# Structure

1.0 Introduction  
1.1 Purpose of Data Base Systems  
1.2 Data Abstraction  
1.3 Instance, Schema and Sub-Schema  
1.4 Data Independence  
1.5 Data Models  
1.6 Data Base Languages  
1.7 Responsibilities Of Database Manager  
1.8 Database Administrator  
1.9 Database Users  
1.10 Overall System Structure Components  

## Learning Objectives

After studying this unit, student will able to learn  

- Data base system, data abstraction, instance schema and sub-schema.  
- Understand data models, data base languages and database administrator.
1.0 Introduction

The most important component of any computer-based information system is the databases. A data base can defined as a collection of inter-related relevant data stored together to serve multiple applications.

A DBMS is a collection of interrelated data and a set of programs to access those data. The collection of data, usually called as database, contains about one particular organization or enterprise.

Or

A database management system (DBMS) can b defined as a collection of software packages for processing the database.

(or)

A data base management system (DBMS) can be defined as a collection of programs that are used to define, manipulate, control and process database.

A DBMS is essentially a tool to build a database structure, store and operate on the data contained in it. The main objective of designing any database management system is to provide its users an appreciably higher level frame work (as compared to files access facilitates provided in high-level programming languages) such that the relevant interrelated data can be collected and stored together to serve multiple application. ADBMS creates and maintains a database, loads and manipulates data within a database.

1.1 Purpose of Data Base Systems

One way to keep the information on a computer is to store it in permanent system files. To allow users to manipulate the stored information, the system has a number of application programs that manipulate the organized files. These application programs are written by system programmers in response to the needs of the organizations. New application programs are added to the system as the need arises. Thus, as the time goes more files and more application programs are added to the system. A typical file processing system described above is the system used to store information before the advent of DBMS.

Keeping the information of an organization in a file processing system has a number of disadvantages, namely

- **Data Redundancy and Inconsistency**: Since the files and applications programs are created by different programmers over a long period, the various files are likely to have different formats and the programs may be written in several programming languages. Moreover, the same information
may be duplicated in several places. This redundancy leads to higher storage and access cost. In addition, it may lead to data inconsistency.

- **Difficulty in Accessing Data**: The file processing system does not allow needed data to be retrieved in a convenient and efficient manner.

- **Data Isolation**: In a file processing system, as the data are scattered in various files, and files may be in different formats. It is very difficult to write new application programs to retrieve the appropriate data.

- **Integrity problems**: The data values stored in the database must satisfy certain types of consistency Constraints (Conditions). For example, the minimum balance in a bank account may never fall below an amount of Rs. 500. Developers enforce these constraints in the system by adding appropriate code in the application programs. However, when new constraints are added, it is difficult to change the application programs to enforce them.

- **Security problems**: Not every user of the database system should be able to access all the data. In the file processing systems, as the application programs are added to the system in an adhoc manner, it is difficult to enforce security.

The above disadvantages can be overcome by use of DBMS and it provides the following advantages.

1. Provides for mass storage of relevant data.
2. Make easy access of the data to user.
3. Allows for the modification of data in a consistent manner.
4. Allows multiple users to be active at a time
5. Eliminate or reduce the redundant data.
6. Provide prompt response to the users request for data.
7. Supports Backup and recovery of data.
8. Protect data from physical hardware failure and unauthorized access.
9. Constraints can be set to database to maintain data integrity.

### 1.2 Data Abstraction

A DBMS is a collection of interrelated files and set of programs which allows the users to access and modify these files. A major purpose of a database system is to provide users with an abstract view of the data. That is, the system hides certain details of how the data are stored and maintained.
Level of Abstraction: basically, Abstraction can be divided into 3 levels.

They are

1. **Physical Level**: The lowest of abstraction describes how the data are actually stored. At the physical level, complex low-level data structures are described in detail.

2. **Logical Level (Conceptual Level)**: This next higher level of abstraction describes what data are stored in the database, and what relationship exist among those data. This level of abstraction is used by Database Administrators (DBA), Who must decide what information is to be kept in the database.

3. **View Level**: This Highest level of abstraction describes only part of the entire database. The use of simpler structures at the logical level, some complexity remains, because of the large databases. Many users of the database system will not be concerned with all this information. Such users need to access only a Part of the database. So that their interaction with the system is simplified, the view level of abstraction is defined. The system may provide views for the same database.

### 1.3 Instance, Schema and Sub-Schema

Database change over time as information is inserted and deleted. The collection of information stored in the database at a particular moment is called an ‘Instance’ of the database.

The overall design of the database is called the database ‘schema’. Schemas are changed frequently.
A database scheme corresponds to the programming language type definition. A variable of a given type has a particular value at a given instant. Thus, the value of a variable in programming languages corresponds to an instance of a database schema.

Database systems have several schemas, portioned according to the level of abstraction. At the lowest level is Physical scheme, at the intermediate level is logical schema and at the higher level is a subschema.

In general, database system supports one physical schema, one logical schema and several subschema's.

### 1.4 Data Independence

The ability to modify a schema definition in one level without affecting a schema definition in the next higher level is called data Independence. There are two levels of data independence.

They are

1. **Physical Data Independence**: The ability to modify the physical schema without causing application programs to be rewritten. The modifications at this level are occasionally necessary to improve performance.

2. **Logical Data Independence**: The ability to modify the logical schema without causing application programs to be rewritten. The modifications at this level are necessary whenever the logical structure of the database is altered.

Logical data independence is more difficult to achieve than is physical data independence, since application programs are more dependent on the logical structure of data that they access.

### 1.5 Data Models

The data model plays an important role in database design. The physical or logical structure of a database is spelt out by the data model.

A data model is a collection of conceptual tools used for describing data, data relationships, data semantics and data constraints.

Evaluation of different data models is still in progress as the primary objective is to evolve a high level data model. The model should enable the designer to incorporate a major portion of semantics of the database in the schema. Numerous data models have been proposed which can be broadly classified in the following categories.
Classification Of Data Models

1. Object based data models
2. Record – based data models
3. Physical data models

1.1.5.1 Object – Based Data Models

Object-based logical models are used in describing data at logical and view levels. They are characterized by the fact they provide flexible structuring capabilities and allow data constraints to be specified explicitly. There are many different data models, some of them are

(i) The Entity-relationship model
(ii) The Object-oriented model
(iii) The semantic data model
(iv) The Functional data model

In this book, we find the entity-relationship model and the Object Oriented model as representative of the class of Object-based logical models. The Entity – Relationship model which is an object – based model is widely used in practice as an appropriate database design tool, explored in Unit-II.

(i) The Entity-relationship Model: This data model is based on a perception of real world that consists of a collection of basic objects, Entities, Entity sets, relationship and relationships sets. The overall logical structure of a
database can be expressed graphically by an E-R-Diagram, which is made up of components. Some of they are

- **Rectangles**: Which represent entity sets.
- **Ellipses**: Which represent attributes
- **Diamonds**: Which represent relationship sets
- **Lines**: Which link attributes to entity sets and entity sets to relationship sets.

**(ii) The Object-Oriented Model**: This model is based on a collection of objects. An object contains values stored in instance variables within the object. An object also contains bodies of code that operate on the object. These bodies of code are called Methods.

Objects that contain the same types of values and the same methods are grouped together into classes. A class may be viewed as a type definition for objects. This combination of data and methods combining a type definition is similar to a programming language (OOPS) abstract data type.

The only way in which one object can access the data of another object is by invoking a method of that other object. This action is called sending a message to the object.

For Example, let us consider an object representing a Bank account. Such an object contains instance variables ‘account-number’ and ‘Balance’. It contains a method ‘pay-interest’, which adds interest to the balance.

### 1.5.2 Record – Based Data Models

These models are used to specify the overall logical structure of the database. With some models a higher level description of the implementation of the structure of the database can also be specified explicitly. The data integrity constraints cannot be specified explicitly with these models.

In record based data models, the database is structured in fixed formats records of several types. Each record defines fixed number of fields (attributes) and each field is fixed length. These models are used to specify the overall logical structure of the database and are used in describing the database at conceptual level.

**The three widely accepted record – based data models are**

(a) Relational model

(b) Network model
(c) Hierarchical model

(a) **Relational Data Model**: The relational model is currently the most popular data model in the database management systems. The popularity is because of simplicity and understandability. This data model is developed by E.F. Codd in 1970 which is based on relation, two dimensional table.

The relational data model uses a collection of tables (also called as relation) to both data and the relationships among those data. Each table has multiple columns and each column has unique name. A relation consists of rows and columns. The row in table (relation) is called as Tuple and column name are known as attribute.

**Ex: Customer Table**

<table>
<thead>
<tr>
<th>Customer Name</th>
<th>UID</th>
<th>Address</th>
<th>Account No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tanveer</td>
<td>A12345</td>
<td>Hyd</td>
<td>A – 101</td>
</tr>
<tr>
<td>Ramesh</td>
<td>B23456</td>
<td>Sec’bad</td>
<td>A – 215</td>
</tr>
<tr>
<td>Ravi</td>
<td>C34567</td>
<td>Charminar</td>
<td>A – 305</td>
</tr>
<tr>
<td>Prasad</td>
<td>A345789</td>
<td>Banglore</td>
<td>A – 201</td>
</tr>
<tr>
<td>Smith</td>
<td>Z459087</td>
<td>Delhi</td>
<td>A - 405</td>
</tr>
</tbody>
</table>

**Account Table**

<table>
<thead>
<tr>
<th>Account No</th>
<th>Balance</th>
</tr>
</thead>
<tbody>
<tr>
<td>A – 101</td>
<td>500</td>
</tr>
<tr>
<td>A – 215</td>
<td>800</td>
</tr>
<tr>
<td>A – 305</td>
<td>400</td>
</tr>
<tr>
<td>A – 201</td>
<td>900</td>
</tr>
<tr>
<td>A - 405</td>
<td>750</td>
</tr>
</tbody>
</table>

**Advantages**

1. In this model, data redundancy is controlled to a greater extent
2. The relational data model allows many-to-many relationships.
3. The relational data model structures are very simple and easy to build
4. Faster access of data is possible and storage space required is greatly reduced.

(b) **Network Data Model**: Data in the network model are represented by collections of records and relationships among data are represented by links, which can be viewed as pointers. The records in the database can be organized as a collection of arbitrary graphs. The Network data model is similar to Hierarchical model except that one data can have more than one parent. Any record in the database is allowed to own sets of other type of record.

**Advantages**

1. It can be used to represent many-to-many relationships
2. It offers integration of data
3. The storage space is reduced considerably due to less redundancy
4. It provides faster access of data.

(c) **Hierarchical Data Model**: In this model the relationship among the data is represented by records and links. It consists of records which are connected to another through links. A link can be defined as an association between two records. This hierarchical data model can do considered as an upside–down tree, with the highest level of tree kept as root.

**Advantages**

1. The hierarchical model, allows one-to-one-and one-to-many relationships.
2. The model has got the ability to handle large amount of data.

**Disadvantages**

1. The model involves with complicated querying.
2. As duplication of data takes place, there is wastage of storage space.
3. During updating of data inconsistency exists.
4. The model does not allow many-to-many relationships.

**1.5.3 Physical Data Models**

Physical data model are used to describe data at the lowest level. In contrast to logical data models, there are few number of physical data models which are in use. very few physical data models have been proposed so far.
Two of these well known models are the unifying model and the frame memory model.

### 1.6 Data Base Languages

A database system provides two different types of Languages, one will specify the schema, and other will express database queries and updates. They are

1. Data-Definition Languages (DDL)
2. Data-Manipulation Language (DML)

1. **Data-Definition Languages (DDL)**: A database scheme is specified by set of definitions which are expressed by special language called Data Definition Language (DDL). The result of compilation of DDL statements is a set of tables that is stored in a special file called ‘Data dictionary’ or ‘data directory’.

A data dictionary is a file that contains metadata, i.e. Data about data. This file is consulted before actual data are read or modified in the database system.

The storage structure and access methods used by the database system are specified by a set of definitions in a special type of DDL called a ‘data storage and data definition language’. The result of consultation of these definitions is a set instruction to specify the implementation details of the database schemas. Which are usually hidden form the users.

The DDL commands are

2. **Data-Manipulation Language (DML)**: A DML is language which enables users to access or manipulate data as organized by appropriate data model. The goal is to provide efficient human interaction with the system. The DML allows following

   (a) The retrieval information from the database
   (b) The Insertion of new information into existing database
   (c) The deletion of existing information from database
   (d) The modification of information stored in the database.

A DML is language which enables users to access or manipulate data. There are basically two types.

- **Procedural DML**: This requires a user to specify what data are needed and how to get those data from existing database.
• Non procedural DML: Which require a user to specify what data are needed ‘without’ specifying how to get those data.

Nonprocedural DMLs are usually easier to learn and use than procedural DMLs. A user does not have to specify how to get the data, these languages may generate code that is not as that produced by Procedural DML. Hence we can make remedy this difficulty by various optimization techniques.

A Query is a statement, a request for retrieval information. The portion of a DML, that involves information retrieval is called a ‘Query Language’.

### 1.7 Responsibilities Of Database Manager

A Database Manager is a program module which provides the interface between the low level data stored in the database and the application program and queries submitted to the system.

**Responsibilities of Database Manager**

1. **Interaction with File Manager**: The row data is stored on the disk using the file system which is usually provided by conventional operating system.

2. **Integrity Enforcement**: The data values stored in the database must satisfy certain types of consistency constraints.

3. **Security Enforcement**: Not every user of the database needs to have access to the entire content of the database.

4. **Backup and Recovery**: It is the responsibility of database manager to detect such failures and restore the database to a state that existed prior the occurrence of the failure this is usually accomplished through the backup and recovery processor.

5. **Concurrency Control**: It is necessary for the system to control the interaction among the concurrent users, and achieving such control is one of the responsibilities of database manager.

6. **Authorization Control**: This module checks that the user has necessary authorization to carry out the required function.

7. **Command Processor**: Once the system has checked that the user has authority to carry out the operation control satisfies all necessary integrity constraints such as key constraints.

8. **Integrity Checker**: For an operation that changes the database the integrity checker checks that the requested operation satisfies all necessary integrity constraints such as key constraints.
9. **Query Optimizer**: This module determines an optimal strategy for the query execution.

10. **Scheduler**: This module is responsible for ensuring that concurrent operations or transactions on the database proceed without conflicting with one another.

11. **Recovery Manager**: This module ensures that the database remains in a consistent state in the presence of failures. It is responsible for transaction commit and abort that is success or failure of transaction.

12. **Buffer Manager**: It is responsible for the transfer of data between main memory and secondary storage such as disk and tape. The recovery manager and buffer manager are sometimes collectively referred to as data manager. The buffer manager is sometimes known as cache manager.

13. **Query Processor**: This is responsible for receiving query language statements and changing them from the English-like syntax of the query language to from the DBMS can understand. The query language processor usually consists of two parts i.e. 1. The parser 2. Query Optimizer.

### 1.8 Database Administrator

A person who has central control of both the data and programs over the system is called database administrator. Or Database administrator (DBA) is the person or group, who is responsible for the supervision and control of the databases, within an organization.

**The functions or responsibilities of DBA includes**

1. **Schema Definition**: The DBA creates the original database schema by writing a set of definitions which are translated by DDL compiler to a set of tables that is stored permanently in the data dictionary.

2. **Storage structure and access Method Definition**: A DBA creates appropriate storage structures and access methods by writing a set of definitions, which are translated by the data-storage and DDL compiler.

3. **Schema Physical Organization and Modification**: The DBA allows the users to accomplish modifications in the database scheme or storage organization through a set of definitions and make these modifications to be appropriate in the internal system tables.

4. **Granting of Authorization for data access**: By granting different types of authorization, the database administrator can regulate which of the database various users can access.
5. Routine Maintenance: DBA is the final authority to regulate the daily activities.

As a whole, the DBA jobs are

- Creating primary database storage structures
- Modifying the structure of the database
- Monitoring database performance and efficiently
- Transferring data between the database and external file
- Monitoring and reestablishing database consistency
- Controlling and monitoring user access to the database
- Manipulating the physical location of the database.

1.9 Database Users

The primary goal of any database is to provide its users an environment for retrieving and storing new information into the database. An important aspect of the DBA’s working is that, it has liaison with users at all levels of the organization. The DBA must be able to communicate effectively with the users and must be conversant with the technical aspects of the system as well as the working procedures of the organization.

Depending on the way that the users expect to interact with the database system, the users are classified into

1. Application Programmers: Application Programmers are computer professionals interacting with the system through DML calls, embedded in a program written in a language like high level languages like COBOL, C, etc.

2. Sophisticated users: These users interact with system without writing programs. They form their request by writing queries in a database query language. Those are submitted to a query processor that breaks a DML statement down into instructions for the database manager’s module.

3. Unsophisticated users: Who interact with the system by using permanent applications. Ex ATM

4. Specialized users: These users write specialized database applications that do not fit into the traditional data processing framework. These applications include Computer-aided design (CAD) systems, Knowledge base expert systems etc.
Unsophisticated Users
Application Programmers
Sophisticated Users
Database Administrator

Embedded DML Precompiler
DML Compiler

Application Programs
Query Evaluation Engine

Transaction Manager
Buffer Manager
File Manager

Indices
Statistical Data

Data files
Data dictionary
1.10 Overall System Structure Components

A database system is partitioned into modules that deal with each of the responsibilities of the overall system. Some of the functions of the database system may be provided by the computer’s operating system. In most cases, the computer’s operating system provides only the basic services, and the database must build on that base. Thus, the design of a database system must include consideration of the interface between the database system and the operating system.

The functional components of a database system are divided into two groups as query processor components and storage manager components.

The query processor components include DML compiler, Embedded DML pre-compiler, DDL interpreter and Query evaluation engine. The storage manager components provide the interface between the low level data stored in the database and the application programs and queries submitted to the system. These components include Authorization and integrity manager, Transaction manager, File manager and Buffer manager.

In addition to the above components, as part of the physical system implementation, several data structures are required. These include data files, data dictionary, indices and statistical data.

Short Answer Type Questions

1. What is database?
2. What is Data Processing?
3. What is a DBMS?
4. What are components of DBMS?
5. Define Instance.
7. What is data independence?
8. What is difference between physical and logical data independence?
9. Who will be called as DBA.
10. What is Meta data or Data Dictionary?
11. What the three levels of data abstraction?
12. Write different types of database users.
13. Expand the terms DDL, DML, DCL?

14. Write the commands of DDL, DML, DCL

Long Answers Type Questions

1. What are the advantages of DBMS over File Processing system?

2. Explain about different data models.

3. What is data abstraction? Explain in detail?

4. Explain DDL, DML and DCL commands with examples.

5. What are the responsibilities of Database Manager? Explain

6. What are the functions of DBA?

7. Discuss briefly about different types database users.
2.0 Introduction
2.1 Entity sets
2.2 Attributes
2.3 Relationship Sets
2.4 Mapping Constraints
2.5 Entity – Relationship Diagrams
2.6 Drawing E-R diagrams
2.7 Procedure for conversion of ER Diagram into a database table

Learning Objectives
After studying this unit, the student will be able to

• Understand about different entity sets
• Understand about datatypes and attributes
• Understand about different mapping constraints
• Know about entity relationship diagrams
• Learn how to convert entity relationship diagrams into database table
2.0 Introduction

The Entity – Relationship (E - R) data model was developed to facilitate database design by allowing specification of an enterprise schema that represents the overall logical structure of a database. The E-R model lies in its representation of the several semantic data models; the semantic aspect of the model lies in its representation of the meaning of the data. The E-R model is very useful in mapping the meanings and interactions of real-world enterprises into a conceptual schema. Because of this usefulness, many database design tools draw on concepts from the E – R Model. The E – R data model employs three basic notions: Entity, Relationship sets and Attributes.

2.1 Entity Sets

An Entity is an “object” that exists and is distinguishable from other objects. An Entity is represented by a set of attributes. These attributes are the descriptive properties possessed by each entity. For example, Roll Number of a Student, is an entity, it is uniquely identifies a person in a class. i.e the Roll Number of a student is distinguishable from one other.

2.1.1 Entity Sets

The Entity Set is a set of entities of the same type, that share the same properties or Attributes. Ex: Persons having an account at bank. or Each student having Admission Number.

The set of all students in a class., can be defined as the entity-set. Who are students of an university, for example, can be defined as the entity set student. Similarly, the entity set Admission might represent the set of all admission awarded by a particular university. The individual entities that constitute a set are said to be the extension of the entity set. Thus, all the individual students are the extension of the entity set student.

Example: Student Table

<table>
<thead>
<tr>
<th>Student Name</th>
<th>Address</th>
<th>Admission No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Salman</td>
<td>Hyderabad</td>
<td>27096-12-101</td>
</tr>
<tr>
<td>Muneer</td>
<td>Sec’bad</td>
<td>27096-12-102</td>
</tr>
<tr>
<td>Imran</td>
<td>Charminar</td>
<td>27096-12-103</td>
</tr>
<tr>
<td>Ravi</td>
<td>Banglore</td>
<td>27096-12-104</td>
</tr>
<tr>
<td>Raja</td>
<td>Delhi</td>
<td>27096-12-105</td>
</tr>
</tbody>
</table>
Bank Account

<table>
<thead>
<tr>
<th>Account No</th>
<th>Name of the Bank</th>
<th>Branch</th>
<th>Balance</th>
</tr>
</thead>
<tbody>
<tr>
<td>A – 101</td>
<td>HDFC</td>
<td>Malakpet</td>
<td>500</td>
</tr>
<tr>
<td>A – 215</td>
<td>SBI</td>
<td>Saidabad</td>
<td>800</td>
</tr>
<tr>
<td>A – 305</td>
<td>SBH</td>
<td>YMCA</td>
<td>400</td>
</tr>
<tr>
<td>A – 201</td>
<td>CANARABANK</td>
<td>Narayanguda</td>
<td>900</td>
</tr>
<tr>
<td>A - 405</td>
<td>HDFC</td>
<td>Malakpet</td>
<td>750</td>
</tr>
</tbody>
</table>

2.2 Attributes

Each entity has certain characteristics known as attributes. For instance, the student entity might include the following attributes: Student name, Roll Number etc. For each attribute, there is a set of permitted values, called the **domain**, or **value set**, of that attribute. An attribute of an entity set is a function that maps from the entity set into a domain. Since an entity set may have several attributes, each entity can be described by a set of (attribute, data value) pairs, one pair for each attribute of the entity set.

**The attributes can be classified into**

1. Simple attributes
2. Complex/composite attributes
3. Single–valued attributes
4. Multi-valued attributes
5. Derived attribute
6. Null Attribute

An attribute, as used in the E–R model, can be characterized by the following attribute types.

- **Simple attributes**: The attributes have been simple; that is, they have not been divided into subparts. Example: Student class Roll Number.

- **Composite attributes**: The attributes, which can be sub divided into subparts.

Example: Student Name, which can be divided into parts like First name, Middle name and Last name. Note also that a composite attribute may
appear as a hierarchy. In the composite attribute address, its component attribute
street can be further divided into street_number, street_name, and Door_number
etc.

- **Single valued attributes**: The attribute which contain/ accept only
  one value/character.

**Example**

**Sex**: Male or Female

**Marital status**: Married or Unmarried

- **Multivalued attributes**: The attributes which has set of values for
  a specific entity. in our example all have a single value for a particular entity.

**Example 1**: Number of dependents in a family may 0,1,2,3,4…..

**Example 2**: A student may have several phone numbers, and different
students may have different numbers of phones.

- **Derived attribute**: The value of this type of attribute can be derived
  from the values of other related attributes or entities. The value of a derived
  attribute is not stored but is computed when required.

- **Null Attributes**: An attribute takes a null value when an entity
does not have a value for it. The null value may indicate “not applicable” – that
is, that the value does not exist for the entity.

### 2.3 Relationship Sets

A **Relationship** is an association among several entities. For example,
we can define a relationship between student entity and bank account entity.
The student named Rahul having bank account in HDFC, Malakpet branch
with an account number A-101.

**The Relationship also can be define as**

A **Relationship set** is a set of relationships of the same type. Formally,
it is a mathematical relation on n e”2 (possibly non distinct) entity sets. If E1,
E2, ……., E_n are entity sets, then a relationship set R is a subset of

\[
\{ (e_1, e_2, \ldots, e_n) | e_1 \circ E_1, e_2 \circ E_2, \ldots, e_n \circ E_n \}
\]

Where (e_1, e_2, ……., e_n) is a relationship.

The association between entity sets is referred to as participation; that
is, the entity sets E_1, E_2, ……., E_n **participate** in relationship set R. A
relationship instance in an E–R schema represents an association between the named entities in the real–world enterprise that is being modeled.

The function that an entity plays in a relationship is called that entity’s role. Since entity sets participating in a relationship set are generally distinct, roles are implicit and are not usually specified. However, they are useful when the meaning of a relationship needs clarification. Such is the case when the entity sets of a relationship set are not distinct; the same entity set participates in a relationship.

A relationship may also have attributes called descriptive attributes. A relationship instance in a given relationship set must be uniquely identifiable from its participating entities, without using the descriptive attributes.

2.4 Mapping Constraints

An E–R enterprise schema may define certain constraints to which the contents of a database must conform. In this section, we examine mapping cardinalities, key constraints, and participation constraints.

Cardinalities: Mapping Cardinalities, or cardinality ratios, express the number of entities to which another entity can be associated via a relationship set.

Mapping Cardinalities are most useful in describing binary relationship sets, although they can contribute to the description of relationship sets that involve more than two entity sets. In this section, we shall concentrate on only binary relationship sets.

For a binary relationship set R between entity sets A and B, the mapping cardinality must be one of the following.

There are 4 types of mapping cardinalities.

1. ONE–to–ONE Relationship
2. MANY–to–MANY Relationship
3. ONE–to–MANY Relationship
4. MANY–to–MANY Relationship

1. ONE–to–ONE Relationship: An entity in A is associated with at most one entity in B is also associated with at most one entity in A.
Example: Relationship between the entities principal and college. i.e., principals can lead a single college and a principal can have only one college.

2. Many – to – One Relationship: An entity set in A is associated with at most one entity in B. An entity in B however can be associated with any number of entities in A.

Example: Relationship between the entities Districts and state. i.e. many districts belong to a single state but many states cannot belong to single district.

3. One – to - Many Relationship: An entity set A is associated with any number of entities in B. An entity in B, however can be associated with at most one entity in A.

Example: Relationship between the entities class and student i.e., a class can have many students but a student cannot be in more than one class at a time.
4. MANY – to – MANY Relationship: An entity set in A is associated with any number of entities in B and an entity set in B is associated with any number of entities in A.

Example: Relationship between the Entities College and course. i.e. a college can have many courses and course can be offered by many colleges.

The appropriate mapping cardinality for a particular relationship set obviously depends on the real – world situation that the relationship set is modeling.

2.5 Entity – Relationship Diagrams

An E-R diagram can express the overall logical structure of a database graphically. E-R diagrams are simple and clear – qualities that may well account in large part for the widespread use of the E-R model. Such diagram consists of the following major components.
- **Rectangles**: Which represent entity sets.
- **Ellipses**: Which represent attributes
- **Diamonds**: Which represent relationship sets
- **Lines**: Which link attributes to entity sets and entity sets to relationship sets
- **Double ellipses**: Which represents multivalued attributes
- **Dashed ellipses**: Which denote derived attributes.
- **Double Lines**: Which indicate total participation of an entity in a relationship set.
- **Double Rectangles**: Which represent weak entity sets

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Represented ERD Property</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Entity" /></td>
<td><strong>Entity</strong></td>
</tr>
<tr>
<td><img src="image" alt="Weak Entity" /></td>
<td><strong>Weak Entity</strong></td>
</tr>
<tr>
<td><img src="image" alt="Relation Ship" /></td>
<td><strong>Relation Ship</strong></td>
</tr>
<tr>
<td><img src="image" alt="Identifying Relationship" /></td>
<td><strong>Identifying Relationship</strong></td>
</tr>
<tr>
<td><img src="image" alt="Attribute" /></td>
<td><strong>Attribute</strong></td>
</tr>
</tbody>
</table>
### 2.6 Drawing E-R diagrams

Example I - ER diagram with the entity sets Customer (Customer-name, Social-security, Street, Customer-city), Account (Account-Number, Balance) with a relationship Custacct (date), i.e. date is attribute of relationship as shown below.
Example : 2

ER diagram showing the cardinality of the customer account relationship as one-to-many is as shown below.

Note: Observe an arrow mark towards customer on the line joining customer and custacc. i.e, arrow mark indicates the cardinality one, on arrow mark indicates many.

Example : 3

ER diagram showing the many - to one relationship between customer and account is as shown below. (Observe the arrow mark is towards account)
Example: 4

ER diagram showing the one-to-one relationship between customer and account is as shown below. (Observe the arrow mark is towards account both customer as well as account).

Note: In the first ER diagram, the cardinality is many-to-many, because there are no arrows towards both the entity sets.

Role in E-R Diagrams

The function that an entity plays in a relationship is called its role. Roles are normally explicit and not specified.

They are useful when the meaning of a relationship set needs clarification.

For example, the entity sets of a relationship may not be distinct. The relationship works - for might be ordered pairs of employee (first is manager, second is worker).
In the E-R diagram, this can be shown by labeling the lines connecting entities (rectangles) to relationships (diamonds).

**Weak Entity sets E-R Diagrams**

**Weak entity set**: An entity set that does not possess sufficient attributes to form a primary key is called a weak entity set.

**Strong Entity set**: An entity set that has a primary key is called a strong entity set.

**For example**

The entity set transaction has attributes transaction-number, date and amount. Different transactions on different accounts could share the same number. These are not sufficient to form a primary key (uniquely identify a transaction). Thus, transaction is a weak entity set.

For weak entity set to be meaningful, it must be part of a one-many relationship set. This relationship set should have no descriptive attributes.

The idea of strong and weak entity sets is related to the existence dependencies seen earlier.

Member of a strong entity set a dominant entity. Member of a weak entity set is a subordinate entity.

A weak entity set does not have a primary key, but we need a means of distinguishing among the entities. The discriminator of a weak entity set is a set of attributes that allows this distinction to be made.

The primary key of a weak entity set is formed by taking the primary key of the strong entity set on which its existence depends (see Mapping Constraints) plus its discriminator.


**Example : 5**

Transaction is a weak entity. It is existence-dependent on account. The primary key of account is account-number.

Transaction-number distinguishes transaction entities within the same account (and is thus the discriminator).

So the primary key for transaction will be (account-number, transaction-number).

**Note:** The primary key of a weak entity is found by taking the primary key of the strong entity on which it is existence-dependent, plus the discriminator of the weak entity set.

A weak entity set is indicated by a doubly-outlined box. For example, the previously-mentioned weak entity set transaction is dependent on the strong entity set account via the relationship set log.

**Example : 6**

E-R diagram of weak entities. Observe that transaction is placed in double rectangle.

![E-R Diagram](image)

**Example : 7**

Non-binary ER diagram (ternary) between three entity sets: customer, Account, Branch as shown below which says that a customer may have several accounts, each located in a specific bank branch, and that an account may belong to several different customers.
Reducing E-R Diagram into tables

A database conforming to an E-R diagram can be represent by a collection tables. Let us see how it can be done.

- For each entity set and relationship set, there is a unique table which is assigned the name of the corresponding set.
- Each table has a number of columns with unique names.

Rules to followed to reduce ER diagrams into tables are as given below.

- Primary keys allow entity sets and relationship sets to be expressed uniform as relation TABLES that represent the content of the database.
- Relationship TABLE that is assigned the entity set.
- A strong entity set reduces to a TABLE with the same attributes.
- A weak entity set becomes a table that includes a column for the primary key of the identifying strong entity set.
Consider ER diagram containing weak entity set as shown below. Here payment is weak entity set.

Example

The table for payment = (loan _ number, payment _ number, payment _ date payment _ amount)

The respective table is as shown below.

<table>
<thead>
<tr>
<th>Loan-number</th>
<th>payment-number</th>
<th>payment-date</th>
<th>payment-amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>L-11</td>
<td>53</td>
<td>7 June 2011</td>
<td>125</td>
</tr>
<tr>
<td>L-14</td>
<td>69</td>
<td>28 May 2011</td>
<td>500</td>
</tr>
<tr>
<td>L-15</td>
<td>22</td>
<td>23 May 2011</td>
<td>300</td>
</tr>
<tr>
<td>L-16</td>
<td>58</td>
<td>18 June 2011</td>
<td>135</td>
</tr>
<tr>
<td>L-17</td>
<td>5</td>
<td>10 May 2011</td>
<td>50</td>
</tr>
<tr>
<td>L-17</td>
<td>6</td>
<td>7 June 2011</td>
<td>50</td>
</tr>
<tr>
<td>L-17</td>
<td>7</td>
<td>17 June 2011</td>
<td>100</td>
</tr>
<tr>
<td>L-23</td>
<td>11</td>
<td>17 May 2011</td>
<td>75</td>
</tr>
<tr>
<td>L-93</td>
<td>103</td>
<td>3 June 2011</td>
<td>900</td>
</tr>
<tr>
<td>L-93</td>
<td>104</td>
<td>13 June 2011</td>
<td>200</td>
</tr>
</tbody>
</table>
• Each TABLE has a number of column (generally corresponding to attributes), which have unique names.

• A many-to-many relationship set is represented as a TABLE with attributes for the primary keys of the two participating entity sets, and any descriptives attributes of the relationship set.

Example

TABLE for relationship set borrower

borrower = (customer_id, loan_number)

The corresponding table is as shown below.

<table>
<thead>
<tr>
<th>Customer_id</th>
<th>Loan_number</th>
</tr>
</thead>
<tbody>
<tr>
<td>019-28-3746</td>
<td>L-11</td>
</tr>
<tr>
<td>019-28-3746</td>
<td>L-23</td>
</tr>
<tr>
<td>244-66-8800</td>
<td>L-93</td>
</tr>
<tr>
<td>321-12-3123</td>
<td>L-17</td>
</tr>
<tr>
<td>335-57-7991</td>
<td>L-16</td>
</tr>
<tr>
<td>555-55-5555</td>
<td>L-14</td>
</tr>
<tr>
<td>677-89-9011</td>
<td>L-15</td>
</tr>
<tr>
<td>963-96-3963</td>
<td>L-17</td>
</tr>
</tbody>
</table>

• Composite attributes are flattened out by creating a separate attribute for each component attribute.

Example

Given entity set customer with composite attribute name with component attributes first-name and last-name the table corresponding to the entity set has two attributes.

name, first_name and name, last_name

• A multivalued attribute M of an entity E is represented by a separate table EM. Table EM has attributes corresponding to the primary key of E and an attribute corresponding to multivalued attribute M.
Example

Multivalued attribute dependent - names of employee is represented by a table.

employee - dependent - names (employee-id, dname)

- Each value of the multivalued attribute maps to a separate row of the table EM.

Example

An employee entity with primary key 19444 and dependents John and Maria maps to rows (19444, John) and (19444, Maria)

Example

Reduce the following ER diagram into tables.

Each entity set customer and account will have a table and the relationship Custacct will have a table hence there will be three tables with the columns as given below.

1. Customer table with columns customer_name, Social_security, Street, Customer_city.

2. Account table with columns account_number, balance.

3. Custacct table with attributes Social_security, account_number, date.
2.7 Procedure for conversion of ER Diagram into a database table

1. The E–R diagram of any database can be represented by a collection of tables.
2. For each entity set and for each relationship set there is unique table to which is assigned the name of the corresponding entity set or relationship.
3. Each table has a number of columns which again have unique names i.e. attributes.
4. The values of all attributes are called records.
5. The column value which uniquely identifies the record in the table will be defined as primary key.
6. Other keys will be defined according to the relationship with other tables / entities.

Quick Reference

• The entity relationship (E-R) data model is a widely used data model for database design. It provides a convenient graphical representation to view data, relationships, and constraints.

• An entity is an object that exists in the real world and is distinguishable from other objects. We express the distinction by associating with each entity a set of attributes that describes the objects.

• A relationship is an association among several entities. A Relationship set is a collection of relationships of the same type. And an entity set is a collection of entities of the same type.

• Mapping Cardinalities express the number of entities to which another entity can be associated via a relationship set.

• An entity set that does not have sufficient attributes to form a primary key is termed a weak entity set. An entity set that has a primary key is termed a strong entity set.

• Domain It is a set of permissible values for each attribute simply a column is known as a domain.

• Degree It is number of columns associated with a table. This property is used to describe the total number of columns a table consists.
• **Cardinality** This is the number of rows in any given table.

• **Object** An Entity which is a “thing” in the real world with an independent existence.

• **Foreign Key** A Foreign Key is a column whose values are derived from the primary key or unique key of some other tables. It represents the relationships between tables.

• **Tuple** Tuples are the rows / records of a table. The tuple gives the total set of properties, attributes of an entity which can be used to describe it.

**Short Answer Type Questions**

1. What is Entity and Entity set?
2. What is Relationship and Relationship set?
3. What is Weak Entity and strong Entity?
4. What is an attribute? What are the different types in it?
5. What is a Domain?
6. What is tuple?
7. What is Degree of table?
8. What are the symbols used in E-R diagram?
9. What are mapping cardinalities?

**Long Answers Type Questions**

1. Explain the mapping constraints with neat diagram.
2. Draw an ER diagram by showing the relationship between a student and Bank.

(Nota: Practice more on ER Diagrams using different examples and different symbols like weak, strong entities, attributes etc)
3. Write the procedure to reducing of an E-R diagram into table.
3.0 Introduction

3.1 Table structure

3.2 Attributes

3.3 Keys

3.4 Query languages

3.5 Codd rules

Learning objectives

After studying this unit, the student will be able to

• Understand the relational model of database management

• Understand about the rules laid by Codd

• Understand the tables, attributes and keys

3.0 Introduction

The relational model for database management is a database model based on first-order predicate logic, first formulated and proposed in 1969 by Edgar F. Codd. In the relational model of a database, all data is represented in terms of tuples, grouped into relations. A database organized in terms of the relational model is a relational database.
In the relational model, related records are linked together with a “key”.

The purpose of the relational model is to provide a declarative method for specifying data and queries: users directly state what information the database contains and what information they want from it, and let the database management system software take care of describing data structures for storing the data and retrieval procedures for answering queries.

**Relational Database:** One of the major advantages of using a relational database is its structural flexibility. It allows the users to retrieve the data in any combination.

A relation is a two-dimensional array, consisting of horizontal rows and vertical columns. Each row, column ie a cell contains a unique value and no two rows are identical with respect to one another.

Columns are always self-consistent in the sense that it has the same meaning in every row. This means that the database management system (DBMS) is not concerned with its appearance, either first or next. The table will be processed the same way, regardless of the order of the columns.

Relations are commonly referred as tables. Every column in a database table acts as attribute since the meaning of the column is same for every row of the database. A row consists of a set of fields and hence commonly referred as a record.

**Properties of Relational Database:** The important properties of a relational database are listed below:

1. A relational database is a collection of relations.
2. The database tables have a row column format.
3. Operators are available either to join or separate columns of the database table.
4. Relations are formed with respect to data only.
5. The tables can be accessed by using simple non-procedural statements.
6. The data is fully independent, that is it will be the same irrespective of the access path used.
3.1 Table Structure

Structure of Relational Database:

Relational database systems are the most common DBMS today. These relational DBMSs organize data into separate structures called tables, which can be linked via common information to make data storage more efficient. A DBMS is like a traditional filing system in that it stores individual groups and pieces of information. Like a filing system, a DBMS consists of separate components, like the cabinet, drawers and folders. A relational DBMS has the following basic components:

- database - the complete collection of information
- tables - a group of data items with a common theme
- records - an individual data item
- fields - a separate piece of information which describe the data item

A relational database consists of a collection of tables, each of which is assigned a unique name. A row in a table represents a relationship among a set of values.

Consider the EMPLOYEE table as under.

<table>
<thead>
<tr>
<th>ENAME</th>
<th>ENO</th>
<th>ESAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAMANA</td>
<td>CM-001</td>
<td>6000</td>
</tr>
<tr>
<td>NAIDU</td>
<td>CM-002</td>
<td>5000</td>
</tr>
<tr>
<td>SURESH</td>
<td>CM-003</td>
<td>4500</td>
</tr>
</tbody>
</table>

The Employee Table

The above table has three column headers namely, ENAME, ENO AND ESAL. These headers are referred as attributed and for each attribute, there is a set of permitted values, called the domain of that attribute. For the attribute ENAME, the domain is the set of all employee-names.

The relational database structure consists of three layers namely, Logical layer, Physical layer and Native layer.

Physical Layer: is the layer in which database tables are created in two dimensional form i.e in the form of form of columns and rows. In a database
Logical layer: This layer is constructed with the logical tables or derived tables.

Native layers: Native layer is the innermost layer in the relational database structure. This is the layer which maintains the physical database in the form of files.

The Relational Database Management (RDBMS) is based on Relational Model. This supports relational databases i.e. the databases which are represented in the form of TABLES. The main objective of relational database design is to generate group of relation schemes which allows the users to retrieve information easily without any redundancy.

Inside a relational database, is stored in the form of tuples and attributes, which ensures data integrity. The organization of data in the form of tuples and attributes also offers complete flexibility in the database design.

Example: Employee:

Example of Employee table of information in relational database

<table>
<thead>
<tr>
<th>Eno</th>
<th>Ename</th>
<th>Design</th>
<th>Salary</th>
<th>Dno</th>
<th>Dname</th>
<th>Joindate</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>Naresh</td>
<td>Clerk</td>
<td>3000</td>
<td>3</td>
<td>Sales</td>
<td>20-jul-12</td>
</tr>
<tr>
<td>30</td>
<td>Suresh</td>
<td>Cashier</td>
<td>5000</td>
<td>5</td>
<td>Servicing</td>
<td>10-aug-12</td>
</tr>
<tr>
<td>50</td>
<td>Kumar</td>
<td>Supdt</td>
<td>7000</td>
<td>1</td>
<td>Marketing</td>
<td>03-oct-10</td>
</tr>
<tr>
<td>25</td>
<td>Amar</td>
<td>Programmer</td>
<td>5000</td>
<td>2</td>
<td>Computer</td>
<td>16-mar-10</td>
</tr>
</tbody>
</table>

The table gives the information about a list of employees working in an organization. Every employee number (ENO) corresponds to a particular employee in the organization. ENO, ENAME, DESGN, SDLARY, DNO, DNAME and JOINDATE are called the columns (attributes) of the table EMPLOYEE. For each of the above attributes, there is a set of values, which is known as the DOMAIN of the attribute.

For example, for the DESGN attributes, the domain is the set of all designations of employees and for DNAME attribute, the domain is the set of all department names and so on.
In the above relation EMPLOYEE, there are four tuples (Rows), each row identifying the details of a particular employee with respect to attributes.

For example, the first row specifies that employee number is 20, named Naresh, who is clerk in the Sales department earning 3000 and joined the organization on 20th July 1995.

**Tuple:** “Row of relation (Table) is referred as tuple. Tuple having a set of ‘n’ numbers of attributes are called as n-tuple.

**Domain:** The values for an attribute or a column are drawn from a set of permitted values known as a Domain. The domain of an attribute contains the set of values that the attribute may assume. In the relation model, no two rows of relation are identical and the ordering of rows is not significant, the domain of Die is 1, 2, 3, 4, 5 and 6. Similarly, domain of coin is Head or Tail.

**Degree of a table:** A relationship’s degree indicates the number of associated entities or participants. A **unary relationship** exists when an association is maintained within a single entity. A **binary relationship** exists when two entities are associated. A **ternary relationship** exists when three entities are associated. Although higher degrees exist, they are rare and are specifically named.

Unary relationship: Course ———- participant

Binary Relationship: Lecturer ——— Batches ——— Class

Ternary relationship: contributor — CFR —— recipient

Fund

An attribute takes a null value when an entity does not have a value for it. The null value may indicate “not applicable” – that is, that the value does not exist for the entity.

### 3.2 Attributes - types of attributes

Each entity has certain characteristics known as attributes. For instance, the student entity might include the following attributes, Student name, Roll Number etc. For each attribute, there is a set of permitted values, called the **domain**, or value set, of that attribute. An attribute of an entity set is a function that maps from the entity set into a domain. Since an entity set may have several attributes, each entity can be described by a set of (attribute, data value) pairs, one pair for each attribute of the entity set.
The attributes can be classified into:

1. Simple attributes
2. Complex/composite attributes
3. Single-valued attributes
4. Multi-valued attributes
5. Derived attribute
6. Null attribute

An attribute, as used in the E-R model, can be characterized by the following attribute types.

**Simple** attributes: The attributes have been simple; that is, they cannot be subdivided into parts. Example: Age, Sex etc.

**Composite** attributes. The attributes, which can be subdivided into subparts. Example: Student Name, which can be divided into First name, Middle name, and Last name. Note also that a composite attribute may appear as a hierarchy. In the composite attribute address, its component attribute street can be further divided into street_number, street_name, and Door_number etc.

![Composite Attribute Diagram]

**Note** also that a composite attribute may appear as a hierarchy. In the composite attribute address, its component attribute street can be further divided into street_number, street_name, and apartment_number.

**Single-valued attributes**: The attribute which contains/accepts only one value/character.

**Example**: Sex: Male or Female  Marital status: Married or Unmarried
Multivalued attributes. The attributes which has set of values for a specific entity, in our example all have a single value for a particular entity.

Example 1: Number of dependents in a family may be 0, 1, 2, 3, 4, ….

Example 2: A student may have several phone numbers, and different students may have different numbers of phones.

Derived attributes: The value of this type of attribute can be derived from the values of other related attributes or entities. The value of a derived attribute is not stored but is computed when required.

Null Attributes: An attribute takes a null value when an entity does not have a value for it. The null value may indicate “not applicable” – that is, that the value does not exist for the entity.

3.3 Keys

We must have a way to specify how entities within a given entity set are distinguished. Conceptually, individual entities are distinct; from a database perspective, however, the difference among them must be expressed in terms of their attributes.

The values of the attribute values of an entity must be such that they can uniquely identify the entity. In other words, no two entities in an entity set are allowed to have exactly the same value for all attributes.

A key allows us to identify a set of attributes that suffice to distinguish entities from each other. Keys also help uniquely identify relationships, and thus distinguish relationships from each other.

The keys can be categorized into:

1. Super Key: A super key is a set of one or more attributes that, taken collectively, allow us to identify uniquely an entity in the entity set. For example, the ‘student_id’ attribute of the entity set student is sufficient to distinguish one student entity from another. Thus, ‘student_id’ is a super key.

2. Candidate Key: A minimal super key. A super key that does not contain a subset of attributes, that is itself super key.

3. Primary Key: A Candidate key selected to uniquely identify all other attribute values in any given row. It cannot contain NULL entries. The primary key of a relational table uniquely identifies each record in the table.

4. Secondary Key: An attribute (or) combination of attributes used strictly for data retrieval purposes.
5. **Foreign key** :- An attribute or Combination of attributes in one table whose values must either match the primary key in another table or be NULL.

### 3.4 Query Languages

A Query Language is language in which a user requests information from the database. These languages are typically of level higher than that of a standard programming language.

Languages can be classified into the following two types.

1. Formal Query language
2. Commercial Query language

#### 3.4.1 Formal query languages

- Formal Query Languages are formal in the sense that they lack of ‘syntactic behavior’ of commercial query languages. These can be used for extracting data from the database. These can be further classified as procedural or non-procedural.

In Procedural Language, the user instructs the system, to perform a sequence of operations on the database to compute the desired result. In a non-procedural language, the user describes the information desired without giving a specific procedure for obtaining that information.

Some of the formal Query Languages are listed below:

- The Relational Algebra
- Tuple Relational Calculus
- Domain Relational Calculus

The relational algebra is a formal Procedural Query language. It consists of a set of operations that take one or two relations as input and procedure a new relation as their result. The fundamental operations in the relational algebra are SELECT, PROJECT, UNION, SET DIFFERENCE, CARTESIAN PRODUCT AND RENAME, etc. In addition to other operations namely, SET INTERSECTION, NATURAL JOIN, DIVISION and ASSIGNMENT.

The SELECT, PROJECT and RENAME operations are called UNARY Operations, because they operate on ONE REALTION. The Other operations operate on pairs (two or more) of relations and are called BINARY OPERATIONS.
3.4.2 Commercial query languages: Commercial Query Languages are needed for the commercial database systems. These languages are more user-friendly. Commercial query languages, besides querying a database, includes features for defining the structure of the data, for modifying data in the database and for specifying security constraints.

Some of the commercial Query languages are listed below

- SQL (Structured Query Language)
- QBE (Query – by – Example)
- Quel (Query Language)

Commercial Query Languages possess such much “syntactic behavior compared to formal query languages. Commercial data languages include construct for update, Insertion and deletion of information as well as for queries to the data bases.

Differences between formal and commercial languages

<table>
<thead>
<tr>
<th>Formal Query Language</th>
<th>Commercial Query Language</th>
</tr>
</thead>
<tbody>
<tr>
<td>These languages lack of syntactic behavior of other languages.</td>
<td>These languages possess syntactic behavior like high level languages.</td>
</tr>
<tr>
<td>These are not used by commercial database systems.</td>
<td>These are the languages used by the commercial database systems.</td>
</tr>
<tr>
<td>The user can query the tables, insert new tuples, delete tuples, and update/modify tuples, using these languages.</td>
<td>These languages are used to query a database, defining the structure of data modifying data for specifying security constraints.</td>
</tr>
<tr>
<td>Examples: Relational Algebra, Tuple Relational Calculus, Domain Relational Calculus</td>
<td>Examples: SQL, QBE, Quel</td>
</tr>
</tbody>
</table>

3.5 CODD rules

Edgar F. Codd, proposed thirteen rules (numbered zero to twelve) and said that if a Database Management System meets these rules, it can be called as a Relational Database Management System. These rules are called as Codd’s 12 rules. Hardly any commercial product follows all.
Rule Zero

- The system must qualify as relational, as a database, and as a management system. For a system to qualify as a relational database management system (RDBMS), that system must use its relational facilities (exclusively) to manage the database.

The other 12 rules derive from this rule. The rules are as follows:

**Rule 1**: The information rule:

All information in a relational database (including table and column names) is represented in only one way, namely as a value in a table.

**Rule 2**: The guaranteed access rule:

All data must be accessible. This rule is essentially a restatement of the fundamental requirement for primary keys. It says that every individual scalar value in the database must be logically addressable by specifying the name of the containing table, the name of the containing column and the primary key value of the containing row.

**Rule 3**: Systematic treatment of null values:

The DBMS must allow each field to remain null (or empty). Specifically, it must support a representation of “missing information and inapplicable information” that is systematic, distinct from all regular values (for example, “distinct from zero or any other number”, in the case of numeric values), and independent of data type. It is also implied that such representations must be manipulated by the DBMS in a systematic way.

**Rule 4**: Active online catalog based on the relational model:

The system must support an online, inline, relational catalog that is accessible to authorized users by means of their regular query language. That is, users must be able to access the database’s structure (catalog) using the same query language that they use to access the database’s data.

**Rule 5**: The comprehensive data sublanguage rule:

The system must support at least one relational language that

1. Has a linear syntax
2. Can be used both interactively and within application programs,
3. Supports data definition operations (including view definitions), data manipulation operations (update as well as retrieval), security and integrity
constraints, and transaction management operations (begin, commit, and rollback).

**Rule 6:** The view updating rule:

All views that are theoretically updatable must be updatable by the system.

**Rule 7:** High-level insert, update, and delete:

The system must support set-at-a-time insert, update, and delete operators. This means that data can be retrieved from a relational database in sets constructed of data from multiple rows and/or multiple tables. This rule states that insert, update, and delete operations should be supported for any retrievable set rather than just for a single row in a single table.

**Rule 8:** Physical data independence:

Changes to the physical level (how the data is stored, whether in arrays or linked lists etc.) must not require a change to an application based on the structure.

**Rule 9:** Logical data independence:

Changes to the logical level (tables, columns, rows, and so on) must not require a change to an application based on the structure. Logical data independence is more difficult to achieve than physical data independence.

**Rule 10:** Integrity independence:

Integrity constraints must be specified separately from application programs and stored in the catalog. It must be possible to change such constraints as and when appropriate without unnecessarily affecting existing applications.

**Rule 11:** Distribution independence:

The distribution of portions of the database to various locations should be invisible to users of the database. Existing applications should continue to operate successfully:

1. when a distributed version of the DBMS is first introduced; and
2. when existing distributed data are redistributed around the system.

**Rule 12:** The non subversion rule:

If the system provides a low-level (record-at-a-time) interface, then that interface cannot be used to subvert the system, for example, bypassing a relational security or integrity constraint.
Short Answer Type Questions

1. What is a Relational Database?
2. What is Domain in Relational Table
3. What is Degree of able on Relational Model
4. What is Tuple?
5. What is super key?
6. What is Candidate key?
7. What is Primary Key?
8. What are the Formal Query Languages?
9. What are the Commercial Query Languages?
10. What are fundamental operations in Relational Algebra?
11. What are Unary operations?
12. What are Binary operations?

Long Answer Type Questions

1. Explain about Relational Data Model
2. What is an attribute? Write about types in it.
3. What is Key? Write about types of Keys.
4. Write the Differences between Formal and Commercial Languages
5. What are the Codd rules in Relational Model
6. Write short notes on Fundamental Operations in Relational Algebra.
UNIT 4

SQL & PLSQL

Structure

4.0 Introduction
4.1 Data type in SQL
4.2 DDL, DML and DCL commands
4.3 Set operator and joins
4.4 Sub queries and data base object
4.5 Introduction to PLSQL
4.6 Data base Triggers

Learning Objectives

After studying this unit, student will able to learn

• Different types of data types in SQL
• Understand DDL, DML and DCL commands
• Understand different set operator and joins
• Understand sub queries, data base object, PL/SQL and Triggers.

4.0 Introduction

SQL was invented and developed by IBM in early 1970s. SQL stands for Structured Query Language. IBM was able to demonstrate SQL, which could be used to control relational database.
The SQL implemented by ORACLE Corporation is 100% compliant with the ANSI/ISO standard SQL data language. Oracle’s database language is SQL, which is used for storing and retrieving information in Oracle. A table is a primary database object of SQL that is used to store data. A table holds data in the form of row and columns.

**To communicate with the database, SQL supports the following commands**

- **Data definition language**: Create, alter, drop, grant, and revoke commands.
- **Data manipulation language**: Insert, select, delete and update commands.
- **Transaction control language**: Commit, save point and rollback commands.

**The following are the benefits of SQL**

- Non-procedural language, because more than one record can be accessed rather than one record at a time.
- It is the common language for all relation database (i.e) portable and it requires only small modification to make use of in other database.
- Very simple commands for querying, inserting, deleting and modifying data and objects.

**Oracle Internal Data type**

In order to create a table we need to specify a datatype to individual columns in the `create table` command. In order to fulfill the above requirement, Oracle supports the following datatype.

**Character Data type**

The following are the character datatype supported by Oracle.

**Char data type**

This datatype is used when a fixed length character string is required. It can store alphanumeric values. The column length of such a datatype can vary between 1-255 bytes. By default it is one byte.

- If the user enters a value shorter than the specified length then database valuable blank-pad to the fixed length.
• In case if the use enters a value larger than the specified length then the database would return an error.

**Varchar 2 data type**

It supports a variable length string. It also alpha numeric values. Maximum size for this datatype would be from 1-2000 bytes. While defining this datatype we should specify the size. Using varchar 2 we can save disk space when compared to char.

This statement can be justified with the help of an example. Considering a column assigned with varchar2 datatype of size 30 bytes if there enters 10 bytes of character, then the column length in that row would only be 10 bytes and not 30 bytes. In the case of char, it would occupy 30 bytes because the remaining would be blank padded by oracle.

**Long data type**

This type is used to store variable character length. Maximum size is 2 GB. Long datatype has several characters similar to varchar2 datatype. It’s length would be restricted based on the memory space available in the computer. The following restrictions needs to be fulfilled when a long datatype attribute is cast on a column in a table.

• Only one column in a table can have long datatype, which should not contain unique or primary key constraint.

• The column cannot be indexed.

• Procedures or stored procedures cannot accept long datatype as arguments.

**Number data type**

This datatype can store positive number, negative number, zeroes, fixed point number floating point number of magnitude ranging between $1.0 \times 10^{130}$ to $9.9 \times 10^{125}$ with a precision of 38.

• Column name number { p = 38, s = 0 }

• Column name number { fixed point }

• Column name number { floating point }

Where P is the precision which refers to the total number of digits, it varies between -38, s P is the scale which refers to number of digits to the right of the decimal, point which varies between -84 to 127.
4.1 Data types

Datatype is used to store data and time in a table. Oracle database makes use of its own format to store data in fixed length of 7 bytes each for century, month, day, year hour, minute and second. Default data datatype is ‘dd-mm-yr’. To view system’s data and time we can use the SQL function sysdate(). Valid date is from Jan 1, 4712 BC to Dec 31, 4712 AD.

Raw size

Raw datatype is used to store byte oriented data like binary data or byte string and the maximum size of this datatype is 255 bytes. While using this datatype the size should be mentioned because by default it does not specify any size. Raw datatype can be indexed.

Long raw

Long datatype is used to store binary data of variable length, which can have a maximum size of 2GB. This datatype cannot be indexed. Further all limitations faced by long datatype also holds goods for long raw datatype.

4.2 DDL, DML and DCL commands

Introduction to data definition language

The data definition language is used to create an object (e.g. table) alter the structure of an object and also to drop the object created. Now let us took at the concepts related to data definition language.

Table Definition

A table is a unit of storage which holds data in the form of rows and columns. The data definition language used for table definition can be classified into the following four categories.

• Create table command.
• Alter table command
• Drop table command

We will go about explaining the above classified commands by giving the syntax followed by an example one by one. Initially to work with oracle database it should contain at least one table. So let us now see how to create a table.

Command to create a table

Create table <table name> (column definition 1, column definition 2...);
In a table

• We should specify a unique column name.

• We should specify proper datatype along with its width.

• We can include ‘Not null’ condition when needed, by default it is ‘Null’.

The following example illustrates how to create a table with custid, name, address and city as its column.

Example

SQL > create table customer (custid number (6) name characteristics (45) address varchar2 (30) city character (30), rapid number (2) credit limit number (8,21);

If the above command is executed successfully the message table created is displayed.

The table name in the above example should adhere strictly to the following norms.

• While naming a table the first letter should be an alphabet.

• Oracle reserved words cannot be used to name a table.

• Maximum length for table name is 30 characters.

• Two different tables should not have the same name.

• Underscore, numerals and letters are allowed but not blank space and single quotes.

If the users uses double quotes for naming his table like ‘inf’ then upper and lower case are not equivalent. For instance ‘inf’ ‘INF’ ‘and ‘inF’ are not same.

So far we learn how to create a table. After creating a table if the sue feel like adding a column or modify the existing column definition he can do so by using the alter table command.

Command to alter a table

Alter table <table name> modify (column definition...);

Alter table <table name> and (column definition....);
Alter table commands creates to the need of the following situations

- When a user wants to add a new column.
- When user wants to modify the existing column definition say from Null to Not null or to change the width of the datatype or datatype itself.
- To include to drop integrity constraints (integrity constraints are dealt in the later part of this session).

The following two examples reinforce the above concept

We proceed in the following way to modify the width of an existing column custid from number (6) to number (6) belonging to customer table.

Example

SQL > alter table customer modify (custid number (8));

The following example explains how to add a column type long to the customer table.

Example

SQL > alter table customer add (comments long);

In order to drop the above table, one can use the drop table command supported by SQL.

Command to drop a table

Drop table <table name>

Example

SQL> Drop table customer;

If this statement is successfully executed then the message ‘table dropped’ would be displayed.

The following command would delete all the record from the table.

Command to truncate a table

Truncate table <table-name>;

Example

SQL > truncate table customer;
Implementation of this command would delete all the rows associated with the table, only the structure of the table remains.

If the user wants to view the structure of the table, the following command helps to achieve the same.

**Command to view the table’s structure**

```
Desc <table name>;
```

**Example**

```
SQL > Desc customer
```

The above command will display the structure of the customer table.

**Integrity Constraints**

An integrity constraint is a mechanism used by Oracle to prevent invalid data entry into the table. It is nothing but enforcing a rule for the column in a table. The following are the various types of integrity constraints.

**Domain integrity constraints**

Maintains value according to the specification like ‘Not null’ condition, so that the user has to enter a value for the column on which it is specified ‘Not null’ and check constraints fall under this category.

**Entity integrity constraints**

Maintains uniqueness in a record.

**Referential integrity constraint**

Enforces relationship between tables.

One can either enable or disable a constraint. The former enforces the constraints and the latter will not enforce the rule, even though the syntax would as it is in the data dictionary. By default the syntax would be enabled. Further we can define a constraint either at table or column level. If it is defined at the table level then, it can be enforced to any number or columns in a table. On the other hand, if it is defined at the column level, then it holds good only for the column for which it is defined.

**Domain integrity - ‘Not null’ constraints**

We know that by default all columns in a table allow null values. When a ‘Not null’ value
Example

SQL > create table customer (custid number (6) constraints cust not null name varchar (30));

Where cust is the constraints name

This command will ensure that the enters a value for the custid column on the customer table, failing which it returns an error message.

Check Constraints

There are rules governed by logical expressions or Boolean expressions. Check conditions cannot contain subqueries. The following example will help us to understand it much better.

The following example creates a table order_info with check constraints to restrict the values of ordid to be within 100.

Example

SQL > create table order_info (ordid number (4) constraints check it check (ordid >100), orderdate date, complain character (1), custid number (6) shipdate date, total number (8,2));

Entity Integrity Constraints

An entity is any data recorded in a database. Each entity represents a table and each row of a table represents an instance of that entity. To identify each row in a table uniquely we need to use this constraint.

Unique Constraints

Usage of the unique key constraint is to prevent the duplication of values with the rows of a specified column or a set column in a table. Columns defined with this constraint can also allow null values. If a unique key constraint unique key. In the above case, combination of columns be duplicated. Maximum combination of columns that a composite unique key can contain is 16.

Consider the following example in which a constraint is enforced in a column level.

Example

SQL > create table price (prodid number (6), stdprice number (8,2), minprice number (8,2), startdate date, enddate date constraint unidate unique));
The same constraint can be enforced using table level syntax also

**Example**

SQL > create table price (prodid number (6), stdprice number (8,2), enddate date, startdate date, constraints unitab unique (enddate));

The above example prevents the duplications of values to be entered in enddate column.

Example for composite unique key follows

**Example**

SQL > create table customer1 (custid number (6), name varchar2 (30), constraints compuni unique (custid name));

In this case combination of columns custid and name cannot be duplicated.

**Primary Key Constraints**

This constraint avoids duplication of rows and does not allow Null value, when enforced in a column or set of columns. As result it is used to identify a row. A table can have only one primary key. If a primary key constraint is assigned to more than one column (i.e) or combination it is said to be composite primary key, which can contain maximum of 16 columns.

**Example**

SQL > create table customer (custid number (6) constraints prim_con primary key, name, char (45), address varchar2 (30), city char (30), rapid number (4), credit limit number (8,2));

The same constrains can be enforced using table level also.

**Example**

SQL > create table item (itemid number (6), ordid number (6), prodid number (6), actual price number (8,2), qty number (8,2), constraints prim_id primary key (itemid));

The following examples illustrate composite primary key

**Example**

SQL > alter table order_info add primary key (ordid, orderdate) disable;
The above command ensures that no two rows in the table have a same value for both ordid and orderdate. Since the constraint is disabled, it can only be defined in the Oracle database dictionary but not enforced.

Referential integrity constraints

To establish a ‘parent-child’ or ‘master-detail’ relationship between two tables having a common column, we make use of referential integrity constraints. To implement this, we should define the column in the parent table as primary key and the same column in the child table as foreign key referring to the corresponding parent entry.

Basic concepts related to referential integrity are:

- **Foreign**: A column or combination of columns included in the definition of referential integrity which would refer to a referenced key.

- **Referenced key**: It is a unique or a primary key which is defined on a column belonging to the parent table.

- **Child table**: This table depends upon the values present in the referenced key of the parent. Table which is referred by a foreign key.

- **Parent table**: This table determines whether insertion or updation of data can be done in child table. This table would be referred by child’s table foreign key.

Example

SQL > create table order_info (ordid number (4), orderdate, complain character (1), shipdate date, total number (8,2), custid number (6) constraints fk_custid references customer (custid));

The above command creates a table order info and will also enable a foreign key on custid column which would refer to the primary key on the custid column of the customer table.

Before enabling this constraint, we ensure that custid column of the customer table has been defined with either unique key or primary key constraint. The foreign key constraint ensures that all values in the column custid in order_info table have a corresponding custid value in the customer table.
Example

SQL > create table order_info2 (ordid number (4), orderdate date, 
complan character (1), shipdate date, total number (8,2), 
custid number (6), constraint fk_custid foreign key (custid) 
reference customer (custid));

On delete cascade clause

If all row under the reference key column in a parent table are deleted, 
then all rows in the child table with dependant foreign key column will also be 
deleted automatically. The following example illustrates the same concept for 
better understanding.

Example

SQL > create table oder_info3 (ordid number (4), orderdate date, 
complan character (1), ship_date date, total number (8,2), 
custid number (6), constraint fk-custid references customer 
(custid) on-delete-cascade);

Because of the on-delete cascade option we find that deleting a 
particular custid value from the customer result in the deletion of the 
corresponding rows from the order table also. For example, if custid 123 is 
deleted from customer table, then, the dependent rows in the order table 
also get deleted.

Referential integrity constraints with composite key

A composite foreign key can contain a maximum of 15 columns. It 
must either refer to a composite unique key or a composite primary key. 
Further to maintain referential integrity constraint with foreign keys, each row 
in the child table must satisfy one of the conditions mentioned below.

• The value of atleast one of the column, which make up the foreign 
key can be Null.

• The values of foreign key columns must match the values of 
referenced key columns.

Data Manipulation Commands (DML)

Data manipulation commands are the most frequently used SQL 
commands and they are as follows.

• Insert

• Select
Let us discuss about these commands in details

**Insert Commands**

The insert command is used to add one or more rows a table. While using this command the values are separated by commands and the datatype char and date are enclosed in apostrophes. The values must be entered in the same order as they are defined in the table.

```
Insert into <table_name> values (a list of data values)
```

The following examples illustrate the insert command.

**Example**

```
SQL > Insert into order_info values (123, '12-Jan-74', 'b', 5, '19-Jan-74', null);
```

The following example illustrates concepts of inserting more than one record using a single insert command.

**Example**

```
SQL > Insert into order-info values ( &no, '&ordate', '&comm', &shipdate', &total);
```

If a user wants to skip any one of the fields then he can enter null against the column value. In example 1, null has been inserted for the column total.

**Inserting data values**

The following example introduces a new function called to date function to insert date values other than the standards format for date.

**Example**

```
SQL > Insert into order_info values (122, '7 Jan 88', 'b', to_date ('14/2/88 98 : 30', 'dd/mm/yy hh : :mi'), 9798);
```

We know that the standards format for the datatype of ‘DD/MON/YR’ in the table order_info for columns like ord-date and ship date when the user enters a date with a format say ‘MM/DD/YY’ then it can be converted to the standards data format with the help of the TO-Date function.
Date forming function TO-DATE consists of two arguments, the first argument specifies the char value that contains the date and time, and second one specifies the format according to which date and time has to be displayed. SQL* plus interprets char value according to the format, converts it to the standard date value and then inserts it into the table.

**Insert Command with Query**

We can use an insert command along with a query to select row from one table and insert them into another table. Only those columns and rows selected by the query will be inserted.

The following examples copies all the rows from order_info table to ord table, provided a table named ord exist having the same structure as order_info.

**Example**

SQL > Insert into ord (select * from order_info);

**Select command**

To perform a query we use the select command. The query is a request for information. It is the most common database operation used. The syntax for select command is given below:

Select column_name ..... from table_name;

The following examples selects all the columns from the table customer.

**Example**

SQL> select * from order_info;

We can either display all the columns in a table or only specific columns from the table. For example if we want only columns sip date ord date custid from order table, then we include only these columns in the select statement. The order of column name in the select command specifies the order in which they should be displayed.

**Example**

SQL > select shipdate, orderdate, custid from order_info;

The above examples displays ship date first followed by ord date and suctis. On the other hand, if we entire the column name in the reverse order, then it would de displays accordingly.
Selecting disjuncts rows

As we know how to select rows or specific columns from table let us go about discussing how to prevent the selection of duplicate rows. To prevent the selection of duplicate rows we include distinct clause in the select command. The following example would eliminate duplicate values present in repid column of the customer table.

Example

SQL > select distinct repid from customer;

Select command with ‘where’ clause

To select specific rows from a table include where clause in the select command. It can appear only after the from clause. We can retrieve only the rows which satisfy the where condition. To arrange the displayed rows according to some pre-defined order we can use the order by clause. It is also used to arrange rows in descending or ascending order. The order by clause can also be used to arrange multiple columns. The syntax for the select command along with the where clause is given below.

Select column....... from table where conditions [order by];

In order to retrieve rows from the customer table for which repid equals 20 and to order the column name in ascending order, the following command is used.

Example

SQL > select custid, name from customer where repid = 20 order by name;

The above where clause can also use a character or date value. In such a case they should be enclosed within parenthesis.

Example

SQL > select * from customer order by name, creditlimit desc;

Select command to create a table

We can also create a table and copy the records into it with a single statements, by including select clause in create table command. The following example will create a new table cust form the existing table customer along with its records.
Example

SQL > create table cust as select * from customer;

Update command

Update command is used to alter the columns values in a table. The update command consist of a set clause and an optional where clause. The syntax for the update command is given below

Update table_name set field = value, ...... where conditions;

In the above syntax the where clause and the set clause can also include queries update sets each field with the value that we supply, provided it satisfies the where condition. In the following example a query has been included in the where clause to select the rows which are to be updated.

In the following example the custid in the customer table is incremented if the same is present in order_info table.

Example

SQL > update item set actual price = 100.50 where prodid in (select prodid from product);

The result of the above command can be verified by selecting rows from the customer table.

Delete command

After inserting rows in a table we can also delete them if require. The delete command consist of a form clause followed by an optional where clause.

Delete from <table_name> where conditions;

To delete several rows from a table, select the rows with appropriate conditions in a ‘where’ clause. The ‘where’ clause can also include a query. The following example will delete a row from customer whose comm_type is ‘b’.

Example

SQL > delete from customer where custid in (select custid from order_info where complain = ‘b’);

Operators in SQL * Plus

The following are the operators supported by SQL * Plus

• Arithmetic operators
• Comparison operators
• Logical operators

**Arithmetic operators**

To perform calculations based on number, values we include arithmetic expressions in SQL commands. An arithmetic expression consist of column names with number datatype and an arithmetic operator connecting them. The arithmetic operator are (addition) + (subtraction) - (multiplication) and (division). Consider the following examples.

**Example**

SQL > select prodid, std price, mini price, std price + miniprice from price where enddate = ‘4 Jan 89’;

We know the std price miniprice is not a column in the table price, yet SQL * plus would display it as a separate column. Arithmetic can also be performed in a where clause.

If there are several operator in an arithmetic expression then the precedence of each operator must be known and have equal higher precedence whereas + and - have equal lower precedence.

An example given below illustrates the precedence of operators.

**Example**

SQL > select ordid, itemid, 100 * (actual price + qty) from item where prodid = 111;

In the above example only after adding actual price with qty it is multiplied by 100. Parenthesis is omitted then multiplications will be performed first followed by addition. Thus we can control the order of evaluations by using parenthesis.

We now move on to study about comparison operators.

**Comparison operators**

Comparison operators are used in conditions to compare one expression with another. The comparison operators are =, !=, <, >, <=, >=, between (to check between any two values) in (to match with any of the values in the list) like (to match a character pattern and is null (to check whether it is null).

The last four operators mentioned above can also be used for checking the NOT conditions like NOT BETWEEN, NOT LIKE and so on. The following examples illustrates the comparison greater than.
Example

SQL > select * from order_info where total > 7000;

The above examples display the rows for which the total is greater than 7000. An example discussed below is used to check for negations (i.e. check for NOT conditions)

Example

SQL > select * from order_info where not (custid = 2 or custid = 4);

The above command will display all columns where custid is neither 2 nor 4. The IN operator can be sued to select rows that match one of the values in a list.

Example

SQL > select * from order_info where shipdate in ('12 January 88', '30-Jan 67');

The above examples list all the columns in a table for which ship date lies in the given list. When we search for character values using IN operator, the column name must exactly match with the values present in the list. In the case of LIKE operator which is used to search a character pattern, we need not know the exact character value. The LIKE operator recognizes special characters like % and _. The former can match zero or more character while the later matches exactly one characters.

Example

SQL > select name, address, city from customer where name like 'v%';

The above command will display the columns name address, city and phone whose name begins with 'v'.

Example

SQL > select * from customer where name like 'V_P';

The above command lists all the columns from customer table whose names are three letters, starting with 'V' and ending with 'P'.

Logical Operator

A logical operators is a used to combined the result of two condition as to produce a single result. The logical operators are AND, NOT and OR.
The following exemplified and AND operator which displays all the columns from order_info table provided both the conditions mentioned below are satisfied.

**Example**

```
SQL > select * from order_info where shipdate > '4-Jan-11 AND total < 9000;
```

We have now gone through the various operators supported by SQL.

* Plus let us have a look at the order in which these operators are evaluated.

**Operator Procedure**

The precedence of the operators discussed above are given below

- Arithmetic operators - Highest precedence
- Comparison operators
- NOT logic operator
- AND logic operator
- OR logic operator - lowest precedence

**Privilege Commands**

Privilege is the right to access another user’s object (Table, view). We can grant privileges (insert, select,...) to others and can also withdraw the granted privilege by using the privilege commands - grant and revoke.

**Grant Privilege command**

For example, if a user creates a table named customer, there is no necessity for the user to be given any privilege to use it. He becomes the owner of the table. In case, the user wants to share an object with others, the appropriate privileges can be granted on that particular object to others. Objects are logical data storage structures like tables, views, sequences, indexes, synonyms etc.

Object privileges can be granted to others using the SQL command GRANT. The syntax is given below.

```
Grant privileges on <object-name> to <username>;
```

The following example grants privileges like select, insert, delete to a single user. We can also specify ALL to grant all the privileges.
Example

SQL > grant select, insert, delete on costumer to accounts;

After successful execution of the above command, the message “grant succeeded” will be displayed.

The following example grants ‘insert’ privilege on columns ship_date and total to production.

Example

SQL > grant insert (shipdate, total) on order_info to production;

The following example grants the select privilege on customer table to adams along with “with grant option”, so that the adams can pass the respective privileges to others.

Example

SQL > grant select on customer to adams with grant option;

Revoke privilege command

To withdraw the privilege which has been granted to a user, we use the revoke command. This command is closely similar to that of the grant command in its formats. The syntax is given below.

Revoke privileges on <object-name> from <username>;

To withdraw select, insert and delete privileges on the object “Customer” from a user, we can use the following command.

Example

SQL > revoke select, insert, delete on customer from mktg;

After successful execution of the above command, the message “revoke succeeded” will appear.

Transactions Control Commands

A transaction is a logical unit of work. All changes made to the database between commit and/or rollback operations can be referred to as a transaction. Transaction changes can be made permanent to a database only if they are committed. A transactions begins with an executable SQL statement and ends explicitly with either rollback or commit statements and implicitly, i.e. automatically, when a DDL statement is used.
Commit

This command is used to end a transaction. Only with the help of the commit command transaction changes can be made permanent to the database. This command also erases all save points in the transactions thus releasing the transaction locks. The syntax is given below.

commit work; or
commit;

Rollback

A rollback command is used to undo the work done in the current transaction. We can either rollback the entire transaction so that all changes made by SQL statements are undone, or rollback a transaction to save point so that the SQL statements after the save point are rolled back. The syntax is given below. To rollback the entire transactions, we give

rollback work; or
rollback;

To rollback to a particular stage in a transaction i.e. a save point, we say

rollback to save point save_pt;

where save_pt is the save point.

Save point

Save points are like markers to divide a very lengthy transactions to the smaller ones. They are used to identify a point in a transaction to which we can later rollback. Thus save point is used in conjunction with rollback, to rollback portions of the current transactions.

save point savepoint_id;

The following example is illustrative of the above two concepts.

Example

SQL > update order_info set total =7100 where ordid =400;
      save point Jan;
      update order_info set total =7400 where ordid =600;
      save point Feb;
      select sum (total) from order_info;
rollback to save point jan;
update order_info set total=7500 where ordid=600;

In the above example, we have two markers Jan and Feb and a rollback statement which rolls back the SQL statements to the save point Jan.

4.3 SET Operators and joins

Set operators combine the results of two queries into a single one. The following set operators aid SQL in joining queries to retrieve rows. Now let us discuss them in detail.

- Union
- Union all
- Intersect
- Minus

The column in the select statements joined using the set operators should adhere strictly to the norms mentioned below.

- The queries which are related by a set operator should have the same number of columns and the corresponding columns must be of the same datatype.
- Such a query should not contain any column of type long.
- The label under which the rows are displayed are those from the first select statement.

**Union**

The union operator returns all distinct rows select by either query. The following example combines the results of two queries with the union operator which eliminate duplicate rows.

**Example**

SQL > select custid from customer union select ordid from order_info;

**Union all**

The ‘union all’ operator returns all rows selected by either query including duplicates. The following example combines the result with the ‘union all’ operator, which does not eliminate duplicate rows.
Example

SQL > select custid from customer union all select ordid from order_info;

Intersect

Intersect operator returns only rows that are common to both the queries.
The following example is illustrative of the above statement.

Example

SQL > select ordid from order_info intersect select itemid from item;

Minus

Minus operator returns all distinct rows selected only by the first query
and not by the second. The following example illustrates this.

Example

SQL > select ordid from item minus select ordid from order_info;

while using the ‘order by’ clause, it must follow the last select statement
and we must order by integer and not by column name. Consider the following
example.

Example

SQL > select ordid, custid from order_info union select itemid, prodid
    from item order by 2;

The above example displays distinct rows selected by either query,
ordered by the second column stated in both the queries. Since the column
names are different in the above queries, we use an integer in the ‘order by’
clause instead of a column name.

Relating data through join concept

The purpose of a join is to combine the data spread across tables. A
join is actually performed by the ‘where’ clause which combines the specified
rows of tables. The syntax for joining tables is follows.

Select columns from table1, table2, where logical expressions;

The logical expression specifies how the tables are joined. There are
basically three different types of joins. They are

- Simple join
- Self join
• Outer join

Let us discuss them in detail

**Simple join**

It is the most common type of join. It reviews rows from two tables having a common column and is further classified into equi-join and non-join. Let us discuss them in detail.

**Equi-join**

A join which is based on equalities is called an equi-join. Consider the following example.

**Example**

```sql
SQL > select * from item, order_info where item. ordid = order_info ordid;
```

If the above statement, item ordid=info . Ordid perform the joint operation. It retrieves rows from both the tables provided they both have the same ordid as specified the ‘where’ clause. Since the ‘where’ clause uses comparison operator equal to (=), to perform a joint, it is said to be an equi join.

**Non equi-join**

A non-equi join specifies the relationship between columns belonging the different tables by making use of the relation operator (> ,<,<=,>=,<> ) other than =. The following example is illustrative of this.

**Example**

```sql
SQL > select * from customer, order_info where customer custid>
order_info, custid and customer, repid = 10;
```

The following example join the rows of customer table to that of order_info table provided custid’s belonging to customer table are greater than custid’s belonging to order_info table. Yet it retrieves rows only when the second expression, i.e., repid = 10 is true.

**Table aliases**

To prevent ambiguity in a query we include table names in the select statements. Table aliases are used to make table queries shorter and more readable. As a result, we give in alias to the table in the ‘from’ clause and use it instead of the table name throughout the query. The following example is illustrative of the same.
Example

SQL > select c. *, o. * from customer c, order_info where c.custid > o.custid and repid = 10;

The above example is same as example 2 but uses tables aliases, where 'o' refers to order_info table and 'c' refers to customer table. Table aliases which are defined in the 'from' clause are separated from the table name by spaces.

Self join

Joining of a table to itself is known as a self join, i.e., it joins one raw in a table to another. It can compare each row of the table to itself and also with other rows of the same table. The following example will illustrate this concept.

Example

SQL > select o. * from order_info o, order_info u where o.ship-date >= u.ord_date and o.custid = u.custid;

The above example will display only those rows from the order_info table whose ship-date is either greater than or equal to the order-date provided they both have the same custid.

Outer Join

The outer join extends the result of a simple join. An outer join returns all the rows returned by simple join as well as those rows from one table that do not match any row from the other table. The symbol, (+) represent outer join. The following example helps us for better understanding.

Example:

SQL > select order_info. ordid, prodid, item. ordid from item, order_info where order_info. ordid (+) = item. ordid;

The above example will also retrieve rows from item table which does not have any matching records in the order_info table because of the presence of an outer join (+).

Usage of Subqueries

Nesting of queries, one within the other, is termed as subquery. A statement containing subquery is called a parent statement. Subqueries are used to retrieve data from tables which depend on the values in the table itself.
The following example is illustrative of this.

**Example**

```sql
SQL > select repid, address, name from customer
    where name = (select name from customer where repid = 10);
```

### 4.4 Subqueries and Database Object

The above example used a subquery which returned a single value. Subqueries can also return more than one value. In such cases, we should include operators like `any`, `all`, `in` or `not in`, between the comparison operator and the subquery.

The following example illustrates the usage of the `any` operator.

**Example**

```sql
SQL > select name, city, address from customer
    where repid < any (select repid from customer
    where credit limit = 45345);
```

In the above example the subquery will display the area which has `maxcredit` equal to 45345. The main query will display details about customer, if area is lesser than any of the values returned by the subquery.

**Example**

```sql
SQL > select name, city, address from customer
    where repid all (select repid from customer
    where credit limit = 45345);
```

In the above example the subquery will display area which has `maxcredit` equal to 45345. The main query will display details about customer only if area is lesser than all the values returned by the subquery.

### Multiple Subqueries

Oracle places no limit on the number of queries included in a `where` clause.

Consider the following example which illustrates multiple queries.

**Example**

```sql
SQL > select name, repid from customer
```
where repid = (select repid from customer where custid = 3) 
or creditlimit > (select creditlimit from customer where credit = 2);

The above example will select the columns name, repid from customer provided one of the above subqueries is true.

A subquery itself can contain a subquery. The following example is illustrative of this statement.

Example:

SQL > select * from order-info where ordid = (select ordid from order_info where custid = (select custid from order_info where total = 7000));

A subquery can retrieve information from more than one table.

Consider the following example.

Example:

SQL > select name, city, address from customer where custid in (select custid from order_info o, item i where o.ordid = i.ordid);

In the above example the subquery refers to two table, order_info and item. It will display rows from customer table for custid returned by subquery.

Correlated subquery

A subquery is evaluated once for the entire parent statement whereas correlated subqueries is evaluated once per row processed by the parent statement. Consider the following example which returns rows for total which are greater than avg (total) which reference to ordid.

Example:

SQL > select ordid, orderdate, shipdate, from order_info o where shipdate <any (select orderdate from order_info where o.ordid = ordid);

The behavior of a correlated subquery is evaluated as follows

- Ordid of the row is first determined.
- Ordid is then used to evaluate parent query.
- If that row’s shipdate is greater than any of the order dates, the row is returned.
- The supplementary is evaluated once for each row of the order_info table until all the rows of the table have been tested.
More Data Objects

Once of the database objects namely, table was the topic of discussion in previous sessions. In the session we will be discussing the database objects mentioned below

- View
- Synonym
- Sequence
- Index

View

A view is an imaginary table and it contains no data and the tables upon which a view is based are called base tables. The advantages of view are as follows.

They provide table security by restricting access to predetermined set of rows or columns of a table.

They simplify commands for the use because they allow them to select information from multiple tables.

They provides data in a different perspective than that of base table by remaining columns without affecting the base table.

The syntax for creating a view is given below

Create view <view name> (column alias name...) as query (with check option constraint);

Example:

SQL > create view custview (customer, city, representative) as select custid, city,repid from customer where repid = 20 with check option;

In the above example the view only about representative whose repid is 20. Because of the check option we should insert only those customer whose repid is 20.

We all know how to perform manipulations on table data. The same is applicable to views also. Let us discuss them in details.

To display the contents of the view, select statement is used.

Example:
SQL > select * from cust view;

Similarly we can perform updations, deletions and insertions in a view with certain restrictions. To illustrates this, consider the following views which are derived from the same table item.

Example:

SQL > create view ve1 (item, product)
    as select itemid, prodid from item;

Example:

SQL > create view ve2 (quantity price)
    as select qty, actual_price from item.

We can insert rows into both the views, provided other column in the base table accept null values. If those column were assigned as not null, then we cannot insert rows. We can perform updations or deletions on ve1 because itemid is declared as primary key which could be used to identify a row. Only after identifications, the corresponding can be deleted or updated. In the case of ve2 we can perform updation or deletion with some restrictions. We need to specify which row we actually mean by using the where clause. Joining of table is also possible in a view. But we cannot update, delete or insert through a view if it selects columns from more than one table. The following example creates a view from two tables.

Example:

SQL > create view joinview (customername, maximumcredit, total) as
    select name, creditlimit, total from customer, order_info where
    customer. custid = order_info.custid;

In the above example the view join view will contain information about customer and also details about maximum credit than he can avail. Since this information is spread across two table, we use the concept of join to retrieve them.

Function in view

Single row functions comprising number, character, date, group function and expression can also be used in view. Consider the following example which illustrates the usage of expression n view.

Example:
SQL > create view discount (quantity, ordid, price) as select qty, ordid, actualprice/50 from item;

The above example finds the discount offered for a product.

**Synonym**

A synonym is a database object which is used as an alias (alternative name) for a table view or sequence. They are used to

• Simplify SQL statement.
• Mask the name and owner of an object
• Provide public access to an object.

Synonym can either be private or public. The former is created by the use, which is available only to that person whereas the latter is created by the DBA, which can be availed by any database user. The syntax for creating a synonym is a given below.

Create (public) synonym <synonym_name> for <table_name>

(.@database link)

Consider the following example which represents in the table customer with a different name say, cust.

**Example :**

SQL > create synonym cust for customer;

**Sequences**

A sequence is database object which can generate unique, sequential integer values. It can be used to automatically generate primary key or unique key value. A sequence can be either an ascending or a descending sequence. The syntax for creating a sequence is as follows.

Create sequence <seq_name> increment by n start with n (max value n) (min value n) (cycle/noncycle) (cache/noncache);

For creating sequence we have to define the following terms.

Increments by n : ‘n’ is an integer which specifies the interval between sequence number. The default is 1 if n is positive then the sequence ascends and if it is negative the sequence descends.

Start with n : Specifies the first sequence number to be generated.
Minvalue n : Specifies the minimum value of the sequence. By default, it is 1 for an ascending sequence and 10e26-1 for a descending sequence.

Max value : It specifies the maximum value that the sequence can generate. By default it is -1 and 10e27-1 for descending and ascending sequence respectively.

Cycle : Specifies that the sequence continues to generate values after reaching either max or min value.

No cycle: Specifies that the sequence cannot generate more values after reaching either its max or min value. The default value is ‘no’ cycle.

Cache: Allows faster generations of sequence number. Oracle preallocates sequence number of keep than in memory for faster access. The cache values must be less than max value minus min value and by default it is 20.

No cache : The default value ‘noncache’ does not preallocate sequence number for faster access.

Example :

SQL > create sequence custeq
    increment by 1 start with 1
    max value 10
    min values 1
    cycle cache 4;

After creating a sequence we can access it values with the help of pseudo columns like currval and nextval. A pseudo column behaves like a table. We can select values from pseudo columns but cannot perform manipulations on their values.

Nextval : Nextval will return initial value of the sequence when referred to, for the first time. Later reference to nextval will increment the sequence value by INCREMENT BY clause and return the new value.

Currval : Currval returns the current value of the sequence which is the value returned by the last reference to nextval.

The following example uses custeq to insert values in the id column belonging to the temp table created for this example.
Example:

SQL > Create table temp (id number (3), name varchar (15)) ;
SQL > insert into temp values (custeq.nextval, 'vijay') ;

To find out the current sequence number, we issue the following SQL statement.

Example:

SQL > select custeq .currval from temp;

Alter sequence

We can alter the sequence when we want to do the following

• Set or eliminate min or max value.
• Change the increment value.
• Change the number of cached sequence number.

The following example will change the max value of the sequence from 10 to 15.

Example:

SQL > alter sequence custeq max values 15;

Index

An index is a database object. Its purpose is to increase the performance of data retrieval and thereby provide faster access path to the table data. Indexes can be created on more than one column of a table. The syntax is given below.

Create (unique) index <index_name> on <table_name> (column, column..);

Where the keyword ‘unique’ ensure that the tables does not allow two rows with identical values in all indexed columns. Square brackets indicate that ‘unique’ is optional.

Consider the following example which will create an index to the prodid column of the price table, to give direct access to the rows in customer table.

Example:
SQL > Create index ind_pri on price (prodid);

**Drop database object**

We can remove a database object from the database by issuing the following syntax.

drop database object <object_name>;

The following example a view named discount from the database.

**Example:**

SQL > drop view discount ;

4.5 **Introduction to PL-SQL**

Introduction to PL/SQL block can contain and TCL statements. A PL/SQL block can also contain any number of SQL statements integrated with flow of control statements. Using PL/SQL we can also trap run time errors.

Thus PL/SQL combines the data manipulating power of SQL with the data processing power of procedural languages.

**Advantages of PL/SQL**

PL/SQL is a completely portable, high performance transaction processing language which offers the following advantages.

**Support to SQL**

PL/SQL allows us to tell all SQL data manipulations commands, transaction and commands SQL function (except group function) operator and pseudocolumns thus allowing us to manipulate data values in a table more flexibility and effectively.

**Higher productivity**

PL/SQL can be used to include procedural constructs in non-procedural tools like SQL * forms and Oracle from 4.5 to build applications. For example, we can use the entire block in an SQL * form triggers. Further PL/SQL remains the same in all environments.

**Better Performance**

Without PL/SQL oracle must process SQL statements one at time. With PL/SQL an entire block of statements can be processed in a single
command line statement. This reduces the time taken to communicate between the applications and the oracle server. Thus it helps in improving performance.

**Portability**

Applications written in PL/SQL are portable to any operating system or platform on which oracle ver 6.0 or 7.0 runs.

**Integration with oracle**

Both PL/SQL and oracle have their foundations in SQL. PL/SQL supports all the SQL datatypes and it integrates PL/SQL with the oracle data dictionary.

**Architecture**

PL/SQL blocks are executed by the PL/SQL engine. The PL/SQL engine executes only the procedural statements and sends the SQL statements to the SQL statement executor in oracle server. The PL/SQL engine reside in the oracle server or an oracle tools such as SQL forms 3.1, oracle forms 4.5 reports 2.5.

In the above block diagram, we find that the PL/SQL engine resides in the oracle server. It executes only the procedural statements in the blocks and sends the SQL statements to the SQL statement executor. The latter also resides in the oracle server.

When an oracle tools contain PL/SQL engine, it executes all procedural statements and sends only SQL statements to oracle. If the block contains on
SQL statements then the engine executes the entire block without referring to the server. In the figure shown below SQL * from contains PL/SQL engine.

**Introduction to PL/SQL block**

A PL/SQL can be divided into three parts, namely, a declarative part, an executable part and an exception handling part. The order is as shown.

```plaintext
DECLARE
}
BEGIN
}
EXCEPTION
}
END;
```

Objects can be declared in the declarative part which can be used in the executable part for further manipulations. All procedural statements are included in between the BEGIN and END statements. Errors that occur during execution are dealt in the exception handling part.
Before proceeding to learn about the above three parts, let us have a brief idea about the character set and lexical units used in the PL/SQL block.

The PL/SQL character set includes the following:

- Upper and lower case letters. They are not case sensitive except within strings and character literals.
- Numerals from 0 to 9.
- All special symbols and characters.
- Tab, space, and carriage returns.

PL/SQL text can contain groups of characters known as lexical units. The following are the lexical units:

- Identifiers
- Literals
- Comments
- Delimiter (simple and compound symbols)

The following example illustrates the features discussed above.

```
Total = salary * 0.90, -- to compute total
```

In the above example:

```
Total salary  ----- identifiers
* and  ----- simple symbols
=  ----- compound symbols
0.90  ----- numeric literals
--  ----- represents comment
```

Some of the simple symbols are +, -, *, /, =, <, >, % (attribute indicator); (statements terminates) and ; (host variable indicator). The compound symbols consist of two characters like < >, !=, := (assignment), || (concentration) -- (single line comment), ** (exponentiation) // (multi-line comment), ... (range operator), <<< >>> (label delimiter)
With this knowledge of the structure and basic elements of PL/SQL, let us proceed to discuss each of them in greater detail. The following pages discuss the three parts of a PL/SQL block separately.

**Datatype and their usage**

PL/SQL, datatypes can be classified into scalar and composite datatypes. Scalar datatype include all SQL datatypes and ANSI standard datatype which comprise char, varchar 2, long, long raw, raw, date, boolean and binary - integer. Scalar datatypes comprising boolean, binary-integer and number are discussed in this session, whereas, composite datatypes are discussed in the next sessions.

**Boolean**

Boolean datatypes can be used to store the values TRUE, FLASE, or NULL. They do not take any parameters. We cannot insert boolean datatype in a database column. We cannot fetch column values into a boolean variable.

**Binary_IntegEr**

It is used to store signed integers. The magnitude range of a binary integer value to \(-2^{31-1} \ldots 2^{31-1}\). For convenience PL/SQL predefines the following binary-integer subtypes.

- Natural (0 -- \(2^{31-1}\))
- Positive (1 -- \(2^{31-1}\))

We can use these subtypes to restricts a variable to positive integer values.

**Number**

It is similar to the SQL number datatypes. In addition to this, it also includes ANSI standard types which comprise the following datatypes.

- Dec/decimal
- Int.integer
- Real

**Variables**

We can declare variables in the declarative part and use them elsewhere in the body of PL/SQL block i.e., we can use them in the SQL and procedural
statements. To declare a variable named in_stock to hold a 4 digit number we issue the following statements.

- In_stock number (4) ;

- Similarly we can assign a Boolean datatypes to a variables as shown below.

- Done Boolean ;

Constants

We can declare constants in the declarative part and can use them elsewhere in the executable part. To declare a constant we must make use of the keyword constant. This keyword must proceed the datatype as shown in below.

Credit limit constant real := 7000.00;

In the above example, we have assigned 7000.00 to the constant named credit limit. After this, no more assignment to the constant is allowed, i.e., 7000.00 will be the initial and final value to the constant.

Attributes

Attributes allow us to refer to datatypes and object from the database. PL/SQL variables and constant can have attributes. The following are the types of attributes supported by PL/SQL.

- % type
- % row type

% type

%type attribute is used when declaring variables that refers to the database columns. Consider the following example where a variable called prod is declared to be or type itemid in the item table using % type attribute.

Declare

prod item. itemid % type

Where prod is the variable name, item is the name of the table and itemid is the name of the columns.

The advantages of using % types are listed below
We need not know the exact datatype of the column ‘itemid’. If the database of ‘itemid’ is changed, then, the datatype of ‘prod’ changes accordingly at run time.

**%rowtype**

%row type attribute provides a record type that represents a row in a table. The record can store an entire row of data selected from the table or fetched by a cursor. (This is explained in the next session). In the following example, a record named ‘cust_inf’ will have the same names and datatypes as the columns in the customer table.

Declare

cust_info customer % row type

In the above example, cust_inf stores a row selected from the customer table. The example that are covered in the next session give a clear idea about the % row type attribute.

**Logical Comparison**

PL/SQL supports the comparison of variables and constant in PL/SQL statements. These comparison called ‘Boolean expression’, are often connected by logical expression AND, OR and NOT. The following are the kinds of Boolean expressions.

- Numeric
- Character
- Date

**Numeric Boolean expression**

We can compare numbers using numeric boolean expressions. The table below illustrates relational operators and their meanings when used with numeric values.

<table>
<thead>
<tr>
<th>Operator</th>
<th>Meaning</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>=</td>
<td>Is equal to</td>
<td>a = 434</td>
</tr>
<tr>
<td>!=</td>
<td>Is not equal to</td>
<td>c! = 5656</td>
</tr>
<tr>
<td>&lt;</td>
<td>Is lesser than</td>
<td>a &lt; 1</td>
</tr>
<tr>
<td>&gt;</td>
<td>Is greater than</td>
<td>b &gt; 4</td>
</tr>
<tr>
<td>&lt;=</td>
<td>Is lesser than or equal to</td>
<td>a &lt;=b</td>
</tr>
<tr>
<td>&gt;=</td>
<td>Is greater than or equal to</td>
<td>a &gt;=c</td>
</tr>
</tbody>
</table>
Characters Boolean expressions

A sequence of characters or individual characters enclosed in a string can be compared using character boolean expressions. Their evaluations is based on the alphabetical ordering of the character strings. The table below illustrates and their meanings when used with character strings.

<table>
<thead>
<tr>
<th>Operator</th>
<th>Meaning</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>=</td>
<td>Is the same as</td>
<td>a = 434</td>
</tr>
<tr>
<td>!=</td>
<td>Is not the same as</td>
<td>c! = 5656</td>
</tr>
<tr>
<td>&lt;</td>
<td>Comes alphabetically before</td>
<td>a &lt; 1</td>
</tr>
<tr>
<td>&gt;</td>
<td>Come alphabetically after</td>
<td>b &gt; 4</td>
</tr>
<tr>
<td>&lt;=</td>
<td>Comes alphabetically before or is the same as</td>
<td>a &lt;=b</td>
</tr>
<tr>
<td>&gt;=</td>
<td>Comples alphabetically after or is the same as</td>
<td>a &gt;=c</td>
</tr>
</tbody>
</table>

Date Boolean expressions

We can compare 2 dates using Date Boolean expressions. The table below illustrate on relational operator and their meanings when used with dates.

<table>
<thead>
<tr>
<th>Operators</th>
<th>Meaning</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>=</td>
<td>Is the same as</td>
<td>ord_date = ‘12-Jan-74’</td>
</tr>
<tr>
<td>!=</td>
<td>Is not the same as</td>
<td>Ship_date != ‘14-Feb-88’</td>
</tr>
<tr>
<td>&lt;</td>
<td>Is earlier than</td>
<td>Birthdate &lt; ‘19-Jun-88’</td>
</tr>
<tr>
<td>&gt;</td>
<td>Is later than</td>
<td>Joining date &gt; ‘30-Jun-77’</td>
</tr>
<tr>
<td>&lt;=</td>
<td>Is earlier than or the same as</td>
<td>Next day (ship_date ‘Friday’) &lt;= ‘1-Jan-97’.</td>
</tr>
<tr>
<td>&gt;=</td>
<td>Is latter than or the same as</td>
<td>Expiry date &lt;= ‘25-DEC - 67’</td>
</tr>
</tbody>
</table>

Control Structure

In addition to SQL commands, PL/SQL can also process data using flow of control statements. The flow of control can be classified under the following categories
• Conditional control
• Iterative control
• Sequential control

Before discussing them in detail, let us get acquainted with a package named `dbms_standards`. This package provides language facility to interact with oracle. The examples that follow illustrate the usage of control structures on oracle along with this package.

**Conditional Control**

Sequence of statements can be executed based on some condition using the statements. There are three forms of if statements, namely, if then, if then else and if then elsif. The simplest form of an if statements is the if then statements. The syntax is

```sql
    If condition then
    Sequence of statements ;
    End if ;
```

The sequence of statements is executed only if the condition evaluates to true. If it is also or null, then, the control passes to the statements after `end if` clause in the `if then` statement defines what is to be done if the condition is false or null. An `if then elsif` statement can be used to select one of several mutually exclusive alternatives. The following example illustrates the if then else statements.

**Example:**

```sql
    declare
    Commission oder_info.complan % types;
    begin
    select complan into commission from order_info
    where custid = 4;
    if commission = 'a' then update order_info
    set shipdate = '12-mar-12' where custid= 4
    else
```
update order_info set shipdate = '12-apr-12'
where custid = 4;
end if;
end;

The above example will change ‘shipdate’ to ‘12-mar-96’ if the condition is met, else it changes ‘shipdate’ to ‘12-apr-96’.

Iterative control

A sequence of statements can be executed any number of times using loop constructs. Loops can be broadly classified as

• Simple loop
• For loop
• While loop

Simple Loop should be placed before the first statements in the sequence and the keyword end loop after the last statements in the sequence. The following example illustrates this concept and the sequence of statements are repeated until a = 250.

Syntax for a simple loop follows

Loop
-- sequence of statements
end loop;

Example :

declare
a number := 100;
begin
loop
a := a + 25;
exit when a = 250;
end loop;
While loops

The while loop statements includes a condition associated with a sequence of statement. If the conditions evaluates to true, then the sequence of statement will be executed, and again control resumes at the top of the loop. If the condition evaluates to false, then the loops is bypassed and the control passes to the next statements.

Syntax for a while loop follows

While <condition>
loop
sequence_of_statements;
ends loop;

The following example illustrates the while loop statements.

Example:

declare
i number := 0;
j number := 0;
begin
while i <= 100 loop
j := j + 1;
i := i + 2;
end loop;
dbms_output.put_line(to_char(j));
end;

For loop

The number of iterations for a while loop is known until the loop terminates, whereas the number of iterations in a for loop is known before the
loops gets executed. The for loop statement specifies a range of integers, to execute the sequence of statements once for each integer.

The syntax is given below

For counter in (Reverse) lowerbound .. upperbound
loop
sequence_of_statements;
end loop;

By default, iteration proceeds from lowerbound to upperbound. If we use the optional keyword reverse, then, iteration proceeds downwards from upperbound to lower bound.

The following example executes the update statement once for each integer specified in the ‘for loop’

Example:

declare
begin
for i in 1..2
loop
update order_infoset complain = 'c' where ordid = 5000;
end loop;
end;

Sequential control

The Goto statement allows us to branch to a label unconditionally. The label, which is enclosed by double angle brackets must precede and executable SQL statement or a PL/SQL block. When executed, the goto statement transfer control to the labeled statement or block.

Example:

declare

cost price.miniprice% type;
begin
select stdprice into cost from price where prodid is (select prodid from product where prodesc = ('Shampoo')) ;
if cost > 7000 then
goto updation;
end if;

update price set minprice=6999 where prodid = 111;

Concept of Error Handling

Error condition in PL/SQL is termed as an exception. There are two types of exceptions.
They are

• Predefined exceptions
• User-defined exceptions

An exception is raised when an error occurs. In that case, normal execution stops and the control is immediately transferred to the exception handling part of the our PL/SQL block. Predefined exceptions are raised automatically by the system during run time, whereas user-defined exceptions must be raised explicitly by using RAISE statements.

Predefined exceptions

An exception is raised implicitly when a PL/SQL program violates oracle rule. The following are the predefined exception supported by PL/SQL.
syntax for predefined exception is as follows

begin
sequence_of_statements
exception
when <exception_name> then
sequence_of_statements
sequence_of_statements

<table>
<thead>
<tr>
<th>Predefined Exception</th>
<th>Explanations</th>
</tr>
</thead>
<tbody>
<tr>
<td>No_data_found</td>
<td>This exception is raised when select statement returns no rows.</td>
</tr>
<tr>
<td>Cursor_already_open</td>
<td>This exception is raised when we try to open a cursor which is already opened.</td>
</tr>
<tr>
<td>Dup_val_on_index</td>
<td>This exception is raised when we insert duplicate values in a column, which is defined as unique index.</td>
</tr>
<tr>
<td>Storage_error</td>
<td>This exception is raised if PL/SQL runs out of memory or if the memory is corrupted.</td>
</tr>
<tr>
<td>Program_error</td>
<td>The exception is raised if PL/SQL has an internal problem.</td>
</tr>
<tr>
<td>Zero_error</td>
<td>This exception is raised when we try to divide number by zero.</td>
</tr>
<tr>
<td>Invalid_cursor</td>
<td>This exception is raised when we violate cursor operation. For example when we try to close a cursor which is not opened.</td>
</tr>
<tr>
<td>Login_denied</td>
<td>This exception is raised when we try to enter Oracle using invalid user/name/password.</td>
</tr>
<tr>
<td>Invalid_number</td>
<td>This exception is raised if the conversion of a character string to a number fails because the string does not represent a valid number. For example, inserting ‘john’ for a column of type number will raise this exception.</td>
</tr>
<tr>
<td>Too-many-rows</td>
<td>Raised when the select into statement returns more than one row.</td>
</tr>
</tbody>
</table>

when others then /* the last expression in the exception will go unhandled. A PL/SQL can have only one ‘others’ handler.

The following example illustrates the concept of exceptions

Example :

```
declare
```
price item.actual price % type;
begin
select actual price into price from item where qty = 678 ;
exception
when no_data-found then
dbms_output. put_line ('item missing');
end;

User defined exception

A user defined exception should be declared and raised explicitly by a ‘raise’ statement. It can be declared only in the declarative part of the PL/SQL block. The syntax is

<exception_name> exception ;

The syntax for a ‘raise’ statement follows

Raise <exception_name>

The following example is used to find out whether price per item is zero or Null. If it is so then it raises an exception named zero_price.

Example:

declare
zero_price exception;
price number (8, 2) ;
begin
select actual price into price from item where ordid = 40 ;
if price = 0 or price is null then
raise zero_price;
end if;
exception
when zero_price then
dbms_output. put_line ('raised zero_price exception') ;
end;
4.6 Database Triggers

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

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

Syntax for creating trigger

The syntax for creating a trigger is given below

Create or replace trigger <trigger_name>

(befor/after) (insert/update/delete) on <table-name> (for each statement/ for each row) (when <condition>

A database trigger can also have declarative and exception handling parts.

Parts of a trigger

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

Trigger statements

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

Trigger body

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

Trigger Restriction

Restrictions on a trigger can be achieved using the WHEN clause as shown in the above syntax. They can be included in the definition of a row...
trigger, wherein, the condition in the WHEN clause is evaluated for each row that is affected by the trigger.

**Types of Trigger**

Triggers are categorized based on the when they are fired

- Before
- After
- For each row
- For each statement (default)

**Before /After Options**

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

**For Each Row Statement**

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

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

- Before update row/statement
- Before delete row / statement
- Before insert/row /statement
- After update row/statement
- After delete row / statement
- After insert row/statement

In general each table can have four trigger (before/after/statement/row) on statements (insert/update/delete). Thus, we can assign twelve trigger to a single table.
Example:

```sql
create or replace trigger trig
before insert
on item
for each row
declare
item_idet item.Itemid %type;
begin
select itemid into item_idet from item where qty = 4543;
if item.ident = 1000 then
raise_application_error (-200001, 'enter some order number')
end if;
end;
```

The above example also illustrates the ‘BEFORE ROW’ trigger. Oracle fires this trigger before adding each row by using the insert statement.

The following examples uses triggers to avoid transaction during Saturday and Sunday.

Example:

```sql
create or replace trigger sun_trig
before insert or update or delete
on order_info
declare
shipping_date char;
begin
shipping_date := to_char(sysdate, 'dy')
if shipping_date in ('sat', 'sun') then
raise_application_error(-20001, 'try on any weekdays')
end if;
end;
```
end if;
end;

The above trigger will fire if insert, update or delete statements affect the order_info table. Since it is a before statements trigger, oracle fires it once before executing the trigger statement.

There are two variables namely, : old and : new which retain the old and values of the column updated in the database. The values in these variable can be used in database trigger for manipulations.

**Enabling and disabling triggers**

A trigger can be either or disabled. An enabled trigger executes the trigger body if the triggering statement is issued. By default trigger are enabled. A disabled trigger does not execute the trigger body even if the triggering statement is issued. We can disabled a trigger using the following syntax.

```sql
alter trigger <trigger_name> disable;
```

The following examples disables the trigger named trig

**Example:**

```sql
SQL > alter trigger trig disable ;
```

To disable all trigger associated with the table, we can issue the following syntax.

```sql
alter table <table name> disable all triggers.
```

**Dropping triggers**

To drop trigger from the database, the drop trigger command issued.

The syntax follows.

```sql
drop trigger <trigger_name> ;
```

**Short Answer Type Questions**

1. What are different datatypes in SQL.
2. Write any two DDL commands.
3. Write any two DCL commands.
4. What is a trigger ?
5. What is sub query ?
Long Answer Type Questions

1. Explain in detail three DDL commands with examples.
2. Explain any four DML commands with examples.
3. Write in detail about different SET operators and Joins.
4. Explain the PL-SQL architecture.
5. Write about different types of Trigger with example.
5.1 Definition of System, Analysis and Design

5.1.0 Definition of System

A **system** is a set of interacting or interdependent components forming an integrated whole or a set of elements (often called ‘components’) and relationships which are different from relationships of the set or its elements to other elements or sets.

The word “System” is derived from the greek word “Systema” which means an organized relationship.

Fields that study the general properties of systems include Systems science, systems theory, cybernetics, dynamical systems, thermodynamics, and
complex systems. They investigate the abstract properties of systems’ matter and organization, looking for concepts and principles that are independent of domain, substance, type, or temporal scale.

**Ex:** Financial accounting system, net Banking system etc.,

Some systems share common characteristics, including

- A system has structure, it contains parts (or components) that are directly or indirectly related to each other;
- A system has behavior, it contains processes that transform inputs into outputs (material, energy or data);
- A system has interconnectivity: the parts and processes are connected by structural and/or behavioral relationships.
- A system’s structure and behavior may be decomposed via subsystems and sub-processes to elementary parts and process steps.

**Elements of a system**

Following are considered as the elements of a system in terms of Information system:

1. Inputs and outputs
2. Processor
3. Control
4. Environment
5. Feedback
6. Boundaries and interface

**5.1.1 Sub System**

One of the number of component parts of a system. All the subsystems must function together in an integrated manner for the system to operate as designed.

System development has two major components, System Analysis and System Design. System analysis and design refers to the process of examine a business with intent to improve it through better methods and procedure. System analysis is the process of gathering and interpreting facts, diagnosing problems to determine and improve to the system.
5.1.2 Analysis

System analysis is, process of gathering and interpreting facts, diagnosing problems and using the information to recommend in the improvement of the system. This phase of analysis begins with Fact finding techniques. The fact finding technique are methods of gathering information from the users by using Interviewing, Questionnaires, Observation and Record inspection.

System Analysis is conducted with the following objectives:

• Identify the customer’s need.
• Evaluate the system concept for feasibility.
• Perform economic and technical analysis.
• Allocate functions to hardware, software, people, database and other system elements
• Establish cost and schedule constraints
• Create a system definition that forms the foundation for all subsequent engineering work.

5.1.3 Design

The good of system design is like panning a new business or replace or improve the performance of existing system. It like a blue print for a building.

Software design is an iterative process through which requirements are translated into a “BLUEPRINT” for constructing the software. The design is represented at a high level of abstraction i.e., a level that can be directly traced to specific data, functional, and behavioral requirements.

**Systems design** is the process of defining the architecture, components, modules, interfaces, and data for a system to satisfy specified requirements. Systems design could see it as the application of systems theory to product development. There is some overlap with the disciplines of systems analysis, systems architecture and systems engineering.

5.1.4 System Analyst

A person who analyses a system by the use of scientific techniques in order to determine where and how the improvements can be made with a view to meet objectives in a more efficient and economical manner is called System Analyst. System Analysis is done by System Analyst.
5.2 Study of System Development Life Cycle

The System Development life cycle (SDLC) is organized way to build an information system. SDLC (Software Development Life Cycle) is the process of developing software through business needs, analysis, design, implementation and maintenance. The product developed which achieves customer satisfaction is not done in a single step.

The Methodology used to describe the process for building information systems, intended to develop information systems in a very deliberate, structured and methodical way, reiterating each stage of the life cycle.

Software has to go through various phases before it is born it involves series of steps in a development process. This is needed to develop quality products with error free products to achieve customer satisfaction. There are many models available in the software development process.

But majority of software development process follow the model named as software development life cycle. This software develop life cycle has number of steps in it. Software development life cycle model is also called as waterfall model which is followed by majority of systems. This software development life cycle process has the following seven stages in it namely.
1. System Requirement Analysis
2. Feasibility study.
3. Design
4. Development of Software
5. Testing
7. Implementation

For the development of a new software system or to modify an existing system the above phases should be considered.

5.3 Requirement Analysis, design, Development, Testing, Implementation and Maintenance

5.3.1 System Requirement Analysis

Before developing any system, investigation can be considered. The project request must be examined to know what the client needs. The important aspect of the preliminary investigation is the request for information system feasibility. Basically there are two major areas to be considered while determining the feasibility of a project is

1. **Technical feasibility**: The analyst finds out whether current technical resources of an organization are capable of handling the user’s requirement. If not, then analyst should confirm whether the technology is available and capable of meeting user requirement.

2. **Economic feasibility**: The analyst finds out whether the user can meet the financial requirement of the project or not.

For example, the first essential or vital thing required for any software development is system. Also the system requirement may vary based on the software product that is going to get developed. So a careful analysis has to be made about the system requirement needed for the development of the product. After the analysis and design of the system requirement phase the system required for the development would be complete and the concentration can be on the software development process.

5.3.2 Feasibility Study

Developing information system is a detailed understanding of all important facts of the business needs. Analyst working with employees and managers must study the business process to answer the following questions.
• What is being done?
• How is it being done?
• What type of transactions (or) decisions takes place?
• Does problem exist?
• If a problem exist, what is the effect
• If a problem exist what is the reason.

5.3.3 Design

The most challenging phase of the SDLC is system design. The term design describes a final system and the process by which it is developed. It refers to the technical specifications often called logical design and physical design. In design phase various types of Actors will be identified. Different types of diagrams used in designing software are as follows.

1. Data Flow Diagram
2. Object Diagram
3. Sequence Diagram
4. Collaboration Diagram
5. State Chart Diagram
6. Deployment Diagram

The following are the few Software Designing Models

1. Water Fall Model
2. V – Model
3. Fish Model
4. Spiral Model.

5.3.4 Development of Software

The system development is based on design specification. The software developers may install software (or) they write new custom design, programs known as physical design.

5.3.5 Testing

After completion of development the system is used experimentally to ensure that the software does not fail i.e., that software will run according to the
user expectations for testing raw data for processing and examine the result. The system testing checks the readiness, and accuracy of the system to access, update and retrieve data from new files.

System testing does not test the software but rather the integration of each module in the system. It also tests to fine dependencies between the system and its original objectives current specifications and system documentation.

Testing of each component of computer system by using actual data (or) by pathetical data and analyzing the result in order to ensure that the system trouble free.

**Types of Software Testing**

Basically there are 3 different types of testing is involved in testing of a software system. Those are

- Unit Testing
- White Box Testing
- Black Box Testing

**Unit Testing**

Unit Testing mainly focus on the verification of the smallest unit of software design or the module of a system. Using the procedural design description as a guide, important control paths are tested to uncover errors within the boundary of a module. The unit test is normally white – box oriented, and the step can be conducted in parallel for multiple modules.
White - Box Testing

White box testing, sometimes called GLASS BOX TESTING, is a test case design method that uses the control structure of the procedural design to derive test cases.

Black – Box testing

Black – Box testing deals with the functional requirements of the software. i.e. Black – Box testing enables the software engineer to derive sets of input conditions that will fully exercise all functional requirements for a program.

Other Types of Testing

1. Peak Load testing
2. Storage testing
3. Performance time testing
4. Recovery testing
5. Procedure testing
6. Human factor testing.

1. Peak Load testing

These are critical types in many systems particularly on line systems.

Ex: Banking systems this is real time.

2. Storage Testing

Analyst specifies a capacity for the system when it is designed and constructed storage testing. Often require entering data until capacity is reached.

3. Performance time testing

This test will be conducted prior to implementation to determine how long it takes to receive a response to inquiry, make a backup of a file or send a transmission and receive a response.

4. Recovery Testing

Analyst must always assume that the system will fail and data will be damaged or lost. By creating a failure or data loss event where the users forced to reload and recover from a backup copy, Analyst can readily determine whether recovery procedures and adequate, the best designed plans usually or adjusted.
5. Procedure testing

Analyst should concentrate on removing diskettes before powering down, instructions about when to depress the enter key etc., and explain to user or the should give proper messages to user on screen with regarding procedures.

6. Human factor testing

This testing includes finding answers to questions about how people will react to the system in ways not anticipated.

5.3.6 Maintenance

After the testing phase is completely through and the system is found to be error free it is delivered to the customer. But no real system would be error free even then.

Once installed, the software is often used for many years. The organization and users must be changed according to period of time. Therefore the software has to be maintained i.e. modifications and changes will be made to the software, files and procedures to meet user’s requirement.

5.3.7. Implementation

In the implementation phase, the project team finishes necessary hardware for the system users and then installs the hardware and software in the user environment. Then the users start using the system to perform work. The process of moving from the old system to the new is called “Conversion”. There are different conversion methods.

1. Direct Conversion: All users stop using the old system at the same time, then being using the new.

2. Parallel conversion: User continuous to use the old system while an increasing amount of data is processed through the new system. The outputs from the two systems are compared, if they agree, the switch is made.

3. Phase Conversion: Users start using the new system components by component. This option works only for systems that can be compartmentalized.

The implementation phase is less creative than system design. It is preliminary concerned with user training, site preparations and file conversion. This method lastly but provides assurance against errors in the system.
Data Dictionary

Data Dictionary is a repository that contains descriptions of all data objects consumed or produced by the software.

**Short Answer Type Questions**

1. What is System and sub system
2. What is System Analysis?
3. Who will be called as System Analyst?
4. What are different methods of system testing?
5. What are the methods of system implementation?
6. What is Data Dictionary?

**Long Answer Type Questions**

1. Explain different stages of Software Development Life Cycle in detail?
2. What is testing? Write different types of testing can be considered while testing a software product?


**Reference Books**

1. Data Base Management Systems — Korth and Sudershlan

2. Data Base Management — C.J. Date

3. Software Engineering — Roger Pressman