❖ **Codd’s Laws for Full Functional Relational Database Management System: -**

The most popular data storage model is the relational database, which grew from the seminal paper entitled ‘A Relational Model of Data for Larger Shared Data Banks’ by Dr. E.F.Codd., written in 1970. Dr. Codd defined 13 rules, oddly enough referred to as Codd’s 12 rules, for the relational model.

1. ***Management of Database: -***

   A relational database management system must be able to manage database entirely through its relational capabilities.

1. ***Information Rule: -***

   All information relational databases, including table and column names are represented explicitly as a value in tabular format.

2. ***Guaranteed Access: -***

   Every value in a relational database is granted to be accessible by using a combination of table name, primary key value and column name.

3. ***Systematic Null value Support: -***

   The database management system provides systematic support for the treatment of null values which are distinct from default value and independent of any domain.

4. ***Active, Online Relational Catalog: -***

   The description of the database and its content is represented at the logical level in tabular format and can therefore e require using the database language.

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

   At least one supported language must have a well-defined syntax and be comprehensive. It must support data manipulation, integrity values, authorization & transaction.

6. ***View Updating Rule: -***

   All views that are theoretically updated can be updated through the system.

7. ***Set Level Insertion, Update & Deletion: -***

The database management system support not only set level retrievals but also set level insert, update & delete.

8. ***Physical Data Independence: -***

   Application program logically unaffected when physical access methods or storage structures are altered.

9. ***Logical Data Independence: -***

   Application programs are altered logically unaffected to the extent possible when changes are made to the table structures.

10. ***Integrity Independence: -***

    The database language must be capable of defining integrity rules, these rules must be stored in the online catalog and they cannot be passed.

11. ***Distribution Independence: -***

    Application programs are logically unaffected when data is first distributed.

❖ Database Administrator (DBA): -

Once of the main reason for using the database systems is to have a central control on both data and programs that access those data. A person who has such central control over the system is called database administrator. The functions provided by the database administrator are.

- ✓ *Schema definition.*
- ✓ *Storage structure and access method definition.*
- ✓ *Schema and physical organization modification.*
- ✓ *Granting of authorization to access data.*
- ✓ *Integrity constraint specification.*
- ✓ *Several utilities are available to help you to maintain and control the oracle server. The following topics are included in this section.*
- ✓ *Enterprise Manager.*
- ✓ *SQL * Leader.*
- ✓ *Export and Import.*

❁ ***Enterprise Manager: -***

Enterprise manager allows you to monitor and control on oracle database. All administrative operations are executed by enterprise manager have both GUI (graphical User Interface) and LMI (Line Mode Interface).

Enterprise manager uses a superset of ANSI/ISO standard SQL command. The most common administrative commands are available in the menus of enterprise manager. GUI commands used less frequently can be typed into an enterprise manager SQL worksheet and executed.

❁ ***SQL * Leader: -***

*SQL * leader is used both database administrators and users of oracle. It leads data from standard operating system titles i.e. files in text or data format into oracle database table.*

❁ ***Export & Import: -***

The export and import utilities allow you to move existing data in oracle format to and from oracle database.

Export files can archive data or move data among different oracle database that run on same or different operating system.

❖ **SQL ® Navigator: -**

SQL navigator provides the tool you need to cut oracle PL/SQL Server Side development time in half.

PL/SQL development can be a tedious & time consuming job. It often initializes the variable time and effort of oracle developers. SQL ® navigator is a PL/SQL development solution that streamlines workflow by adding a drag & drop, graphical user interface to the PL/SQL development environment.

SQL navigator speeds the development of oracle based applications & combinations coding, tuning, debugging, web *development & version control to deliver higher quality applications and save valuable time.*

Choose the SQL navigator to must suitable for your environment. Following are editions of several SQL navigators.

1. ***SQL Navigator Standard: -***

   Provides everything the oracle professional needs to code, edit & manage database object.

2. ***SQL Navigator Professional: -***

   Includes all of the features of SQL navigator standard addition plus an integrated PL/SQL debugger for trouble shooting stored database code.

3. ***SQL Navigator Xpert: -***

   Includes all of the features and functionalities of SQL navigator professional edition plus out xpert tuning & quest SQL_optimizer for oracle.

4. ***SQL Navigator Suit: -***

   The SQL navigator suit includes all of the features & functionalities of SQL navigator for SQL plus data factory ® developer addition. Benchmark ® factory oracle and Qdesigner ™ physical are hited. This complete development suit provides all the tools necessary to design, develop, test and manage your database.

❖ Oracle User Manager: -

The user manager in oracle is meaning that to manage the user of oracle who performs the various operations on the database created by him.

1. ***Create User: -***

   In create user we are going to create the user. Here, the system which controls the oracle is referred. It can be done by two ways.

   ➢ ***Create user authenticated by oracle passwords.***

   ✓ ***Syntax: -***

   CREATE USER <user name>

IDENTIFIED BY <password>;

- ***Create user with complex password.***
  - ✓ ***Syntax: -***

    CREATE USER <user name>

    IDENTIFIED BY "<password>";

2. ***<u>Alter User: -</u>***

   In alter user operation the system change the information of user to its relevant information.

- ***Change Password.***
  - ✓ ***Syntax: -***

    ALTER USER <user name>

    IDENTIFIED BY <new password>;

- ***Change default password from oracle information.***
  - ✓ ***Syntax: -***

    ALTER USER SYS

    IDENTIFIED BY <new password>;

3. ***<u>Drop User: -</u>***

   In drop user operation the system disconnect the user. It can also be used to delete the user from the system.

   - ✓ ***Syntax: -***

     DROP USER <user name>

     IDENTIFIED BY <password>;

4. ***<u>Become User: -</u>***

   The become user operation allows guarantee to act as only other user. One can grant himself to be one of the users and perform its works.

❖ **<u>Index: -</u>**

When the user fires a select statement to search for a particular record, the oracle engine must first locate the table. The oracle engine reads the system information and locates the *starting location of a table record on the current storage media. The oracle engine to perform a sequential search to locate records that match usrer_defined criteria.*

Indexing on table is an access strategy that is a way to sort & search records in the table. Indexes are essential to improve the speed with which the records can be located and retrieve from a table.

An index is an order list of content of column or a group of columns of a table.

Indexing involves forming a two dimensional matrix completely independent of the table on which it is being created.

- *A column which will hold sorted data is extracted from the table.*
- *An address field called ROWID identifies the location of record in oracle database.*

When data is inserted in the table the oracle engine inserts the data value in the index. For every data value the oracle engine inserts a unique ROWID value, without exception. The records in the index are sorted in ascending order of index column.

If the select statement has a where clause for the index column the oracle engine will scan the index sequentially looking for a match of the search criteria. The sequential search is done using an ASCII compare routine to scan the column of an index. Since the data is sorted, the sequential search ends as soon as the oracle engine reads an index data value that does not meet the search criteria.

Address Field in Index: -

The address field is called ROWID. ROWID is an internal generated & maintains binary value. The information in ROWID column provides oracle engine the location of the table & a specific record in oracle database.

Types of Index: -

Oracle allows the creation of two types of Indexes. The indexes can be classified in two ways. – Columns and – Records.

- ***Columns: -***

  There are two types of indexes if we classify indexes as number of columns. – Simple Index and – Composite Index.

  - ***Simple Index: -***

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

  - ***Composite Index: -***

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

- ***Records: -***

  There is another way of classifying types of index whether the index can have duplicate value or not. – Unique Index and – Duplicate Index.

  - ***Unique Index: -***

    The index that is created using the keyword UNIQUE is called unique index. The unique index cannot have duplicate value.

  - ***Duplicate Index: -***

    The index that is not enforced not to have duplicate value is called duplicate index.

- ***Creation of Index: -***
  - ***General Syntax: -***

    ```
    CREATE INDEX <Index Name>
    ON <Table Name> (Column Name [, Column Name]);
    ```

  - ***Syntax For Unique Index: -***

    ```
    CREATE UNIQUE INDEX <Index Name>
    ON <Table Name>;
    ```

  When the user defines a primary key constraint, the oracle engine automatically creates unique index on the primary key or unique key column.

- ***Dropping Index: -***

Indexes associated with the table can be removed using the DROP INDEX command.

- ✓ ***Syntax: -***

DROP INDEX <Index Name>;

When a table is dropped the oracle engine automatically drops the entire associated index.

Multiple Indexes on a Table: -

In oracle engine allows the creation of multiple indexes on table. The oracle engine prepares a query plan to decide on the index that must be used for specific data retrieval based on the where clause or the order by clause.

The indexes are created to improve speed but if we have many indexes on one table or we have unnecessary indexes, than the speed benefit may cause the slowness.

ROWID: -

ROWID is an internal generated and maintained binary value which identifies a record. The information in the ROWID column provides oracle engine the location of a table and specific record in the oracle data type.

The ROWID format used by oracle is as follow.

BBBBBBBB.RRRR.FFFF.

Where BBBBBBBB is the block number in which the record is stored. Each data file is further divided into data block and each block is given a unique number. The unique number assign to the first data block in a data file 0. These block number can be used to identify the data block in which a record is stored.

RRRR is a unique record number each data block can store one or more records, thus each record in the data block is given a unique record number.

FFFF is a unique number given by the oracle engine to each data file. Data files are the files used by oracle engine to store user data.

Each time a record is stored into the table, oracle locates free space in the data blocks in the data files. Oracle then inserts records in the table.

❁ ***Deleting Using ROWID: -***

If the user enters duplicate records in a table, a delete statement with a where clause will delete all the records that satisfy the where condition.

If we required returning one record and deleting all other records which are duplicate, the where clause must be define a value that uniquely identifies a record which we can use ROWID to do the job.

```
DELETE FROM EMP
WHERE ROWID NOT IN (SELECT MIN (ROWID) FROM EMP
GROUP BY empno);
```

❖ Inner Select Statement: -

To create a record set of identical records from a table, the records must be grouped on all the columns in the table by using a group by clause.

The select statement will than retrieve the ROWID of the first row in each set at the duplicate records. The first row can be extracted by using MIN function.

```
SELECT MIN (ROWID) FROM EMP
GROUP BY empno, ename, sal;
```

❖ ROWNUM: -

For each returned row, the ROWNUM PSUDO column returns a number indicating the order in which oracle engine selects the row.

The oracle engine assign a ROWNUM value to each row as it is retrieved, before rows are sorted on the column in the order by clause. The order in which data is retrieved id dependent upon the indexes created on the table.

If an index is created on the column in the order by clause, the oracle engine uses the index to retrieve data, thus the ROWNUM will be in the order of rows retrieved from the index.

If no index is created the oracle engine will retrieve data from the table in the order data creation.

❖ **Views: -**

After a table is created and populated with data, it may become necessary to prevent all users from accessing all columns of a table for data security reason. This would mean creating several tables having the appropriate number of columns as required. This will answer data security but will give rise to great deal of redundant data being resident in table.

To reduce redundant data to the minimum possible, oracle allow the creation of an object called a view. A view is mapped to select sentences. The table on which the view is created is described in form clause. The select clause consists of a subset of columns of the table, thus a view, which is mapped to a table will in effect have a subset of actual columns. This technique offers a simple, effective way of hiding columns.

The view is stored only as a definition in oracle system catalogue, when a reference is made to a view, its destination is scanned the base table is opened and the view is created on top of the base table.

Hence, a view holds no data at all until a specific call to the view is made. This reduces redundant data on hard disk to a very large extent. When a view is used the base table is completely invisible. This will give the level of data security required.

View can be queried exactly as through it was a base table however a query fired on a view will run slowly then a query fired on a base table. This is because the view definition has to be retrieved from oracle's system catalogue, the base table has to be identified and opened in the memory.

Two types of view available – Read Only View and – Updateable View.

❁ ***Read Only View: -***

A view that is used to only look at table data and nothing else is called a read only view.

- ***Updateable View: -***

  A view that is used to look at data as well as insert, update and delete table data is called an updateable view.

- ***The reasons Why Views are Created: -***
  - *When data security is required.*
  - *When data redundancy is to be kept to the minimum possible while maintaining data security.*

- ***Creation of View: -***
  - ***Syntax: -***

```
CREATE VIEW <View Name>
AS
SELECT <Column List> FROM <Table Name>
[WHERE <Condition>]
[GROUP BY <Column List>
[HAVING <Condition>]]
[WITH READ – ONLY];
```

Order by clause cannot be used while creating a view. Columns of the table are related to the view on a one to one relationship.

E.g.
```
CREATE VIEW viewemp AS
    SELECT empno FROM EMP;
```

- ***Selecting Data From a View: -***

  Once a view is created, it can be queried exactly like a base table.
  - ***Syntax: -***

```
SELECT <Column List> FROM <View Name>;
```

- ***Updateable View: -***

  View can also be used for data manipulation. Views on which data manipulation can be done are called updateable view. When you give an updateable view name in update, insert or delete SQL statement, modification to data will be passed to the underlining table.

For a view to be updateable it should meet the following criteria.

- ***Views Define From Single Table: -***
  - *If the user wants to insert records with the help of a view than the primary key and all not null columns must be included in the view.*
  - *The user can update, delete records with the help of view even if the primary key and not null columns are excluded from the view.*

Views can also be created from more than one table for the purpose of creating the view. These tables will be linked by join condition specified in the where clause.

The behavior of the view will very from insert, update, delete and select table statement depending upon,

- *Whether the tables were created using a referencing clause.*
- *The tables were created without any referencing clauses are actually stand alone tables.*

- ***Views Defined From Multiple Tables With Referencing Clause: -***

If a view is create using multiple tables with referencing clause that is a view created between the tables than through primary key and not null columns are included in the view, its behavior will be,

- *An insert operation is not allowed.*
- *The delete/update operation does not have effect on the master table.*
- *The view can be used to update the columns at the detail table included in the view.*
- *If a delete operation is executed on the view the correspondence table will be deleted.*

- ***Views Defined From Multiple Tables Without Referencing Clause: -***

If a view is created from multiple table without referencing clause, than though primary key and not null columns are included in view its behavior will be,

- *Insert, update or delete operations are not allowed.*
-

❁ ***Common Restrictions on Updateable View: -***

The following conditions holds true irrespective of the view being created from a single table or multiple tables.

For the view to be updateable, the view definition must not include,

- *Aggregate functions,*
- *Distinct, group by, having clause.*
- *Sub queries.*
- *Constants, strings or value expression like sell price * 1.05.*
- *Union, intersect and minus.*

If a view is defined from another view, the second view should be updateable.

❁

❁ ***Dropping a View: -***

✓ ***Syntax: -***

DROP VIEW <View Name>;

❖ **Sequence: -**

The quickest way to retrieve the data from the table is to have a column in the table whose data uniquely identify a row. By using this column and specific data value in the where clause the oracle engine will be able to identify and retrieve the row fastest.

To achieve this, a constraint is attached to a specific column that ensure that column is never left empty and data values are unique since data entry is done by human being. It is likely that duplicate value will be entered which violates the constraint and the entire row is rejected.

If the value to be entered into this column is machine generated, it will always fulfill the constraint oracle provide an object called a sequence that can generate numeric values. The value generated can have a max of 38 digits a sequence can be define to,

- *Generate number in ascending or descending order.*
- *Provide intervals between numbers.*
- *Caching of sequence number in memory etc.*

A sequence is an independent object and can be used with any table that requires its output.

Creating Sequence: -

The minimum information required for generating numbers using sequence is,

- *The starting number.*
- *The max number that can be generated by a sequence.*
- *The increment value for generating the next number.*
- *The information is provided to oracle at the time of sequence creation.*

✓ ***Syntax: -***

```
CREATE SEQUENCE <Sequence Name>
[INCREMENT BY <Number>
START WITH <Number>
MAXVALUE <Number> / NOMAXVALUE
MINVALUE <Number> /NOMINVALUE
CYCLE / NOCYLCE
CACHE <Number> / NOCACHE
ORDER / NOORDER];
```

Keywords and Parameters: -

Increment By: -

Specifies the interval between sequence numbers, It can be any value accept 0 if this clause is omitted the default value is 1.

Minvalue: -

Specifies the sequence min value.

Nominvalue: -

Specify the minimum value of 1 from an ascending sequence and -1026 for a descending sequence.

Maxvalue: -

Specify the maximum value that a sequence can generate.

❋ ***Nomaxvalue: -***

Specifies the max of 1027 for an ascending sequence of -1 for a descending sequence, this is the default clause.

❋ ***Start With: -***

Specifies the first sequence number to be generated, the default for an ascending sequence is Minvalue of 1 and for a descending sequence it is Maxvalue of -1.

❋ ***Cycle: -***

Specifies that the sequence continues to generate repeat values after reaching either it's Maxvalue.

❋ ***No cycle: -***

Specifies that a sequence cannot generate more values after reaching the Maxvalue.

❋ ***Cache: -***

Specifies how many value of a sequence oracle pre allocates and kept in memory for faster access the Minvalue for this is two.

❋ ***No cache: -***

Specifies that values of a sequence are not pre allocates, if the cache/no cache clause omitted, oracle caches 20 sequence numbers by default.

❋ ***Order: -***

This guarantees that sequence number are generated in the order of request. This is only necessary if you are using parallel sever in parallel mode option. In exclusive mode option, a sequence always generates a number in order.

❋ ***No order: -***

This does not guarantee that sequence numbers are generated in order of request. If order/no order clause is omitted, a sequence takes the no order clause by default.

The order/no order clauses have no significance, if oracle is configured with single server option.

Referencing a Sequence: -

Once a sequence is created SQL can be used to view the values held in its cache.

- ✓ ***Syntax: -***

```
SELECT <Sequence Name>.NEXTVAL FROM
DUAL;
```

This will display the next value held in cache on the monitor screen every time next value references a sequence. Its output is automatically incremented from the old value to the new value ready for use.

Altering a Sequence: -

- ✓ ***Syntax: -***

```
ALTER SEQUENCE <Sequence Name>
[INCREMENT BY <Number>
MAXVALUE <Number> / NOMAXVALUE
MINVALUE <Number> /NOMINVALUE
CYCLE / NOCYLCE
CACHE <Number> / NOCACHE
ORDER / NOORDER];
```

The start value of the sequence not altered.

Drop a Sequence: -

- ✓ ***Syntax: -***

```
DROP SEQUENCE <Sequence Name>;
```

- ***Example: -***

Create sequence Seq_Patient. Using this sequence create primary key for patients. In this table Pat_ID is a primary key & have following constraint. First character of name + number e.g. A1, A2, A3,……, An, B1, B2, B3,……, Bn.

```
CREATE SEQUENCE seqpatients
INCREMENT BY 1
START WITH 1
```

```
MAXVALUE 9
MINVALUE 1
CYCLE
NOCACHE;

CREATE TABLE patients1
(pat_id char (5) PRIMARY KEY,
 patname varchar2 (10));

CREATE TABLE asci
(ascno number (10));

INSERT INTO asci VALUES (65);
SET SERVEROUTPUT ON;
DECLARE
    Patientid patients1.pat_id %TYPE;
    pat_name patients.patname %TYPE;
    ass number(10);
    id varchar2(10);
    sid varchar2(10);
BEGIN
    pat_name:= '&patname';
    SELECT MAX (pat_id) INTO id FROM patients1;
    SELECT SUBSTR (id, 2, 1) INTO sid FROM DUAL;
    IF sid = '9' THEN
        UPDATE asci SET ascno = ascno + 1;
    END IF;
    SELECT ascno INTO ass FROM asci;
    SELECT CHR (ass) ||seqpatients. nextval INTO patientid FROM
    DUAL;
    INSERT INTO patients1 VALUES (patientid, pat_name);
END;
/
```

❖ Granting and Revoking Permissions: -

Oracle provides extensive security features in order to safe guard information from unauthorized viewing and damage. Depending on user status and responsibility, appropriate rights on *oracle resource can be assigned to the user. The rights that allow the user of some of all of oracle resources on the server are called privileges.*

Objects are controlled by the user if a user wishes to access any of the objects belonging to another user; the owner of the object will have to give permission of privileges.

Privileges once given can be taken back of the owner of the object; this is called revoking of privileges.

Grant: -

- ✓ ***Syntax: -***

```
GRANT {OBJECT PRIVILEGES}
ON <Object Name>
TO <User Name>
[WITH GRANT OPTION];
```

Object Privileges: -

- ***Alter: -***

  Allows the guarantee to change the table definition with the alter table command.

- ***Delete: -***

  Allows the guarantee to use delete command on object specified.

- ***Index: -***

  Allows guarantee to create an index on the table.

- ***Insert: -***

  Allows guarantee to add records.

- ***Select: -***

  Allows guarantee to query the table.

- ***Update: -***

  Allows guarantee to modify records.

With grant option allows the guarantee to in turn grant object privileges to other user.

From SYBCA12.

E.g.

```
GRANT ALL
ON EMP
TO sybca12
WITH GRANT OPTION;

GRANT SELECT, UPDATE
ON dept
TO sybca41
WITH GRANT OPTION;
```

- *View the contents of employee belongs to sybca13.*

```
SELECT * FROM sybca12.EMP;
GRANT SELECT
ON sybca12.EMP
TO sybca20;
```

- ***Revoke: -***
  - ✓ *Syntax: -*

```
REVOKE {OBJECT PRIVILEGES}
ON <Object Name>
FROM <User Name>;
```

The revoke command is used to revoke object privileges that the user previously granted directly to revoke.

The revoke command cannot be used to revoke the privileges granted through the operating system.

❖ <u>PL/SQL (Programming Language / Structural Query Language): -</u>

❁ ***<u>Disadvantages of SQL: -</u>***

Though the SQL is natural language of the database administrator, it suffers from various disadvantages when used as a conventional programming language.

- *SQL does not have any procedural capability that is SQL does not provide programming technique of conditional programming, looping and branching.*
- *SQL statements are passed to the oracle engine one at a time each time a SQL statement is executed a call is made to the engine resources. This adds to the traffic on the network there by decreasing the speed of data processing especially in a multi user environment.*
- *While processing on SQL sentence if an error occur the oracle engine displays its own error message. SQL has no facility for programmed handling of errors that arise during manipulation of data.*

❁ ***<u>Advantages of PL/SQL: -</u>***

Although SQL is a very powerful tool, its set of disadvantages prevent it from being a fully structured programming language, for a fully SPL, oracle provides PL/SQL.

As the name suggest, PL/SQL is a superset of SQL. It is block structure language that enables developers to combine the power of SQL with procedural statements. It bridges the gap between database technology and procedural programming language. Following are the advantages of PL/SQL over SQL.

- *PL/SQL is development tool that only support SQL data manipulation but also provides facilities of conditional looping, branching and checking.*
- *PL/SQL sends an entire block of statements to the oracle engine at a time. The communication between programming block and oracle engine reduces considerably. This is turn to reduce network traffic, the oracle engine processes this block of code much faster.*

- *PL/SQL also permits declaring with errors as required and facilities displaying user friendly message when errors are encountered.*
- *PL/SQL allows declaration & use of variables in blocks of code. They can be used to store intermediate result of a query for later processing. PL/SQL variables can be used only where either in SQL statements or in PL/SQL block.*
- *Via PL/SQL all sorts of calculations can be done quickly & efficiently without the use of the oracle engine. This considerably improves transaction performance.*
- *Application written in PL/SQL are portable to any computer hardware & operating system where oracle is operational, hence PL/SQL code block written for a DOS version of oracle will run on its UNIX version without any modification at all.*

❖ <u>Generic PL/SQL Block: -</u>

PL/SQL permits the creation of structured logical block of code the describe processes which have to be applied to data. A single PL/SQL code block consist of a set of SQL statements, clubbed together & passed to the oracle engine entirely. A PL/SQL block has a definite structure. Following are the sections of PL/SQL block.

✓ ***Syntax: -***

```
        DECLARE
        Declaration of memory constants, cursor etc in
PL/SQL.;
        BEGIN
        SQL executable statements;
        PL/SQL executable statements;
        EXCEPTION
        SQL / PL/SQL code to handle errors that may arise
during execution of code block;
        END;
```

❁ ***Declare Section: -***

Code blocks start with declaration section, in which memory variables & other oracle object can be declared & it require initialize.

❁ ***Begin Section: -***

It consists of a set of SQL & PL/SQL statements which describe processes that have to be applied to the data. Actual data manipulations, retrieval, looping and branching constructs are specified in this section.

❁ ***Exception Section:-***

This section deals with handling of errors that arise during execution of the data manipulation statements which make up the PL/SQL block.

❁ ***End: -***

This marks the end of PL/SQL block.

❖ PL/SQL Execution Environment: -

The PL/SQL engine resides in the oracle engine, the oracle engine can process not only single SQL but also entire PL/SQL blocks. These blocks are sent to the PL/SQL engine where procedural statements are executed and SQL statements are sent to the SQL executor in the oracle engine. Since the PL/SQL engine resides in the oracle engine this is all efficient operation.

The call to the oracle engine needs to be made only once to execute any number of SQL statements, if these can bundled inside a PL/SQL block therefore the speed of SQL statement execution is vastly enhanced.

❖ Lexical Units: -

Words used in a PL/SQL block are called lexical units. Blank spaces can be freely inserted between lexical units in a PL/SQL block. The spaces have no effect on the block.

❖ Literals: -

A literal is a numeric value or character string used to represent itself.

❁ ***Numeric Literals: -***

These can be integer or floats if a float is being represented then the integer part must be separated from the float part by a period.

❁ ***String Literals: -***

These are represented by one or more legal characters and must be enclosed within single quotes. Single quotes character itself can be represented by writing it twice.

❁ ***Character Literals: -***

These are string literals consisting of single characters.

❁ ***Logical Literals: -***

These are predetermined constant, the value it can take are; TRUE, FALSE and NULL.

❖ Displaying User Messages on the Screen: -

Dbms_output is a package that includes a number of procedures and functions that accumulate information in a buffer so that it can be retrieved letter. This function can also be used to display message to the user.

*To display message to the user, the server output should be set to on. It is a SQL * PLUS environment parameter that display the information passed as a parameter to the function.*

✓ ***Syntax: -***

```
SET SERVEROUTPUT [ON/OFF];
```

❀ ***Put (Value): -***

Put can have value of any type, which display in a same line.

❀ ***Put Line (Value): -***

It displays contents on the new line with a new character at the end of every time it is called.

❀ ***Newline (Value): -***

This will add the new line.

❖ Comments: -

A comment can have two forms,

- *The comment line begins with double hyphen (--). The entire line treated as comment.*
- *The comment line begins with a slash followed by an Asterisk (/*) till the occurrence of an Asterisk followed by a slash (*/). All lines treated as comment.*

❖ If: -

PL/SQL allows the use of if statement to central the execution of block of code.

✓ ***Syntax: -***

```
IF <Condition> THEN
        <Executable Statements>
ELSIF <Condition> THEN
        <Executable Statements>
ELSE
        <Executable Statements>
END IF;
```

❖ Iterative Control: -

Iterative control indicates the ability to repeat or skip section of a code block. A loop marks sequence of statements that has to be repeated.

Loop: -

- ✓ ***Syntax: -***

```
LOOP
        <Statements>
END LOOP;
```

This loop is a simple loop statement or we can say that it is infinite loop; if we want to terminate the loop we must write a termination condition.

While Loop: -

- ✓ ***Syntax: -***

```
WHILE <Condition>
LOOP
        <Statements>
END LOOP;
```

The keyword loop has to be placed before the first statement in the sequence of statement to be repeated while the keyword end loop is placed immediately after the last statement into the sequence. Once of loop begins to execute, it will go on forever, hence a conditional statement that controls the number of times a loop is executed by always accompanies loop.

For Loop: -

- ✓ ***Syntax: -***

```
FOR <Variable> IN [REVERSE] <Start>..<End>
LOOP
        <Statements>
END LOOP;
```

The variable in the for loop need not to be declared, also the incremental value cannot be specified. For loop variable is always incremented by 1.

Go to Statement: -

- ✓ ***Syntax of Label: -***

<<Label>>;

- ✓ ***Syntax: -***

GOTO <Label>;

The Go to statement changes the flow of control within a PL/SQL block. This statement allows execution at a section of code, which is not in the normal flow of control. The entry point into such a block of code is marked using the tags << Usrer_defined Name >>, the Go to statement can then make use of this user defined name to jump into the block of code for execution.

❖ Oracle Transaction: -

A series of one or more SQL statements that are logically related or a series of operations performed on oracle table data is termed as a transaction. Oracle treats this logical unit as a single entity oracle treats changes to table data as a two-step process, first the change request are done, to make these permanent a commit statement has to be given at the SQL prompt. A rollback statement given at the SQL prompt can be used to undo a part of or the transaction.

A transaction begins with the first executable SQL statement after a commit, rollback or connection made to the oracle engine. All changes made to an oracle table data via unit transaction are made at one sentence.

Specifically a transaction is a group of event that occurs between any of the following event.

- *Connecting to oracle.*
- *Disconnecting to oracle.*
- *Committing changes to the database table.*
- *Rollback.*

A transaction can be closed by using either a commit or a rollback statement. By using these statements table data can be changed or all the changes made can be undone.

Commit: -

✓ ***Syntax: -***

SQL> COMMIT;

❁ ***Rollback: -***

Rollback does exactly the opposite of commit. It ends the transaction but undoes any changes made during the transaction. All transactional locks dequeried on table are released.

✓ ***Syntax: -***

ROLLBACK [WORK] [TO [SAVEPOINT] <Savepoint>];

Work is optional end is provided for ANSI compatibility.

Savepoint is optional and is used to rollback a particular transaction, as for as the specified Savepoint.

Savepoint as a Savepoint name created during the current transaction.

Savepoint marks & save the current point in the processing of a transaction. When a savepoint is used with a rollback part of a transaction can be undone. An active savepoint is one that is specified since the last commit or rollback.

✓ ***Syntax: -***

SAVEPOINT <Savepoint Name>;

Rollback statement can be fired from the SQL prompt with or without the savepoint clause. The implication of each is follows.

A rollback operation performed without savepoint clause, element to the following,

- *Ends the transaction.*
- *Undoes all the changes in the current transaction.*
- *Erase all savepoint in that transaction.*
- *Release the transaction lock.*
- *A rollback operation performed with the to savepoint clause amounts to the following,*
- *A predetermined portion of the transaction is rollback.*
- *Returns the savepoint rollback too but loses all created after the named savepoint.*
- *Reduces all transactional lock that was quires since the savepoint was taken.*

❖ **Cursor: -**

The oracle engine uses a work area for its internal processing in order to execute an SQL statement. This work is as private to SQL operation & is called a cursor.

The data that is stored in the cursor is called the active data set. The size of the cursor in memory is the size required to hold the number of rows in the active data set. This is conceptual however; the actual size is determined by the oracle engine built in memory management capabilities & the amount of memory available.

The values retrieved from the table are held in a cursor opened in memory on the oracle engine, thus data is then transferred to the client machine via the network. If the number of rows are more than the area available in the cursor on the client the cursor data & retrieved data is swapped between wings swap are & memory under the control of the client operating system.

When a cursor is loaded with multiple rows via a query, the oracle engine opens & maintains a row pointer.

❁ ***Types of Cursor: -***

Cursors are classified depending on the circumstances under which they are opened. There are two types of cursors. – Implicit Cursor and – Explicit Cursor.

✵ ***Implicit Cursor: -***

If the oracle engine for its interval processing has opened a cursor, they are known as an implicit cursor. The oracle engine implicitly opens a cursor on the server to process cache SQL statement.

Since the implicit cursor is opened & manage for the oracle engine internally. The function of reserving all area in memory, population this area with data processing the data & releasing the memory area is takes care of by oracle engine.

Implicit cursor attribute can be used to access information about the status of last insert, update, delete or single row select statements. This can be done by preceding the implicit cursor attribute with the cursor name that is SQL.

✠ ***Attributes of Implicit Cursor: -***

1. ***%ISOPEN: -***

   The oracle engine automatically opens & closes the SQL cursor, thus the SQL%ISOPEN attribute of an implicit cursor cannot be referenced outside its SQL statement, as a result, SQL%ISOPEN always evaluates to false.

2. ***%FOUND: -***

   Evaluates to true, if an insert, update or delete affected one or more rows or a single row select statement returns one or more rows.

3. ***%NOTFOUND: -***

   It is logically opposite of %FOUND.

4. ***%ROWCOUNT: -***

   Returns the number of rows affected by an insert, update, delete or select into statement.

   Both SQL%FOUND & SQL%NOTFOUND evaluate to null until they are set by implicit or explicit cursor operation.

❋ ***Explicit Cursor: -***

A user can also open a cursor for processing data as required such user defined cursor are known as explicit cursor. When individual record in a table have to be processed inside PL/SQL code block a cursor is opened. This cursor call be declared & mapped to an SQL query in the declare section of PL/SQL block & used within the executable section. A cursor thus created & used is known as explicit cursor.

✠ ***Explicit Cursor Management: -***

The steps involved in using an explicit cursor & manipulating data in its active set are,

- *Declare a cursor mapped to an SQL select statement that retrieves data for processing.*
- *Open the cursor.*
- *Fetch data from the cursor into variables.*
- *Process the data held in variables as required using a loop.*
- *Exit from the loop after processing is complete.*
- *Close the cursor.*

✠ ***Cursor Declaration: -***

A cursor is defined in the declaration part of the PL/SQL block. This is done by naming the cursor & mapping it to a query. When a cursor is declared the oracle engine is informed that a cursor of the said name needs to be opened, the declaration is only information; there is no memory allocation at this point. The three commands used to control the cursor subsequently are open, fetch & close.

✠ ***The Functionality of Open, Fetch and Close commands: -***

Initialization of a cursor takes place via the open statement this,

- *Define a private SQL area named after the cursor name.*
- *Executes a query with associated the cursor which retrieves table data & populate the named private SQL area in memory that is creates the active data set.*
- *Sets the cursor row pointer in the active data set to the first record.*

  A fetch statement then moves the data held in the active data set into memory variables. Data can be processed as desired.

The fetch statement is placed inside a loop end loop construct which causes the data to be fetched & processed until all the rows are processed.

✓ ***Syntax: -***

```
        DECLARE
        CURSOR <Cursor Name> IS <SQL Select
    Statement>;
        BEGIN
        OPEN <Cursor Name>;
        FETCH <Cursor Name> INTO <Variable List>;
        CLOSE <Cursor Name>;
        END;
```

✠ ***Attributes of Explicit Cursor: -***

The attributes of explicit cursor are same as the attributes of implicit cursor.

❋ ***Cursor For Loop: -***

Another technique is commonly used to control the loop. End loop within a PL/SQL block is for variable in value construct. This is an example of machine defined loop exit that is when all the values in for construct are executed looping steps.

✓ ***Syntax: -***

```
        DECLARE
        CURSOR <Cursor Name> IS <SQL Select
    Statement>;
        BEGIN
        FOR <Variable> IN <Cursor Name>
        LOOP
        <Statement>;
        END LOOP;
        END;
```

Here the verb for automatically creates the variable of the %ROWTYPE. Each record in the opened cursor becomes a value for the memory variable of the %ROWTYPE. For verb ensures that a row from the cursor is loaded in the declared variable & the loop execute once. This goes on until all the rows of the cursor have been loaded into the variable.

A cursor for loop automatically does the following.

- *Implicitly declares its loop index as a %ROWTYPE record.*
- *Opens a cursor.*

- *Fetches a row from the cursor for each loop iteration.*
- *Closes the cursor when all rows have been processed.*

Cursor can be closed even when an exit or a Go to statement is used to leave the loop prematurely, or if an exception is raised inside the loop.

Parameterized Cursor: -

Till now, all the cursors that have been declared & used fetch a predetermined set of records. Records which satisfy conditions set in the where condition of the select statement mapped to the cursor.

In other words, the criterion on which the active data set is determined is hard coded and never changes.

Commercial application require that the query which defines the cursor be generic & the data that is retrieved from the table be allowed to change according to need.

Oracle recognized this & permits the creation of a parameterized cursor prior opening; hence the contents of the opened cursor will constantly change depending upon a value passed.

Since the cursor accepts values or parameterized. It is called as parameterized cursor. The parameters can be either constant or variable.

✓ ***Syntax: -***

```
CURSOR <Cursor Name> (Variable_Name Data Type)
IS <SQL Select Statement>;
```

Opening a parameterized cursor & passing values to the cursor.

✓ ***Syntax: -***

```
OPEN <Cursor Name> (Value/Variable/Expression);
```

Concurrency Control: -

Users manipulate oracle table data via SQL or PL/SQL sentence. An oracle transaction can be made up of a single SQL sentence or several SQL sentences. This gives rise to single query transactions and multiple query transactions.

These transaction access an oracle table or tables, since oracle works on a multi-user platform, it is more than likely that *several people will access data either for viewing or for manipulating from the same tables at the same time via different SQL statement. The oracle table is therefore a global resource that is it is shared by several users.*

Tables contain valuable data on which business decisions are based. There is a definite need to ensure the integrity of data in a table is maintained each time that its data is accessed. The oracle engine has to allow simultaneous access to table data without causing damage to the data.

The technique employed by oracle engine to protect table data when several people are accessing it is called concurrency control.

Oracle uses a method called locking to implement concurrency control when multiple users access a table to manipulate its data at the same time.

Locks: -

Locks are mechanisms used to ensure data integrity allowing maximum concurrent access to data. Oracle's locking is fully automatic & requires no user interaction.

Implicit Locking: -

The oracle engine automatically locks table data while executing SQL statements. This type of locking is called implicit locking it has to decide on two issues.

- *Type of lock to be applied.*
- *Level of lock to be applied.*

Types of Locks: -

The type of lock to be placed on a resource depends on the operation being performed on that resource. Operations on tables can be distinctly grouped into two categories.

- *Read operations: select statement.*
- *Write operations: insert, update and delete statements.*

Since read operations make no changes to data in a table and are meant only for viewing purpose simultaneous read operation *can be performed on a table without any danger to the table's data. Hence, the oracle engine places a shared lock on a table when its data is being viewed.*

On the other hand write operations cause a change in table data that is any insert, update or delete statement affects table data directly and hence, simultaneous write operation can adversely affect table data integrity simultaneous write operation will cause 'Loss of Data Consistency' in the table. Hence, the oracle engine places an exclusive lock on a table.

✠ ***The Rules of Locking: -***

Data being changed cannot be read.
Writers wait for other writers, if they attempt to update the same rows at the same time.

✠ ***Two Types of Locks: -***

1. ***Shared Lock: -***
   - *Shared locks are placed on resource whenever a read operation is performed.*
   - *Multiple shared locks can be simultaneously set on a resource.*

2. ***Exclusive Lock: -***
   - *Exclusive locks are placed on resource whenever write operations are performed.*
   - *Only one exclusive lock can be placed on a resource at a time.*

✠ ***Levels of Lock: -***

A table can be decomposed into rows and a row can be further decomposed into fields. Hence, if an automatic locking system is designed so as to be able to lock the fields of a record it will be the most flexible locking system available.

It would mean that more than one user can be working on a single record in a table that is each on a different field of the same record in the same table.

Oracle does not provide a field level lock. Oracle provides following three levels of locking.

- *Row Level.*
- *Page Level.*
- *Table Level.*

The oracle engine decides on the level to be used by the presence or absence of a where condition in the SQL sentence.

- *If the where clause evaluates to only one row in the table, a row level lock is used.*
- *If the where clause evaluates to a set of data, a page level lock is used.*
- *If there is no where clause, a table level lock is used.*

Explicit Lock: -

The technique of lock taken on a table or its resources by a user is called explicit locking.

Oracle provides facilities by which the default locking strategy can be overridden table or row can be explicit lock by using either the select. For update statement, or lock table statement.

Select. For Update Statement: -

It is used for acquiring exclusive row level locks in anticipation of performing updates on records. This clause is generally represents that data currently being used need to be updated. It is often followed by one or more update statements with a where clause.

The select. For update cannot be used with the following.

- *Distinct and the group by clause.*
- *Set operators and group functions.*

✓ ***Syntax: -***

```
LOCK TABLE <Table Name> [, <Table Name>]
IN {ROW SHARE / ROW EXCLUSIVE / SHARE /
EXCLUSIVE} MODE
[NOWAIT];
```

Table Name: -

Indicates the name of table, view to be locked, In case of views, the lock is placed on underlying tables.

❁ ***In: -***

Decides what other locks on the same resource can exist simultaneously.

✵ ***Exclusive: -***

They allow query on the locked resource but prohibit any other activity.

✵ ***Share: -***

It allows queries but prohibits updates to a table.

✵ ***Row Exclusive: -***

Row exclusive locks are the same as row share locks, also prohibit locking in shared mode. These locks are acquiring when updating, inserting or deleting.

✵ ***Share Row Exclusive: -***

They are used to lock at a whole table, to selective updates & to allow other users to look at rows in the table but not lock the table in share mode or to update rows.

❁ ***Nowait: -***

Indicate the oracle engine should immediately return to the user with a message, if the resources are busy. It omitted, the oracle engine will wait the resources are available forever.

❖ Error Handling in PL/SQL: -

❁ ***Oracle's Named Exception Handlers: -***

The oracle engine has a set of predefined oracle error handlers called named exceptions. These error handlers are referenced by their name. The following are some of the predefined oracle named exception handlers.

✵ ***Predetermined Internal PL/SQL Exceptions: -***

1. ***CURSOR_ALREADY_OPEN: -***

   This exception is raised if an open statement tries to open a cursor that is already open.

2. ***DUP_VAL_ON_INDEX: -***

   The exception is raised if an insert or update would have caused a duplicate value in a unique index.

3. ***INVALID_CURSOR: -***

   This exception is raised if you want to open an undeclared cursor or close one that was already closed or fetch from one that was not open.

4. ***INVALID_NUMBER: -***

   This exception is raised on a conversion error from a character string to a number when the character string does not contain proper number.

5. ***NO_DATA_FOUND: -***

   This exception is raised if a select statement returns 0 rows.

6. ***NOT_LOGGED_ON: -***

   This exception is raised if we attempt any source of database calls without being logged on.

7. ***PROGRAM_ERROR: -***

   This exception is raised if PL/SQL itself as a problem in execution of code.

8. ***ROWTYPE_MISMATCH: -***

   This exception is raised if a host cursor variable and PL/SQL cursor variable involve in an assignment have incompatible return type.

9. ***STORAGE_ERROR: -***

   This exception is raised if PL/SQL need more memory then is available or if it detect corruption of memory.

10. ***SUBSCRIPT_BEYOND_COUNT: -***

This exception is raised when a program referenced a nested table or varray element using an index number larger than the number of elements in the collection.

11. ***SYS_INVALID_ROWID: -***

This exception is raised if the conversion of character string into ROWID fails because the character string does not represent a valid ROWID.

12. ***TIMEOUT_ON_RESOURCE: -***

This exception is raised when a resource oracle is waiting for is not available when it should be this usually means at as an abnormal termination.

13. ***TOO_MANY_ROWS: -***

This exception is raised when a select statement that is supposed to return just one row but return more than one row.

14. ***VALUE_ERROR: -***

This exception is raised when the data type or data size is invalid.

15. ***ZERO_DIVIDE: -***

This exception is raised when a statement tries to divide a number by 0.

❁ ***User Named Exception Handler: -***

The technique that is used to bind a numbered exception handler to a name using "PRAGMA EXCEPTION_INIT ()". This binding of a numbered exception handler to a name is done in the declare section of the PL/SQL block.

All objects declared in the declare section of a PL/SQL block are not created until actually required within the PL/SQL block. However, the binding of a numbered exception handler to a name *must be done exactly when declared not when the exception handler is invoked due to an exception condition.*

The pragma action word is a call to a pre-compiler which immediately binds the numbered exception handler to a name when encountered.

The function exception_init () takes two parameters the first is used defined exception name and the second is the oracle engines exception number. These lines will be included in the declare section of the PL/SQL block.

✓ ***Syntax: -***

```
        DECLARE
                Exception Name EXCEPTION;
                PRAGMA EXCEPTION_INIT (Exception Name,
Error Code);
        BEGIN;
```

Using this technique it is possible to bind appropriate numbered exception handlers to names & use these names in exception section of a PL/SQL block when this is done the default exception handling code of the exception handler is overridden & the user defined exception handling code is executed.

✓ ***Syntax: -***

```
        DECLARE
                Exception Name EXCEPTION;
                PRAGMA EXCEPTION_INIT (Exception Name,
Error Code);
        BEGIN
                <Statement>;
        EXCEPTION
                WHEN Exception Name THEN
                <Statement>;
        END;
```

<u>User Defined Exception Handler (For Input/output Validations): -</u>

The manner of defining and handling an exception in a PL/SQL block of code is shown in the following example.

✓ ***Example: -***

Two client's machines (Client A and Client B) are accessing the same Client Master table using identical PL/SQL code blocks for updating the Bal_Due column of the table Client_Master.

```
SQL> CREATE TABLE Client_Master
 (Client_No VARCHAR2 (6),
 Bal_Due NUMBER (10, 2));

        DECLARE
        Bal_Amt NUMBER (10, 2);
        Trans_Amt NUMBER (10, 2);
        Cl_No VARCHAR2 (6);
        Trans_Type CHAR (1);
        Resource_Busy EXCEPTION;
        PRAGMA EXCEPTION_INIT (Resource_Busy, -00054);
BEGIN
        Trans_Amt:= &Trans_Amt;
        Cl_No:= '&Cl_No';
        Trans_Type:= '&Trans_Type';
        SELECT Bal_Due INTO Bal_Amt FROM Client_Master
        WHERE Client_No = Cl_No FOR UPDATE NOWAIT;
        IF LOWER (Trans_Type) ='d' THEN
                UPDATE Client_Master
                SET Bal_Due = Bal_Due – Trans_Amt
                WHERE Client_No = Cl_No;
        ELSIF LOWER (Trans_Type) ='c' THEN
                UPDATE Client_Master
                SET Bal_Due = Bal_Due + Trans_Amt
                WHERE Client_No = Cl_No;
        END IF;
EXCEPTION
        WHEN Resource_Busy THEN
                DBMS_OUTPUT.PUT_LINE ('the Row Is In Use');
END;
```

❁ ***User Defined Exception Handling (For Business Rules Validations): -***

In commercial application data being manipulated needs to be validated against business rule. If the data violates a business rule, the entire record must be rejected. In such cases, the insert SQL data manipulation language actually runs successfully. It is the data values that the insert statement is placing in the table which violates the business rule.

Since the SQL data manipulation language did not cause any system error, the oracle engine cannot object to the erroneous data value being inserted into a table.

To trap business rules being violated the technique of raising user defined exceptions & then handling them, is used.

All business rule exceptions are completely transparent to the oracle engine & hence the skill of translating a business rule into appropriate PL/SQL user defined exception handling code is vital to a programmer.

User defined error condition must be declared in the declaration part of any PL/SQL block. In the executable part, a check for the condition that needs special attention is made. If that condition exists then call to the user defined exception is made using a raise statement. The exception once raised is then handled in the exception handling section of the PL/SQL code block.

- ✓ ***Syntax: -***

```
DECLARE
        <Exception_Name> EXCEPTION;
BEGIN
SQL Statement;
        IF <Condition> THEN
                RAISE <Exception_Name>;
        END IF;
EXCEPTION
WHEN <Exception_Name> THEN
                {User Defined Action to be taken};
END;
```

❖ **Trigger: -**

- ✓ ***Syntax: -***

```
CREATE [OR REPLACE] TRIGGER <Trigger Name>
```

```
                    {BEFORE / AFTER / INSTEAD OF}
                    {DELETE, INSERT, UPDATE [OF <Column List>]}
                    ON <Table Name>
                    [REFERENCING {OLD AS old, NEW AS new}]
                    [FOR EACH {ROW / STATEMENT}]
                    [WHEN <Condition>]
                    PL/SQL Block;
```

❖ **Stored Procedure: -**

✓ ***Syntax: -***

```
        CREATE [OR REPLACE] PROCEDURE <Procedure Name>
        [(Argument [IN / OUT / INOUT] Data Type [DEFAULT
        <Default Value>]
        [,    “    “    “    “    “    “    “    “    “    “
    )]
        {IS / AS}
        PL/SQL Block;
```

❖ **Stored Function: -**

✓ ***Syntax: -***

```
        CREATE [OR REPLACE] FUNCTION <Function Name>
        [(Argument [IN / OUT / INOUT] Data Type [DEFAULT
    <Default Value>]
        [,    “    “    “    “    “    “    “    “    “    “
    )]
        RETURN <Data Type>
        {IS / AS}
        PL/SQL Block;
```

❖ **Package: -**

❁ ***Package Specification: -***

✓ ***Syntax: -***

```
            CREATE [OR REPLACE] PACKAGE <Package Name>
            {IS / AS}
                    <Package Specification>;
            END [<Package Name>];
```

❁ ***Package Body: -***

✓ ***Syntax: -***

```
CREATE [OR REPLACE] PACKAGE BODY <Package Name>
    {IS / AS}
        <Package Body>;
END [<Package Name>];
```

❁ ***Altering Package: -***

✓ ***Syntax: -***

```
ALTER PACKAGE <Package Name>
COMPILE;

ALTER PACKAGE <Package Name>
COMPILE BODY;
```

❁ ***Dropping Package: -***

✓ ***Syntax: -***

```
DROP PACKAGE <Package Name>;
```

❁ ***Granting Package: -***

✓ ***Syntax: -***

```
GRANT EXECUTE
ON <Package Name>
TO <User Name>;
```

✓ ***Example: -***

➢ *Customer (AcCode number(3), Cname varchar2(50), Balance number(8,2));*

➢ *Trans (TransNo number(4), AcCode number(3), TransType char(1), Narration varchar2(50), Amt number(8,2));*

➢ *Note: - TransType = "C" for Credit and "D" for Debit.*
Create package BankMgt, this package must have following members.

- *Procedure: - AddCust (Name varchar2, Bal number)); To add new customer, AcCode is a next possible number.*
- *Procedure: -CrDr (ACode number, Narration varchar2, Amount number, CrDr char); To record transaction in Trans table, Automatically generate new TransNo. Update customer balance as per Credit or Debit amount.*
- *Function: - function number StudBalance (ACode number); Return balance of customer.*

```
CREATE TABLE Customer
(Accode NUMBER (3),
 Cname VARCHAR2 (50),
 Balance NUMBER (8, 2));

CREATE TABLE Trans
(TransNo NUMBER (4),
 Accode NUMBER (3),
 TransType CHAR (1),
 Narration VARCHAR2 (50),
 Amt NUMBER (8, 2));

CREATE SEQUENCE seqcode
MINVALUE 1;

CREATE SEQUENCE seqtrans
MINVALUE 1;

CREATE OR REPLACE PACKAGE BankMgt
AS
    PROCEDURE AddCust (name varchar2, bal number);
    PROCEDURE CrDr (acode number, narration varchar2, amount ı ɔer,
    CrDr char);
    FUNCTION StudBalance (acode number) RETURN number;
END BankMgt;

CREATE OR REPLACE PACKAGE BODY BankMgt
```

```
AS
   PROCEDURE AddCust (name varchar2, bal number)
   AS
   BEGIN
      INSERT INTO customer
      VALUES (seqcode. nextval, name, bal);
   END AddCust;
   PROCEDURE CrDr (acode number, narration varchar2, amount r   ber,
   CrDr char)
   AS
      t number;
      b number;
   BEGIN
      SELECT   balance   INTO   t   FROM   customer   WHERE
   accode=acode;
      IF LOWER (CrDr) = 'c' THEN
         INSERT INTO TRANS
         VALUES (seqtrans. nextval, acode, CrDr, narration, amount);
         UPDATE customer
         SET balance = t + amount
         WHERE accode=acode;
      ELSIF LOWER (CrDr) ='d' THEN
         INSERT INTO Trans
         VALUES (seqtrans. nextval, acode, CrDr, narration, amount);
         UPDATE customer
         SET balance= t - amount WHERE accode=acode;
      ELSE
         DBMS_OUTPUT.PUT_LINE ('you can Insert "D" for Debit o   " fo
   Credit');
      END IF;
   EXCEPTION
      WHEN NO_DATA_FOUND THEN
         DBMS_OUTPUT.PUT_LINE ('Account Does not exists');
   END CrDr;
   FUNCTION StudBalance (acode number) RETURN number
   AS
      a number;
```

```
    BEGIN
        SELECT balance INTO a FROM customer WHERE
    accode=acode;
        RETURN a;
    EXCEPTION
        WHEN NO_DATA_FOUND THEN
            DBMS_OUTPUT.PUT_LINE ('Account Does not exists');
            RETURN 0;
    END StudBalance;
END BankMgt;

SQL> EXECUTE BankMgt. AddCust ('&name’, &bal);

SQL> EXECUTE BankMgt. CrDr (&acode, '&narration', &amount,
    '&CrDr');

DECLARE
    temp number;
BEGIN
    temp:=&number;
    temp:=BankMgt. StudBalance (temp);
    DBMS_OUTPUT.PUT ('Balance of Customer = ');
    DBMS_OUTPUT.PUT_LINE (temp);
END;
/
```

❖ Object Relational Database Management System: -

❁ ***Abstract Datatype: -***

Abstract datatype are data types that consists of one or more subtypes. We can use abstract datatype to create object tables. In an object table, the columns of the table map to the columns of the abstract datatype. To create abstract datatype uses create type command.

✓ ***Syntax: -***

```
CREATE TYPE <Type Name> AS OBJECT
(<Member Name> <Member Datatype>
 [, <Member Name> <Member Datatype>]);
```

Create type command can also contain other abstract datatype as member, so abstract datatype can be a nested.

- ✓ ***Example: -***

```
CREATE TYPE address_ty AS OBJECT
(street varchar2 (50),
city varchar2 (50),
state varchar2 (30),
pin number (10));
```

❁ ***<u>Object Table: -</u>***

The table that contain at least one object column.

Example: -

```
Create table employee
(empno number(3),
ename varchar2 (30),
address address_ty);
```

✠ ***<u>Inserting records into table based on abstract datatype: -</u>***

To insert record into table based on abstract datatype use constructor method. A constructor method is a program that is named after the datatype. Their parameters are names if the attributes defined for the datatype.

- ✓ ***Example: -***

```
INSERT INTO employee
VALUES (1,'Munavvar', address_ty ('Navsari
Bazar', 'Surat', 'Gujarat', 395002));
```

✠ ***<u>View Records from Table: -</u>***

- ✓ ***Example: -***

```
SELECT e.address.street, e.address.city FROM
employee e;
```

✠ ***<u>Update Record: -</u>***

- ✓ ***Example: -***

```
UPDATE employee e
SET e.address.pin = 395003
WHERE e.address.city = 'Surat';
```

- ***Delete Record: -***
  - ✓ ***Example: -***

```
DELETE FROM employee e WHERE
    e.address.city = 'Surat';
```

- ***To Drop Abstract Datatype: -***
  - ✓ ***Syntax: -***

```
DROP TYPE <Type Name>;
```

Index By Table: -

An index by table is a composite data structure composed of a collection of homogenous elements similar to an array. The elements in this collection are unbounded, distributed and index by binary integer.

The individual elements in the index by table are elements of a scalar data type of a PL/SQL record. Lower and upper limit of binary integers are -2^{31} -1 and 2^{31} -1.

✓ ***Syntax: -***

```
TYPE <Table_Type_Name> IS TABLE OF <Type> INDEX BY
    BINARY_INTEGER;
```

To use index by table first create variable of that type. The elements in index by table are called row, as they have a two-column structure consisting of the index as first column & the corresponding value as a second column, to access individual element of index by table.

✓ ***Syntax: -***

```
<Table_Var_Name> < (Index) >;
```

With index by table, a row is created only when a value is assigned. So before this happens, the row doesn't exist. The index by *table has no lower or upper bound values. Accessing an undefined row raise a PL/SQL error called a NO_DATA_FOUND exception.*

❅ ***Populating an Index by Table: -***

✠ ***Individual Row Created by Assignment: -***

Tmp_tab (100):= 10;

✠ ***Assigning a Table Using a Second Table: -***

Tmp_tab:= tmp1_tab;

Here Tmp_tab and tmp1_tab is a same type of PL/SQL index by table.

✓ ***Example: -***

```
DECLARE
    TYPE num_tab IS TABLE OF number INDEX BY
        BINARY_INTEGER;
    v_tmp_num_tab num_tab;
BEGIN
    FOR i IN 1..5
    LOOP
        v_tmp_num_tab (i):= i * 10;
    END LOOP;
    FOR i IN 1..5
    LOOP
        DBMS_OUTPUT.PUT_LINE (v_tmp_num_tab (i));
    END LOOP;
END;
```

❅ ***Methods of Index By Table: -***

✠ ***Exists: -***

A function that check for the existence of a particular row, return TRUE if row is defined & FALSE otherwise.

- ✓ ***Syntax: -***

```
tmp_tab.EXISTS (Index);
```

- ✠ ***<u>Count: -</u>***

  A function that returns the number of elements defined.

  - ✓ ***Syntax: -***

```
tmp.tab.COUNT;
```

- ✠ *<u>Delete:</u> -*

  A procedure that deletes some or all elements of index by table.

  - ✓ ***Syntax: -***

```
tmp_tab.DELETE;
tmp_tab.DELETE (Index);
tmp_tab.DELETE (Start Index, End Index);
```

- ✠ ***<u>First: -</u>***

  A function that return lowest index, it returns null if the table is empty.

  - ✓ ***Syntax: -***

```
tmp_tab.FIRST;
```

- ✠ ***<u>Last: -</u>***

  A function that returns highest index of table.

  - ✓ ***Syntax: -***

```
tmp_tab.LAST;
```

- ✠ ***<u>Next: -</u>***

  A function that returns index on the element that is next to the element specified by an index. It returns null if such row does not exist.

  - ✓ ***Syntax: -***

```
tmp_tab.NEXT (Index);
```

- ✠ ***<u>Prior: -</u>***

A function that returns index on the element that is previous to the element specified by an index.

- ✓ ***Syntax: -***

```
                tmp_tab.PRIOR (Index);
```

- ✓ ***Example: -***

```
DECLARE
    TYPE my_tab IS TABLE OF number INDEX BY BINARY_INTEGER;
    tmp_my my_tab;
    idx BINARY_INTEGER;
BEGIN
    tmp_my (10):= 100;
    tmp_my (-10):= 30;
    tmp_my (5):= 60;
    IF tmp_my.count>0 THEN
        idx:= tmp_my.first;
        LOOP
            DBMS_OUTPUT.PUT_LINE (tmp_my (idx));
            EXIT WHEN idx:= tmp_my.last;
            idx:= tmp_my.next (idx);
        END LOOP;
    ELSE
        DBMS_OUTPUT.PUT_LINE ('No Row Exists');
    END IF;
END;
```

Varying Array: -

A varying array is a set of objects, each with the same datatype. The size of the array is limited when it is created, when we create a varying array in a table, the array is treated as a column in the main table. Conceptually, varying is a nested table with a limited set of rows. Maximum size of varray is 2 GB.

Creating a Varying Array: -

We can create a varying array based on either an abstract datatype or one of oracle's standard datatype. Varray contain only one column.

- ✓ ***Syntax: -***

```
CREATE [OR REPLACE] TYPE <Varray Name>
AS VARRAY (<Size>) OF <Datatype>;
```

✓ ***Example: -***

```
CREATE TYPE Hobbies_VA AS VARRAY (5) OF
Varchar2 (20);
```

❁ ***<u>To Use Varray in Table:</u> -***

```
CREATE TABLE hobby
(Name varchar2 (50),
 Hobbies Hobbies_VA);
```

❁ ***<u>To Insert Record into Table Having Varray:</u> -***

```
INSERT INTO hobby
VALUES ('abc', Hobbies_VA ('Travel', 'Music', 'Reading'));
```

If we try to enter more than length of varray, oracle gives us an error.

❁ ***<u>Selecting Data From Varying Array:</u> -***

There are two ways to select data from varray.

1. ***<u>Use PL/SQL Block:</u> -***

✠ ***<u>To Use This Method We Have to Use 2 Properties of Varray:</u> -***

- *<u>Limit:</u> Return maximum number of entries per row.*
- *<u>Count:</u> Return current number of entries per row.*

✓ ***Example: -***

```
DECLARE
    CURSOR cur_hobby IS SELECT * FROM hobby;
BEGIN
    FOR hobby_rec IN cur_hobby
    LOOP
        DBMS_OUTPUT.PUT_LINE ('Person Name' ||
            hobby_rec.Name);
        FOR i IN 1..hobby_rec.hobbies.count
        LOOP
```

```
            DBMS_OUTPUT.PUT_LINE (hobby_rec.hobbies (i));
        END LOOP;
    END LOOP;
END;
```

2. ***Use of SQL: -***

   To use SQL we have to use table function that can treat the varying array as a table. To use function take name of varying array as input and its output is alias for table.

   ✓ ***Example: -***

   ```
   SELECT h.Name, v.*
   FROM hobby h, TABLE (h.hobbies) v;
   ```

❅ ***Update Data of Varray: -***

In varray we cannot update the particular number (element) of varray, but varray we have to create new object of varray.

✓ ***Example: -***

```
UPDATE hobby
SET hobbies = Hobbies_VA ('Music', 'Travel')
WHERE Name = 'abc';
```

If we want to assign null value then

✓ ***Example: -***

```
UPDATE hobby
SET hobbies = null
WHERE Name = 'abc';
```

Varray stored as a part of object table, if the size of varray greater than 4000 bytes then, oracle stores it into BLOB. Otherwise it store it into row value.

❁ ***Nested Table (PL/SQL Table): -***

A nested table is a table within a table. It is a table that is represented as a column within another table. We can have multiple rows in the nested table for each row in the main table.

✓ ***Syntax: -***

CREATE TYPE <Ntype Name> AS TABLE OF <Datatype>;
Here datatype is a standard datatype or user defines datatype.

✓ **Example: -**

```
CREATE TYPE emps_nt AS TABLE OF varchar2 (30);
```

✵ ***Object Table: -***

✓ **Example: -**

```
CREATE TABLE newemp
(dname varchar2(30),
emps emps_nt);
NESTED TABLE emps STORE AS emps_nt_tab;
```

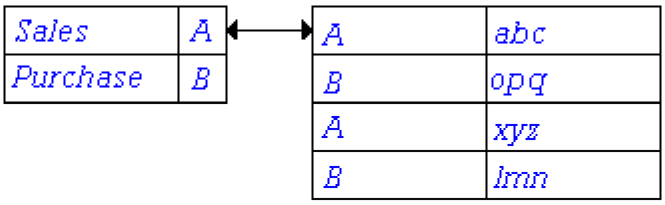
In nested table oracle store parent table & child table separately, it only maintain pointer between them. In above example newemp table stores separately. This table has single column dname, while emps column stored separately in emps_nt_tab. This is a nested table of newemp; emps_nt_tab is not a standard table.

✵ ***Inserting Records into Nested Table: -***

✓ **Example: -**

```
INSERT INTO newemp
VALUES ('Sales', emps_nt ('abc', 'xyz'));
```

To insert record into nested table we have to use constructor of nested table. In nested table system generates NESTED_TABLE_ID, which is 16 bytes in length, correlates the parent row with the rows in its corresponding storage table.



Sales	A
Purchase	B

A	abc
B	opq
A	xyz
B	lmn

❁ ***<u>Selecting Data from Nested Table:</u> -***

To select data from nested table, we have to select nested table within parent table. For that THE function is useful.

✓ ***Example: -***

```
SELECT * FROM THE (SELECT emps FROM newemp WHERE
    dname = 'sales');
SELECT n.dname, t.* FROM newemp n, TABLE (n.emps) t;
```

❁ ***<u>Query to Enter New Data Into Nested Table:</u> -***

```
INSERT INTO THE (SELECT emps FROM newemp WHERE
    dname = 'sales')
VALUES ('zzz');
```

❁ ***<u>Query to Change Existing Data of Nested Table:</u> -***

```
UPDATE THE (SELECT emps FROM newemp WHERE dname =
    'sales')
SET COLUMN_VALUE = 'xxx'
WHERE COLUMN_VALUE = 'xyz';
```

❁ ***<u>Query to Delete Data from Nested Table:</u> -***

```
DELETE FROM THE (SELECT emps FROM newemp WHERE
    dname = 'sales')
WHERE column_value = 'xxx';
```