNCTION fun1
        (noin number)
        RETURN varchar2;
END;
```

/

Package Body Code

Create Package body code for implementing procedure or function that are defined package specification. Once you implement execute this program.

```
CREATEor REPLACE PACKAGE BODY pkg1
IS
    PROCEDURE pro1(noin number,info our varchar2)
        IS
        BEGIN
            SELECT*INTO temp FROM emp1 WHERE eno =no;
        END;
    FUNCTION fun1(noin number)return varchar2
        IS
        name varchar2(20);
        BEGIN
            SELECT ename INTO name FROM emp1 WHERE eno =no;
            RETURN name;
        END;
END;
```

/

PL/SQL Program calling Package

Now we have a one package pkg1, to call package defined function, procedures also pass the parameter and get the return result.

pkg_prg.sql

```
DECLARE
    no number :=&no;
    name varchar2(20);
BEGIN
    pkg1.pro1(no,info);
    dbms_output.put_line('Procedure Result');
    dbms_output.put_line(info.eno || ' ' ||
        info.ename || ' ' ||
        info.edept || ' ' ||
        info.esalary || ' ' || );
    dbms_output.put_line('Function Result');
    name := pkg1.fun1(no);
    dbms_output.put_line(name);
END;
/
```

Output:

```
SQL>@pkg_prg
no number &n=2
Procedure Result
2 marks jems Program Developer 38K
Function Result
```

Advantages of Packages

1. You can create package to **store all related** functions and procedures are grouped together into single unit called packages.
2. Packages are reliable to **granting a privileges**.
3. All function and procedure within a package can **share variable** among them.
4. Packages are support **overloading** to overload functions and procedures.
5. Packages are **improve the performance** to loading the multiple object into memory at once, therefore, subsequent calls to related program doesn't required to calling physically I/O.

Package is **reduce the traffic** because all block execute all at once.

Q: What is Exception Handling?

PL/SQL provides a feature to handle the Exceptions which occur in a PL/SQL Block known as exception Handling. Using Exception Handling we can test the code and avoid it from exiting abruptly. When an exception occurs a messages which explains its cause is recieved.

PL/SQL Exception message consists of three parts.

- 1) Type of Exception
- 2) An Error Code
- 3) A message

By Handling the exceptions we can ensure a PL/SQL block does not exit abruptly.

2) Structure of Exception Handling.

The General Syntax for coding the exception section

DECLARE

Declaration section

BEGIN

Exception section

EXCEPTION

WHEN ex_name1 THEN

-Error handling statements

WHEN ex_name2 THEN

-Error handling statements

WHEN Others THEN

-Error handling statements

END;

General PL/SQL statments can be used in the Exception Block.

When an exception is raised, Oracle searches for an appropriate exception handler in the exception section. For example in the above example, if the error raised is 'ex_name1 ', then the error is handled according to the statements under it. Since, it is not possible to determine all the possible runtime errors during testing fo the code, the 'WHEN Others' exception is used to manage the exceptions that are not explicitly handled. Only one exception can be raised in a Block and the control does not return to the Execution Section after the error is handled.

If there are nested PL/SQL blocks like this.

DELCLARE

Declaration section

BEGIN

DECLARE

Declaration section

BEGIN

Execution section

EXCEPTION

Exception section

END;

EXCEPTION

Exception section

END;

In the above case, if the exception is raised in the inner block it should be handled in the exception block of the inner PL/SQL block else the control moves to the Exception block of the next upper PL/SQL Block. If none of the blocks handle the exception the program ends abruptly with an error.

3) Types of Exception.

There are 3 types of Exceptions.

a) Named System Exceptions

b) Unnamed System Exceptions

c) User-defined Exceptions

a) Named System Exceptions

System exceptions are automatically raised by Oracle, when a program violates a RDBMS rule. There are some system exceptions which are raised frequently, so they are pre-defined and given a name in Oracle which are known as Named System Exceptions.

For example: NO_DATA_FOUND and ZERO_DIVIDE are called Named System exceptions.

Named system exceptions are:

- 1) Not Declared explicitly,
- 2) Raised implicitly when a predefined Oracle error occurs,
- 3) caught by referencing the standard name within an exception-handling routine.

Exception Name	Reason	Error Number
CURSOR_ALREADY_OPEN	When you open a cursor that is already open.	ORA-06511
INVALID_CURSOR	When you perform an invalid operation on a cursor like closing a cursor, fetch data from a cursor that is not opened.	ORA-01001
NO_DATA_FOUND	When a SELECT...INTO clause does not return any row from a table.	ORA-01403
TOO_MANY_ROWS	When you SELECT or fetch more than one row into a record or variable.	ORA-01422
ZERO_DIVIDE	When you attempt to divide a number by zero.	ORA-01476

For Example: Suppose a NO_DATA_FOUND exception is raised in a proc, we can write a code to handle the exception as given below.

```
BEGIN
    Execution section
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        dbms_output.put_line ('A SELECT...INTO did not return any row.');
```

END;

b) Unnamed System Exceptions

Those system exception for which oracle does not provide a name is known as unnamed system exception. These exception do not occur frequently. These Exceptions have a code and an associated message.

There are two ways to handle unnamed sysyem exceptions:

1. By using the WHEN OTHERS exception handler, or
2. By associating the exception code to a name and using it as a named exception.

We can assign a name to unnamed system exceptions using a **Pragma** called **EXCEPTION_INIT**. **EXCEPTION_INIT** will associate a predefined Oracle error number to a programmer_defined exception name.

Steps to be followed to use unnamed system exceptions
are They are raised implicitly.

If they are not handled in WHEN Others they must be handled explicitly.
To handle the exception explicitly, they must be declared using Pragma
EXCEPTION_INIT as given above and handled referencing the user-defined
exception name in the exception section.

The general syntax to declare unnamed system exception using
EXCEPTION_INIT is:

```
DECLARE
    exception_name EXCEPTION;
    PRAGMA
        EXCEPTION_INIT (exception_name, Err_code);
BEGIN
    Execution section
EXCEPTION
    WHEN exception_name THEN
        handle the exception
END;
```

For Example: Lets consider the product table and order_items table from sql
joins.

Here product_id is a primary key in product table and a foreign key in
order_items table. If we try to delete a product_id from the product table when it
has child records in order_id table an exception will be thrown with oracle code
number -2292. We can provide a name to this exception and handle it in the
exception section as given below.

```
DECLARE
    Child_rec_exception EXCEPTION;
    PRAGMA
        EXCEPTION_INIT (Child_rec_exception, -2292);
BEGIN
    Delete FROM product where product_id= 104;
EXCEPTION
    WHEN Child_rec_exception
THEN Dbms_output.put_line('Child records are present for this
    product_id.');
```

END;

/

c) User-defined Exceptions

Apart from system exceptions we can explicitly define exceptions based on
business rules. These are known as user-defined exceptions. Steps to be
followed to use user-defined exceptions:

They should be explicitly declared in the declaration section.

They should be explicitly raised in the Execution Section.

They should be handled by referencing the user-defined exception name in the exception section.

Triggers

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

- ❖ A database manipulation (DML) statement (DELETE, INSERT, or UPDATE)
- ❖ A database definition (DDL) statement (CREATE, ALTER, or DROP).
- ❖ A database operation (SERVERERROR, LOGON, LOGOFF, STARTUP, or SHUTDOWN).
- ❖ Triggers can be defined on the table, view, schema, or database with which the event is associated.

Purpose of Triggers

- ❖ To generate data automatically
- ❖ To enforce complex integrity constraints
- ❖ To customize complex security authorization
- ❖ To maintain replicate tables
- ❖ To audit data modifications.

Creating Triggers

```
CREATE [OR REPLACE ] TRIGGER trigger_name  
  {BEFORE | AFTER | INSTEAD OF }  
  {INSERT [OR] | UPDATE [OR] | DELETE} [OF col_name] ON table_name  
  [REFERENCING OLD AS o NEW AS n] [FOR EACH ROW] WHEN  
  (condition)
```

DECLARE Declaration-statements

BEGIN Executable-statements

EXCEPTION Exception-handling-statements

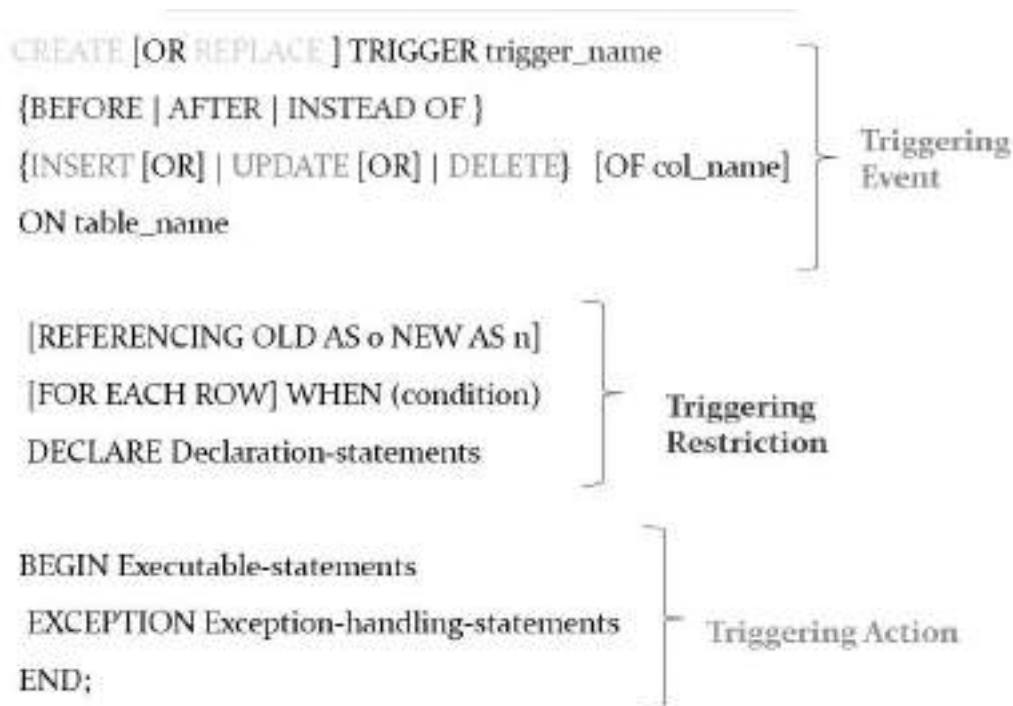
END;

- ❖ CREATE [OR REPLACE] TRIGGER trigger_name – Creates or replaces an existing trigger with the *trigger_name*.
- ❖ {BEFORE | AFTER | INSTEAD OF} – This specifies when the trigger will be executed. The INSTEAD OF clause is used for creating trigger on a view.
- ❖ {INSERT [OR] | UPDATE [OR] | DELETE} – This specifies the DML operation.
- ❖ [OF col_name] – This specifies the column name that will be updated.
- ❖ [ON table_name] – This specifies the name of the table associated with the trigger.
- ❖ [REFERENCING OLD AS o NEW AS n] – This allows you to refer new and old values for various DML statements, such as INSERT, UPDATE, and DELETE.
- ❖ [FOR EACH ROW] – This specifies a row-level trigger, i.e., the trigger will be executed for each row being affected. Otherwise the trigger will

execute just once when the SQL statement is executed, which is called a table level trigger.

- ❖ **WHEN (condition)** – This provides a condition for rows for which the trigger would fire. This clause is valid only for row-level triggers.

Parts of Triggers



Types of PL/SQL Triggers

There are two types of triggers based on the which level it is triggered.

- 1) Row level trigger** - An event is triggered for each row updated, inserted or deleted.
- 2) Statement level trigger** - An event is triggered for each sql statement executed.

PL/SQL Trigger Execution Hierarchy

The following hierarchy is followed when a trigger is fired.

- 1) BEFORE statement trigger** fires first.
- 2) Next BEFORE row level trigger** fires, once for each row affected.
- 3) Then AFTER row level trigger** fires once for each affected row. This events will alternates between BEFORE and AFTER row level triggers.
- 4) Finally the AFTER statement level trigger** fires.

For Example: Let's create a table 'product_check' which we can use to store messages when triggers are fired.

```
CREATE TABLE product
(Message varchar2(50),
Current_Date number(32) );
```

Let's create a BEFORE and AFTER statement and row level triggers for the product table.

1) BEFORE UPDATE, Statement Level: This trigger will insert a record into the table 'product_check' before a sql update statement is executed, at the statement level.

```
CREATE or REPLACE TRIGGER Before_Update_Stat_product
BEFORE
UPDATE ON
product Begin
INSERT INTO product_check Values('Before update, statement
level',sysdate); END;
/
```

2) BEFORE UPDATE, Row Level: This trigger will insert a record into the table 'product_check' before each row is updated.

```
CREATE or REPLACE TRIGGER Before_Upddate_Row_product
BEFORE
UPDATE ON product
FOR EACH ROW
BEGIN
INSERT INTO product_check Values('Before update row level',sysdate);
END;
/
```

3) AFTER UPDATE, Statement Level: This trigger will insert a record into the table 'product_check' after a sql update statement is executed, at the statement level.

```
CREATE or REPLACE TRIGGER After_Update_Stat_product
AFTER
UPDATE ON product
BEGIN
INSERT INTO product_check Values('After update, statement level', sysdate);
End;
/
```

4) AFTER UPDATE, Row Level: This trigger will insert a record into the table 'product_check' after each row is updated.

```
CREATE or REPLACE TRIGGER After_Update_Row_product
AFTER
insert On product
FOR EACH ROW
BEGIN
INSERT INTO product_check Values('After update, Row level',sysdate);
END;
/
```



STUDENT NAME: _____

SUBJECT _____ CLASS: _____

VILLAGE: _____

PH NO: _____

COLLEGE: _____