Topics:

Introduction to DBMS

- **Basics of Databases**
  - **Data, Information, Field, Record, File**
- **Database Management System**
  - **Meaning**
  - **Advantages and Disadvantages (only list)**
- **Components of DBMS**
  - **Classification of DBMS Users**
  - **Structure of DBMS (only Diagram)**
- **The three-schema architecture for a DBMS**
- **Data Models**
  - **Hierarchical**
  - **Network**
  - **Relational**
- **Relational Data Model**
  - **Concept**
  - **Terminologies**
    - **Tuple, Attribute, domain, relation (definition)**
- **Operations on Data**
  - **DDL**
  - **DML**
- **Keys**
  - **Super key, Candidate Key, Primary Key, Alternate Key, Foreign Key**
- **Relationships and Relationship Types**

Data:

Facts and statistics collected together for reference or analysis.

Information:

When data is processed, organized, structured or presented in a given context so as to make it useful, it is called information.

Database:

A structure set of data held in a computer, especially one that is accessible in various ways.

DBMS:

A DataBase Management System is the software system that allows users to define, create and maintain a database and provides controlled access to the data.

Examples: dBase, FoxPro, IMS and Oracle, MySQL, SQL Server, DB2 etc.

- **BENEFITS OF DBMS:**

1. The amount of data redundancy in stored data can be reduced.
2. No more data inconsistencies.
3. Stored data can be shared by a single or multiple users.
4. Standards can be set and followed.
5. Data integrity can be maintained.
6. Security of data can be simply implemented.
7. Data independence can be achieved.

Record Type:

The physical representation of an entity set is made by aggregating the attributes used to model the entity set. Such a representation is called a record type.

An instance of a record type is a **record occurrence.**

Field:

The stored value of the attribute is referred to as an attribute value, stored field or field.

File:

File is a collection of identical record type occurrences pertaining to an entity set and is labelled to identify the entity set.

Entity:

Entities are the basic units used in modelling classes of concrete or abstract objects. Entities can have concrete existence or constitute ideas or concepts.

Examples: building, room, chair, transaction, course, machine, employee, student etc.

An entity set or entity type is a group of similar objects of concern to an organization for which it maintains data.

Advantages of DBMS:

1. Reduction of Redundancies
2. Shared Data
3. Integrity
4. Security
5. Conflict Resolution
6. Data Independence

Disadvantages of DBMS:

1. Problems associated with centralization
2. Cost of software/hardware and migration
3. Complexity of backup and recovery

Components of DBMS

1) **Classification of DBMS Users**
   1. **Naive Users**
      - Users who need not be aware of the presence of the database system or any other system supporting their usage are considered naïve users.
      - A user of an automatic teller machine falls in this category.
      - The user is instructed through each step of a transaction; he or she responds by a coded key or entering a numeric value.
      - The operations that can be performed by this class of users are very limited and affect a precise portion of the database; in the case of the users of the automatic teller machine, only one or more of her or his own accounts.
      - Other are end users of the database who work through a menu-oriented application program where the type and range of response is always indicated to the users.
      - Thus, a very competent database designer could be allowed to use a particular database system only as a naïve user.
   2. **Online Users**
      - There are users who may communicate with the database directly via an online terminal or indirectly via a user interface and application program.
      - These users are aware of the presence of the database system and may have acquired a certain amount of expertise in the limited interaction they are permitted with the database through the intermediary of the application program.
      - Online users can also be naïve users requiring additional help, such as menus.

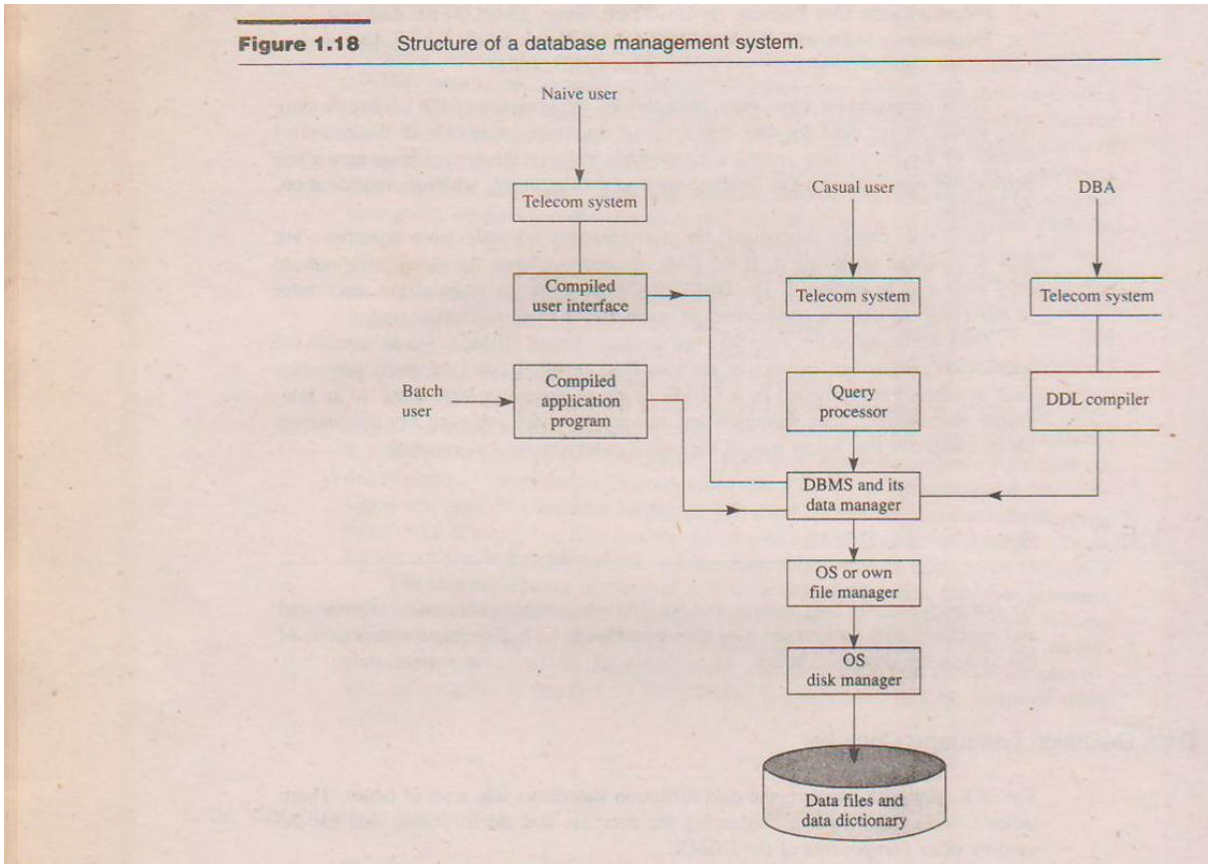
3. Application Programmers

- Professional programmers who are responsible for developing application programs or user interfaces utilized by the naïve and online users fall into this category.
- The application programs would be written in a general-purpose language such as Assembler, C, COBOL, FORTRAN, Pascal, or PL/I and include the commands required to manipulate the database.

4. Database Administrator

- Centralized control of the database is exerted by a person or group of persons under the supervision of a high-level administrator. This person or group is referred to as the **database administrator (DBA).**
- They are the users who are most familiar with the database and are responsible for creating, modifying, and maintaining its three levels.
- The DBA is responsible for granting permission to users of the database and stores the profile of each user in the database. The user profile can be used by the database system to verify that a particular user can perform a given operation on the database.
- The DBA is also responsible for defining procedures to recover the database from failures due to human, natural, or hardware causes with minimal loss of data.

2) Structure of DBMS

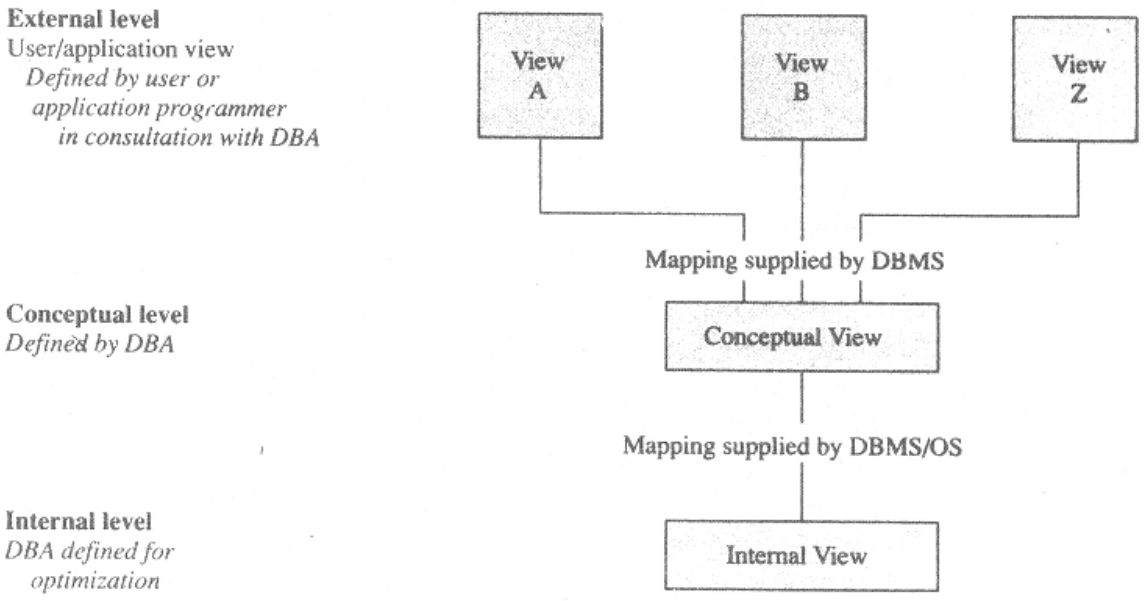


The three-schema architecture for a Database Management System(DBMS) (ANSI-SPARC Model)

A large number of commercial systems and research database models fit this framework. The architecture, shown in Figure A, is divided into three levels: the external level, the conceptual level, and the internal level.

The view at each of these levels is described by a **Schema**.
A **Schema** is an *outline* or a plan that describes the records and relationships existing in the view. The word scheme, which means a systematic plan for attaining some goal, is used interchangeably in the database literature with the word schema. The word schemas are used in the database literature for the plural instead of schemata, the grammatically correct word. The scheme also describes the way in which entities at one level of abstraction can be mapped to the next level.

Figure – A



1. Internal View

We find this view at the lowest level of abstraction, closest to me physical storage method used, It indicates how the data will be stored and describes the data structures and access methods to be used by the database. The internal view is expressed by the **internal schema,** which contains the definition of the stored record, the method of representing the data fields, and the access aids used.

2.Conceptual or Global View

At this level of database abstraction all the database entities and the relationship among them are included. One conceptual view represents the entire database. This conceptual view is defined by the conceptual schema. It describes all the records and relationships included in the conceptual view and, therefore, in the database. There is only one conceptual schema per database. This schema also contains method to deriving the objects in the conceptual view from the objects in the internal view.

The description of data at this level is format independent of its physical representation. It also includes features that specify the checks to retain data consistency and integrity.

3.External or User View

The external or user view is at the highest level of database abstraction where only those portions of the database of concern to a user or application program are included. Any number of user views (some of which may be identical) may exist for given global or conceptual view.

Each external view is described by means of a scheme called an external schema. The external schema consists of the definition of the logical records and the relationships in the external view. The external schema also contains the method of deriving the objects in the external view from the objects in the conceptual view. The objects include entities, attributes, and relationships

INTRODUCTION TO DATA MODELS

- A data model is a picture or description which depicts how data is to be arranged to serve a specific purpose.
- A data model is a mechanism that provides this abstraction for database applications. It can be also said that a data model is a collection of conceptual tools for describing data, data relationships, data semantics(meaning) and consistency constraints.
- Data modeling is used for representing entities of interest and their relationships in the database. It allows the conceptualization of the association between various entities and their attributes. A number of models for data representation have been developed. As with programming languages, there is

no one "best" choice for all applications. Most data representation models provide mechanisms to structure data for the entities being modelled and allow a set of operations to be defined on them.

- The models can also enforce a set of constraints to maintain the integrity of the data. These models differ in their method of representing the associations amongst entities and attributes.
- The data model depicts what that data items are required, and how that data must look.
- Some data models, which data records are connected or related within a file structure. These are called **record or structural data models**.
- Some data models are used to identify the subjects of corporate data processing - these are called **entity-relationship data models**.
- Still another type of data model is used for analytic purposes to help the analyst to solidify the semantics associated with critical corporate or business concepts.
- A model is an abstraction process that hides redundant details while highlighting details relevant to the applications at hand. The main models that we will study are the hierarchical, network, and relational models.
- The logical design of a database shows an abstract model of how the data should be structured and arranged to meet information needs. It involves identifying relationships among different data items and grouping them in orderly fashion. They are developed in accordance to particular conceptual data model.

➢ DATA MODELS can be classified….

1. Hierarchical
2. Network
3. Relational

1) Hierarchical

➢ The hierarchical data model (HDM) organizes the data in a tree structure.
➢ The nodes of the tree are the record types representing the entity sets and are connected by pointers or links.
➢ A pointer or link as in the network data model represents a relationship between exactly two records. However, in the hierarchical model this relationship, as in the genealogical (family) tree, is that of a parent and child.

- To create links between these record types, the hierarchical model uses Parent Child Relationships.
- There is a hierarchy of parent and child data segments.
- This structure implies that a record can have repeating information, generally in the child data segments.
- Data in a series of records, which have a set of field values attached to it. It collects all the instances of a specific record together as a record type.
- These record types are the equivalent of tables in the relational model, and with the individual records being the equivalent of rows.
- A parent record type can have any number of children record types. Two record types in a hierarchical tree can have at most one relationship between them and this relationship is that of one-to-one or one-to-many.
- i.e. it represent a one-to-many relationship between two entities where the two are respectively parent and child
- For example, an organization might store information about an employee, such as name, employee number, department, salary.
- The organization might also store information about an employee's children, such as name and date of birth. The employee and children data forms a hierarchy, where the employee data represents the parent segment and the children data represents the child segment.
- If an employee has three children, then there would be three child segments associated with one employee segment.
- In a hierarchical database the parent-child relationship is one to many. This restricts a child segment to having only one parent segment.
- Hierarchical DBMSs were popular from the late 1960s, with the introduction of IBM's Information Management System (IMS) DBMS, through the 1970s.

The hierarchical data model has the following constraints:

- Each hierarchical tree can have only one root record type and this record type does not have a parent record type.
- The root can have any number of child record types, each of which can itself be a root of a hierarchical (sub-) tree.
- Each child record type can have one parent record type; thus a many-to many relationship cannot be directly expressed between two record types.
- Data in a parent record applies to all its children records.
- Each occurrence of a record type can have any number of occurrences of each of its child record types.
- A child record occurrence must have a parent record occurrence; deleting a

parent record occurrence requires deleting all its children record occurrences.

- A hierarchical tree can have any number of record occurrences for each record type at each level of the hierarchical tree.
- In the implementation of the hierarchical data model the pointers are normally from a parent record to a child record only.
- The hierarchical database can be represented using a structure similar to the data structure diagram used in the network data model.
- The records are represented by rectangular boxes and the relationships between records are represented by arcs pointing from a root toward the leaf.
- The arcs are not labeled, since the relationship is always that of a parent and a child.
- Such structure diagrams are called tree structure diagrams, definition trees, or hierarchical definition trees.

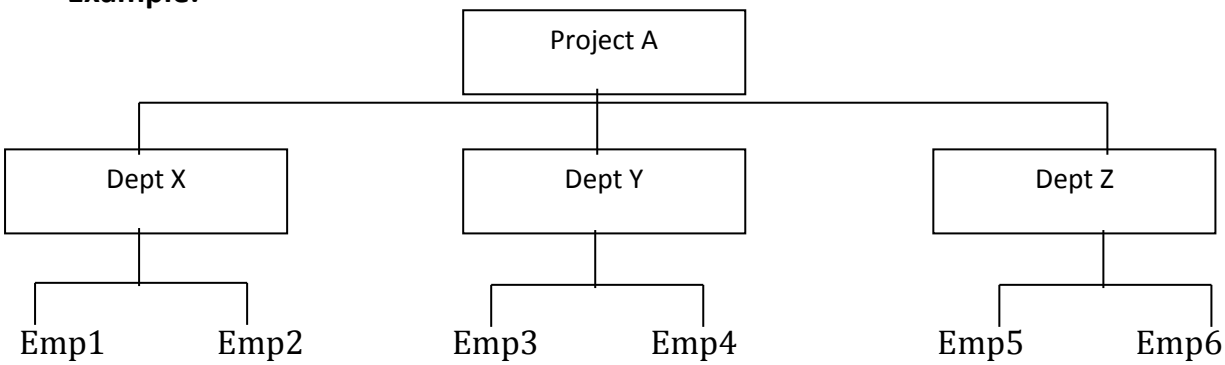
Advantages:

- Simplicity – The design is simple and more logical
- Data Integrity – The data can always referred to the parent data for data integrity.

Disadvantages:

- Navigation through the hierarchical database is a problem. To locate any data, one has to start from the root and navigate down from parent to child and reach the desired record.
- If the relationships are one-many, then manipulations are required.

Example:



The first record (Project A in the example) is known as root. It may have one or more child records. These records may have their own child records. Hence the relationship is one-many. That is no child can have more than one parent.

2) Network Model

- The popularity of the network data model coincided with the popularity of the hierarchical data model.

- The network data model (NDM) represents data for an entity set by a logical record type.
- The data for an instance of the entity set is represented by a record occurrence of the record type.
- Some data were more naturally modeled with more than one parent per child. So, the network model permitted the modeling of many-to-many relationships in data.
- In 1971, the Database Task Group of the Conference on Data Systems Languages (DBTG / CODASYL) formally defined the network model. The basic data modeling construct in the network model is the set construct.
- A set consists of an owner record type, a set name, and a member record type.
- A member record type can have that role in more than one set, hence the multi parent concept is supported.
- An owner record type can also be a member or owner in another set.
- The data model is a simple network, and link and intersection record types (called junction records by IDMS) may exist, as well as sets between them.
- Thus, the complete network of relationships is represented by several pair wise sets; in each set some (one) record type is owner (at the tail of the network arrow) and one or more record types are members (at the head of the relationship arrow).
- Usually, a set defines a 1:M relationship, although 1:1 is permitted. The CODASYL network model is based on mathematical set theory.

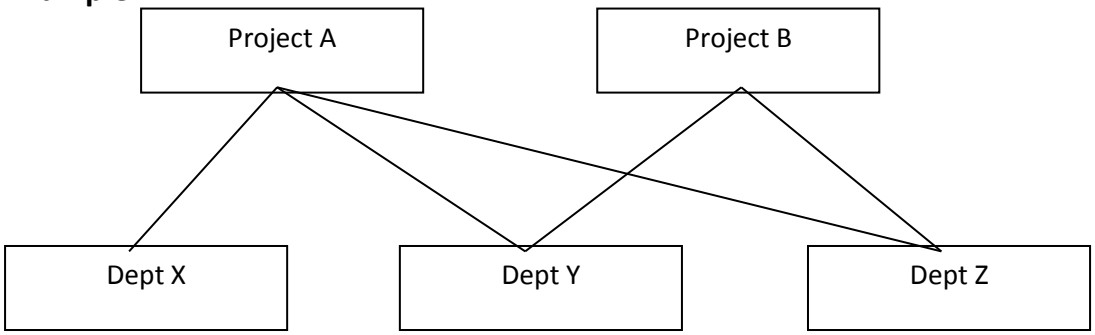
Advantages:

- The network database model is a great improvement over Hierarchical model. It is also conceptually simple and easy to operate.
- It has the capability to operate one to one and many to many relationship.
- The data access is easier.
- The network model is very useful for high volume operations encountered in powerful mainframes.
- Generally, changes in data characteristics do not require changes in application programs, that is a kind of data independence built in.

Disadvantages:

- The model has inherent complexities. Hence the user must be quite acquainted with the model to accomplish a query or an update.
- The data access method is navigational.

Example:



In the network model, two departments Y and Z come under Project A and Project B. That is, the elements of this model represent many to many relationship, which is difficult to handle in hierarchical model.

3) The Relational Data Model:

Concept:

A relational database matches data by using common characteristics found within the data set.
The resulting groups of data are organized and are much easier for many people to understand.
One of the main reasons for introducing this model was to increase the productivity of the application programmer by eliminating the need to change application programs when a change is made to the database. Users need not know the exact physical structures to use the database and are protected from any changes made to these structures. They are, however, still required to know how the data has been partitioned into the various relations.

For example, a data set containing all the real estate transactions in a town can be grouped by the year each transaction occurred, the sale price, a buyer's last name and so on.
Such a grouping uses the relational model (a technical term for this is schema).
Hence such a database is called a relational database.
The software used to do this grouping is called a relational database management system (RDBMS). The term "relational database" often refers to this type of software.

Terminology:

The term relational database was originally defined by and is attributed to Edgar Codd at IBM Almaden Research Center in 1970.
Relational Database theory uses a set of mathematical terms, which are equivalent to SQL database terminology.

- **Relation** is defined as a set of tuples that have the same attribute.

- Only applies to logical structure of the database, not the physical structure.
- A relation is usually described as a table which is organized into rows and columns.

- **Attribute** refers to characteristics of entity.
- **Domain** is a pool of values from where one or more attributes can draw their actual values
- **Tuple** is a row of a relation.
- **Degree** is a number of attributes in a relation.
- **Cardinality** is a number of tuples in a relation.
- **Relational Database** is a collection of normalized relations.

Alternative Terminology (Relational Model)

Formal terms	*Alternative 1*	*Alternative 2*
Relation	Table	File
Tuple	Row	Record
Attribute	Column	Field

Properties of Relational Table:

- Values are atomic.
- Each row is unique.
- Column Values are of the same kind.
- The sequence of column is insignificant.
- The sequence of rows is insignificant.
- Each column has a unique name.
- Certain fields may be designated as keys, which mean that searches for specific value of that field will use indexing to speed them up.
- Where fields in two different tables take the values form the same set, a join operation can be performed to select related records in the two tables by matching values in both fields.

Advantages:

- The relational model is simple as the database designers are freed from the physical storage details. They can concentrate on logical part.
- It is easier to control, more flexible and more intuitive.
- The greatest benefit comes here from ad-hoc query capability. The SQL is very powerful.
- It is very useful to express DBMS independent database designs.

Disadvantages:

- Conceptual data models are easier to comprehend than relational data models, because they confirm to the way we naturally look at things.
- The update anomalies occur because of relationships between attributes.

Examples:

Project
Project Number
Description
Dept.Number

Department
Dept. Number
Dept Name
Manager

Examples of current RDBMS products

SQL Server, Oracle, MySQL, Access, DB2, Teradata, dBase, SyBase, FoxPro

Operations on data(DDL , DML)

- SQL is a language that provides an interface to a rational database system.
- SQL was developed by IBM in the 1970 for use in system.
- SQL is often pronounced SEQUEL
- In common use SQL also encompasses DML, DDL, used for creating and modifying tables and other database structures.

Data Definition Language (DDL)

- Database management systems provide a facility known as data definition language (DDL), which can be used to define the conceptual scheme and also give some details about how to implement this scheme in the physical devices used to store the data.
- This definition includes all the entity sets and their associated attributes as well as the relationships among the entity sets. The definition also includes any constraints that have to be maintained, including the constraints on the value that can be assigned to a given attribute and the constraints on the values assigned to different attributes in the same or different records.
- These definitions, which can be described as metadata about the data in the database, are expressed in the DDL of the DBMS and maintained in a compiled form (usually as a set of tables).
- The compiled form of the definitions is known as a data dictionary, directory, or system catalog.
- The data dictionary contains information on the data stored in the database and is

consulted by the DBMS before any data manipulation operation.

- The database management system maintains the information on the file structure, the method used to efficiently access the relevant data (i.e., the access method).
- It also provides a method whereby the application programs indicate their data requirements. The application program could use a subset of the conceptual data definition language or a separate language.
- The database system also contains mapping functions that allow it to interpret the stored data for the application program. (Thus the stored data is transformed into a form compatible with the application program.)
- The internal schema is specified in a somewhat similar data definition language called data storage definition language. The definition of the internal view is compiled and maintained by the DBMS.
- The compiled internal schema specifies the implementation details of the internal database, including the access methods employed. This information is handled by the DBMS; the user need not be aware of details.
- DDL is set of SQL commands used to create, modify and delete database structures but not the data.
- These commands are normally used by a general user, who should be accessing the database via an application.
- DDL statements are used to define the database structure or schema.

Some examples:

- **CREATE** - to create objects in the database
- **ALTER** - alters the structure of the database
- **DROP** - delete objects from the database
- **TRUNCATE** - remove all records from a table, including all spaces allocated for the records are removed
- **COMMENT** - add comments to the data dictionary
- **RENAME** - rename an object

Data Manipulation Language (DML)

- The language used to manipulate data in the database is called data manipulation language (DML).
- Data manipulation involves retrieval of data from the database, insertion of new data into the database, and deletion or modification of existing data.
- The first of these data manipulation operations is called a query. A query is a statement in the DML that requests the retrieval of data from the database. The subset of the DML used to pose a query is known as a query language; however, we use the terms DML and query language synonymously.

- The DML provides commands to select and retrieve data from the database.
- Commands are also provided to insert, update, and delete records. They could be used in an interactive mode or embedded in conventional programming languages such as Assembler, COBOL, FORTRAN, and Pascal.
- The data manipulation functions provided by the DBMS can be invoked in application programs directly by procedure calls or by preprocessor statements.
- The latter would be replaced by appropriate procedure calls by either a preprocessor or the compiler.
- An example of a procedure call and a preprocessor statement is given below:
- Procedure call: Call Retrieve (EMPLOYEE. Name, EMPLOYEE. Address) Preprocessor statement: %select EMPLOYEE. Name, EMPLOYEE. Address from EMPLOYEE;
- These preprocessor statements, indicated by the presence of the leading % symbol, would be replaced by data manipulation language statements in the compiled version of the application program.
- Commands in the conventional languages allow permissible operations on the database such as data retrieval, addition, modification, or deletion.
- The DML can be procedural; the user indicates not only what to retrieve but how to go about retrieving it. If the DML is nonprocedural, the user has to indicate only what is to be retrieved.
- The DBMS in this case tries to optimize the exact order of retrieving the various components to make up the required response.
- Data definition of the external view in most current DBMSs is done outside the application program or interactive session.
- Data manipulation is done by procedure calls to subroutines provided by a DBMS or via preprocessor statements. In an integrated environment, data definition and manipulation are achieved using a uniform set of constructs that forms part of the user's programming environment.
- DML is the area of SQL that allows changing data within the database.
- DML statements are used for managing data within schema objects.

Some examples:

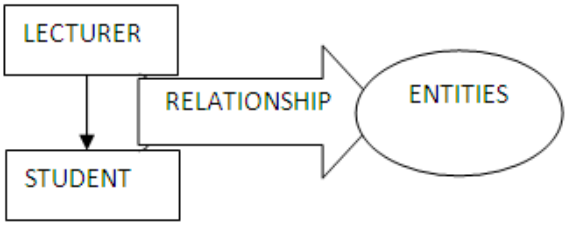
- **INSERT** - insert data into a table
- **UPDATE** - updates existing data within a table
- **DELETE** - deletes all records from a table, the space for the records remain
- **CALL** - call a PL/SQL or Java subprogram
- **EXPLAIN PLAN** - explain access path to data

- **LOCK** TABLE - control concurrency

Relationships and relationship types

- In addition to the associations that exist between the attributes of an entity, relationships exist among different entities.
- Relationships are used to model the interactions that exist among entities and the constraint that specifies the number of instances of one entity that is associated with the others.
- Even though a relationship may involve more than two entities, we concentrate on the relationship between two entities because it is the most common type encountered in database applications. Such a relationship is known as a binary relationship. It may be one-to-one (I : I), one-to-many (I: M), or many-to-many (M: N).
- The I: I relationship between entity sets E1, and E2 indicates that for each entity in either set there is at most one entity in the second set that is associated with it, The I:M relationship from entity set E1, to E2 indicates that for an occurrence of the entity from the set E1, there could be zero, one, or more entities. From the entity set E2 associated with it, Each entity in E2 is associated with at most one entity in the entity set E,. In the M: N relationship between entities sets E1, and E2, there is no restriction as to the number of entities in one set associated with an entity in the other set.
- A relationship can be defined as an association among ENTITIES.
- An association of several entities in a r-y model is called relationship.
- The relationship represents the fact that the LECTURER teaches several STUDENTS and the STUDENT, And STUDENT is taught by several LECTURERS.

ENTITIES AND RELATIONSHIP



Three Types of Relationship exists among entities….

These Are….

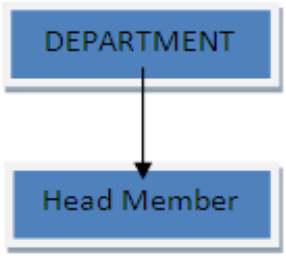
1. **One-to-One (1:1)**
2. **One-to-Many (1:M)**
3. **Many-to-Many (M:M)**

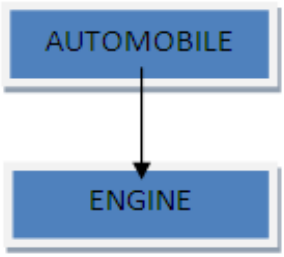
1. **One-to-One (1:1)**

- A One-to-One relationship is an association between two entities.

  For Ex…

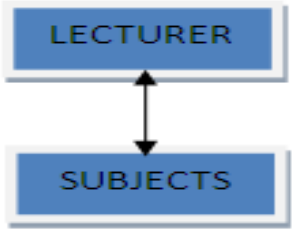
- In University each Department have one head of department.
- More Over, One member cannot be a head of more than one department.

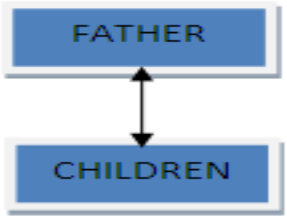




2. **One-to-Many**

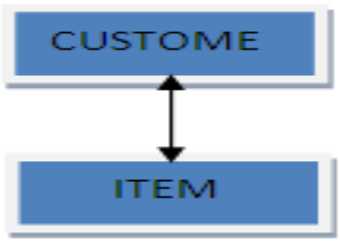
- A One-to-Many relationship exists when one entity is related to more than one entity.
- For Ex…A father may have many children but, a child has only one father.
- More than one subject is taught by one lecturer only.

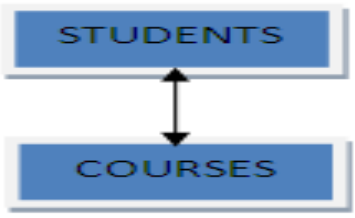




3. **Many-to-Many Relationship**

- A many-to-many relationship describes entities that may have many relations in both the directions.
- For Ex…
- One customer may buy many items and one item may be bought by many customers.
- Another Ex. A student can take many courses in university, and many students can register for given course.


CUSTOME
ITEM


STUDENTS
COURSES

Topics:

Structured Query Language - 1

- **SQL**
  - **Introduction**
  - **Advantages**
  - **Disadvantages**
- **Data types**
- **Types of SQL Commands**
  - **DDL**
  - **DML**
  - **DCL**
  - **TCL**
- **Working with SQL*Plus**
  - **Overview and basic commands of SQL plus**
- **Tables**
  - **Creation**
  - **Removal**
  - **Alternation**
- **Null Values**
- **Dual Table**
- **Tab Table**
- **Table data**
  - **Insertion**
  - **Updation**
  - **Deletion**
  - **Selection**

SQL

SQL is the language used to communicate with the server to access, manipulate and control data.

Features of SQL

- Can be used by a range of users, including those with little or no programming experience.
- Is a non-procedural language**.**
- Reduces the amount of time required for creating and manipulating systems.
- It is an English like language.
- SQL commands may be entered on one or many lines.
- Clauses are usually placed on separate lines for readability and ease of editing.
- Tabs and indents can be used to make code more readable.
- Command words cannot be split across line or abbreviated.
- SQL statements are not case sensitive.
- An SQL command entered at the SQL prompt, and subsequent lines are numbered. This is called the SQL buffer.
- Statement can be executed in a number of ways.
    - **a.** Place a semicolon (;) at the end of last clause
    - **b.** Place a semicolon or slash on the last line in the buffer.
    - **c.** Place a slash at the SQL prompt.
    - **d.** Issue a SQL*Plus START command at the SQL prompt.

Advantages of SQL

Standard independent language:

The widespread rules of the SQL have been recognized by ANSI and ISO. Therefore the SQL is an open language, involving it is not owned or controlled by a single company. Nowadays, SQL is offered by all the foremost DBMS vendors. SQL is not just for an exacting product; it also works with oracle, Microsoft SQL server and Sybase, and lot more. Almost all the data base vendors may have proprietary extensions to SQL, but the fundamentals of SQL are almost equal across all the data base vendors.

Cross-platform abilities:

The prevalent adversity of using a programming language to access database is that it infrequently produces an accurate cross platform application. SQL may be moderately innovative to some of the for the most part of popular programming languages like COBOL or C. But it has been in use on dissimilar hardware platforms for years. Majority of the cases, the same SQL statement can be used on a mainframe, desktop and a server.

Easy to learn and use:

SQL statements look like very simple English sentences that are easy to understand. SQL has been created such that it is instinctive, easy, and maps human's cognitive model. While it is non-procedural in nature, the user host has to type SQL declaration

and afterwards it has to be handed over to the DBMS. Then it is executed by the DBMS, internal code and returns a set, which is a logically defined group of data.

Speed:

For the past ten years, SQL data engines have been focusing the powerful effort to get better performance. The passionate competition among data base vendors has resulted in quicker, more vigorous database management systems that work at lower costs.

Disdvantages of SQL

- SQL cannot be used for programming because SQL does not provide the programming techniques of condition checking, looping and branching etc.
- SQL statements are passed to oracle engine one at a time which increases the traffic on the network which results in decrease of speed of data processing.

DATA TYPE

- **Varchar/Varchar2(size):** variable length alphanumeric data
  - Maximum length upto 4000 characters
  - Values of VARCHAR is compared using non-padded comparison semantics i.e. the inserted values will not be padded with spaces.
- **CHAR (size):** Fixed length character strings.
  - The maximum number of characters (i.e. the size) this data type can hold is 255 characters.
  - The data held is right-padded with spaces to whatever length specified.
  - For example: in case of Name CHAR(60), if the data held in the variable Name is only 20 characters in length, then entry will be padded with 40 characters worth of spaces.
- **NUMBER (P, S):** used to store numbers (fixed or floating point).
  - Numbers of virtually of any magnitude may be stored up to 38 digits of precision.
  - Numbers may be expressed in two ways: first, with the numbers 0 to 9, the signs + and -, and a decimal point(.); second, in scientific notation such as, 1.85E3 for 1850.
  - The precision (P), determines the maximum length of the data, whereas the scale (S), determines the number of places to the right of the decimal. If scale is omitted then the default is zero. If precision is omitted, values are stored with their original precision up to the maximum of 38 digits.
- **DATE:** represents Date and time values.
  - The standard format is DD-MON-YY.
  - To enter dates other than the standard format, use the appropriate functions.
  - It stores date in the 24-hour format.
  - By default the time in a date field is 12:00:00 am, if no time portion is specified.
  - The default date for a date field is the first day of the current month.
- **LONG:** Variable length character data up to 2 gigabytes.

  - Used to store arrays of binary data in ASCII format.
  - Only one LONG value can be defined per table.
  - LONG values cannot be used in subqueries, functions, expressions, where clauses or indexes and the normal character functions such as SUBSTR cannot be applied to LONG values.
  - A table containing LONG value cannot be clustered.
- **RAW** and **LONG RAW:** used to store binary data such as digitized picture or image.
  - RAW data type can have a maximum length of 255 bytes.
  - LONG RAW data type can contain up to 2 GB.
  - Values stored in columns having LONG RAW data type cannot be indexed.

NULL VALUE CONCEPT

Often there may be records in a table that do not have values for every field. This could be because the information is not available at the time of data entry or because the field is not applicable in every case. If the column was created as NULLABLE, Oracle will place a NULL value in the column in the absence of a user-defined value.

NULL value is **different from a blank or zero**. NULL value can be inserted into columns of **any data type.**

Principles of NULL Values

- Setting a NULL value is appropriate when the actual value is unknown, or when a value would not be meaningful.
- A NULL value is **not equivalent** to a **zero** if the data type is **number** and is not equivalent to **spaces** if the data type is **character.**
- A NULL value will evaluate to NULL in any expression (**e.g.** NULL multiplied by 10 is NULL)
- NULL value can be inserted into columns of **any data type.**
- If the column has a NULL value, Oracle ignores any UNIQUE, FOREIGN KEY, CHECK constraints that may be attached to the column.

TAB Table

To list existing created objects in oracle.

SQL> Select * from Tab;

Above command is used to determine the tables to which a user has access. The tables created under the currently selected tablespace are displayed.

Displaying the Table Structure

Syntax: Describe <TableName>;

This command displays the column names, the data types and the special attributes connected to the table.

DUAL TABLE IN ORACLE

Dual is a table owned by SYS. SYS owns the data dictionary, and DUAL is part of the data disctionary. DUAL is a small oracle worktable, which consists of only one column and one row, and contains the value x in that column. Besides arithmetic calculations, it also supports **date** retrieval and it's formatting.

As dual contains exactly one row (unless someone fiddled with it), it is guaranteed to return exactly one row in select statements. Therefore, dual is the preferred table to select a pseudo column (such as sysdate) Example: select sysdate from dual;

Structure of DUAL

DESC dual;
Output:

NAME	NULL?	TYPE
Dummy		VARCHAR2(1)

Select * from dual;
Output:

D
X

Example: select 2*2 from dual;
Output:

2*2
4

WORKING WITH SQL*PLUS

Basic SQL*PLUS Commands

Ed **:** Load the SQL * PLUS buffer into an editor. By Default, saves the file to AFIEDT.BUF (ED <filename> / Path)

Start **:** Run an SQL Script (START <filename.sql> / Path)

Get **:** Retrieve the previously stored command file (Get <Filename>)

Save **:** Save the contents of the SQL*PLUS buffer in a command file (Save <filename>)

Exit **:** Commit, logoff and exit

Connect : connect to a database as a specified user. (Connect username/password)

Set linesize : display or change SQL * PLUS line size setting.

Set pagesize : display or change SQL * PLUS page size setting.

Host : Execute host operating system command

The CREATE TABLE statement

Create tables to store data by executing the SQL CREATE TABLE statement. This statement is one of the data definition language (DDL) statements. DDL statements are subset of SQL statements used to create, modify or remove database structure. These statements have immediate effects on the database, and they also record information in the data dictionary.

To create a table, a user must have the CREATE TABLE privilege and a storage area in which to create object.

SYNTAX	EXAMPLE
CREATE TABLE <table name> (column1 data type <(size)> default <value>, . . . default <value>, . . . default <value>, Column N <datatype> <(size)> default <value>);	CREATE TABLE emp (Eno number(5), Ename varchar2(15), Design varchar2(15) default clerk, Salary number(7,2));

Default Statement: - value followed by DEFAULT key word will be automatically inserted in the table if no value will be inserted at the time of insertion record in the table.

The Alter Table Statement:

Use the Alter Table Statement to:

1. Add a new column.
2. Modify an existing column.
3. Define a default value for the new column.
4. Drop a column.
5. Rename existing column

You can add, modify, drop and rename columns to a table by using the Alter Table statement.

❖ **Adding a column:**

SYNTAX	EXAMPLE
ALTER table <Table Name> ADD (<Column name> <datatype> <(size)>);	ALTER table Emp ADD (Comm number(6,2));

Guideline:

1. You can add or modify columns.
2. You cannot specify where the column is to appear. The new column becomes the last column.

- **Note:** If a table already contains rows when a column is added, then the new column is initially null for all the rows.

❖ **Modifying a column:**

Column modification can include changes to a column data type; size and default value.

SYNTAX	EXAMPLE
ALTER table <Table Name> MODIFY column (<Column name> <new datatype> <new (size)>);	ALTER table Emp MODIFY column (Ename varchar2(25));

Guideline:

1. You can increase the width or precision of numeric column and also width of character column.
2. You can decrease the width of a column only if the column contains only null-values or if the table has no rows.
3. You can change the data type only if the column contains null values.
4. You can convert a CHAR column to VARCHAR2 data type or convert a VARCHAR2 column to the CHAR data type only if the column contains null values or if you do not change the size.

5. A change to the default value of a column affects only subsequent insertion to the table.

❖ **Dropping a column:**

You can drop a column from a table by using Alter Table.

SYNTAX	EXAMPLE
ALTER table <Table Name> DROP column <Column name> ;	ALTER table Emp DROP column comm;

Guideline:

1. The column may or may not contain data.
2. Using a Alter Table statement, only one column can be dropped at a time.
3. The table must have at least one column remaining in it after it is altered.
4. Once a column is dropped, it cannot be recovered.

❖ **Renaming a column:**

You can rename a column in a table by using Alter Table.

SYNTAX	EXAMPLE
ALTER table <Table Name> RENAME column <old Column name> to <new Column name> ;	ALTER table Emp RENAME column comm to commission;

Guideline:

1. The column may or may not contain data.
2. Using a Alter Table statement, only one column can be renamed at a time.

DROPPING a Table:

The drop table statement removes the definition of an oracle table. When you drop a table, the database loses all the data in the table and all the indexes associated with it.

SYNTAX	EXAMPLE
DROP TABLE <Table name>;	DROP TABLE emp;

Guideline:

1. All data is deleted from the table.
2. Any views and synonyms are remain but are invalid.
3. Any pending Transaction are committed.
4. Only the creator of the table as user with the DROP ANY TABLE privilege can remove a table.

Note: The DROP TABLE statement once executed is irreversible.

MANIPULATING DATA

- **Data Manipulation Language:**
  DML is a core part of SQL. When you ADD, UPDATE or DELETE data in the database, you execute a DML statement. A collection of DML statements that form a logical unit of work is called a transaction.

INSERT COMMAND

- **Adding a new Row to a Table:**

You can add new rows to a table by issuing the INSERT statement.

SYNTAX	EXAMPLE
INSERT into <table name> VALUES (<exp1>,<exp2>,……<expN>);	INSERT into Emp Values (1,'XYZ', 'CLERK', 10000);
INSERT into <tablename> (columnname1, columnname2) VALUES (<expression1>,<expression2>);	INSERT into Emp (Eno, Ename,Salary) Values (1, 'PQR', 25000);

NOTE: This statement with the values clause adds only one row at a time to a table.

1. Because you can insert a new row that contains values for each column, the column list is not required in the INSERT clause. However, if you do not use the column list, the values must be listed according to the default order of the columns in the table, and a value must be provided for each column.
2. Enclose character and date values within single quotation marks it is not recommended to enclose numeric values within single quotation marks.
3. Number values should not be enclosed in single quotes, because implicit conversion may take place for numeric values assigned to NUMBER data type columns if single quotes are included.

- **Methods for Inserting Null Values.**

1. **Implicit:** Omit the column from the column list.
2. **Explicit:** Specify the NULL keyword in the values list, specify the empty string (‘ ‘) in the VALUES list for character strings and dates.

The Oracle server automatically enforces all data types, data ranges and data integrity constraints. Any column that is not listed explicitly obtains a null value in the new row.

❖ **Common errors that can occur during user input:**

- Mandatory value missing for a NOTNULL column.
- Duplicate value violates uniqueness constraint.
- Foreign key constraint violated.
- CHECK constraint violated.
- Data type mismatch.
- Value too wide to fit in column.

You can use SQL functions to enter special values in your table.

You can also use the USER function when inserting rows in the table. The user functions records the current username.

❖ **Creating a Script:**

- Use & substitution in a SQL statement if prompt for values.
- & is a place holder for the variable value.
- Use ‘&’ Substitution for accepting a value for a column from the users.

You can save command with substitution variables to a file and execute the commands in the file.

SYNTAX	EXAMPLE
INSERT into <table name> VALUES (&columnname1,&columnname2, &columnname3,. . . . &columnnameN) /	INSERT into Emp Values(&eno,‘&Ename’,’&desig’,&salary) /
INSERT into <tablename> (col1, col2) VALUES (&col1value, &col2value) /	INSERT into Emp (Eno, Ename,Salary) Values (&eno, ‘&Ename’,&salary) /

UPDATE statement:

You can modify existing rows by using the UPDATE statement.

SYNTAX	EXAMPLE

UPDATE <Tablename> SET <columnname>= <expression>, <columnnname>=<expression> where <condition>;	UPDATE emp SET salary =20000 Where eno=2;

1. The UPDATE statement modifies specific rows if the WHERE clause is specified.
   If you omit the WHERE clause, all the rows in the table are modified.
2. You can update multiple columns in the SET clause of an UPDATE statement by writing multiple sub queries.

Note: If NO rows are updated, a message "0 rows updated" is returned

The DELETE statement:

You can remove existing rows by using the DELETE statement.

SYNTAX	EXAMPLE
DELETE FROM <Table Name> where <condition>;	DELETE from Emp where salary = 10000

NOTE: If no rows are deleted, a message "0 rows deleted" is returned.

- You can delete specific rows by specifying the WHERE clause in the DELETE statement.
- If you omit the WHERE clause, all rows in the table are deleted.
- You can use sub queries to delete rows from a table based on the values from another table.
- If you attempt to delete a record with a value that is tied to an integrity constraint.

Que: Explain use of Substitutional variable in Oracle.

Answer: Substitutional Variable (&)

When running a report, users often want to restrict the data returned dynamically. SQL*PLUS provide this flexibility by means of user variables. Use an ampersand (&) to identify each variable in your SQL statement. You do not need to define the value of each variable.

& (user variable): Indicates a variable in a SQL statement; if the variable does not exist, SQL*PLUS prompts the user for a value (SQL*PLUS) discards a new variable once it is used.

- With a single ampersand (&) we can accept a value from the user for **INSERT INTO <table name> VALUES** statement too.

- With a single ampersand, the user is prompted every time the command is executed, if the variable does not exist.
- When SQL*PLUS detects that the SQL statement contains an &, you are prompted to enter a value for the substitution variable named in the SQL statement. Once you enter a value and click the submit for execution button, the results are displayed in the output area of your SQL*PLUS session.
- **Specifying character and date values with substitution variables:**
  In a WHERE clause, date and character values must be enclosed within single quotation marks. The same rule applies to the substitution variables.

  You can also use functions such as UPPER and LOWER with the ampersand. Use UPPER ('&Job_title').

- **Specifying Column Names, Expressions and Text:**
  - Not only can you use the substitution variables in WHERE clause of SQL statement, but these variables can also be used to substitute for column names, expressions, or text.
  - If you do not enter a value for the substitution variable, you will get an error when you execute the preceding statement.

Topics:

Structured Query Language - 2

- **Pseudo columns (Definition and use)**
  - **ROWID**
  - **ROWNUM**
  - **UID**
  - **USER**
  - **SYSDATE**
- **Operators**
  - **Arithmetic, Relational, Logical, Range Searching, Pattern matching**
- **Data Constraints**
  - **Introduction, Column level, Table level**
  - **Primary key, Foreign key, Unique, Not null, Check**
- **Default value concept**
- **USER_CONSTRAINTS table**
- **Modifying Constraints using ALTER statement**
- **Functions**
  - **Introduction**
  - **Types (Scalar and Aggregate)**
- **Aggregate Functions (AVG, SUM, COUNT, MIN, MAX)**
- **Scalar Functions**
  - **Numeric Functions (ABS, MOD, POWER, ROUND, SQRT, TRUNC)**
  - **Character Functions (LOWER, UPPER, INITCAP, LENGTH, SUBSTR, TRIM, LPAD, RPAD)**
  - **Date Functions (ADD_MONTHS, LAST_DAY, NEXT_DAY, MONTHS_BETWEEN)**
  - **Conversion Functions (TO_NUMBER, TO_CHAR, TO_DATE)**
  - **Miscellaneous Functions (NVL, DECODE)**

Pseudo Columns

1) ROWID

For each row in the database, the ROWID pseudo column returns the address of the row. Oracle Database rowid values contain information necessary to locate a row:

2) The data object number of the object
3) The data block in the data file in which the row resides
4) The position of the row in the data block (first row is 0)
5) The datafile in which the row resides (first file is 1). The file number is relative to the tablespace.

Usually, a rowid value uniquely identifies a row in the database.

Rowid values have several important uses:

- They are the fastest way to access a single row.
- They can show you how the rows in a table are stored.
- They are unique identifiers for rows in a table.

Example: This statement selects the address of all rows that contain data for employees in department 20:

SELECT ROWID, lastname FROM employees WHERE deptno = 20;

6) ROWNUM

Rownum is a pseudo column. It numbers the records in a result set. The first record that meets the where criteria in a select statement is given rownum=1, and every subsequent record meeting that same criteria increases rownum.
After issuing a *select* statement, one of the last steps that oracle does is to assign an increasing (starting with 1, increased by 1) number to each row returned. The value of this row number can always be queried with rownum in a select statement:

Example: select **rownum**, empname,salary from emp;

It is important to realize that the first row's rownum is *always* 1.

7) UID

The **uid** function returns the integer value corresponding to the UserID of the user currently logged in.

Syntax: uid [into <variable>]

For Example: select uid from dual;

8) USER

This function returns the user name of the user who has logged in. The value returned is in varchar2 data type.

Syntax: User

Example: select user from dual;

9) SYSDATE

SYSDATE returns the current date and time set for the operating system on which the database resides. The datatype of the returned value is DATE, and the format returned depends on the value of the NLS_DATE_FORMAT initialization parameter. The function requires no arguments.
In distributed SQL statements, this function returns the date and time set for the operating system of your local database.
You cannot use this function in the condition of a CHECK constraint.

Example: SELECT sysdate FROM DUAL;

Operators

1) Arithmetic operators :
Arithmetic operators manipulate numeric operands. The - operator is also used in date arithmetic.

Table Arithmetic Operators

Operator	Description	Example
+ (unary)	Makes operand positive	SELECT +3 FROM DUAL;
- (unary)	Negates operand	SELECT -4 FROM DUAL;
/	Division (numbers and dates)	SELECT SAL / 10 FROM EMP;
*	Multiplication	SELECT SAL * 5 FROM EMP;
+	Addition (numbers and dates)	SELECT SAL + 200 FROM EMP;
-	Subtraction (numbers and dates)	SELECT SAL - 100 FROM EMP;

2) Logical operator :

Logical operators manipulate the results of conditions.

Table Logical Operators

Operator	Description	Example
NOT	Returns TRUE if the following condition is FALSE. Returns FALSE if it is TRUE. If it is UNKNOWN, it remains UNKNOWN.	SELECT * FROM EMP WHERE NOT (job IS NULL) SELECT * FROM EMP WHERE NOT (sal BETWEEN 1000 AND 2000)
AND	Returns TRUE if both component conditions are TRUE. Returns FALSE if either is FALSE; otherwise returns UNKNOWN.	SELECT * FROM EMP WHERE job='CLERK' AND deptno=10
OR	Returns TRUE if either component condition is TRUE. Returns FALSE if both are FALSE. Otherwise, returns UNKNOWN.	SELECT * FROM emp WHERE job='CLERK' OR deptno=10

3) Range Searching (Between….And)

The BETWEEN operator selects values within a range. The values can be numbers, text, or dates.
The range consists of a beginning, followed by an AND keyword and an end expression.

Syntax: SELECT <*columnName*>
FROM <*table_name*>
WHERE <*columnName*> **BETWEEN** *value1* **AND** *value2;*

Example: List Employee name from emp table where the department number lies in 15 and 25.

```
SELECT empno, ename, deptno FROM EMP WHERE
deptno BETWEEN 15 AND 25;
```

4) Pattern Matching (Like,In)

Use of Like Predicate:

The `LIKE` expression allows comparison of one `string value` with another `string value`. This is achieved by Wildcard character.
Two wildcards character are used:

- A percent sign (`%`) matches any string of any length.
- An underscore (_) matches on a single character.

Example: Select the records of that employee whose name starts with first character ‘B’ and third character ‘A’

```
SELECT * FROM emp where
ename like ‘B_A%’;
```

Use of IN and NOT IN Predicates:

The arithmetic oprator(=) compare a single value to another single value. In case a value needs to be compared to a list of values then IN predicate is used.

Example: Select the employee whose named aaa,bbb,ccc.

```
SELECT name FROM emp where
ename IN('aaa','bbb','ccc');
```

FUNCTIONS: (Scalar, Aggregate and Miscellaneous)

Scalar Function: (Numeric, Character, Date and Conversion function)

NUMERIC FUNCTIONS :

1) ABS :
The abs () function returns the absolute value of a numeric column.
Syntax : Abs()
SELECT Abs(column_name) FROM table_name;
For example : select abs(-10) from dual;

3) Mod :
This function is used to find mod value of a given value.
Syntax : mod(expression)
For example : select mod(5) from dual;

4) Power:
This function returns power value of a given expression.
Syntax : power(expression);
For example : select power(2) from dual;
Output : 4

5) Round:
Returns n, rounded to m places to the right of a decimal point.
Syntax : round(n,m)
For example : select round(15.19,1) from dual;
o/p : 15.2

6) Sqrt :
It returns square root of n.
Syntax : sqrt(n)
For example : select sqrt(25) from dual;
Output: 5

7) Trunc :
Returns a number truncated to a certain number of decimal places.
Syntax : trunk(number,decimal_places)
For example : Select trunk(125.815,1) from dual;

Output: 125.8

CHARACTER FUNCTIONS :

1) lower :
Returns character with all letters in lower case
Syntax : lower(char)
Example : select lower("HI") from dual;
o/p : hi

2) upper :
Returns character with all letters in upper case
Syntax : upper (char)
Example : select upper("hi") from dual;
o/p : HI

3) initcap :
Returns a string with first letter capital of every word.
Syntax : initcap(string)
Example : select initcap('hi hello')from dual;
o/p : Hi Hello

4) substr :
It returns a portion of character, beginning at character m and going upto character n.
Syntax : substr(string, start_pos, length)
Example : select substr('secure',3,4) from dual;
o/p : cure

5)Length :
Returns the length of a word.
Syntax : length(word)
Example: select length('hello') from dual;
o/p : 5

6) trim :
Remove all specified character from beginning and end of a string.
Syntax : trim(leading | trailing | both from string)
Example : select trim(' hello ') from dual;
o/p : hello

7)LPAD :

Returns **char1**, left-padded to length **n** with the sequence of characters specified in **char2**. If char2 is not specified Oracle uses blanks by default.

Syntax: Lpad(char1,n[,char2])

Example: select Lpad('Page 1',10,'*') "Lpad" from dual;

o/p:

Lpad

****Page 1

8) RPAD :

Returns **char1**, right-padded to length **n** with the sequence of characters specified in **char2**. If char2 is not specified Oracle uses blanks by default.

Syntax: Rpad(char1,n[,char2])

Example: select Rpad('Page 1',10,'*') "Rpad" from dual;

o/p:

Rpad

Page 1****

DATE FUNCTIONS :

1) add_months :
Returns date after adding the number of months specified in the function.
Syntax : add_months(d,n)
Example : select add_months(sysdate,4) from dual;
o/p : 01-nov-04

2) last_day :
It returns last day of the given month.
Syntax : last_day(d)
Example : select last_day(sysdate) from dual;
o/p : 31-jul-04

3) next_day :
It returns next day of given date.
Syntax : next_day(date,char)
Example :
Select next_day('06-july-02','saturday') from dual;
o/p : 13-july-02

4) months_between :
It returns number of months between d1 and d2.
Syntax : months_between(d1,d2).
Example : select months_between('02-feb-92','02-jan-92') from dual;
o/p : 1

CONVERSION FUNCTIONS :

1) to_char :
This function convert date value into character value .
Syntax : to_char (date_value, [format])
Example : select to_char(sysdate,'dd-mm-yy') from dual;
o/p : 01-07-04

2) to_date :
It convert character value into date value.
Syntax : to_date (char_value, [format])
Example : select to_date('06/07/02','dd/mm/yy') from dual;
o/p : 06-jul-02.

3) to_number
It convert character or numeric value into number value.
Syntax : to_number (char_value, [format])
Example : select to_number('0607') from dual;

AGGREGATE FUNCTIONS :

1) max :
It returns maximum value of a given expression.
Syntax : max(expression)
Example : select max(curbal) from dual;
o/p : 5000

2) min :
It returns minimum value of a given expression.
Syntax : min(expression)
Example : select min(curbal) from dual;
o/p : 1000

3) sum :
It returns sum value of a given expression.
Syntax : sum(expression)
Example : select sum(curbal) from dual;
o/p : 10000

4) count :
It count number of records from a given table .
Syntax : count(expression)
Example : select count(emp_ip) from dual;
o/p : 10

5) Avg :
It returns an average value of a given expression, ignoring null values in column.
Syntax: avg(expression)
Example: select avg(salary) from emp;

MISCELLANEOUS FUNCTIONS:

1) NVL

The NVL function can be used to return a value when a null occurs.

For example, the expression NVL (COMM, 0) returns 0 if COMM is null or the value of COMM if it is not null. Here COMM is the field from emp table and type of that is NUMBER so will be replaced by ZERO.

If user wants to convert the null value with any user define string value then first the field must been converted into the character type.

For example, NVL (TO_CHAR (COMM),'Null Value') returns a string 'Null Value' if the COMM is null or the value of COMM if it is not null

2) DECODE

SELECT ename, empno,
DECODE (deptno, 10,'ACCT', 20,'SALES','PURCHASE') FROM EMP;

In above query the employee name, number and department number fields will be displayed but with deptno the DECODE function is used.

In that the deptno is the field name and of the record is having the value 10 then 'ACCT' will be displayed, if 20 then 'SALES' will be displayed and if not 10 or 20 then 'PURCHASE' will be displayed.

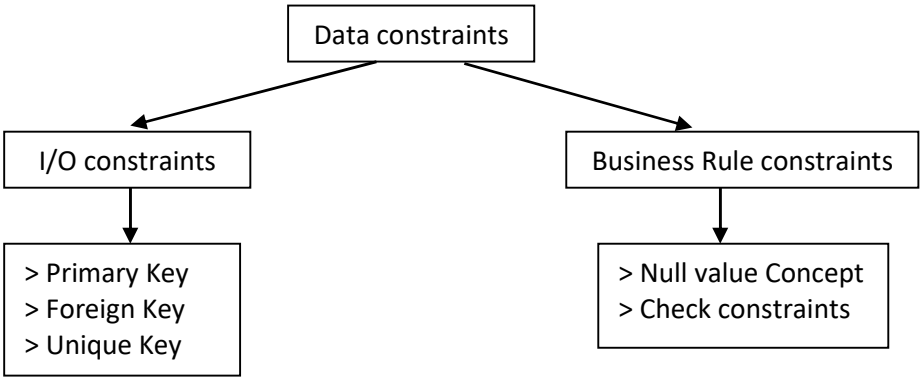
CONSTRAINTS

Constraints are used to limit the type of data that can go into a table.

Constraints can be specified when a table is created (with the CREATE TABLE statement) or after the table is created (with the ALTER TABLE statement).

We will focus on the following constraints:

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



1) PRIMARY KEY :

The PRIMARY KEY constraint uniquely identifies each record in a database table.A primary key column cannot contain NULL values.

A Single column Primary key is called a Simple key. A multicolumn primary key is called a Composite Primary key.

Features Of Primary Key:

- Primary key is a column or set of column which uniquely identifies a record in a database table.
- Primary keys must contain unique values means not contain duplicate value.
- Primary key column can not contain NULL values.
- Primary Key is not compulsory but necessary.
- Primary Key can not be Long or Double.
- Only one Primary Key is allow at column level per table
- Unique Index created automatically if there is a Primary key.
- One table can combine upto 16 columns in a Composite Primary Key.

Primary Key Constraint Defined At Column Level:

Syntax: <columnName> <Datatype>(<size>) PRIMARY KEY

Example: The following SQL creates a PRIMARY KEY on the "S_Id" column when the "Student" table is created:

```
CREATE TABLE Student
(
S_Id number(3) NOT NULL PRIMARY KEY,
Name varchar(20) NOT NULL,
Address varchar(25));
```

Primary Key Constraint Defined At Table Level:

Syntax: PRIMARY KEY (<columnName>,<columnName>)

Example: The following SQL creates a PRIMARY KEY on the "S_Id" column when the "Student" table is created:

```
CREATE TABLE Student
(
S_Id number(3) NOT NULL,
Name varchar(20) NOT NULL,
Address varchar(25),
PRIMARY KEY(S_Id));
```

Dropping Constraints:

Any constraint that you have defined can be dropped using the ALTER TABLE command with the DROP CONSTRAINT option.

For example, to drop the primary key constraint in the EMPLOYEES table, you can use the following command:

```
ALTER TABLE EMPLOYEES DROP CONSTRAINT EMPLOYEES_PK;
```
OR
```
ALTER TABLE EMPLOYEES DROP PRIMARY KEY;
```

2) FOREIGN KEY :

The FOREIGN KEY represents relationship between tables. A foreign key is a column (or a group of columns) whose values are derived from the primary key or unique key of same or some other table.

Foreign key defined table is called Foreign table or Detail table.
Primary key or Unique key defined table is called Primary table or Master table.
'References' keyword is used to define foreign key.

Features Of Foreign Key:

- A table can have more than one foreign key.
- Record cannot INSERT or UPDATE of a value, if a corresponding value does not exist in the primary key table Primary key column can not contain NULL values.

- Parent must reference a PRIMARY KEY or UNIQUE column(s) in primary key table.
- Foreign key can be specified on child but not on parent.
- Master table cannot be updated if child record exists.

Foreign Key Constraint Defined At Column Level:

Syntax: <columnName> <Datatype>(<size>)
REFERENCES <tableName>(<ColumnName>)

Example: The following SQL creates a FOREIGN KEY on the "S_Id" column when the "Student" table is already created:

```
CREATE TABLE Marks
(
S_Id number(3) REFERENCES student(s_no),
Marks1 number(3) NOT NULL,
Marks2 number(3) NOT NULL,
Marks3 number(3) NOT NULL);
```

Foreign Key Constraint Defined At Table Level:

Syntax: FOREIGN KEY (<columnName> [,<columnName>])
REFERENCES <tableName>(<ColumnName>,< ColumnName >)

Example: The following SQL creates a FOREIGN KEY on the "S_Id" column when the "Student" table is already created:

```
CREATE TABLE Marks
(
S_Id number(3),
Marks1 number(3) NOT NULL,
Marks2 number(3) NOT NULL,
Marks3 number(3) NOT NULL,
FOREIGN KEY (S_Id) REFERENCES student (s_no));
```

3) UNIQUE KEY :

The UNIQUE KEY column constraint permits multiple entries of NULL INTO THE COLUMN. UNIQUE and PRIMARY KEY constraints both provide a guarantee for uniqueness for a column or set of columns.
A PRIMARY KEY constraint automatically has a UNIQUE constraint defined on it.

Features Of Unique Key:

- Unique keys will not contain duplicate value.
- Unique Index created automatically.
- Unique key can combine upto 16 columns in a Composite Unique Key.
- Unique Key can not be Long or Long Raw data type.

Unique Key Constraint Defined At Column Level:

Syntax: <columnName> <Datatype>(<size>) UNIQUE

Example: The following SQL creates a PRIMARY KEY on the "S_Id" column when the "Student" table is created:

```
CREATE TABLE Student
(
S_Id number(3)  UNIQUE ,
Name varchar(20) NOT NULL,
Address varchar(25));
```

Unique Key Constraint Defined At Table Level:

Syntax: UNIQUE (<columnName1>,<columnName2>)

Example: The following SQL creates a PRIMARY KEY on the "S_Id" column when the "Student" table is created:

```
CREATE TABLE Student
(
S_Id number(3),
Name varchar(20) NOT NULL,
Address varchar(25),
UNIQUE (S_Id));
```

4) NOT NULL Cosntraint

NOTE: NOT NULL Constraint can only be defined At Column Level:

Syntax: <columnName> <Datatype>(<size>) NOT NULL

Example: Not null constraint defined at column name of student in "Student" table is created:

```
CREATE TABLE Student
(
S_Id number(3)  UNIQUE ,
Name varchar(20) NOT NULL,
Address varchar(25));
```

5) CHECK Constraint :

The check constraints must be specify the constraints as a logical expression that evaluate either TRUE or FALSE (due to a null).
The condition of a CHECK constraint can refer to any column in the table, but it cannot refer to columns of other tables.

Restrictions on CHECK Constraints:

- The condition must be a BOOLEAN expression that can be evaluated using the values in the row being inserted or updated.
- The condition cannot contain sub queries or sequences.
- The condition cannot include the SYSDATE, UID, USER or USERENV SQL functions.

Check Constraint Defined At Column Level:

Syntax: <columnName> <Datatype>(<size>) CHECK (<Logical Expression>)

Example: The following SQL check data inserted into column on "S_Id" must start with **S**.

```
CREATE TABLE Student
(
S_Id number(3)  CHECK(s_id LIKE 'S%'),
Name varchar(20) NOT NULL,
Address varchar(25));
```

Check Constraint Defined At Table Level:

Syntax: CHECK (<Logical Expression>)

Example: The following SQL check data inserted into column on "S_Id" must start with **S**.

```
CREATE TABLE Student
(
S_Id number(3),
Name varchar(20) NOT NULL,
Address varchar(25),
CHECK(s_id LIKE 'S%'));
```

DEFAULT VALUE CONCEPT:

At the time of column of table creation a 'default value' can be assigned to this column. When the user is loading a 'record' with values and leaves this cell empty, the Oracle engine will automatically load this cell with the default value specified. The data type of default value should be match with the type of the column.

Syntax: <columnName> <Datatype>(<size>) DEFAULT (<value>)

Example: The following SQL creates a column gender with default gender value Female:

```
CREATE TABLE Student
(
```

S_Id number(3) ,
Name varchar(20) NOT NULL,
Gender char(1) DEFAUL ‘F’);

USER_CONSTRAINTS TABLE:

Oracle stores constraint information in this table. This table comprises of multiple columns, some of which are described below:

COLUMN Name	Description
OWNER	The owner of the constraint
CONSTRAINT_NAME	The name of the constraint
TABLE_NAME	The name of the table associated with the constraint
CONSTRAINT_TYPE	The type of constraint: P: Primary Key R: Foreign Key U: Unique Constraint C: Check Constraint
SEARCH_CONDITION	The search condition used. (for CHECK constraint)
R_OWNER	The owner of the table referenced by the FOREIGN KEY constraint
R_CONSTRAINT_NAME	The name of the constraint referenced by a FOREIGN KEY constraint.

Exa: View the constraints of the table CLIENT_MST.

Select constraint_name,constraint_type,search_condition from user_constraints where table_name='CLIENT_MST’;

Note: Write down table name in above query as all letters must be in upper case.

Defining Integrity Constraints Via Alter Table Command

1) Primary Key
   Exa: Alter the table Emp_MST by adding a primary key on the column EMP_NO.

   Alter table emp_mst add primary key(emp_no);

2) Foreign Key

Exa: Alter the table Emp_mst by adding foreign key on deptno column owned by dept table.

```
Alter table emp_mst add constraint fk_dno foreign key(deptno) references dept(deptno);
```

3) Check

Exa: Alter the table Emp_mst by adding check constraint on salary column as salary must be greater than zero.

```
Alter table emp_mst add constraint chk_sal  check(sal>0);
```

4) Unique

Exa: Alter the table dept_mst by adding unique on deptname column

```
Alter table dept_mst add constraint uq_dname unique(deptname);
```

5) Not Null

Exa: Add not null constraint on empname in emp table.

```
Alter table emp modify(empname not null);
```

Topics:

Structured Query Language - 3

- **Grouping using Group by and Having**
- **Sub Query (using in, any, all)**
- **Joining tables, types of join**
  - **Inner Join, Outer Join, Cross Join, Self-Join – ANSI and THETA style**
- **Creation of database objects**
  - **Indexes**
    - **Uses, types - simple and composite (only definition, syntax and example)**
  - **Views**
  - **Sequences**
  - **Synonym**
- **Data Control Language statements**
  - **Grant**
  - **Revoke**
- **Transaction Control Language statements**
  - **Commit**
  - **Rollback**
  - **Savepoint**

GROUP BY Clause

The **GROUP BY** clause is another section of the **select** statement. This optional clause tells Oracle to group rows based on distinct values that exist for **specified columns.** The GROUP BY clause creates a data set, containing several sets of records **grouped together** based on a condition.

Syntax:-	Example:-
SELECT <ColumnName1>, <ColumnName2>, <ColumnNameN>, AGGREGATE_FUNCTION (<Expression>) FROM TableName WHERE <condition> GROUP BY <ColumnName1>, <ColumnName2>, <ColumnNameN>;	1) Select dno,sum(sal) from emp group by dno; 2) Select eno,dno,ename,sum(sal) from emp where dno=22 group by dno,eno,ename;

HAVING Clause:-

The **HAVING** clause can be used in conjunction with the **GROUP BY** clause, **HAVING** imposes a condition on the **GROUP BY** clause, which further filters the groups created by the **GROUP BY** clause. Each column specification specified in the HAVING clause must occur within a statistical function or must occur in the list of columns named in the **GROUP BY** clause.

Syntax:-	Example:-
SELECT <ColumnName1>, <ColumnName2>, <ColumnNameN>, AGGREGATE_FUNCTION (<Expression>) FROM TableName WHERE <condition> GROUP BY <ColumnName1>, <ColumnName2>, <ColumnNameN> HAVING AGGREGATE_FUNCTION (<Expression>) <operator> <value>;	select cust_no,count(fd_no) from emp group by cust_no having count(fd_no) > 2;

Rules for Group By and having Clause:-

- Columns listed in the select statement have to be listed in the GROUP BY clause
- Columns listed in the GROUP BY clause need not be listed in the SELECT statement.
- Only group functions can be used in the HAVING clause.
- The group functions listed in the having clause need not be listed in the SELECT statement.

Sub query

- A **subquery** is a form of an SQL statement that appears inside another SQL statement.
- It is also termed as **nested query**.
- The statement containing a subquery is called a **parent** statement.

- The parent statement uses the rows (i.e. the result set) returned by the subquery.

It can be used for the following:

- To insert records in a target table
- To create tables and insert records in the table created
- To update records in a target table
- To create views
- To provide values for conditions in WHERE, HAVING, IN and so on used with SELECT, UPDATE, and DELETE statements.

Syntax:-	Example:-
Select <ColumnName1>,<ColumnName2>,.. from <tablename> where <expr> <operator> (select <columname1>,<columnname2> from <tablanme> where <condition> group by <ColumnName> having <condition> Order by <columName>);	1. Select eno,ename,sal from emp where sal > (select sal from emp where name='Priya'); 2.Select eno,ename,sal,design from emp Where sal < any (select sal from emp where designation='manager');

- A subquery must be enclosed with simple brackets or peranthesis.
- Place the subquery on the right side of the comparison condition.
- One common error with subquery is more than one row returned with single row subquery. At that time use the IN, ANY or ALL operator.

Two Types of Subquery:-

1) Single Row Subquery (=, >, <, >=, <=, <>)
2) Multiple Row Subquery (IN, ANY, ALL)

ANY:- The ANY operator compares value to each value returned by the subquery. ANY is equivalent to IN.

< ANY :- Less than the maximum.

> ANY :- Greater than the minimum.

ALL:- The ALL operator compares a value to every value returned by a subquery.

>ALL :- Greater than the maximum.

<ALL :- Less than the minimum.

EXISTS: - The exists operator ensure that the search in the inner query terminates when at least one match is found. The exists operator is usually used with co-related queries.

NOT EXISTS: -The not exists operator enables to test whether s value retrieved by the outer query is not a part of the result set of the value retrieved by the inner query.

Syntax:- (Exists)	Syntax:- (Not Exists)
Select <columnanme1, <columnanme2>,...	Select <columnanme1,<columnanme2>,…….
From <tablename> where exists(select statement);	From <tablename> where not exists(select statement);

One of the value return by the inner query is NULL value and hence the entire query returns no rows. The reason is that all condition compare NULL value result in a NULL. So whenever NULL values are likely to be part of the result set of a subquery **do not use NOT IN operator.** So use the **<> operator.**

Creation And Manipulation of Database Objects- Indexes, Views, Sequences and Synonyms

Index

- When a select statement is fired to search for a particular record, the Oracle engine must first locate the table on the hard disk. The oracle engine reads system information and finds the start location of a table's record on the current storage media. The oracle engine then performs a sequential search to locate records that matches user-defined criteria as specified in the **select.**
- Indexing involves forming a two dimensional matrix completely independent of the table on which the index is being created. This two dimensional matrix will have a **single column**, which will hold sorted data, extracted from the table column(s) on which the index is created.
- Another column is called **the address field** identifies the location of the record in the oracle database.
- For every data value held in the index the oracle engine inserts a unique ROWID value. This is done for every data value inserted into the index, without exception.
- Users and application developers can also use ROWIDs for the following functions:
    1. Rowids are the fastest means of accessing particular rows.
    2. Rowids can be used to see how a table is organized.
    3. Rowids are unique identifiers for rows in a given table.

➢ The ROWID format used by oracle is as follows:

BBBBBBB.RRRR.FFFF

- Where, **FFFF** is a unique number given by the oracle engine to **each Data file**. Data files are the files used by the oracle engine to store user data.
- Each data file is further divided into Data Blocks and each block is given a unique number. Thus block number can be used to identify the data block in which a record is stored. **BBBBBBB** is the **Block number** in which the record is stored.
- Each data block can store one or more records. Thus each record in the data block is given a unique **Record number**. **RRRR** is a unique record number.

Duplicate / Unique Index

Oracle allows the creation of two types of indexes.

1. Index that allows duplicate values for the indexed columns i.e. **Duplicate Index**
2. Index that deny duplicate values for the indexed columns i.e. **Unique Index**

Creation of an Index:-

An index can be created on one more columns. Based on the number columns included in the index, an index can be:

- Simple index
- Composite index

Creating a simple index

An index created on a single column of a table is called a simple index.

Syntax:	Example:-
Create index <IndexName> ON <TableName> (<ColumnName>);	Create indx_emp on emp (eno);

Creating a Composite index

An index created on more than one column is called a composite index.

Syntax:	Example:-
Create index <IndexName> ON <TableName> (<ColumnName1>,<ColumnName2>);	Create indx_emp on emp (eno,dno);

Creating of unique index

A unique index can also be created on one or more columns. If an index is created on a single column it is called simple index.

Syntax:	Example:-
Create UNIQUE index <IndexName> ON <TableName> (<ColumnName>);	Create unique indx_emp on emp (eno);

Creating of composite unique index

If an index is created on more than one column it is called composite index.

Syntax:	Example:-
Create UNIQUE index <IndexName> ON <TableName> (<ColumnName1,<ColumnName2>)	Create unique indx_emp on emp (eno,dno);

Views

- After a table is created and populated with data, it may become necessary to prevent all users from accessing all columns of a table, for data security reasons. This would mean creating several tables having the appropriate number of columns and assigning specific users to each table.

- To reduce redundant data to the minimum possible, oracle allows the creation of an object called a **view.** A view is mapped to a select sentence. The table on which the view is based is described in the FROM clause of the select statement. The select clause consists of a sub-set of the columns of the table.
- Thus a view which mapped to a table will in infect have a sub-set of the actual columns of the table from which it is built.

Creating View:-

Syntax:	Example:
create view <view_name> as select <columname1>,<columnanme2> from <table_name> where <columnanme>=<expression list>	create view view_emp as select eno,ename,sal from emp **where eno=33;**

❖ The **ORDER BY** clause **cannot** be used while creation a view.

Read Only View:-

If a view is used to only look at table that view is called a **Read Only** view. User cannot perform Insert, Update and Delete Operations. Read Only view is created using **WITH READ ONLY option**.

Syntax:	Example:
create view <view_name> as select <columname1>,<columnanme2> from <table_name> WITH READ ONLY;	create view view_emp as select eno,ename,sal from emp WITH READ ONLY;

Updatable View:-

Views can also be used for data manipulation i.e user can perform Insert, Update, Delete operations. Views on which data manipulation can be done are called **Updatable Views**.
For a view to be updatable, it should meet the following criteria.

From a single Table	From Multiple Tables: (If Table is created using referencing clause i.e using a foreign key)
▪ If the user wants to **INSERT** records with the help of as view, then the **PRIMARY KEY** column(s) and **NOT NULL** column(s) must be **included** in the view. ▪ The user can **UPDATE, DELETE** records with the help of a view even if the **PRIAMRY KEY** column and **NOT NULL** column(s) are **excluded** from the view definition.	▪ The **INSERT** operation is not allowed ▪ The **DELETE** or **MODIFY** operations **do not** affect the **Master** table. ▪ The view can be used to **MODIFY** the columns of the **child table** included in the view. ▪ If a **DELETE** operation is executed on the view, the corresponding records from the **child table** will be deleted.

Example :- Create View using single Table	Example: Create View using Multiple Tables.
Create view view_emp as select eno,dno,ename,sal from emp;	Create view view_emp1 as select eno,e.dno,ename,sal from emp e,dept d where e.dno-d.dno;

Restrictions on UPDATABLE VIEW:-

For the view to be updatable the view definition must not include

- Aggregate Functions
- Disinct, Group By and Having Clause
- Sub-Queries
- Union, Intersect and Union clause
- Constant and value Expressions like sal*0.10

Sequence

- The quickest way to retrieve data from a table is to have a column in the table whose data uniquely identifies a row.
- A constraint is attached to a specific column in a table that ensure that the column is never left empty Not Null) and data values in the columns are unique (Primary Key). Since users do data entry, and entered the duplicate value, which violates the constraints and values is rejected, so achieve this sequence is used.
- If the value entered into this column is computer generated it will always fulfill the unique constraint and the row will always be accepted for storage.
- Oracle provides an object called Sequence that can generate a numeric value
- The values generated have a maximum of 38 digits.

The Sequence can be defined to:

- Generate numbers in ascending or descending order
- Provided intervals between numbers.
- Caching of sequence numbers in memory to speed up their availability.

Creating a sequences:-

- The starting number
- The maximum number that can be generated by a sequence.
- The increment value for generating the next number.

Syntax:	Example:
CREATE SEQUENCE <sequenceName> [INCREMENT BY <IntegerValue> START WITH <IntegerValue> MAXVALUE <IntegerValue>/ MAXVALUE MINVALUE <IntegerValue> / MINVALUE CYCLE / NOCYCLE	[Ascending Order] 1. create sequence seq1 increment by 1 maxvalue 20 minvalue 1 cache 5

CACHE <IntegerValue> / NOCHE ORDER / NOORDER]	[Descending Oprder] 2. create sequence seq2 increment by -1 start with 10 maxvalue 10 minvalue 1 cache 20 cycle;

Keywords and Parameters:-

INCREMENT BY: Specifies the interval between sequence numbers. It can be any positive, negative value but not a zero. Default value is 1.

MINVALUE: Specifies the sequence minimum value.

NOMINVALUE: Specifies a minimum value of 1 for an ascending sequence. And –10^26 for descending sequence.

MAXVALUE: Specifies the maximum value that a sequence can generate.

NOMAXVALUE: Specifies a maximum 10^27 for an ascending sequence or -1 for a descending sequence. This is the default clause.

START WITH: Specifies the first sequence number to be generated.

CYCLE: Specifies that the sequence continues to generate repeat values after reaching either its maximum value.

NOCYCLE: Specifies that a sequence cannot more values after reaching the maximum value.

CACHE: Specifies how many values of a sequence Oracle pre-allocates and keeps in memory for faster access.

NOCACHE: Specifies that the values of a sequence are mot pre-allocated.

ORDER: This guarantees that sequence numbers are generated in the order of request.. This is only necessary if using a parallel server in parallel mode option.

NOORDER: This does not guarantee sequence numbers are generated in order of request. This is only necessary if using a parallel server in parallel mode option. The default is NOORDER clause.

Referencing a Sequence: (Pseudocolumns of sequence):-Once a sequence is created then the view the values held in cache. This using the two columns NEXTVAL and CURRVAL columns.

NEXTVAL:

Syntax:-	Example:-
select <SequenceName>.NextVal form Dual;	Select seq1.nextval form dual; Insert into emp (eno,enm) values(seq1.nextval,'vishal');

This will Display the next value held in the cache on the screen. Every time nextval references a sequence its output is automatically incrememted from old value to new value.

CURRVAL:

Syntax:-	Example:-
select <SequenceName>.CurrVal form Dual;	Select seq1.currval form dual;

This gives the current value of a sequence.

Altering a Sequence: A sequence once created can be altered. This is achieved by a ALTER SEQUENCE statement.

Syntax:	Example:-
ALTER SEQUENCE <sequenceName> [INCREMENT BY <IntegerValue> START WITH <IntegerValue> MAXVALUE <IntegerValue>/ MAXVALUE MINVALUE <IntegerValue> / MINVALUE CYCLE / NOCYCLE CACHE <IntegerValue> / NOCHE ORDER / NOORDER]	Alter sequence seq1 increment by 1 maxvalue 50 minvalue 1 cache 5 cycle; i.e change the sequence seq1 with maxbalue 50 instead of 20.

Dropping a Sequence: To DROP SEQUENCE command is used to remove the sequence from the database.

Syntax:	Example:-
DROP SEQUENCE <sequenceName>;	Drop sequence seq2;

Synonym

A **synonym** is an alternative name for objects such as Table, view, sequence, stored procedure, stored function, package, java class and another synonym.

Synonym is an alias for one of the objects:

- The object does not need to exist at the time of its creation.
- Synonyms cannot be used in a drop table, drop view or truncate table statements. If this is tried, it results in a oracle that **table or view does not exist**
- Synonyms provide both data independence and location transparency.
- You can refer to synonyms in the following DML statements: SELECT, INSERT, UPDATE, DELETE and LOCK TABLE.

Create Synonym:

Syntax:	create [or replace] [public] synonym [schema .] synonym_name for [schema .] object_name ;	
Keywords	create	Use the CREATE SYNONYM statement to create a **synonym**, which is an alternative name of objects.
	or replace	The **or replace** phrase allows you to recreate the synonym (if it already exists) without having to issue a DROP synonym command.

	Public	The **public** phrase means that the synonym is a public synonym and is accessible to all users. Remember though that the user must first have the appropriate privileges to the object to use the synonym.
	schema	The **schema** phrase is the appropriate schema. If this phrase is omitted, Oracle assumes that you are referring to your own schema.
	object_name	The **object_name** phrase is the name of the object for which you are creating the synonym.
Example:-	create or replace synonym emp1 for emp; create or replace synonym sb1.dept for dept; create or replace public synonym emp11 for emp11; In first example we create a synonym emp1 for emp table without using schema. In second example we create a synonym dept for dept table using schema sb1. In third example we create a public synonym emp11 for emp11 table that means emp11 synonym accessible by all users.	
Note:-	If this synonym already existed and you wanted to redefine it, you could always use the **or replace** phrase .(example-3)	
How to Use of synonym	Using the select statement we can use the synonym. Select * from synonym_name; **Example:** select *from emp11;	

Dropping a synonym

It is also possible to drop a synonym.

Syntax:	drop [public] synonym [schema .] synonym_name [force];	
Keywords	Drop	Use the drop statement to drop a **synonym**.
	Public	The **public** phrase allows you to drop a public synonym. If you have specified **public**, then you don't specify a **schema**.
	Schema	The **schema** phrase is the appropriate schema. If this phrase is omitted, Oracle assumes that you are referring to your own schema.
	Force	The **force** phrase will force Oracle to drop the synonym even if it has dependencies. It is probably not a good idea to use the **force** phrase as it can cause invalidation of Oracle objects.
Example:-	drop synonym emp1; drop synonym sb1.dept; drop public synonym emp11;	

Join

Joining Multiple Tables (Equi Joins): -

Sometimes it is necessary to work with multiple tables as through they were a single entity. Then a single SQL sentence can manipulate data from all the tables. Joins are used to achieve this. Tables are joined on columns that have the same data type and data width in the tables.

Tables in a database can be related to each other with keys. A primary key is a column with a unique value for each row. The purpose is to bind data together, across tables, without repeating all of data in every table.

The JOIN operator specifies how to relate tables in the query.

Types of JOIN:

- INNER
- OUTER (LEFT, RIGHT, FULL)
- CROSS

INNER JOIN:

- Inner joins are also known as Equi Joins. There are the most common joins used in SQL*Plus.
- They are known as Equi joins because the where statement generally compares two columns from two tables with the equivalence operator =.
- This type of join is by far the most commonly used. In fact, many systems use this type as the default joins. This type of join can be used in situations where selecting only those rows that have values in common in the columns specified in the ON clause, is required.
- In short, the INNER JOIN returns all rows both tables where there is match.

Syntax:-	Example:-

ANSI-style:-
SELECT<ColumnNm1>, <ColumnNm2>, <ColumnNmN>
FROM <TableName1> INNER JOIN <TableName2>
ON
<TableName1>.
<ColumnNm1>=<TableName2>.<ColumnName2>
WHERE <Condition>
ORDER BY <ColumnName1>, <ColumnName2>;
 	
Theta style:-
SELECT <ColumnName1>, <ColumnName2>, <ColumnNameN>
FROM <TableName1>, <TableName2>
WHERE
<TableName1>.<ColumnName1>=<TableName2>.<ColumnName2>
AND <condition>
ORDER BY <ColumnName1>, <ColumnName2>;
	ANSI-style:-
	SELECT eno,e.dno,ename,dname
	from emp e inner join dept d
	on e.dno=d.dno;
	
	Theta style:-
	SELECT eno,e.dno,ename,dname
	from emp e,dept d
	where e.dno=d.dno;
	and eno=111;

In the above syntax :
ColumnName1 in TableName1 is usually that table's Primary Key.
ColumnName2 in TableName2 is a Foreign Key in that table.
ColumnName1 in ColumnName2 must have the same Data Type and for certain data types, the same size.

OUTER JOIN:

- Outer joins are similar to inner joins, but give a bit more flexibility when selecting data from related tables.
- This type of join can be used in situations where it is desired, to select all rows from the tables on the left (or right, or both) regardless of whether the other table has values in common and (usually) enter NULL where data is missing.

Three types of Outer Join :

- Left Outer join
- Right Outer Join
- Full Outer Join

Left Outer Join:-

This type of join can be used in situations where it is desired, to select all rows from the tables on the left where there is match.

Syntax:- (Left Outer Join)	Example:-

ANSI-style
SELECT<ColumnNm1>, <ColumnNmN>
FROM <TableName1> LEFT JOIN <TableName2>
ON <TableName1>.
<ColumnNm1>=<TableName2>.<ColumnName2>
>
WHERE <Condition>
ORDER BY <ColumnName1>, <ColumnName2>;
 	
Theta style:-
SELECT <ColumnName1>, <ColumnName2>,
<ColumnNameN>
FROM <TableName1>, <TableName2>
WHERE
<TableName1>.<ColumnName1>=<TableName2
>.<ColumnName2>(+)
AND <condition>
ORDER BY <ColumnName1>, <ColumnName2>;
	ANSI-style
	SELECT eno,e.dno,ename,dname
	from emp e left join dept d
	on e.dno=d.dno;
	
	Theta style:-
	SELECT eno,e.dno,ename,dname
	from emp e,dept d
	where e.dno=d.dno(+);
	
	(i.e) Display all the data of emp
	table , if dno of both table is match.

Right Outer Join:-

This type of join can be used in situations where it is desired, to select all rows from the tables on the Right where there is match.

Syntax:- (Left Outer Join)	Example:-
ANSI-style:- SELECT<ColumnNm1>, <ColumnNm2>, <ColumnNmN> FROM <TableName1> RIGHT JOIN <TableName2> ON <TableName1>. <ColumnNm1>=<TableName2>.<ColumnName2> WHERE <Condition> ORDER BY <ColumnName1>, <ColumnName2>;	ANSI-style:- SELECT eno,e.dno,ename,dname from emp e right join dept d on e.dno=d.dno;

Theta style:- SELECT <ColumnName1>, <ColumnName2>, <ColumnNameN> FROM <TableName1>, <TableName2> WHERE <TableName1>.<ColumnName1>(+)=<TableName2>.<ColumnName2> AND <condition> ORDER BY <ColumnName1>, <ColumnName2>;	Theta style:- SELECT eno,e.dno,ename,dname from emp e,dept d where e.dno(+)=d.dno; (i.e) Display all the data of dept table , if dno of both table is match.

Full Outer Join:-

Syntax:- (Full Outer Join)	Example:-
ANSI-style:- SELECT<ColumnNm1>, <ColumnNm2>, <ColumnNmN> FROM <TableName1> FULL JOIN <TableName2> ON <TableName1>.<ColumnNm1>=<TableName2>.<ColumnName2> WHERE <Condition> ORDER BY <ColumnName1>, <ColumnName2>	ANSI-style:- SELECT eno,e.dno,ename,dname from emp e Full join dept d on e.dno=d.dno; (i.e) Display all the data of emp table and emp , if dno of both table is match.

CROSS JOIN:

- A cross join returns what's known as a Cartesian product. This means that the join combines every row from the left table with every row in the table.
- As can be imagined, sometimes this join produces a mess, but under the right circumstances, it can be very useful.
- This type of join can be used in situations where it is desired to select all possible combination of rows and columns from both tables.
- This kind of join is usually not preferred as it may run for a very long time and produce a huge result set that may not be useful.

Syntax:-	Example:-
ANSI-style SELECT<ColumnNm1>,<ColumnNm2>, <ColumnNmN> FROM <TableName1> CROSS JOIN <TableName2>	ANSI-style SELECT eno,e.dno,ename,dname from emp e cross join dept d;

SELF JOIN:-

- In some situations, it is necessary to join a table to itself. This is known as a self-join. In a self-join, two rows from the same table combine to form a result row.
- To join a table to itself, two copies of the very same table have to be opened in memory. Hence in the FROM clause, the table name needs to be mentioned twice.
- Since the table names are same, second table will overwrite the first table. To avoid such a problem, each table is opened using an alias.
- Now these table alias cause two identical tables to be opened in different memory locations. This will result in two identical copies to be physically present in the computer's memory.

Syntax:- (Self Join)	Example:-
SELECT<ColumnNm1>,<ColumnNm2>, <ColumnNmN> FROM <TableName> [<Alias1>],<TableName> [<Alias2>] Where <Alias1>. <ColumnName>=<Alias2>.<ColumnName>;	Display all employee from employee table with its manager name. Select e.fname "Employee", m.fname "Manager" from emp e, emp m where e.manager_id = m.employee_id;

Data Control Language

(Granting and Revoking object privileges)

- The Owner of that Objects always gives object Privileges. If a user wishes to access any of the objects belonging to another user, the owner of the object will have to give permissions for such access. This is called Granting of Privileges.
- Privileges once given can be taken back by the owner of the object. This is called Revoking of Privileges.
- Each object privileges that are grant authority to perform some operation on the object. A user can grant all the privileges or grant only specific privileges.

Grant privileges using Grant Statement:-

Grant statement provides various types of access to database objects such as tables, views and sequences and so on.

Syntax:-	GRANT <Object Privileges> ON <Object Name> TO <User Name> [WITH GRANT OPTION];
Object Privileges:-	The list of object privileges are as follows:- ALTER:- Allows the grantee to change the table definition with the ALTER TABLE command. DELETE:- Allows the grantee to removes the records from the table with the DELETE command. INSERT:- Allows the grantee to add records to the table with the INSERT TABLE command. SELECT:- Allows the grantee to query the table with the SELECT command.

	UPDATE:- Allows the grantee to modify the records in the tables with the UPDATE command. INDEX:- Allows the grantee to create an index on the table with the CREATE INDEX command. WITH GRANT OPTION:- Allows the grantee to in turn grant object privileges to other users.
Example:-	SQL> grant select, insert, update on emp to sb1 ; that means give the grant of perform insertion, updating and deletion operation to user sb1;

Revoking Permission using REVOKE Statement:-

The Revoke statement is used to deny the grant given on an object.

Syntax:-	REVOKE <Object Privileges> ON <Object Name> FROM <User Name>
Object Privileges:-	The list of object privileges are as follows:- ALTER:- It take back the permission of to change the table definition. DELETE:- It take back the permission of to removes the records from the table. INSERT:- It take back the permission of to add records to the table. SELECT:- It take back the permission of to query the table. UPDATE:- It take back the permission of to modify the records in the tables.. INDEX:- It take back the permission of to create an index on the table.
Example:-	SQL> REVOKE select, insert, update on emp FROM sb1 ; that means give the grant of perform insertion, updating and deletion operation to user sb1;

Transaction Control Language

(Commit, Rollback and Savepoint)

Definition:

- A series of one or more SQL statements that are logically related or a series of operations performed on Oracle table data is termed as a **Transaction.**
- The Oracle server ensures data consistency based on transactions.
- Transactions give you more flexibility and control when changing data and they ensure data consistency in the event of user process failure or system failure.
- A transaction consists of DML statements that make up one consistent change to the data.

- Oracle treats this logical unit as a single entity. Oracle treats changes to table data as a two-step process. First, the changes requested are done. To make these changes permanent a COMMIT statement has to be given at the SQL prompt. A ROLLBACK statement given at SQL prompt can be used to undo a part of or the entire transaction.
- A transaction begin with first executable SQL statement after a commit, rollback or connection made to the oracle engine.
- A transaction is a group of events that occur between any one of the following events:-
  1) Connection to the oracle
  2) Disconnection from oracle
  3) Committing changes to the database tables.
  4) Rollback

<table>
<tr><td rowspan="3">COMMIT</td><td>Use</td><td>Using the commit statement ends the current transaction and makes permanent any changes made during the transaction.</td></tr>
<tr><td>Syntax</td><td>COMMIT;</td></tr>
<tr><td>Example:-</td><td>SQL> update emp set sal=1000 where eno=1;
2 row updated
SQL> commit;
Commit complete.
That means update two rows permanently in emp table.</td></tr>
<tr><td rowspan="5">ROLLBACK</td><td>Use:-</td><td>A rollback does exactly the opposite of commit; It ends the transaction but undoes any changes made during the transaction.</td></tr>
<tr><td>Syntax:-</td><td>ROLLBACK [TO [SAVEPOINT] <SavePointName>];</td></tr>
<tr><td></td><td>Where,
SAVEPOINT: is optional and is used to rollback a transaction partially, as far as the specified savepoint
SAVEPOINTNAME: is a savepoint created during the current transaction.</td></tr>
<tr><td>Example:-</td><td>SQL> update emp set sal=1000 where eno=1;
2 row updated
SQL> rollaback;
That means undo the data of emp table without updation.</td></tr>
<tr><td colspan="2">A ROLLBACK operation performed without the SAVEPOINT clause amounts to the following:
• Ends the transaction
• Undoes all the changes in the current transaction
• Erases all savepoints in that transaction
• Releases the transactional locks</td></tr>
</table>

<table>
<tr><td rowspan="4">SAVEPOINT</td><td>Use</td><td>Savepoint marks and saves the current point in the processing of a transaction. When a SAVEPOINT is used with a ROLLBACK statements, parts of a transaction can be undone.</td></tr>
<tr><td>Syntax:-</td><td>SAVEPOINT <SavePointName>;</td></tr>
<tr><td>Example:-</td><td>SQL>SAVEPONT s1;
SQL> update a1 set salary=20000 where
ename='Amit';
SQL> savepoint s2;
SQL>delete from a1 where eno=1;
SQL> rollback to savepoint s1;
That means undo the data of emp table upto savepoint s1.</td></tr>
<tr><td colspan="2">A ROLLBACK operation performed with the SAVEPOINT clause amounts to the following:
• A predetermined portion of the transaction is rolled back.
• Retains the save point rolled back to, but loses those created after the named savepoint.
• Releases all transactional locks that were acquired since the savepoint was taken.</td></tr>
</table>