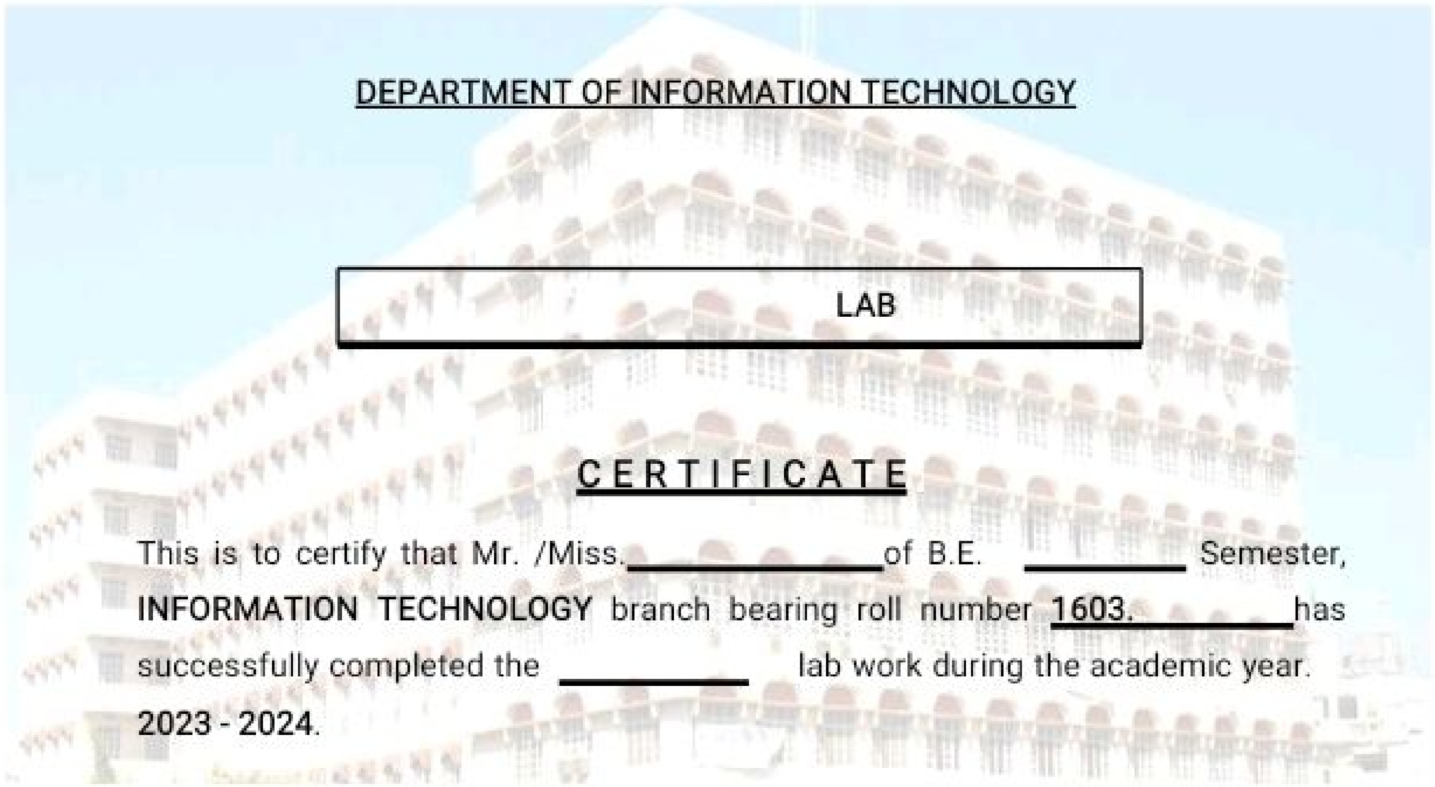
DECCAN COLLEGE OF ENGINEERING AND TECHNOLOGY

Dar-us-Salam, Hyderabad - 500 001.



DEPARTMENT OF INFORMATION TECHNOLOGY

LAB

C E R T I F I C A T E

This is to certify that Mr. /Miss.______________of B.E. _________ Semester, **INFORMATION TECHNOLOGY** branch bearing roll number 1603.________has successfully completed the __________ lab work during the academic year. 2023 - 2024.

Internal Examiner

External Examiner

INDEX

INTRODUCTION TO STRUCTURED QUERY LANGUAGE

SQL is a language that provides an interface to relational database system.SQL was developed by IBM for use in system and is a de facto standard as well as ISO and ANSI standard.In common usage, SQL also encompasses DML(Data Manipulation Language) and DDL(Data Definition Language).

Features of SQL:-
1.SQL can be used a range of users, including those with little or no programming experience.
2.It is not a procedural language.
3.It reduces the amount of time required for creating and maintaining system.

Rules for SQL:-
1.SQL starts with a verb ex: select statement.
2.Each verb is followed by a number of clauses ex: from,where,having.
3.A space separate clauses.
4.A comma(,) separate parameters without a clause.
5.A ';' is used to end SQL statement.
6.Statements may be split across line but keywords may not.
7.Reserved words cannot be used as identifiers unless enclosed with double quotes.
8.Identifiers can contain up to 30 characters and must start with an alphabetic character.

SQL Delimiters:-
Delimiters are symbols or compound symbols which have a special meaning within SQL and PL/SQL statements.

+Addition
- Substraction
* Multiplication
/ Division
= > < Relational
( ) Expression or list
; Terminator

Components of SQL

1.**DDL(Data Definition Language):**
- It is a set of SQL commands used to create,modify and delete database structures but not data.
- These statements are immediate,they are not susceptible to Rollback.
- Every DDL command would commit all the updates as every DDL command implicitly issues a commit command to the database.

2.**DML(Data Manipulation Language):**
- It is the area of SQL that allows changing data within the database.

3.**DCL(Data Control Language):**
- It is the component of SQL statements that control access to data and to the database.
- DCL statements are grouped with DML statements.

4.**DQL(Data Query Language)**:

- It is the component of SQL statements that allows getting data from the database and imposing ordering upon it.It includes ‘select’ statement.

Examples of DDL,DML,DCL and DQL command:-

DDL:

Create: To create objects in the database.
Alter: Alters the structure of database.
Drop: Delete objects from the database.
Rename: Rename objects in database.
Truncate: Removes all records along with space allocated for the records.
Comment: Add comment to data dictionary.
Grant: Give users access privileges with grant command.

DML:

Insert: Inserts data into a table.
Update: Updates existing data within a table.
Delete: Delete all records from a table, the space for the record remains.
Call: Calls a PL/SQL or Java subprogram.
Lock: Table control concurrency.

DCL:

Commit: Saves works done.
Savepoint: Identify a point in a transaction to which you can later rollback.
Rollback: Restore database to original since the last commit.
Set transaction: Change transaction option like what rollback segment to use.

DQL:

Select: Retrieve data from the database.

Table Fundamentals:-

Table is a database object that holds user data. Each table has multiple columns and each column of a table have a specific data type associated with them.

Basic Data types:-

Data types comes in several forms and sizes allowing the programmers to create tables suited to the scope of the project. The decision made in choosing proper data type greatly influences the performance of a database.

Ex: **CHAR(size)**

This data type is used to store character strings values of fixed length. The size in brackets determine the number of characters the cell can hold. The maximum number of characters(i.e size) this data type can hold is 255 characters.

VARCHAR(size)/VARCHAR2(size)

This data type is used to store variable length alphanumeric data.It is more flexible form of the CHAR data type.The maximum this data type can hold up to 400 characters.

DATE
This data type is used to represent date and time. The standard format is DD-MON-YY as in 21-JUN-04.Datetime stores date in the 24-hour format.

NUMBER
The number data type is used to store numbers (fixed or floating point).Numbers of vertically any magnitude may be stored up to 38 digits of precision. Valid values are 0 and positive and negative numbers with magnitude 1.0E-130 to 9.9..E1.25.

LONG
This data type is used to store variable length character strings containing up to 2GB.Long data can be used to store arrays of binary data in ASCII format. Only one long value can be defined per table.

1.DEMONSTRATION OF DATA DEFINITION LANGUAGE COMMANDS

A.CREATE COMMAND

The create table command defines each column of the table uniquely.Each column has a minimum of three attributes a name and data type and size(i.e column width).Each table column definition is a single clause in the create table syntax.Each table column definition is seperated from the other by a comma.Finally,the SQL statement is terminated with a semicolon.

Rules for creating tables:

- A name can have maximum upto 30 characters.
- Alphabets from A-Z,a-z and numbers from 0-9 are allowed.
- A name should begin with an alphabet.
- The use of the special characters like-is allowed and also recommended.
- SQL reserved words not allowed.

Syntax:

Create table tablename(columnName1 Datatype(size),columnName2 Datatype(size),………..columnNamen Datatype(size));

CREATE COMMAND EXAMPLE

CREATION OF STUDENTS TABLE

```
SQL> create table student(sid int,sname varchar(10),sphone int,saddress varchar(20));
Table created.

SQL> desc student;
 Name                    Null?                    Type
 ----------------------------------------- -------- ----------------------------
 SID                                              NUMBER(38)
 SNAME                                            VARCHAR2(10)
 SPHONE                                            NUMBER(38)
 SADDRESS                                         VARCHAR2(20)
```

B. ALTER COMMAND

The structure of a table can be modified by using the Alter table command. With Alter table command it is possible to add or delete columns ,create or destroy indexes, change the data types of existing columns or rename columns or the table itself.Alter table works by making a temporary copy of the original table. The alteration is performed on the copy then the original table is deleted and the new one is renamed.

Alter command is used for the following modification:

Adding a new column using add clause:-

Syntax:

Alter table <tablename>
Add (<newcolumn name1> <Datatype> (<size>) <newcolumn name2> <Datatype> (<size>));

Dropping a column from a table:-

Syntax:
Alter table <tablename>
drop column <columnname>;

Modifying existing columns:-
Syntax:
Alter table <tablename> modify (<columnname> <newdatatype>(<new size>));

ALTER COMMAND EXAMPLE

Alter Student Table(Add)
```
SQL> alter table student21 add(grade varchar(10));
  Table altered.

SQL> desc student21
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 SID                                                NUMBER(38)
 SNAME                                              VARCHAR2(10)
 SPHONE                                             NUMBER(38)
 SADDRESS                                           VARCHAR2(20)
 GRADE                                              VARCHAR2(10)
```

Alter Student Table(Drop)
```
SQL> alter table student21 drop column grade;
Table altered.

SQL> desc student21;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------
 SID                                                NUMBER(38)
 SNAME                                              VARCHAR2(10)
 SPHONE                                             NUMBER(38)
 SADDRESS                                           VARCHAR2(20)
```

Alter Student Table Using Modify
```
SQL> alter table student21 modify(sname varchar(40));
Table altered.

SQL> desc student21;
 Name                                      Null?    Type
 ----------------------------------------- -------- --------------------------
 SID                                                NUMBER(38)
 SNAME                                              VARCHAR2(40)
 SPHONE                                             NUMBER(38)
 SADDRESS                                           VARCHAR2(20)
```

C.DROP COMMAND

Drop table statement can be used to destroy a table and releasing the storage space held by the table. Dropping a table deletes all the data stored (records) in the table and all the indexes associated with it.We cannot roll back the drop statement.

Syntax:

    Drop table <Table Name>;

DROP COMMAND EXAMPLE

```
SQL> drop table s021;
Table dropped.

SQL> desc s021;
ERROR:
ORA-04043: object s08 does not exist
```

D.RENAME COMMAND

Oracle allows renaming of tables. The rename operation is done automatically, which means that no other thread is can access any of the tables while the rename process is running. To rename a table alter and drop privileges on the original table and create and insert privileges on the new table are required.

Syntax:

    Rename <Table Name>   to   <New Name>;

RENAME COMMAND EXAMPLE

```
SQL> rename student21 to s021;
Table renamed.

SQL> desc s021;
 Name                                      Null?    Type
 ----------------------------------------- -------- ---------------------
 SID                                                NUMBER(38)
 SNAME                                              VARCHAR2(10)
 SPHONE                                             NUMBER(38)
 SADDRESS                                           VARCHAR2(20)
```

E.TRUNCATE COMMAND

Truncate table statement removes all rows from a table and releases all the storage space used by that table. The difference between Truncate and Delete statement is: Truncate operation drop and recreate the table which is much faster than deleting rows one by one. Truncate does not generate roll back Information. Truncate operations are not transaction safe. (i.e. an error will allow if an active transaction or an active table lock exist.)

Syntax:

    Truncate table <Table Name>;

TRUNCATE COMMAND EXAMPLE

```
SQL> truncate table s021;
Table truncated.
```

2.DEMONSTRATION OF DATA MODIFICATION LANGUAGE COMMANDS

A.INSERT COMMAND
Once table is created, the most natural thing to do is load this table with data to be manipulated later. When inserting a single row of data into the table the insert operation-
1.creates a new row(empty) in the database table.
2.loads the values passed(by the SQL insert) into the column specified.

Syntax:
Insert into tablename (columnname1,columnname2) values(expression1,expression2);
Characters expressions placed within the insert into statement must be enclosed in single quotes.
In the insert into SQL sentence, table columns and values have one to one relationship (i.e the first value described is inserted into the first column and the second column and so on).

<u>Inserting data into a table from another table.</u>
Syntax:
```
Insert into   Destination_tablename
Select   columnname1,columnname2
   from      Source_tablename;
```

<u>Inserting rows with null values:-</u>
Implicit method omit the column from the column list. Explicit method specify the NULL keyword in the values clause.
Ex: for department table containing

Name	Null?	Type
Department_id	Not Null	Number(4)
Department_name	Not Null	Varchar2(30)
Manager_id		Number(6)
Location_id		Number(4)

Implicit method syntax:
```
Insert into department (department_id,department_name)     values(30,'purchasing');
1 row created.
```

Explicit method syntax:
```
Insert into department values (100,'finance',NULL,NULL);
1 row created.
```

INSERT COMMAND EXAMPLE
<u>INSERTION OF VALUES INTO STUDENT TABLE</u>
```
SQL> insert into student values('&sid','&sname','&sphone','&saddress');
Enter value for sid: 1
Enter value for sname: aaa
Enter value for sphone: 86989
Enter value for saddress: h.no 92-1
old     1: insert into student values('&sid','&sname','&sphone','&saddress')
new     1: insert into boats21 values('1',aaa,86989,h.no 92-1)
```

```
1 row created.
```

B. INSERT COMMAND USING DATA FROM ANOTHER TABLE

Insertion of data into stud01 table from student table:

```
SQL> create table stud01(sid int,sname varchar(10));
Table created.


SQL> insert into stud01 select sid,sname from student01 where sid=1;
1 row created.

SQL> select * from stud01;

       SID     SNAME
---------- --------------------
         1     aaa
```

C.UPDATE COMMAND

Update command is used to change or modify data values in a table. Update is used to either update all the rows from a table, a select set of rows from a table.

Syntax:

For all rows

Update   tablename
Set   columnname1 = expression1,columnname2 = expression2;

A select set of rows

Update   tablename
Set columnname1 = expression1,columnname2 = expression2
Where condition;

UPDATE COMMAND EXAMPLE

DEMONSTRATION OF UPDATE COMMAND:

```
SQL> update student01 set rno=0309501 where rno=1;

1 row updated.

SQL> select * from student01;

       RNO NAME
---------- --------------------
   0309501 aaa

SQL>update emp021 set eid=20 where eid=4;
1 row updated.
```

D.DELETE COMMAND

The delete command deletes rows from the table that satisfies the condition provided by its where clause and returns the number of records deleted. The verb delete in SQL is used to remove either all rows from a table or a set of rows from a table.

Syntax:

For all rows

delete from tablename;

Removal of specific rows.

Delete from tablename where condition;

We can use subqueries in delete statement to remove rows from a table based on values from another table. Ex:

delete from employee
where department_id=(select department_id from department where departmentname like '%public%');

We cannot delete a row that contains a primary key that is used as a foreign key in other table.

DELETE COMMAND EXAMPLES

DELETION WITHOUT WHERE CLAUSE:

```
SQL>delete from student01;
  All rows deleted.
```

E.DELETE COMMAND USING WHERE CLAUSE:

```
SQL> delete from student01 where rno=03095006;

1 row deleted.

SQL>select * from emp021;

       EID EMPNAME                  SALARY
---------- -------------------- ----------
         2 ASD                        3000
         3 DSAZ                      24653
        20 DBD                       57885

SQL>delete from emp021 where eid=2;

1 row deleted.

SQL>select * from emp021;

       EID EMPNAME                  SALARY
---------- -------------------- ----------
         3 DSAZ                      24653
        20 DBD                       57885
```

3. DEMOSTRATION OF DATA CONSTRAINTS

Business rules, which are enforced on data being stored in a table are call constraint. Constraint super control the data being entered into a table for a permanent storage. Oracle permits data constraints to be attached to the table columns via SQL Syntax that checks data integrity prior storage. Once data constraint are part of a table columns construct, the Oracle Database engine checks the data being entered into a table column against the data constraint, the Oracle Database stores the data in the table column. If the data passes the check, else the data is rejected.

- Both the Create table and Alter table SQL verbs can be used to write SQL sentences that attach constraints to a table column.
- Once a constraint is attached to a table column any SQL insert or update statement automatically causes these constraint to be applied to data prior it is being inserted into the table column for storage.
- Constraints prevent the deletion of a table if these are dependencies.
- The following constraint types are valid:
  1. Primary Key
  2. Foriegn Key
  3. Unique Key
  4. Null
  5. Check
- Constraint can be named.
- Constraint can be defined at table level, column level.
- Constraint are stored in data dictionary.

Types of Data Constraint

- There are two types of data constraint that can be applied to data being inserted into a Oracle table.
- One type of constraint is called an I/O constraint (Input/output). This data constraint determines the speed at which data can be inserted or extracted from a Oracle table.
- The other type of constraint is Business rule constraint.
- Constraints can be defined at either column level.

At Column Level,

Syntax:

    Column Name constraint_type

At Table Level

Syntax:

        Constraint_type (column ....)

Column Constraint

References a single column and is defined within a specification for the owning column can define any type of integrity constraint.

Table Constraint

References one more column and is defined separately from the definitions of the columns in the table, can define any constraints except NOT NULL.

I/O Constraint

The Input/output data constraint can be further divided into distinctly different constraints:

i. Primary Key Constraint
ii. Unique Key Constraint
iii. Foreign Key Constraint

A.PRIMARY KEY CONSTRAINT

A primary key is the one or more column in a table used to uniquely identify each row in the table. None of fields that are the part of primary key can contain a Null value. A table can have only one primary key. It defines the column as a mandatory column (i.e. the column cannot be left blank) as the not null attribute is active. The data held across the column must be unique.

- A single column primary key is called a simple key.
- A multicolumn primary key is called a composite key.
- A primary key can be defined in either a create table statement or an alter table statement.
- Unique index is created automatically if there is a primary key.

Primary key constraint defined at column level.

Syntax:

Column Name Datatype(Size) Primary Key

Primary Key constraint defined at table level

Syntax:

Primary Key(Column Name,Column Name)

A composite key is defined only by the table level definition.

PRIMARY KEY EXAMPLE

AT COLUMN LEVEL

```
SQL>create table products(pid int primary key,pname varchar(20));
Table created.
```

AT TABLE LEVEL

```
SQL>create table products(pid int,pname varchar(20),primary key(pid));
Table created.
```

B.FOREIGN KEYCONSTRAINT

Foreign Key represents relationship between tables. A Foreign Key is a column (or a group of columns) whose values are derived from the primary key or unique of some other table. A table in which foreign key is defined is called a foreign table or detail table. The table that defines the primary or unique key and is referenced by the foreign key is called the primary table or master table. Foreign key can be defined in either a create table statement or an Alter table statement. The master table can be referenced in the foreign key definition by using the clause 'References' when defining the foreign key, column attributes in the detail table.

The constraint of the foreign key establishes a relationship between the records i.e. Column data across a master and a detail table.

This relationship ensures:

- Records cannot be inserted into a detail table if corresponding records in the master table does not exists.

- Records of the master table cannot be deleted if corresponding records in the detail table actually exist.

Foreign Key constraints:

- Rejects an insert /or update of a value does not currently exist in the master key table.
- Rejects a delete from the master table if corresponding records in the table exists.
- If on delete cascade option is set a delete operation in the master table will trigger a delete operation for corresponding records in all detail tables.
- if on delete set Null option is set a delete operation in the master table will set the value held by the foreign key of the detail table to Null.

Foreign key constraint defined at the column level
Syntax:
    Column Name Data type(Size) references
    Table Name [Column Name] [ON delete cascade]
Ex:
Create table Account(a_no char(20) primary key, branch_name char(20) references branch, balance integer);

Foreign Key constraint defined at the table level
Syntax:
    Foreign Key (Column Name,Column Name) references Table_Name(Column Name, Column Name)

Ex:
Create table Account (a_no char (20) primary key, branch_name char(20),balance integer, foreign key (branch_name) references branch (branch_name));

FOREIGN KEY EXAMPLE

```
SQL> create table bank(cid integer primary key,cname varchar(20));

Table created.
```

DEMONSTRATION OF FOREIGN KEY: (AT COLUMN LEVEL)

```
SQL> create table accoynt(acno integer,acname char(20),cid integer references bank(cid));

Table created.
```

DEMONSTRATION OF FOREIGN KEY : (AT TABLE LEVEL)

```
SQL> create table account1(acno integer,acname char(20),cid integer,foreign key(cid)
references bank(cid));

Table created
```

C.UNIQUE KEY CONSTRAINT

Unique key constraint permits multiple entries of Null into the column. The NULL values are clubbed at the top of the column in the order in which they were entered into the table. This is essential difference between the primary key and the unique constraint when applied to the table columns.

- Unique will not allow duplicate values.
- Unique index is created automatically.
- A table can have more than one unique key which is not possible in primary key.
- The column included in the definition of the unique key constraint is called the unique key.
- If the unique constraint comprises more than one column, that a group of columns is called a composite unique key. Unique constraint can be defined either at the table level or column level.

<u>Unique constraint defined at the column level.</u>
Syntax:
Column_Name datatype(Size) unique.
Ex:Create table Customer (c_id char(20) unique, c_name char(20),c_address char(20));

<u>Unique constraint defined at the table level.</u>
Syntax:
Create table TableName (Column Name data type(size). Column Name data type(size),unique(Column Name1, Column Name2));

UNIQUE KEY EXAMPLE

<u>AT COLUMN LEVEL</u>

```
SQL>create table employee021(eid int primary key,ename varchar(20) unique);
Table created.
```

<u>AT TABLE LEVEL</u>

```
SQL>create table employee21(eid int ,ename varchar(20) ,primary key(eid),unique(ename));
Table created.
```

BUSINESS RULE CONSTRAINT
D.CHECK CONSTRAINT
Business Rule Constraint can be further divided into check constraint and not null constraint.

- Oracle allows the application of business rules to table columns.
- The rules are applied to data prior the data is being inserted into the table columns.

This ensures that the data (records) in the table have integrity.

- Business rules can be implemented in Oracle by using Check Constraints.
- Check Constraint can be bound to a column or a table using the create table or alter table command.

Check constraint defines a condition that each row must satisfy. A single column can have multiple check constraints which refer to the column in its definition. There is no limit to the number of check constraint which you can define on a column. Check constraint must be specified as a logical expression that evaluate either to True or False.

<u>Check Constraint defined at the column level.</u>
Syntax:

Column Name Data type(Size) Check (Logical Expression);

<u>Check Constraint defined at the column level.</u>
Syntax:

Check (Logical Expression)

Restricted on check constraints
A check integrity constraint require that a condition to be true or unknown for the unknown for row to be processed. If an SQL Statement causes the condition to evaluate to false, an appropriate error message is displayed and processing stops.

Check has following limitations:

- The Condition must be a Boolean expressing that can be evaluated using the values in the row being inserted or update.
- The Construct cannot contain sub queries or sequences.
- The Condition cannot include the SYSDATE, UID, USER or USERENV SQL Functions.

CHECK CONSTRAINT EXAMPLE

<u>CHECKING SALARY>500</u>

```
SQL> create table empl021(eid int primary key,empname varchar(20),salary float
check(salary>500));
Table created.

SQL>insert into empl021 values('&eid','&empname','&salary');
Enter value for eid: 1
Enter value for empname: ABC
Enter value for salary: 300
old   1: insert into empl021 values('&eid','&empname','&salary')
new    1: insert into empl021 values('1','abc','300')
insert into empl021 values('1','ABC','300')
*
ERROR at line 1:
ORA-02290: check constraint (SYSTEM.SYS_C007103) violated
```

(E)NOT NULL CONSTRAINT
Often there may be records in a table that so not have values for every field. This could be because the information is not available at the time of data entry or because the field is not applicable in every case. If the column was created a Null able, Oracle will place a Null value in the column in the absence of a user defined value. A Null value is different from a blank or zero. A Null value can be inserted into column of any data type. Setting a Null value is appropriate when actual value is unknown, or when a value would not be meaningful. A Null value is not equivalent to a value of zero if the data type is a number and is not equivalent to spaces if the data type is character.A Null value will evaluate to Null in any expression. If column is defined as Not Null, then that column becomes a mandatory column. It implies that a value must be entered into the column, if the record is to be accepted for storage in the table.
Syntax:

<Column Name><Data Type>(<Size>) NOT NULL;

NOT NULL CONSTRAINT EXAMPLE

```
SQL> create table lab(lid int primary key,lname varchar(20) NOT NULL);
Table created.

SQL> insert into lab values('&lid','&lname');
Enter value for lid: 1
Enter value for lname:
old   1: insert into lab values('&lid','&lname')
new   1: insert into lab values('1','')
insert into lab values('1','')
                            *
ERROR at line 1:
ORA-01400: cannot insert NULL into ("SCOTT"."LAB"."LNAME")
```

4. SIMPLE CONDITION QUERY CREATION

A.SELECT CLAUSE

A select statement retrieves information from the database. Using the select statement you can do the following.
Projection: You can use the projection capability in SQL to choose the columns in a table that you want returned by your query. You can choose as few or as many columns of the table as you require.
Selection: You can use the selection capability in SQL to choose the rows in a table that you want returned by your query.
Joining: You can use the Join capability in SQL to bring together data that is stored in different tables by creating a link between them.

Syntax:
Select * | { [distinct] column | expression [alias]..}
From table;

Select identifies what column, from identifies which table.

Select-is a list of one or more columns.
*****-select all columns.
distinct-suppress duplicates.
column|expression-selects the named column or the expression.
alias-gives selected columns different headings.
from table-specifies the table containing the columns.

- Multiple columns can be selected by specifying the column names separated by commas.
- Arithmetic operators can be used in select statement.The order of precedence is * / + -
  Ex: select ename,salary,12*salary
  From emp;

AS: As is used for renaming a column heading.It is useful with calculations.
Ex: select ename,12*salary as annual sal
from emp;

|| : || is a concatenation operator. It concatenate columns or character strings to other column.
Ex: select last-name || job as “employee” from emp;

Lateral character string:
A lateral is a character, a number, or a date in the select list.
Date and character lateral value must be enclosed within single quotation marks.
Each character string is output once for each row returned.
Ex: select last-name || ‘is a’ || job-1 as “employee details”
from employees;

Queries on Select Clause

1.Display The Deptno’s From Dept. Table .
Sql> Select Deptno From Dept;

2.Display Salary With Name Of The Employee:
Sql> Select Sal,Ename From Emp;

3.Increase Salary By 1000:
Sql> Select Sal,Sal+1000 From Emp;

4.Display Annual Salary And Employee Name As Last Name:
Sql> Select 12*Sal As Annualsal,Ename As Lastname From Emp;

5.Combination Of Last Name And Job:
Sql> Select Ename || Job As Empdetails From Emp;

6.Display Last Name Name With Job Separated By 'Is A' As Emp Details:
Sql> Select Ename ||'Is A'|| Job As Empdetails From Emp;

B. WHERE CLAUSE
We can restrict the rows returned by using the where clause.

Syntax:
Select * | {[distinct] column | expression [alias]..}
From table
Where condition(s);

Where clause contains a condition that must be met and it directly follows the from clause.
If condition is true the row meeting the condition is returned.

In the syntax;
Where: restricts the query to rows that meet a condition.
Condition: is composed of column names, expressions constants and a comparison operator.

Where clause can compare values in columns, literal values, arithmetic expression or function.
It consists of three elements
1) column name
2) comparison condition
3) column name, constant or list of values.

Character strings and data values are enclosed in single quotation marks.
- All character searches are case sensitive.
- Comparison conditions are =, >,>=, <, <=, < >, ! used in where clause as
- Where expr operator value.
- Ex: where salary>=6000.
- Other comparision conditions are between and -Between two values(inclusive),In(set)-match any of a list of values, is null- is a null value.

Logical condition-And- return true if both component conditions .
Or- returns true if either component condition is true
Not- return true if the following condition is false

Queries On Where Clause

7. Display All Employees In Dept No 30:
Sql> Select * From Emp Where Deptno=30;

8. Display All Details Of Manager:
Sql> Select * From Emp Where Job='Manager';

9.Display Employees Names Whose Sal Is Between 2000 And 2500:
Sql> Select Ename From Emp Where Sal Between 2000 And 2500;

10.Display Employees No And Names With Mgrno Where Mgrno Can Be Either7839,7698:
Sql> Select Empno,Ename,Mgr From Emp Where Mgr In(7839,7698);

11.List The Names Of Employees Who Are Not Managers:
Sql>   Select Ename From Emp Where Job!='Manager';

12.List The Names Of Employees Who Do Not Report To Mgr:
Sql> Select Ename,Job From Emp Where Mgr Is Null;

13.List The Name Of Employees Not Eligible For Commission:
Sql> Select Ename From Emp Where Comm Is Null;

14. List The Names Of Analysts And Salesman:
Sql>   Select Ename From Emp Where Job In('Analyst','Salesman');

15. List The Names Of Employees Working In Dept No 24,40,50 Or As A Clerk:
Sql> Select Ename From Emp Where Deptno In(24,40,50) Or Job='Clerk';

16.List The Names Of Employees Not Working In Dept No 10 Or 20:
Sql> Select * From Emp Where Deptno Not In(10,20);

17.Display The Employees Names Ending With S:
Sql> Select Ename From Emp Where Ename Like '%S';

18. Display Employees Names With Hiredate Hired In 1982:
Sql> Select Ename,Hiredate From Emp Where Hiredate Like '%_%_82';

19.DisplayEmployees The Names Whose Third Letter Is A:
Sql> Select Ename From Emp Where Ename Like '_A%';

20.Display Employees The Names Who Have A In Their Name:

Sql>   Select Ename,Deptno From Emp Where Ename Like '%A%';

C. ORDER BY CLAUSE

The order by clause can be used to sort the rows. Order by must be the last clause of SQL statement.

Syntax:

```
select expr
from table
where condition
order by {column,expr} {Asc [desc]};
```

asc- ascending order – order rows in ascending order, it is a default order.
desc- order the rows in descending order.

- If a column contain Null value then null values are displayed last for ascending sequences and first for descending sequences.
- Column all as can be used in order by clause.
- You can sort query results by more than one column. The sort limit is the number of columns in the given table, separate multiple column names using commas.

21.Display Employees Names In The Order Of Salaries:
Sql> Select Ename,Sal From Emp Order By Sal;

22.Display The Details From Employees Details In The Order Of Emp No:
Sql> Select * From Emp Order By Empno;

23.Display Employees Name Along With Annual Salary In Descending Order:
Sql> Select Ename,12*Sal As Annualsal From Emp Order By Annualsal Desc;

D. GROUP BY CLAUSE:

Group by clause to divide the rows in a table into groups.

Syntax:

```
select column, group-function (column)
from table
where condition
group by group_by_expression
order by Column.
```

- Using where clause we can exclude rows before dividing them into groups.
- Columns alias cannot be used in group by.
- When using the group by clause, make sure that all columns in the select list that are not group functions are included in the Group by clause.

Ex:

```
Select department_id, Avg(salary)
From employee
Group by department_id.
```

Average will be calculated for each department.

- You can return summary results for groups and subgroups by listing more than one group by column.

  Ex:

  ```
  Select did, Job_id, sum(salary)
  From emp
  ```
- Using the count Function

Queries On Group By Clause

24.Display Dept No And Total No Of Employees:
Sql> Select Deptno,Count(Ename) From Emp Group By Deptno;

25.Display Various Jobs And Total Employees
Sql> Select Job,Count(Ename) From Emp Group By Job;

26.Display Dept No And Total Salary:
Sql> Select Deptno,Sum(Sal) From Emp Group By Deptno;

27.Display Each Job With Min Salary:
Sql> Select Job,Min(Sal) From Emp Group By Job;

(E)HAVING CLAUSE
Having clause can be used to satisfy which groups are to be displayed. Thus, further restricting the groups on the basis of aggregate of Information.
Syntax:

```
select column, group function
from table
where condition
group by group_by_expression
having group_conditoin
order by column
```

Oracle server performs the following steps when you use the Having clause
1. Rows are grouped
2. Group function is applied to the group
3. The groups that match the criteria in the having clause are displayed.

Ex:

```
select did, avg(sal)
from emp
group by did
having max (sal)>1000;
```

Queries On Having Clause

28.Display Dept No If More Than 3 Employees Are In Each Dept:
Sql> Select Deptno,Count(Ename) From Emp Group By Deptno Having Count(Ename)>3;

29.List Jobs Whose Total Sal Is Greater Than 4000:
Sql> Select Job,Sum(Sal) From Emp Group By Job Having Sum(Sal)>4000;

(F)ORACLE FUNCTIONS (CASE,STRINGS,DATE)
Functions are a very powerful feature of SQL and can be used to do the following.

- perform calculations on data.
- modify individual data items.
- manipulate output for groups of rows.
- convert column data types.

SQL functions sometimes take arguments and always return a value.

Two types of SQL functions are,
a)Single row function
b)Multiple row function

Single row functions:
These functions operate on single row only and return one result per row.There are different types of single row function.

1) Character
2) Number
3) Date
4) Conversion

Multiple row functions:
Functions can manipulate groups of rows to give one result per group of rows. These functions are known as group functions.

Single row functions are;

1) Character function-Accepts character input and can return both character and number values.
2) Numbers function-Accepts numeric input and returns numeric values.
3) Date function-Operates in values of the data type.
4) Conversion function-Converts a value from one data type to another.
5) General functions-are NVL, NVL2, NULLIF, COAL SELECT, CASE, DECODE.

1.CHARACTER FUNCTIONS:
Accepts character data as input and can return both character and numeric values.Character values can be divided into;

1.CASE MANIPULATION FUNCTIONS.
Examples are;
LOWER(column|expression): convert alpha character values to lowercase.
UPPER(column|expression): converts alpha character values to uppercase.
INIT CAP(column|expression): converts alpha character value to uppercase for the first letter of each word, all other letters in lowercase.

II.Character manipulation functions.

Examples are;

<u>CONCAT(column1|expression1,column2|expression2)</u>: concatenates the first character value to the second character value, equivalent to concatenation operator(||).

<u>SUBSTR(column|expression,m,[n]):</u> returns specified character position m, n characters long(if m is negative count starts from the end of the character value).

<u>LENGTH(column|expression):</u> returns the number of characters in the expression.

<u>INSTR(column|expression,'string',[m],[n])</u>: returns the numeric position of a named string, optionally a position m to start searching can be provided and the occurrence n of the string. m and n default to 1, meaning start the searching at the beginning of the search and report the first occurrence.

<u>LPAD(column|expression,n,'string')</u>: pad character values right justified to a total width of n character positions.

<u>RPAD(column|expression,n,'string'):</u> pad the character values left justified to a total width of n character positions.

2.NUMBER FUNCTIONS:

Number functions accepts numeric input and returns numeric values.

Ex:

<u>Round(column|expression,n)</u>: Rounds the column, expression or value to n decimal places or if n is omitted, no decimal places. If n is negative, numbers to left of the decimal point are rounded.

<u>Trunc(column|expression,n):</u>Truncates the column expression or value to n decimal places or if n is omitted then n defaults to zero.

<u>Mod(m,n): </u>Returns the remainder of m divided by n.

```
Ex: select Round(45.923,2) , Round(45.923,0) , Round(45.923,-1)
        from dual;
```

```
O/P:    Round(45.923,2) = 45.92
         Round(45.923,0) = 46
         Round(45.923,-1) = 50
```

If second argument is 0 or is missing the value is rounded to zero decimal places.If the second argument is 2 the value is rounded to two decimal places.If -2 the value is rounded to two decimal places to the left.

Dual table:-

Dual table is owned by the user system and can be accessed by all users. It contain one column dummy and one row with the value X, dual table is useful when you want to return a value once only,dual table is generally used for select clause syntax completeness, because both select and from clause are mandatory and several calculations do not need to select from actual tables.

3.DATE FUNCTIONS

Dates:-Oracle database stores dates in an internal numeric format.

Century,year,month,day,hours,minutes,seconds.

Default: date display format is

DD – month – year

19-oct-2000

Sysdate : is a date function that returns the current database server date and time. Sysdate can be used as any other column name.

Ex: select sysdate from dual;

Arithmetic with dates:

Since the database stores dates as numbers, you can perform calculations using arithmetic operators such as addition and subtraction.

Operation	Result
Date + Number	Date
Date – Number	Date
Date – Date	Number of days
Date + Number/24	Date

Arithmetic operators ex:
Select name, (sysdate – hiredate)/7 as weeks from emp;

Date functions:-

Months-Between (Date1, Date2) :Finds the number of months between date1 and date2. The result can be positive or negative.
Add_months (Date, n): Adds n number of calander months to date, the value of n must be an integer and can be negative.
Next_day (date, 'char'): Finds the date of the next specified day of the week ('Char') following date. The value Char can be number representing a day or a character string.
Last_day (date): Finds the date of the last day of the month that contains date.
Round (date, ['fmt']): Returns date rounded to the unit specified by the format model FMT. If FMT is omitted then date is rounded to the nearest day.
Trunc (date, ['fmt']: Returns date with the time portion of the day truncated to the unit specified by the format model FMT. If FMT is omitted, date is truncated to the nearest day.

Ex:
Assume sysdate = '25 – jul – 1995'
Round (sysdate, 'month') – 01 – aug – 1995
Round (sysdate, 'year') – 01 – jan – 1996
Trunc (SysDate, 'Month') 01-JUL-95
Trunc(SysDate, 'YEAR') 01-JAN-1995
*fmt can be month and year.

4.CONVERSION FUNCTION:

Data type conversions can be either:
Implicit Data type conversion;
Explicit Data type conversion.
Implicit Data type conversions are done implicitly by Oracle Server.
Explicit Data type conversion are done by using the conversion Functions.
Ex for Implicit data type conversion (For Assignments)

From	To
Varchar 2 or char	Number
Varchar 2 or char	Date
Number	Varchar 2
Date	Varchar 2

Implicit Data Type conversion (For Expression Evaluation.)

From	To
Varchar 2 or char	Number

Varchar 2 or char Date
Ex: For Explicit Data Type Conversion
To_char: (number | date, [fmt], [nLsparams]) converts a number or date value to a varchar 2 characters string with format model fmt. nLsparam (Number Conversion) specifies the following characters which are returned by numbers format elements.

- Decimal Character
- Group Separator
- Local currency symbol
- International currency symbol

If nLsparams or any other parameter is omitted, this function uses default parameter values for the session.

To_Number: (Char,[fmt],[nLsparams])
Converts a character string containing digits to a number in the format specified by the optional format model fmt.

To_Date:(char,[fmt][nLsparams]
Converts a character string representing a date to a date value according to a fmt specified. If fmt is omitted the format is DD-MON-YY.

5.GENERAL FUNCTION:
These Functions work with any data type and pertain to using nulls

- NVL(expr1, expr2)

Converts a null value to an actual value

- NVL2(expr1, expr2, expr3)

If expr1 is not null, NVL2 returns expr2. If expr1 is null, NVL2 returns expr3. The argument expr1 can have any data type.

- NULLif(expr1, expr2)

Compares the two expressoinms and returns null if they are equal. Or the first expression. If they are not equal.

- Coalesce((expr1, expr2,..................,exprn)

Returns the first non-null expr expressions in the expression list

NVL Funtion
NVL converts a null to an actual value.
Syntax:

```
NVL(expr1,expr2)
```

Expr1 is the source value or expression that may contain null
Expr2 is the target value for converting the null.

Data types that can be used are at date, character and number.
Ex:

```
NVL(commisiion-pet,0)
NVL(hire_date, '01-JAN-97')
NVL(Job-id, 'No Job Yet')
```

Queries on Oracle Functions

1. Character Functions:

30.Display Employeesp Names In Lower Case:
Sql> Select Lower(Ename) From Emp;

31.Display EmployeesNames In Upper Case:
Sql> Select Upper(Ename) From Emp;

32.Display Employees Names In Proper Case:
Sql> Select Initcap(Ename) From Emp;

33.Display Length Of Your Name:
Sql> Select Length('Mak')"Myname" From Dual;

34.Extract Two Characters From The Second Position Of The String Dcet
Sql> Select Substr('Dcet',2,2) From Dual;

2. Number Functions

35.Display Employees name Of Employees Whose Salary Is Even No:
Sql> Select Ename From Emp Where Mod(Sal,2)=0;

3. Date Functions

35.Display Age In Days:
Sql> Select Trunc(Sysdate-To_Date('31-Aug-1991')) From Dual;

36.Display Age In Months:
```
Sql> Select Months_Between(Sysdate,'&Dob') From Dual;
Enter Value For Dob: 10-Jan-1998
Old    1: Select Months_Between(Sysdate,'&Dob') From Dual
New     1: Select Months_Between(Sysdate,'10-Jan-1998') From Dual
```

37.Find The Day Of Nearest Saturday After The Current Date:
Sql> Select Sysdate,Next_Day(Sysdate,'Saturday') From Dual;

38.Display The Date Of Three Months Before The Current Date:
Sql> Select Sysdate,Add_Months(Sysdate,-3) From Dual;

GROUP FUNCTIONS:
QUERIES ON AGGREGATE FUNCTIONS:

39. Display Total Number of Employees Working In A Company.

Sql> Select Count(Ename) From Emp;

40.Display Total Salary Paid To All Employees.
Sql> Select Sum(Sal) From Emp;
41.Display The Average Salary Of The Employees.
Sql> Select Avg(Sal) From Emp;

42.Display Maximum Salary Being Paid To A Clerk.
Sql> Select Max(Sal) From Emp Where Job='Clerk';

43.Display Total Salary Drawn By Analyst In Deptno.=20.
Sql> Select Sum(Sal) From Emp Where Job='Analyst' And Deptno=20;

5. COMPLEX CONDITION QUERY CREATION

1. Subquery (Nested Query): Sub query is also known as nested queries. A Sub query is usually a select query within one of the clause in another select query. Very powerful queries can be designed by using simple sub queries.

Sub queries are of two types.
a)Single Row Sub query:
A Sub query that returns only one row of data. It is also known as scalar Subquery.
b) Multiple Row Sub query:
A Sub query that returns more than one row of data

A. SINGLE ROW SUBQUERY

Syntax
```
select column list
from tablename
where columnname operator
(select columnnames
from tablenames where condition)
```
*Subquery must be enclosed in parentheses
*Order by clause cannot be used in subquery
*subquery is used on the right side of the condition
* relational operators are used in the outer query condition.

creating a table can be carried by using a subquery
Syntax
```
Create table table_name As select query;
```

An existing table can be populated with a sub queryi.e insert with in a sub query does not create a new table
Syntax
```
Insert into table_name(column list) Select column_names from table_names
    where condition;
```
Update using a subquery as
Syntax
```
update table_name
set column_name=value or expression where column operator(select sub query);
```

Delete using subquery

Syntax

delete from table_name where column_name=(select sub query);

B. MULTIPLE ROW SUBQUERY

Multiple row sub query returns more than one row. The operators used in single row sub query cannot be used in multiple row sub query

Special operators used in multiple row sub query are

in: equals to any value in a list

all: compares the given value to every value returned by the sub query

any or some: compare the given value to each value returned by the sub query

Any operator can be used in combination with other relational operators, it serves as or operator

As

<any means less than the maximum value

>any means higher than minimum value in the list

=any means equal to any value in the list (similar to in)

all operator can be used with relational operator it serves as and operator

>all means more than maximum value in the list

<all means less than the minimum value

=all is meaningless because no value can be equal to all values in the list

C.CORRELATED SUBQUERY

In Correlated sub query the inner query can reference columns from the outer sub query .the inner query is executed once for each row in the outer query.

exists and not exists operator

The Exist and Not exist can be used in correlated queries.

The Exist operator checks if the inner query returns at least one row it returns true or false.

Not exist check if the inner query does not return a row, it is opposite of exist operator.

Employees Database Nested Queries:

1)Display Dept Name Where At least Two Employees Are Working.

Sql> select dname from emp,dept where emp.deptno=dept.deptno and emp.deptno in (select deptno from emp group by deptno having count(deptno)>2);

2)Display The Names Of Employees Whose Salary Is Greater Than The Average Salary Of Employees.

sql> select ename from emp where sal>(select avg(sal) from emp);

3)Display The Emp Names Who Has The Highest Salary.

sql> select ename from emp where sal=(select max(sal) from emp);

5)Display Emplyee Names Working As Clerk And Caching Highest Salary.

sql> select ename from emp where sal in (select max(sal) from emp group by job having job='clerk');

6)Display Names Of Clerks Whose Salary Is Greater Than The Lowest Salary Of The Sales Man.
sql> select ename from emp where job='clerk' and sal>(select min(sal) from emp where job='salesman');

7)Display The Emp Names Who Earns Highest Salary In Respected Departments.
sql> select ename from emp where sal in (select max(sal) from emp group by deptno);

8)Display The Emp Names Who Earns Highest Salary In Job Group.
sql> select ename from emp where sal in (select max(sal) from emp group by deptno);

9)Display The Second Highest Salary Of Employee.
sql> select ename from emp where sal in (select max(sal) from emp group by job);

6. PL/SQL

There many disadvantages of SQL:
1. SQL does not have any procedural capabilities i.e.,SQL does not provide the programming techniques of condition checking looping and branching.
2. SQL statements are passed to the oracle engine one at a time.
While processing an SQL sentence if an error occurs, the oracle engine displays its own error messages.

PL/SQL is a superset of SQL,PL/SQL is a block structured language that enables developers to combine the power of ,SQL with procedural statements.
Advantages of PL/SQL:
1. PL/SQL sends an entire block of SQL statements to the oracle engine all in one go.
2. PL/SQL also permits dealing with errors as required, and facilitates displaying user-friendly messages when errors are encountered.
3. PL/SQL allows declaration and use of variables in block code. There variables can be used to store intermediate results of a query for later processing or calculate values and insert them into an oracle table later.
4. Application written in PL/SQL is portable to any computer hardware and os.

GENERIC PL/SQL BLOCK:
PL/SQL permits the creation of structured logical block of code that describe processes which have to be applied to data. A single PL/SQL block consists of a set of SQL statement clubbed together and passed to the oracle engine entirely.

PL/SQL block has definite structure, which can be divided into
1. Declare Section:
Code blocks starts with a declaration section in which memory variables and other oracle objects can be declared and if required initialize.
2. Begin Section:

It consists of a set of SQL and PL/SQL statements, which describe processes that have to be applied to table data, actual data manipulation, retrieval, looping and branching constructs are specified in this section.
3. Exception Section:
This section deals with handling of errors that arises during exception of data manipulation statement, which make up the PL/SQL code block.
4. End Section:
This marks the end of a PL/SQL block.

PL/SQL DATA TYPES:
The default data types that can be declared in PL/SQL are:

Number
Char
Date
Boolean

%type declares a variable or constant to have the same data type as previously defined variable or of a column in a table or in a variable.

PL/SQL VARIABLES
Variables in PL/SQL blocks are named variables. A variable mane must begin with a character and can be followed by a maximum of 29 characters.
Assigning values to variables
The assigning of a value to a variable can be done in two ways:
1) Using the assignment operator:=
2) Selecting or fetching table data values into variables.

Displaying message
DBMS-OUT PUT: This package that include a number of procedures and function
that accumulate information in a buffer so that it can be retrieved later. These function can also be used to display message.
Put-line
It put a piece of information in the package buffer followed by an end of line mark.It can be used to display a message.
To display message the server output should be on.

CONTROL STRUCTURES
The flow of control statement can be classified into following categories
1) Conditional control
2) Interactive control
3) Sequential control

1) CONDITIONAL CONTROL
PL/SQL allows the use of an IF statement to control the execution of a block of code.
Syntax

```
If condition then
   Action
ELSE condition then
   Action
ELSE
   Action
   ENDIF
```

2) ITERATIVE CONTROL

Iterative control indicates the ability to repeat or skip sections of code block. A loop marks of sequence of statement that has to be repeated. A conditional statement that controls the number of times a loop is executed always a company's loops

Syntax: Loop

```
        Loop
        <sequence of statement>
        End loop;
```

Syntax: While loop

```
        While condition
        Loop
                Action
        End loop
```

Syntax:For loop

```
                For variable in[Reverse] start..end
                Loop
                        <Action>
                End loop
```

3) SEQUENTIAL CONTROL

Goto statement changes the flow of control within PL/SQL block. This statement allows execution of a section of code which is not in the normal flow of control.

Syntax

```
Goto   <codeblockname>
```

PL/SQL PROGRAMS

1)<u>PL/SQL PROGRAM TO DEMONSTRATE ADDITION OF TWO NUMBERS:</u>

```
SQL> declare
  2  a number:=&a;
  3  b number:=&b;
  4  c number;
  5  begin
  6  c:=a+b;
  7  dbms_output.put_line(c);
  8  end;
  9  /
Enter value for a: 3
old   2: a number:=&a;
new   2: a number:=3;
Enter value for b: 7
old   3: b number:=&b;
new   3: b number:=7;
10
PL/SQL procedure successfully completed.
```

2) PL/SQL PROGRAM TO DEMONSTRATE WHETHER A NUMBER IS EVEN OR NOT

```
SQL> declare
  2  num number;
  3  begin
  4  num:=&num;
  5  if(mod(num,2)=0)then
  6  dbms_output.put_line('given number is even');
  7  else
  8  dbms_output.put_line('given number is odd');
  9  end if;
 10  end;
 11  /
Enter value for num: 4
old   4: num:=&num;
new   4: num:=4;
given number is even
PL/SQL procedure successfully completed.
```

3)PL/SQL PROGRAM TO DEMONSTRATE FIBBONACCI SERIES

```
SQL> declare
  2  a number;
  3  b number;
  4  c number;
  5  x number;
  6  i number;
  7  begin
  8  a:=&a;
  9  b:=0;
 10  c:=1;
 11  i:=3;
 12  dbms_output.put_line(b);
 13  dbms_output.put_line(c);
 14  while i<=a loop
 15  x:=b+c;
 16  dbms_output.put_line(x);
 17  b:=c;
 18  c:=x;
 19  i:=i+1;
 20  end loop;
 21  end;
 22  /
Enter value for a: 4
old   8: a:=&a;
new   8: a:=4;
0
1
1
2
```

```
PL/SQL procedure successfully completed.
```

4)PL/SQL PROGRAM TO DEMONSTRATE FACTORIAL OF A NUMBER

```
SQL> declare
  2   n number;
  3   i number;
  4   begin
  5   n:=&n;
  6   i:=n-1;
  7   loop;
  8   n:=n*i;
  9   i:=i-1;
 10   exit when i=1;
 11   end loop;
 12   dbms_output.put_line('fact of given   number is');
 13   dbms_output.put_line(n);
 14   end;

SQL> /
Enter value for n: 5
old     5: n:=&n;
new     5: n:=5;
fact of given   number is
120

PL/SQL procedure successfully completed.
```

5) PL/SQL PROGRAM TO DEMONSTRATE WHETHER A NUMBER IS PRIME OR NOT.

```
  SQL>   declare
  2   n number;
  3   a number;
  4   i number(2);
  5   begin
  6   n:=&n;
  7   i:=n/2;
  8   loop
  9   if(mod(n,i)=0)then
 10   dbms_output.put_line('given number is not prime');
 11   else
 12   dbms_output.put_line('given number is prime');
 13   exit;
 14   end if;
 15   i:=i-1;
 16   exit when i=a;
 17   end loop;
 18* end;
SQL> /
Enter value for n: 5
```

```
old     6: n:=&n;
new     6: n:=5;
given number is prime

PL/SQL procedure successfully completed.
```

6)PL/SQL PROGRAM TO CREATE A TABLE.

```
   SQL>   declare
   2   n number;
   3   i number:=1;
   4   a number;
   5   begin
   6   n:=&n;
   7   loop
   8   a:=n*i;
   9   dbms_output.put_line(n||'*'||i||'='||a);
  10   i:=i+1;
  11   exit when i=11;
  12   end loop;
  13* end;
SQL> /
Enter value for n: 4
old     6: n:=&n;
new     6: n:=4;
4*1=4
4*2=8
4*3=12
4*4=16
4*5=20
4*6=24
4*7=28
4*8=32
4*9=36
4*10=40

PL/SQL procedure successfully completed.
```

PROCEDURE AND FUNCTIONS

Procedures and functions are made up of

Declarative part

Executable part

An optional exception-handling part

Declarative part

The declarative part may contain the declaration of cursors, constants, variables, exceptions and sub programs. These objects are local to the procedures or functions.

Executable part:

The executable part is a PL/SQL block consisting of SQL and PL/SQL. Statements that design values, control executions and manipulate data .Actions are coded here. The data that is to be returned back to the calling environment is also returned from here.

Exception handling part:

This part contain code that deal with exception that may be raised during the execution of code in there executable part.

Advantages of using procedures or functions

- Security
  Stored procedures and functions can help enforce data security.
- Performance:
  It improves the performance of DB system by reducing information sent over the network, removing compilation step to execute the code
- Memory allocation
  The amount of memory used reduces as stored procedures or functions of have shared memory capabilities. Only one copy of procedure needs to be loaded for execution by multiple users.
- Productivity:
  By writing procedures and functions, redundant coding can be avoided increasing productivity.

Procedures versus functions:

The differences between procedures and functions are a function must return a value back to the caller. A function can return only one value to the calling SQL block. By defining multiple out parameters in a procedure ,multiple values can be passed to the caller.

SYNTAX(procedure):

```
Create or replace procedure (schema) <procedure N.
(<Argument> {IN,OUT,INOUT} <data type >…..)
{IS, AS}
Variable declaration;
Constant declaration;
Begin
PL/SQL program body;
```

```
                    Exception
            Exception PL/SQL block;
END;
```

SYNTAX (function)

```
          Create or replace function [schema]<function var (<Argument>1N<Datatype>…)
          Return<Datatype> {IS, AS}
          Variable declaration;
          Constant declaration;
          Begin
          Pl/sql program body
          Exception
          Exception pl/sql block;
          End;
```

<u>Deleting a stored procedure/Function</u>

Syntax:

```
         Drop procedure<procedure Name>;
         Drop function<function Name>;
```

7. PROCEDURES

A. PROCEDURE FOR DISPLAYING SALARY OF AN EMPLOYEE.

```
SQL> create or replace procedure displaysal21(name emp21.ename%type) is vsal
emp21.sal%type;
  2   begin
  3   select sal into vsal from emp21 where ename=name;
  4   dbms_output.put_line('the sal of'||name||'is'||vsal);
  5   end;
  6   /

Procedure created.

SQL> execute displaysal21('KING');
the sal ofKINGis5000
PL/SQL procedure successfully completed.
```

B. PROCEDURE FOR INCREMENTING SALARY OF AN EMPLOYEE.

```
SQL> create or replace procedure proc21(name emp21.ename%type,inc number) is sal
emp21.sal%type;
  2   begin
  3   update emp21 set sal=sal+inc where ename=name;
  4   dbms_output.put_line('the sal of'||name||'is'||sal);
  5   end;
  6   /
Procedure created.
```

```
SQL> execute proc21('SMITH',600);
the sal ofSMITHis 600
PL/SQL procedure successfully completed.
```

C. PROCEDURE TO REVERSE A STRING

```
SQL>    create or replace procedure ReverseOf21(input IN varchar2) IS
   2            reverse varchar2(50);
   3                begin
   4            for i in reverse 1..length(input) LOOP
   5                    reverse := reverse||''||substr(input, i, 1);
   6            end loop;
   7            dbms_output.put_line(reverse);
   8              end;
   9               /
Procedure created.

SQL> execute ReverseOf21('apple');
leppa
PL/SQL procedure successfully completed.
```

D.PROCEDURE FOR UPDATING SALARY

```
SQL> desc empproc21;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 EMPNO                                              NUMBER(38)
 ENAME                                              VARCHAR2(20)
 SAL                                                NUMBER(38)
 DEPTNO                                             NUMBER(38)

create or replace procedure update_sal21(id in number,p_sal in number)
      as
    begin
         update empproc21 set sal = p_sal where empno = id;
dbms_output.put_line('the salary of employe with id'|| id||' updated to '|| p_sal);
     end;/

SQL> select * from empproc21;

     EMPNO ENAME                       SAL     DEPTNO
---------- -------------------- ---------- ----------
         1 abc                       20800         10
         1 abc                       30800         10

SQL> execute update_sal21(1,99999);
PL/SQL procedure successfully completed.

SQL> select * from empproc21;

     EMPNO ENAME                       SAL     DEPTNO
---------- -------------------- ---------- ----------
```

```
        1 abc                       99999          10
        1 abc                       99999          10
```

8. PROCEDURE FOR DATA VALIDATION

```
SQL> create table empproc21(empno int,ename varchar(20),sal int,deptno int);
Table created.

SQL> create procedure pesal21(p_empno empproc21.empno%type,p_ename
empproc21.ename%type,p_sal empproc21.sal%type,p_deptno empproc21.deptno%type)as
invalid_sal exception;
  2   begin
  3   if p_sal<100 or p_sal>100000 then
  4   raise invalid_sal;
  5   end if;
  6   insert into empproc21(empno,ename,sal,deptno)
values(p_empno,p_ename,p_sal,p_deptno);
  7   commit;
  8   exception
  9   when invalid_sal then
 10   raise_application_error(-20001,'salary must be in between 100 and 100000');
 11   end;
 12   /
Procedure created.

SQL> save pesal21;
Created file pesal21.sql

SQL> execute pesal21(1,'abc',10,10);
BEGIN pesal21(1,'abc',10,10); END;

*
ERROR at line 1:
ORA-20001: salary must be in between 100 and 100000
ORA-06512: at "SCOTT.PESAL21", line 10
ORA-06512: at line 1
```

9.FUNCTIONS

A. FUNCTION TO CONVERT A CURRENCY.

```
SQL> create or replace function doll21(d number) return number is
  2   begin
  3   return(d*70);
  4   end;
  5   /

Function created.

SQL> select doll21(10) from dual;

  DOLL21(10)
----------
        700
```

B. FUNCTION TO CONVERT TEMPERATURE TO FAHRENHEIT.

```
SQL> create or replace function fah21(f number) return is
  2   c number;
  3   begin c:=5/9*(f-32);
  4   return(c);
  5   end;
  6   /

Function created.

SQL> select fah21(30) from dual;

   FAH21(30)
----------
-1.1111111
```

C. FUNCTION TO FIND A NUMBER IS PRIME OR NOT.

```
  SQL> create or replace function pr21(n number) return varchar2 is
  2   b varchar2(20):='prime';
  3   begin
  4   for i in 2..n-1 loop
  5   if mod(n,i)=0 then
  6   b:='not prime';
  7   end if;
  8   end loop;
  9   return(b);
 10* end;
SQL> /

Function created.
```

```
SQL> select pr21(5) from dual;

PR21(5)
--------------------------------------------------------------------------------
prime

SQL> select pr21(4) from dual;
PR21(4)
--------------------------------------------------------------------------------
not prime
```

D. FUNCTION TO REVERSE A STRING.

```
SQL> create or replace function rev21(st varchar2) return varcahr2 is
  2   k number;
  3   s varchar2(20);
  4   begin
  5   k:=length(st);
  6   for i in reverse 1..k loop
  7   s:=s||substr(st,i,1);
  8   end loop;
  9   return(s);
 10   end;
 11   /
Function created.

SQL> select rev21(3336875) from dual;
REV21(3336875)
--------------------------------------------------------------------------------
5786333
```

10. CURSORS

The oracle engine used a work area for its internal processing in order to execute an SLQ statement. They work are is private to SQL operation and is called cursor. The data that is stored in the cursor that is called the active data set. The values that are retrieved from the table are held in a cursor opened in memory by the oracle engine. This data is then transferred to the client machine via network. In order to hold this data, a cursor is opened at the client end.

Types of cursors

Cursor is classified depending on the circumstances under which they are opened. If the oracle engine opened, a cursor for its internal processing it is known as implicit cursor.
A cursor opened for processing data through a PL/SQL block on demand is known as explicit cursor.
General cursor attributes
When an oracle engine creates an implicit or explicit cursor, cursor control variables are also created to control the execution of cursor.
These are a set of four system variables which keep track of the current status of cursor.
The cursor variables can be accessed and used in PL/SQL code block.

ATTRIBUTR NAME	DESCRIPTION
%ISOPEN	Return true if cursor is opened, False otherwise
%FOUND	Return true If record was fetched success, False otherwise
%NOT FOUND	Return True If record was not fetched success, False otherwise
%ROW COUNT	Return Number of records processed from the cursor

EXPLICT CURSOR:
When individual records in a table have to be processed inside a PL/SQL code a cursor is used. Thus cursor will be declared and mapped to an SQL query in the declare section of the PL/SQL block and used within its executable section.

Cursor declaration:
A cursor must be named and a query has to be mapped to it.Three commands are used to control the cursor subsequently are OPEN, FETCH, and CLOSE

Open: Initialization of a cursor takes place via the open statement. It defines a private SQL area named after the cursor name. Executes the query associated with cursor. Retrieves table data and populate them in private SQL area in memory. Sets the cursor row pointer in Active Data Sets to the first record.
Syntax for creating a cursor:

```
cursor cursor_name is select statement:
```

Syntax for open:

```
Open cursor name.
```

Fetch: The fetch statement retrieves the rows from the active set opened in the server into memory enables declared in PL/SQL code block on client one row at a time. Each time the fetch is executed. The cursor pointer is advanced to the next row in the Active Data Set.
Syntax for fetch:
Fetch cursor name into variable1, variable2………

CLOSING A CURSOR:

The close statement disables the cursor and the active set becomes undefined. This will release the memory occupied by the cursor and the data set on the client and on the server.
Syntax for close:
Close cursor name;

Parameterized cursors:
The contents of the parameterized cursor will constantly change depending upon the value passed to the parameter.

```
        Syntax:
                Cursor or cursor_name(variable name Datatype) is
                <select statement>
Opening:
                Open cursor_name(value/variable/expression)
```

10. CURSORS USING PROCEDURE

```
1)SQL> create table course021(cid number(10), cname varchar2(20) ,fee number(10),duration number(10),stat
us varchar2(20),primary key(cid));
Table created.

SQL> select * from course021;
no rows selected

SQL> insert into course021 values('&cid','&cname','&fee','&duration','&status');
Enter value for cid: 101
Enter value for cname: mca
Enter value for fee: 250000
Enter value for duration: 3
Enter value for status: offered
old   1: insert into course1 values('&cid','&cname','&fee','&duration','&status')
new   1: insert into course1 values('101','mca','250000','3','offered')
1 row created.

SQL> select * from course021;

       CID CNAME                       FEE   DURATION STATUS
---------- -------------------- ---------- ---------- --------------------
       101 mca                      250000          3 offered
       102 ma                        30000          2 offered

SQL> create table student021(rno number(10),dob date,cid number(10),name varchar2(20),primary key(rno),foreign key(cid) references course1(cid));
Table created.
SQL> select * from student021;
       RNO DOB              CID NAME
---------- --------- ---------- --------------------
         1 01-JAN-80        101 aaa
         2 06-FEB-81        101 sar
         3 15-MAR-81        101 fds
```

```
            4 16-AUG-81           101 xyz
            5 10-SEP-81          101 asf
            6 10-DEC-80           102 saf
            7 05-MAY-80            103 faadf
7 rows selected.
```

Creation of procedure

```
SQL> create or replace procedure p021(vcid in number)
  2   is
  3   begin
  4   update course021 set status='not offered' where cid=vcid;
  5   end;
  6   /

Procedure created.
```

Creation of cursor

```
SQL> save p1.sql;
Created file p1.sql
SQL> declare
  2   cursor c is select cid from student021 group by cid having count(cid)<5;
  3   vcid student021.cid%type;
  4   begin
  5   open c;
  6   loop
  7   fetch c into vcid;
  8   exit when c%notfound;
  9   p1(vcid);
 10   end loop;
 11   close c;
 12   commit;
 13   end;
 14   /
PL/SQL procedure successfully completed.
SQL> select * from course021;

       CID CNAME                          FEE     DURATION STATUS
---------- -------------------- ---------- ---------- --------------------
       101 mca                         250000              3 offered
       102 ma                           30000              2 notoffered
```

11. CURSOR WITHOUT PROCEDURE

```
SQL> declare
  2   cursor c is select cid from student021 group by cid having count(cid)<3;
  3   vcid student021.cid%type;
  4   begin
  5   open c;
  6   loop
  7   fetch c into vcid;
  8   exit when c%notfound;
  9   update course021 set status='not offered' where cid=vcid;
 10   end loop;
 11   close c;
 12   commit;
 13   end;
 14   /

PL/SQL procedure successfully completed.

SQL>select * from course021;

       CID CNAME                       FEE   DURATION STATUS
---------- -------------------- ---------- ---------- --------------------
       101 mca                      250000          3 offered
       102 ma                        30000          2 notoffered
```

12.CURSORS WITHIN PROCEDURES TO FETCH EMPLOYEE NAME

```
SQL> create or replace procedure get_emp_names21 (dept_num in number) is
  2      emp_name        varchar2(10);
  3      cursor        c1 (depno number) is
  4                      select ename from emp
  5                         where deptno = depno;
  6   begin
  7      open c1(dept_num);
  8      loop
  9          fetch c1 into emp_name;
 10          exit when c1%notfound;
 11          dbms_output.put_line(emp_name);
 12      end loop;
 13      close c1;
 14   end;
 15   /

Procedure created.

SQL> execute Get_emp_names21(10);
CLARK
KING
MILLER
Shakir

PL/SQL procedure successfully completed.
```

13. PACKAGE

A package is an encapsulated collection of related program objects (for example, procedures, functions, variables, constants, cursors, and exceptions) stored together in the database.
Using packages is an alternative to creating procedures and functions as standalone schema objects. Packages have many advantages over standalone procedures and functions. For example, they:

- Let you organize your application development more efficiently.
- Let you grant privileges more efficiently.
- Let you modify package objects without recompiling dependent schema objects.
- Enable Oracle Database to read multiple package objects into memory at once.
- Can contain global variables and cursors that are available to all procedures and functions in the package.
- Let you **overload** procedures or functions. Overloading a procedure means creating multiple procedures with the same name in the same package, each taking arguments of different number or data type.

```
create or replace package pkg_test1
   as
      function getArea (i_rad NUMBER) return NUMBER;
        procedure p_print (i_str1 VARCHAR2 :='hello',
                           i_str2 VARCHAR2 :='world',
                           i_end VARCHAR2   :='!' );
   end;
/

create or replace package body pkg_test1
   as
      function getArea (i_rad NUMBER)return NUMBER
      is
          v_pi NUMBER:=3.14;
       begin
          return v_pi * (i_rad ** 2);
       end;
       procedure p_print(i_str1 VARCHAR2 :='hello',
                         i_str2 VARCHAR2 :='world',
                         i_end VARCHAR2   :='!' )
      is
       begin
           DBMS_OUTPUT.put_line(i_str1||','||i_str2||i_end);
       end;
   end;
/
SQL> select pkg_test1.getArea(34) from dual;
PKG_TEST1.GETAREA(34)
--------------------
              3629.84

SQL> execute pkg_test1.p_print;
hello,world!          PL/SQL procedure successfully completed.
```

14. DATABASE TRIGGERS

Database triggers are stored in the database and is called automatically when a triggering event occurs. A user cannot call a trigger explicitly. Triggering event is based on data manipulation language (DML) statement such as insert, update or delete. A trigger can be created to fire before or after the triggering event. Execution of triggers is also known as firing the trigger.

Syntax:

```
Create [or replace] trigger trigger_name
Before [after |instead of triggering event on table|]
   [for each row]
   [when condition]
Declare
   Declaration statement
Begin
    Executable statement
Exception
    Exception handling statement
End;
```

Triggers are very useful for generating values for derived column keeping track of table access preventing invalid entries, performing validity checks and maintaining security.

Restrictions on triggers

- Triggers cannot use transaction control language statements such as commit, rollback or savepoint.
- Variables in a triggers cannot be declared with long or long raw data type.-Instead of trigger is based on view and it is a row trigger.

A. TRIGGER TO RESTRICT RECORDS TO THREE:

```
SQL> create table studentt21(sname varchar(20),branch varchar(10));
Table created.

SQL> create or replace trigger tristudent21
  2   after insert
  3   on studentt21
  4   declare
  5   c integer;
  6   begin
  7   select count(*) into c
  8   from studentt21;
  9   where branch='CSE';
 10   if(c>3)
 11   then   raise_application_error(-20503,'cannot cross more than three records');
 12   end if;
 13   end;
 14   /
Trigger created.
```

```
SQL>insert into studentt21 values(‘&sname’,’&branch’);
Enter value for sname:aaa
Enter value for branch:cse
Old 1:insert into studentt21 values(“&sname’,’&branch’)
New1:insert into studentt21 values(‘aaa’,’cse’)
1 row created.
SQL>/
Enter value for sname:bbb
Enter value for branch:cse
Old 1:insert into studentt21 values(“&sname’,’&branch’)
New1:insert into studentt21 values(‘bbb’,’cse’)
1 row created.
SQL>/
Enter value for sname:ccc
Enter value for branch:cse
Old 1:insert into studentt21 values(“&sname’,’&branch’)
New1:insert into studentt21 values(‘ccc’,’cse’)
1 row created.

SQL>/
Enter value for sname:ddd
Enter value for branch:cse
Old 1:insert into studentt21 values(“&sname’,’&branch’)
New1:insert into studentt21 values(‘ddd’,’cse’)

ERROR at line 1:
ORA-20508:cannot cross more than three records
ORA-06512:at”IT1.TRISTUDENT21”,line 8
ORA-04088:error during execution of trigge’IT1.TRISTUDENT21’
```

B. TRIGGER FOR RESTICTION OF SALARIES OF COMPANY TO NOT EXCEED MORE THAN 80000.

```
SQL>create table emp7021(ename char(20),salary integer);
Table created.

SQL> create or replace trigger trg021
  2   after insert
  3   on emp7021
  4   declare
  5   c integer;
  6   begin
  7   select sum(salary) into c
  8   from emp7021;
  9   if(c>80000)
 10   then   raise_application_error(-20504,'cannot cross 80000');
 11   end if;
 12   end;
 13   /
```

```
Trigger created.

SQL> insert into emp7021 values('&ename','&salary');
Enter value for ename: dfs
Enter value for salary: 7000000;
old     1: insert into emp7021 values('&ename','&salary')
new     1: insert into emp7021 values('dfs','7000000;')
insert into emp7021 values('dfs','7000000;')
                                      *
ERROR at line 1:
ORA-01722: invalid number
```

15. VIEWS

A view is an oracle object that gives the user a logical view of data from an underlying table or tables. You can restrict what users can view by allowing them to see only a few columns from a table. Views can be either simple view or complex view.

a)Simple view:
It is based on a one table. It can contain group function. Data manipulation is always possible.
b)Complex view:
It is based on one or more tables. It may contain group function. Data manipulation is not always possible.

Syntax:

```
        create [or replace] view view_name [column aliases]
        as <query expression>
        [with check option [constraints constraints_name]]
        [with read only]
```

- Every exception can use all clauses except order by.
- with check option applies to the where clause condition in the sub query. It allows insertion and updating of rows based on the condition that satisfies the view.
  Constraints name can be given to with check option.
- The read only option is to make sure that the data in the underlying table are not changed through the view.
- user_view table contains data about views.
  select view_name from user_view;
  This will give the list of all view_names.
- altering a view
  When you alter an underlying table, the view becomes invalid. You need to recompile that view to make it valid again. The alter view statement is used to recompilations of
a view.

  Alter view deptsal compile;

EXAMPLE FOR CREATION OF A VIEW:

```
SQL> create view c021(courseid,coursename) as select cid,cname from course021;

View created.
```

16. CREATION OF INDEX

An index is another oracle object that is used for faster retrieval of rows from a table. Index can be created explicitly by using the create index statement. Once an index is created the user does not have to open or use the index with a command or a statement the oracle server uses the index to search for a row rather than scanning through the entire table. Indexing reduces both search time and disk I/O.
-All indexes are maintained separately from the tables on which they are bared.
-Creating and removing an index does not affect the table at all.
-When a table is dropped all indexes based on that table are also removed.

Implicit indexes:
Implicit indexes are created when the primary key or unique constraints is defined such indexes get the name of the constraints.

Explicit indexes:
A user can create explicit indexes based on non primary key or non unique columns or on any combinations of columns for faster table access.

Simple indexes:
It is defined on a single column.

Composite indexes:
It is based on the combination of columns.

Syntax-for creating index

```
create index index_name
on table_name(column1,column2……);
```

-Information of indexes is stored in user_indexes table.
-ex:

```
create index stuidx
on student(last, first)
```

Index created

```
select index_name,table_name
from user_indexes
where table_name='student'
```

EXAMPLE FOR INDEX CREATION:

```
SQL> create index sid021 on student021(rno,dob);

Index created.
```

17. DEMONSTRATION OF COMMIT,SAVEPOINT AND ROLLBACK COMMANDS

Transactions:
In oracle the transaction starts with the first executable SQL statement that you issue.

Transaction ends when one of the following occurs
->A comment or rollback transaction control statement is used.
->A DDL (create, alter, drop, rename or truncate) statement is executed (automatic commit)
->A DCL (grant or revoke) statement is execution (automatic commit)
->A user properly exists (commit)
->The system crashes (rollback)

-Once a transaction ends, the new transaction starts with your next executable SQL statement.

-There is an environment variable auto commit by default. It is set to OFF a user can set it to ON or immediate by typing

```
Set autocommit ON
Set autocommit immediate
```

-When autocommit is on every DML statement is written to the disk as soon as it is executed. Every DML statement is committed implicitly and no rollback occurs with autocommit.

-autocommit can be toggled back to off for an explicit commit.

-If system crashes any statement after the last commit are rolled back, so partial changes to tables are not permanently written.
TCL (Transaction Control Language):

There are three TCL statements
<u>a)commit:</u>
Ends the current transaction and write all changes permanently to the disk.

<u>b)savepoint n:</u>
Makes a point in the current transaction.

<u>c)rollback [To savepoint n]:</u>
Ends the current transaction by undoing all changes to the last commit, if a 'To save point' clause is not used. If its used it rollbacks the save point, removing the save point and any changes after the save point, but it does not end the transaction.

<u>EXAMPLE:</u>

```
desc studentt21;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 RNO                                                NUMBER(38)
 NAME                                               VARCHAR2(30)

SQL> select * from studentt21;
no rows selected

SQL> insert into studentt21 values('&rno','&name');
Enter value for rno: 1
Enter value for name: AAA
old   1: insert into studentt21 values('&rno','&name')
new   1: insert into studentt211 values('1','aaisha')

1 row created.

SQL> select * from studentt21;

       RNO NAME
---------- ------------------------------
         1 AAA
         2 BBB
         3 CCC

SQL> savepoint a;

Savepoint created.

SQL> delete studentt21;

3 rows deleted.

SQL> rollback to savepoint a;

Rollback complete.

SQL> select * from studentt21;

       RNO NAME
---------- ------------------------------
         1 AAA
         2 BBB
         3 CCC
```

18. PRIVILEGES

A user access needs to be controlled in a shared multi-user oracle environment. Security can be classified into two types.

a)System Security:

system security defines access to the database at the system level. It is implemented by assigning a username and password allocation disk spaces and providing a user with the ability.

b)Database Security:

Database security defines a user access to various objects and the task a user can perform on them.

-DBA can create users with create user statement. Once a user is created and a password assigned the user needs privileges to do anything.

Syntax-for creating user

```
create user username [profile profilename]
identified by password
[default tablespace tablespacename]
[temporary tablespace tablespacename]
[password expire] [account unlock];
```

-A user needs at minimum two roles connected and resource by

```
grant connect to username
grant resource to username
```

-A user can change his own password with the password command.

-DBA assigns privileges based on the level of a views or on a user or on a users needs these levels are called Roles.

-Roles are created by DBA.

-Object privileges specifies what a user can do with a database object such as table, sequence or view. These are basic 11 objects privileges and each object has a set of privileges out of a total of 11 privileges.

1) alter (table, sequence)
2) insert (table, view)
3) update (table, view)
4) delete (table, view)
5) select (table, view, and sequence)
6) references (table, view)
7) index (table)
8) execute (procedure, function, and package)
9) under (view, user defined type)
10) read (directory)
11) write (directory)

Grant:

-A user has his objects in his own schema. An owner has all possible privileges on the owner's objects. A user (owner) can grant privileges on objects from his own schema to other users or rolls.

Syntax:

```
Grant objectprivileges (columnnames)/All
On objectname
To user/role/public
```

```
        [With grant option];
```

-Column on which privileges are to be granted is specified by columnnames.
-Public grants privileges to all users.
-With grants option clause allows the guarantee to grant privileges to other users and rolls.

```
    Ex:
         a) Grant select
              On dept
              To xman;

         o/p-Grant succeeded

         b) Grant select, insert, update
               On emp
               To student, xman
               With grant option;

         o/p-Grant succeeded
```

Revoke:
The revoke statement takes privileges not only from the granter but also from the users granted privileges by the granter. If with grant option is given.
Syntax:

```
         Revoke object privileges
         On objectname
         From user/role/public
         [Cascade constraints]
```

-Ex:

```
         Revoke ALL
         On employee
         From xman
```

<u>PRIVILEGES(GRANT,REVOKE) EXAMPLE</u>

STAFF USER:

```
SQL>grant select on course01 to scott;
Grant succeeded
```

SCOTT USER:

```
SQL>insert into course01 values('&cid','&cname','&fee','&duration','&status');
Enter value for cid:45
Enter value for cname:sam
Enter value for fee:5678
Enter value for duration:4
Enter value for status:offered
Old 1:insert into course01 values('&cid','&cname','&fee','&duration','&status')
New 1:insert into course01 values('45','sam','5678','4','offered')

ERROR at line 1:
```

OA-01081:insufficient privileges

STAFF USER:

Revoking ganted permission

SCOTT USER:

```
SQL>revoke select on course01 from scott;
Revoke succeded.
```

19.LOCKING

-Oracle uses automatic locking in the least restrictions to provide sharing of data with integrity.
-Oracle has two lock classes
a) Share lock
b) Exclusive lock

-Shared mode lock is used when users requires only a read on the data item.
-Exclusive mode lock is used when users want to insert, update and delete data items.

Syntax:

```
Lock table table_name in Share/Exclusive mode nowait
```

-Ex:

```
To lock emp in Exclusive mode
 Lock table emp in Exclusive mode
```

Releasing locks:
A lock is released when either of the following occurs
->The transactions is committed successfully using commit.
->Rollback is performed
->Rollback to a savepoint will release lock set after the specified savepoint.

Locking rows for update:
-When DML query changes are done rows are locked until changes are committed.
-Suppose we want to view rows and also change them i.e. we want to lock rows for future changes we use select for update of statement for such manual lock.

Syntax:

```
select columnnames
from tablename
[where condition]
for update of columnnames
[nowait];
```

-Use of columnnames in for update of does not that the locking is at the column level. The entire row is locked. Column names are just used for information.
-Nowait clause, tell the user instantaneously if another user has already locked the row if it is omitted (Nowait clause) then we have to wait for any rows that have been locked by other application to be released, you will not be able to do any processing on those rows until then.

-Ex:

```
select ename, salary, commission
from emp
where cid=544
for update of salary, commission nowait
```

SECURITY:
(a) TABLE LOCKING:

```
CLIENT A:
SQL>lock table student01 in exclusive mode nowait;

Tables(s) locked.


CLIENT B:
SQL>delete from student01;


SQL not responding.

CLIENT A:
SQL>rollback;
Rollback completed.

CLIENT B:

All rows deleted.
```

20. FORMS AND REPORTS:

(A)CREATION OF LIBRARY INFORMATION ,PAYOLL,STUDENT:

Creation of library information table?

```
SQL>create table libinfo01(studentname char(20),idno integer,branch
char(20),noofcardsissued integer,challan char(10),booktitle char(20),author char(20),edition
integer,publication char(20);

Table created.
```

Creation of payroll information table?

```
SQL>create table payinfo01(name char(20),job char(10),basic integer,hr integer,ta integer,pf
integer,pt integer);

Table created.
```

Creation of student information table?

```
SQL>create table studinfo01(studentname char(10),studentid integer,branch char(10),year
integer,semester integer,address char(10));

Table created.
```

(B)INSERTING DATA INTO TABLE:

```
SQL>insert into libinfo01
values('&studentname','&idno','&branch','&noofcardsissued','&challan','&booktitle','&au
thor','&edition','&publication');

Enter value for studentname:aaa
Enter value for idno:123
Enter value for branch:it
Enter value for noofcardsissued:3
Enter value for challan:y
Enter value for booktitle:dbs
Enter value for author:Abraham
Ente value for edition:3
Enter value for publication:mcgawhill

Old 1: insert into libinfo01
values('&studentname','&idno','&branch','&noofcardsissued','&challan','&booktitle','&au
thor','&edition','&publication');
```

```
New 1: : insert into libinfo01
values('aaa','123','it','3','y','dbs','abraham','3','mcgrawhill');

1 row created


SQL>insert into payinfo01 values('&name','&job','&basic','&hr','&ta','&pf','&pt');

Enter value for name:bbb
Enter value for job:manager
Enter value for basic:2000
Enter value for hr:300
Enter value for ta:566
Enter value for pf:600
Enter value for pt:300

Old 1: insert into payinfo01 values('&name','&job','&basic','&hr','&ta','&pf','&pt');

New 1: insert into payinfo01 values('bbb','manager','2000','300','566','600','300');

1 row created
```

22. CREATION OF SMALL DATABASE APPLICATION

A. CREATION OF SAILORS DATABASE

CREATION OF SAILORS TABLE

```
SQL> create table sailors21(SID int,Sname varchar(20),rating integer,age float,primary
key(SID));
Table created.
SQL> desc sailors21;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 SID                                       NOT NULL NUMBER(38)
 SNAME                                              VARCHAR2(20)
 RATING                                             NUMBER(38)
 AGE                                                FLOAT(126)
```

CREATION OF BOATS TABLE

```
SQL> create table boats21(bid integer,bname varchar(20),color varchar(20),primary
key(bid));
Table created.

SQL> desc boats21;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 BID                                       NOT NULL NUMBER(38)
 BNAME                                              VARCHAR2(20)
 COLOR                                              VARCHAR2(20)
```

CREATION OF RESERVES TABLE

```
SQL> createtable reserves21(sid integer,bid integer,day date,foreign key(sid) references
sailors21(sid),foreign key(bid)references boats21(bid));
Table created.

SQL> desc reserves21;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 SID                                                NUMBER(38)
 BID                                                NUMBER(38)
 DAY                                                DATE
```

B. INSERTION OF VALUES INTO SAILORS TABLE:

```
SQL> insert into sailors01 values('&sid','&sname','&rating','&age');
Enter value for sid: 22
Enter value for sname: dustin
Enter value for rating: 7
Enter value for age: 46.0
old   1: insert into sailors01 values('&sid','&sname','&rating','&age')
new   1: insert into sailors01 values('22','dustin','7','46.0')

1 row created.
```

CONTENTS OF SAILORS TABLE:

```
SQL> select * from sailors01;

       SID SNAME                    RATING        AGE
---------- -------------------- ---------- ----------
        22 dustin                        7         46
        29 brutus                        1         33
        31 lubber                        8       55.5
        32 andy                          8       25.5
        58 rusty                        10         35
        64 horalt                        7         35
        71 zorba                        10         16
        85 art                           3       25.5
        95 bob                           3       63.5
       74 horatio                    9       35.0


    7   rows selected
```

INSERTION OF VALUES INTO BOATS TABLE:

```
SQL> insert into boats01 values('&bid','&bname','&color');
Enter value for bid: 101
Enter value for bname: interlake
Enter value for color: blue
old   1: insert into boats01 values('&bid','&bname','&color')
new   1: insert into boats01 values('101','interlake','blue')
```

```
1 row created.

CONTENTS OF BOATS TABLE:
SQL> select * from boats01;

       BID BNAME                COLOR
---------- -------------------- ----------
       101 interlake            blue
       102 interlake            red
       103 clipper              green
       104 marine               red

       4 rows selected
```

<u>INSERTION OF VALUES INTO RESERVES TABLE</u>:

```
 SQL> insert into reserves01 values('&sid','&bid','&day');
EntSer value for sid: 22
Enter value for bid: 101
Enter value for day: 10-oct-1998
old   1: insert into reserves01 values('&sid','&bid','&day')
new   1: insert into reserves01 values('22','101','10-oct-1998')

1 row created.
```

```
CONTENTS OF RESERVES TABLE:
SQL> select * from reserves01;

       SID        BID DAY
---------- ---------- ---------
        22        101 10-OCT-98
        22        102 10-OCT-98
        22        103 10-AUG-98
        22        104 10-JUL-98
        31        102 11-OCT-98
        31        103 11-JUN-98
        31        104 11-DEC-98
        64        101 09-MAY-98
        64        102 09-AUG-98
        74        103 09-AUG-98

10 rows selected.
```

C. QUERIES ON SAILOR DATABASE

<u>CONTENTS OF SAILORS TABLE:</u>

SQL> select * from sailor21;

```
       SID SNAME                          RATING        AGE
---------- -------------------- ---------- ----------
        22 dustin                              7            46
        29 brutus                              1            33
        31 lubber                              8          55.5
        32 andy                                8          25.5
        58 rusty                              10            35
        64 horalt                              7            35
        71 zorba                              10            16
        85 art                                 3          25.5
        95 bob                                 3          63.5
        74 horatio                             9            35
```

CONTENTS OF BOATS TABLE:

SQL> select * from boats21;

```
       BID BNAME                 COLOR
---------- -------------------- ----------
       101 interlake            blue
       102 interlake            red
       103 clipper              green
       104 marine               red
```

CONTENTS OF RESERVES TABLE:

SQL> select * from reserves21;

```
       SID        BID DAY
---------- ---------- ---------
        22        101 10-OCT-98
        22        102 10-OCT-98
        22        103 10-AUG-98
        22        104 10-JUL-98
        31        102 11-OCT-98
        31        103 11-JUN-98
        31        104 11-DEC-98
        64        101 09-MAY-98
        64        102 09-AUG-98
        74        103 09-AUG-98
```

Sailors Data Base Queries:

1) Find the sids of sailors who have reserved a red boat.

```
sql> select r.sid from boats21 b,reserves21 r where b.bid=r.bid and b.color ='red';
```

2) Find the color of the boats reserved by lubber.

```
sql> select b.color from sailor21 s,reserves21 r,boats21 b where s.sid=r.sid and r.bid=b.bid
and s.sname='lubber';
```

3) Find the names of sailors who have reserved atleast one boat.

sql> select s.sname from sailor21 s,reserves21 r where s.sid=r.sid;

4) Find the ages of sailors whose name begins and ends with b and has atleast three characters.
sql> select s.age from sailor21 s where s.sname like 'b_%b';

5) Find the names of sailors who have reserved a red or a green boat.
sql> select s.sname from sailor21 s,reserves21 r,boats21 b where s.sid=r.sid and r.bid=b.bid and (b.color='red'or b.color='green');

6) Find all sids of sailors who have rating of 10 or reserved boat 104.
sql> select s.sid from sailor21 s where s.rating=10 union select r.sid from rese
rves21 r where r.bid=104;

Sailors Data Base Nested Queries:

7) Find the names of sailors who have reserved a boat 103.
sql> select s.sname from sailor21 s where s.sid in (select r.sid from reserves21 r where r.bid=103);

8) Find the names of sailors who have reserved a red boat.
sql> select s.sname from sailor21 s where s.sid in (select r.sid from reserves21 r where r.bid in (select b.bid from boats21 b where b.color='red'));

Sailors data base correlated nested queries:

9) Find the names of sailors who have reserved boat no 103.
sql> select s.sname from sailor21 s where exists (select * from reserves21 r whe
re r.bid=103 and r.sid=s.sid);

Sailors data base queries using aggregate functions:

10) Find the average age of all sailors.
sql> select avg(s.age) from sailor21 s;

11) Count the no. of sailors.
sql> select count(*) from sailor21 s;

12) Find the names of sailors who are older than the oldest sailors with rating 10.
sql> select s.sname from sailor21 s where s.age>(select max(s.age) from sailor21 s where s.rating=10);

13) Find the age of younger sailor who is eligible to vote for each rating level with atleast two such sailors.
sql> select s.rating,min(s.age) as minage from sailor21 s where s.age>=18 group by s.rating having count(*)>1;

15) For each red boat,find the no. of reservations for this boat.
sql> select b.bid,count(*) as resercount from boats21 b,reserves21 r where r.bid
=b.bid and b.color='red' group by b.bid;

16) Find the average age of sailors for each rating level tht has atleast two sailors.
sql> select s.rating,avg(s.age) as avgage from sailor21 s group by s.rating havi
ng count(*)>1;

union:

17) Find the name of sailors who have reserved both a red or a green boat.
sql> select sname from sailor21 s,reserves21 r,boats21 b where s.sid=r.sid and r.bid=b.bid and b.color='red' union select s2.sname from sailor21 s2,reserves21 r2, boats21 b2 where s2.sid=r2.sid and r2.bid=b2.bid and b2.color='green';

set comparisons operators:

18) Find sailors whose rating is better than some sailors called 'horatio'.
sql> select s.sid from sailor21 s where s.rating>any(select s.rating from sailor21 s where s.sname='horatio');

19) Find the sailors with highest rating.
sql> select s.sid from sailor21 s where s.rating>=any(select s.rating from sailor21 s);

23. DEMOSTRATION OF DATABASE CONNECTIVITY

In general, to process any SQL statement with JDBC, you follow these steps:

1. Establishing a connection.
2. Create a statement.
3. Execute the query.
4. Process the ResultSet object.
5. Close the connection.

Establishing Connections
First, establish a connection with the data source you want to use. A data source can be a DBMS, a legacy file system, or some other source of data with a corresponding JDBC driver. This connection is represented by a Connectionobject.
Creating Statements
A Statement is an interface that represents a SQL statement. You execute Statement objects, and they generate ResultSet objects, which is a table of data representing a database result set. You need a Connection object to create a Statement objec
Executing Queries
To execute a query, call an execute method from Statement such as the following
Processing ResultSet Objects
You access the data in a ResultSet object through a cursor. Note that this cursor is not a database cursor. This cursor is a pointer that points to one row of data in the ResultSet object. Initially, the cursor is positioned before the first row. You call various methods defined in the ResultSet object to move the cursor.
Closing Connections
When you are finished using a Statement, call the method Statement close to immediately release the resources it is using. When you call this method, its ResultSet objects are closed.

CREATION OF TABLE AND INSERTING DATA

```
SQL> create table empsam21(eno int,ename varchar(20),address varchar(20));
Table created.

SQL> insert into empsam21 values('&id','&name','&address');
Enter value for id: 1
Enter value for name: Jack
Enter value for address: Melbourne
old    1: insert into empsam21 values('&id','&name','&address')
new    1: insert into empsam21 values('1','Jack','Melbourne')
1 row created.

SQL> /
Enter value for id: 2
Enter value for name: Jill
Enter value for address: Detroit
old    1: insert into empsam21 values('&id','&name','&address')
new    1: insert into empsam21 values('2','Jill','Detroit')
1 row created.
```

JAVA PROGRAM FOR CONNECTIVITY

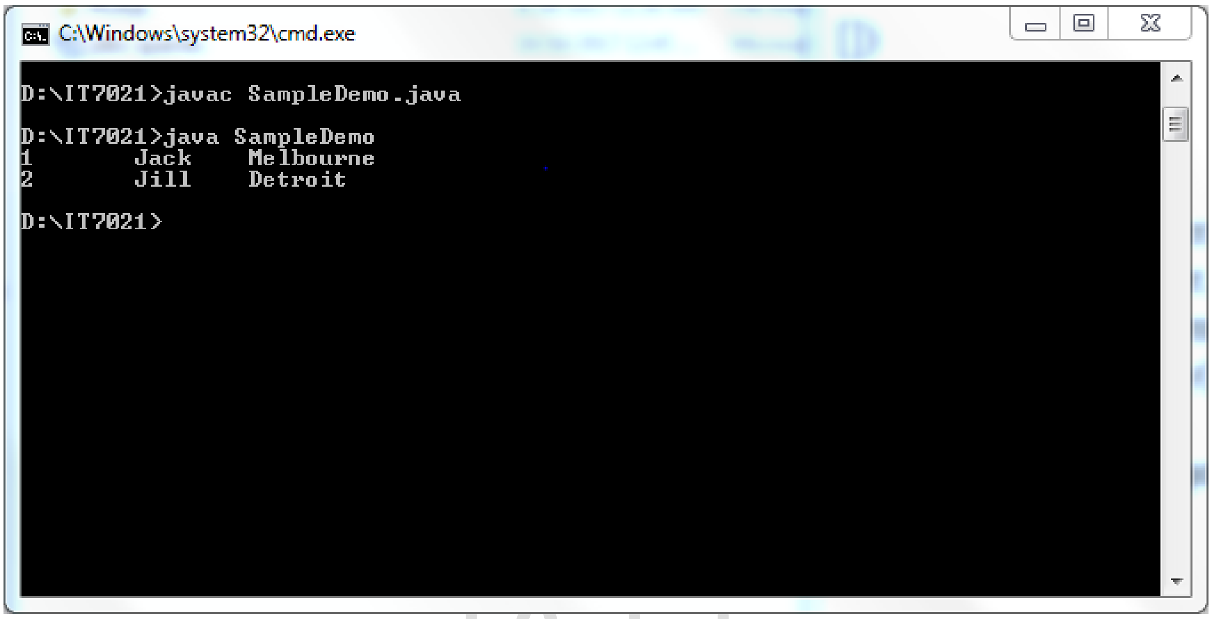
```
import java.sql.*;
class SampleDemo
{
        public static void main(String args[])
        {
                try
                {
                        Class.forName("oracle.jdbc.driver.OracleDriver");
                        Connection con=
DriverManager.getConnection("jdbc:oracle:thin:@192.168.2.100:1521:orcl","scott","tiger");
                        Statement stmt = con.createStatement();
                        ResultSet rs = stmt.executeQuery("select * from empsam21");
                        while(rs.next())
                                System.out.println(rs.getInt(1)+"       "+rs.getString(2)+"
        "+rs.getString(3));
                        con.close();
                }
                catch(Exception e)
                {
                        System.out.println(e);
                }
        }
}
```

OUTPUT:

```
D:\IT7021>javac SampleDemo.java

D:\IT7021>java SampleDemo
1         Jack      Melbourne
2         Jill      Detroit

D:\IT7021>
```



```
D:\IT7021>javac SampleDemo.java

D:\IT7021>java SampleDemo
1        Jack     Melbourne
2        Jill     Detroit

D:\IT7021>
```