DATA BASE MANAGEMENT SYSTEMSLAB MANUAL

Department of Computer Science & Engineering

VEMU INSTITUTE OF TECHNOLOGY::P.KOTHAKOTA

NEAR PAKALA, CHITTOOR-517112

(Approved by AICTE, New Delhi & Affiliated to JNTUA, Anantapuramu)

DATABASE MANAGEMENT SYSTEMS LAB MANUAL

Name:____DBMS Lab_____

Regulation: _____R19_____

Year/Semester: _____II-I_____

Department of Computer Science & Engineering

VEMU INSTITUTE OF TECHNOLOGY::P.KOTHAKOTA

NEAR PAKALA, CHITTOOR-517112

(Approved by AICTE, New Delhi & Affiliated to JNTUA, Anantapuramu)

JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY ANANTAPUR

Syllabus for (R19 Regulations)

DATABASE MANAGEMENT SYSTEMS LAB (19A05302P)

List of Experiments:

Week-1: CREATION OF TABLES

Week-2: QUERIES USING DDL AND DML

Week-3: QUERIES USING AGGREGATE FUNCTIONS

Week-4: PROGRAMS ON PL/SQL

Week-5: PROCEDURES AND FUNCTIONS

Week-6: TRIGGERS

Week-7: PROCEDURES

Week-8: CURSORS

Week-9: CASE STUDY: BOOK PUBLISHING COMPANY

Week-10: CASE STUDY GENERAL HOSPITAL

Week-11: CASE STUDY: CAR RENTAL COMPANY

Week-12: CASE STUDY: STUDENT PROGRESS MONITORING SYSTEM

DATABASE MANAGEMENT SYSTEMS LABINDEX

GENERAL INSTRUCTIONS FOR LABORATORY CLASSES

DO'S

1. Without Prior permission do not enter into the Laboratory.
2. While entering into the LAB students should wear their ID cards.
3. The Students should come with proper uniform.
4. Students should sign in the LOGIN REGISTER before entering into the laboratory.
5. Students should come with observation and record note book to the laboratory.
6. Students should maintain silence inside the laboratory.
7. After completing the laboratory exercise, make sure to shutdown the system properly

DONT'S

8. Students bringing the bags inside the laboratory.
9. Students wearing slippers/shoes insides the laboratory.
10. Students using the computers in an improper way.
11. Students scribbling on the desk and mishandling the chairs.
12. Students using mobile phones inside the laboratory.
13. Students making noise inside the laboratory.

SCHEME OF EVALUATION

S.No	Name of the Experiment	Date	Marks Awarded				Total 30(M)
			Record (10M)	Observation (10M)	Viva-Voce (5M)	Attendance (5M)	
1	Creation of tables						
2	Queries using DDL and DML						
3	Queries using aggregate functions						
4	Programs on PL/SQL						
5	Procedures and functions						
6	Triggers						
7	Procedures						
8	Cursors						
9	Case study: book publishing company						
10	Case study general hospital						
11	Case study: car rental company						
12	Case study: student progress monitoring system						

Signature of Lab In-charge

Week-1: CREATION OF TABLES

Exp 1:

Aim: To create a table called Employee with the following Structure and Answer the following queries.

Name	Type
Empno	Number
Ename	Varchar2(20)
Job	Varchar2(20)
Mgr	Number
Sal	Number

Sql>createtable Employee (Empnonumber,Ename varchar2(20),job varchar2(20), Mgrnumber,Sal number);
Or

Sql>createtable Employee (Empnonumber,Ename varchar2(20),job varchar2(20), Mgrnumber,Sal number); constraintpk_employeesprimarykey (empno), constraintfk_employees_deptnoforeignkey (deptno) references DEPARTMENTS (deptno));

Sql> Select * from Employee;

Output:

a. Add a column commission with domain to the Employee table

Sql> Altertable employee add commission number;

Output:

b. Insert any five records in to the table.

Sql> INSERT INTO Employee VALUES (1, 'King', 'ITmanager', '100', '20000');

Sql> INSERT INTO Employee VALUES (5, 'blake', 'IT', '200', '30000');

Sql> INSERT INTO Employee VALUES (9, 'raj', 'manager', '300', '40000');

Sql> INSERT INTO Employee VALUES (19, 'clarke', 'Assistant', '400', '50000');

Sql> INSERT INTO Employee VALUES (25, 'mohan', 'clerk', '500', '60000');

Output:

c. Update the column details of job

Sql> UPDATE EMPLOYEE SET JOB = 'MANAGER'WHERE JOB IS NULL;

Output:

d.Rename the column of Employ table using alter command.

Sql>ALTER TABLE Employee RENAME COLUMN Ename TO Employname;

Output:

e. Delete the employee whose empno is19.

Sql>DELETEempno FROM Employee WHERE empno=19;

Output:

Exp 2:

Aim: Create department table with the following structure and answer the following quries.

Name	Type
Deptno	Number
Deptname	Varchar2(20)
location	Varchar2(20)

Sql>CREATE TABLE dept (Deptnonumber,Deptnamevarchar2(20), location varchar2(20));**or**

create table dept(deptno number(2,0), dname varchar2(14), loc varchar2(13),

constraint pk_dept primary key (deptno));

Output:

a.Add column designation to the department table.

Sql>Altertable det add designation varchar2(20);

Output:

b. Insert values into the table.

Sql> insert into dept values(101, ‘cse’, ‘ nellore’, ‘assistant’);

Sql> insert into dept values(102, ‘Ece’, ‘ tpty’, ‘assistant’);

Sql> insert into dept values(103, ‘eee’, ‘banglore’, ‘HR’);

Sql> insert into dept values(104, ‘civil’, ‘Hyd’, ‘manager’);

Sql> insert into dept values(101, ‘cse’, ‘ chittoor’, ‘assistant’);

Output:

c. List the records of emp table grouped by dept no

Sql>SELECT empno from emp, dept , GROUP BY deptno;

Output:

d. Update the record where dept no is9.

Sql> Update table dept set deptno=9 where location= ‘tpty’;

Output:

e. Delete any column data from the table

Sql>DELETE location FROM dept;

Output:

Exp 3:

Aim: To create a table called Customer table and answer the following queries.

Name	Name Type
Custname	Varchar2(20)
custstreet	Varchar2(20)
custcity	Varchar2(20)

Sql>CREATE TABLE customer (custname varchar2(20), custstreetvarchar2(20), custcityvarchar2(20));

a.Insert records into the table

Sql> insert into customer values(‘kumar’, ‘4street’, ‘hyd’);

Sql> insert into customer values(‘rmesh’, ‘avanue’, ‘hyd’);

Sql> insert into customer values(‘mahesh’, ‘amerpet’, ‘hyd’);

Sql> insert into customer values(‘vasu’, ‘marthali’, ‘Banglore’);

Sql> insert into customer values(‘hari’, ‘siliconcity’, ‘Banglore’);

Output:

b. Add salary column to the table

Sql> Update table customer add salary number;

Output:

c.Alter the table column domain.

Sql> Alter table customer set custname = ‘cname’;

Output:

d. Drop salary column of the customer table.

Sql> Alter table customer drop column salary;

Output:

e.Delete the rows of customer table whose ust_city is ‘hyd’.

Sql>DELETEFROMcustomer WHERE custcity = ‘hyd’;

Output:

f.Create a table called branch table.

Name	Name Type
branchname	Varchar2(20)
Branch	Varchar2(20)
asserts	Varchar2(20)

Sql> Create table branch (branchname varchar2(20), Branch varchar2(20), asserts varchar2(20);

Output:

Exp 4:

Aim: To increase the size of data type for asserts to the branch and answer the following queries

a) Add and drop a column to the branch table.
Sql> Alter table branch add branchid number;
Output:

Sql> Alter table branch drop column branchid;
Output:

b) Insert values to the table.
Sql> insert into branch values (‘kukatpally’, ‘Iron’, ‘Iron_rods’);
Sql> insert into branch values (‘amerpet’, ‘steel’, ‘steel_plates’);
Sql> insert into branch values (‘SRNagar’, ‘soap’, ‘soapplant’);
Output:

c) Update the branch name column
Sql> update table branch set branchname = ‘bname’;
Output:

d) Delete any two columns from the table
Sql> Alter table branch drop(bname, asserts);
Output:

Exp 5:

Aim: Create a table called sailor table and answer the following queries

Sailors(sid: integer, sname: string, rating: integer, age: real);

SQL>CREATE TABLE sailors (sid integer not null,sname varchar(32),rating integer,CONSTRAINT PK_sailors PRIMARY KEY (sid));

a.Add column age to the sailortable.

```
Sql> alter table sailors add column age real;
```

Output:

b. Insert values into the sailortable.

```
Sql> INSERT INTO sailors ( sid, sname, rating, age ) VALUES ( 22, 'Dustin', 7, 45.0 );
Sql> INSERT INTO sailors ( sid, sname, rating, age ) VALUES ( 22, 'brutes', 9, .60.0 );
Sql> INSERT INTO sailors ( sid, sname, rating, age ) VALUES ( 22, 'luber', 8, 58.0 );
```

c. Delete the row with rating>8.

Sql> delete from sailors where ratting>8;

Output:

d. Update the column details ofsailor.

Sql> Update table sailors set sname = ‘sailorname’;

Output:

e. Insert null values into thetable.

```
Sql> INSERT INTO sailors ( sid, sname, rating, age ) VALUES ( 22, 'Dustin',  , 45.0 );
Sql> INSERT INTO sailors ( sid, sname, rating, age ) VALUES ( 22, '  ', 7, 45.0 );
```

Output:

Exp 6:

Aim: To Create a table called reserves table and answer the following queries

Reserves(sid: integer, bid: integer, day: date)

Sql> CREATE TABLE reserves (sid integer not null, bid integer not null, day datetime not null, CONSTRAINT PK_reserves PRIMARY KEY (sid, bid, day), FOREIGN KEY (sid) REFERENCES sailors(sid), FOREIGN KEY (bid) REFERENCES boats(bid));

a. Insert values into the reserves table.

Sql> INSERT INTO reserves (sid, bid, day) VALUES (22, 101, '1998-10-10');

Sql> INSERT INTO reserves (sid, bid, day) VALUES (31, 101, '1998-10-10');

Sql> INSERT INTO reserves (sid, bid, day) VALUES (22, 102, '1998-10-09');

Sql> INSERT INTO reserves (sid, bid, day) VALUES (64, 102, '1998-10-08');

Output:

b. Add column time to the reserves table.

Sql> Alter table reserves add column bname varchar2(20);

Output:

c. Alter the column day data type to date.

Sql> Alter table reserves modify day date;

Output:

d. Drop the column time in the table.

Sql> Alter table reserves drop column sid where day= '1998-10-10';

Output:

e. Delete the row of the table with some condition.

Sql> Delete table reserves;

Output:

Week 2: QUERIES USING DDL AND DML

SQL (Structured Query Language):

Structured Query Language is a databasecomputer language designed for managing data in relational database management systems(RDBMS), and originally based upon Relational Algebra. Its scope includes data query and update, schema creation and modification, and data access control. SQL was one of the first languages for Edgar F. Codd's relational model in his influential 1970 paper, "A Relational Model of Data for Large Shared Data Banks"[3] and became the most widely used language for relational databases.

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

DATA TYPES:

1. **CHAR (Size):** This data type is used to store character strings values of fixed length. The size in brackets determines the number of characters the cell can hold. The maximum number of character is 255 characters.
2. **VARCHAR (Size) / VERCHAR2 (Size)**: This data type is used to store variable length alphanumeric data. The maximum character can hold is 2000 character.
3. **NUMBER (P, S):** The NUMBER data type is used to store number (fixed or floating point). Number of virtually any magnitude may be stored up to 38 digits of precision. Number as large as $9.99 * 10^{124}$. The precision (p) determines the number of places to the right of the decimal. If scale is omitted then the default is zero. If precision is omitted, values are stored with their original precision up to the maximum of 38 digits.
4. **DATE:** This data type is used to represent date and time. The standard format is DD-MM-YY as in 17-SEP-2009. To enter dates other than the standard format, use the appropriate functions. Date time stores date in the 24-Hours format. By default the time in a date field is 12:00:00 am, if no time portion is specified. The default date for a date field is the first day the current month.
5. **LONG:** This data type is used to store variable length character strings containing up to 2GB. Long data can be used to store arrays of binary data in ASCII format. LONG values cannot be indexed, and the normal character functions such as SUBSTR cannot be applied.
6. **RAW:** The RAW data type is used to store binary data, such as digitized picture or image. Data loaded into columns of these data types are stored without any further conversion.

RAW data type can have a maximum length of 255 bytes. LONG RAW data type can contain up to 2GB.

There are five types of SQL statements. They are:

1. DATA DEFINITION LANGUAGE (DDL)
2. DATA MANIPULATION LANGUAGE (DML)
3. DATA RETRIEVAL LANGUAGE (DRL)
4. TRANSATIONAL CONTROL LANGUAGE (TCL)
5. DATA CONTROL LANGUAGE (DCL)

EXP 7: TO PRACTICE DDL COMMANDS USING ORACLE

1. DATA DEFINITION LANGUAGE (DDL): The Data Definition Language (DDL) is used to create and destroy databases and database objects. These commands will primarily be used by database administrators during the setup and removal phases of a database project. Let's take a look at the structure and usage of four basic DDL commands:

1. CREATE 2. ALTER 3. DROP 4. RENAME

1. CREATE:

(a)CREATE TABLE: This is used to create a new relation and the corresponding

Syntax:
```
CREATE TABLE relation_name
    (field_1 data_type(Size),field_2 data_type(Size), .. . );
```

Example:

```
SQL>CREATE TABLE Student (snoNUMBER(3),snameCHAR(10),class CHAR(5));
```

(b)CREATE TABLE..AS SELECT....: This is used to create the structure of a new relation from the structure of an existing relation.

Syntax:
```
CREATE TABLE (relation_name_1, field_1,field_2,.....field_n) AS
    SELECT field_1,field_2,.....field_nFROM relation_name_2;
```

Example: `SQL>CREATE TABLE std(rno,sname) AS SELECT sno,snameFROM student;`

Output:

```
SQL>Descstd;
```

Output:

2. ALTER:

(a)ALTER TABLE ...ADD...: This is used to add some extra fields into existing relation.

Syntax:
```
ALTER TABLE relation_nameADD(new field_1 data_type(size), new
          field_2 data_type(size),..);
```

Example :
```
SQL>ALTER TABLE std ADD(Address CHAR(10));
```

(b)ALTER TABLE...MODIFY...: This is used to change the width as well as data type of fields of existing relations.

Syntax:
```
ALTER TABLE relation_nameMODIFY (field_1 newdata_type(Size),
          field_2 newdata_type(Size),....field_newdata_type(Size));
```

Example:
```
SQL>ALTER TABLE student MODIFY(snameVARCHAR(10),class
VARCHAR(5));
```

Output:

3. DROP TABLE: This is used to delete the structure of a relation. It permanently deletes the records in the table.

Syntax:
```
DROP TABLE relation_name;
```

Example:
```
SQL>DROP TABLE std;
```

Output:

4. RENAME: It is used to modify the name of the existing database object.

Syntax:
```
RENAME TABLE old_relation_name TO new_relation_name;
```

Example:
```
SQL>RENAME TABLE stdTO std1;
```

Output:

5. TRUNCATE: This command will remove the data permanently. But structure will not be removed.

Syntax: `TRUNCATE TABLE<Table name>`

Example `TRUNCATE TABLE student;`

Output:

EXP 8:

AIM: TO PRACTICE DML COMMANDS USING ORACLE

DATA MANIPULATION LANGUAGE (DML): The Data Manipulation Language (DML) is used to retrieve, insert and modify database information. These commands will be used by all database users during the routine operation of the database. Let's take a brief look at the basic DML commands:

1. INSERT 2. UPDATE 3. DELETE

1. INSERT INTO: This is used to add records into a relation. These are three type of INSERT INTO queries which are as

a) Inserting a single record

Syntax: `INSERT INTOrelationname(field_1,field_2,.field_n)VALUES`

`(data_1,data_2,.....data_n);`

Example: `SQL>INSERT INTO student(sno,sname,class,address)VALUES`

```
                                        (1,'Ravi','M.Tech','Palakol');
```

Output:

b) Inserting all records from another relation

Syntax: `INSERT INTO relation_name_1 SELECTField_1,field_2,field_n`

```
            FROM relation_name_2 WHERE field_x=data;
```

Example: `SQL>INSERT INTO stdSELECT sno,snameFROM student`

```
      WHERE name = 'Ramu';
```

Output:

c) Inserting multiple records

Syntax: `INSERT INTO relation_namefield_1,field_2,.....field_n) VALUES`

```
      (&data_1,&data_2,........&data_n);
```

Example: `SQL>INSERT INTO student(sno,sname,class,address)`

```
      VALUES(&sno,'&sname','&class','&address');
                  Enter value for sno: 101
                  Enter value for name: Ravi
                  Enter value for class: M.Tech
                  Enter value for name: Palakol
```

Output:

2. UPDATE-SET-WHERE: This is used to update the content of a record in a relation.

Syntax: SQL>UPDATE relation name SET Field_name1=data,field_name2=data,

WHERE field_name=data;

Example: SQL>UPDATE student SETsname = 'kumar' WHERE sno=1;

Output:

3. DELETE-FROM: This is used to delete all the records of a relation but it will retain the structure of that relation.

a) DELETE-FROM: This is used to delete all the records of relation.

Syntax: SQL>DELETE FROM relation_name;

Example: SQL>DELETE FROM std;

Output:

b) DELETE -FROM-WHERE: This is used to delete a selected record from a relation.

Syntax: SQL>DELETE FROM relation_name WHERE condition;

Example: SQL>DELETE FROM student WHERE sno = 2;

Output:

DRL(DATA RETRIEVAL LANGUAGE): Retrieves data from one or more tables.

1. SELECT FROM: To display all fields for all records.

Syntax : SELECT * FROM relation_name;

Example : SQL> select * from dept;

Output:

2. SELECT FROM: To display a set of fields for all records of relation.

Syntax: `SELECT a set of fields FROM relation_name;`

Example: `SQL> select deptno, dname from dept;`

Output:

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

Syntax: `SELECT a set of fields FROMrelation_nameWHERE  condition;`

Example: `SQL> select * FROM dept WHERE deptno<=20;`

Output:

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

Syntax: `SELECT a set of fields FROM relation_nameGROUP BY field_name;`

Example: `SQL> SELECT EMPNO, SUM (SALARY) FROM EMP GROUP BY EMPNO;`

Output:

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

Syntax:
```
SELECT a set of fields FROM relation_name
        ORDER BY field_name;
```

Example: `SQL> SELECT empno,ename,job FROM emp ORDER BY job;`

Output:

6. JOIN using SELECT - FROM - ORDER BY: This query is used to display a set of fields from two relations by matching a common field in them in an ordered manner based on some fields.

Syntax: `SELECT a set of fields from both relations FROM relation_1, relation_2 WHERE relation_1.field_x = relation_2.field_y ORDER BY field_z;`

Example: `SQL>SELECT empno,ename,job,dname FROM emp,deptWHERE emp.deptno = 20 ORDER BY job;`

Output:

7. JOIN using SELECT - FROM - GROUP BY: This query is used to display a set of fields from two relations by matching a common field in them and also group the corresponding records for each and every value of a specified key(s) while displaying.

Syntax: `SELECT a set of fields from both relations FROM relation_1,relation_2 WHERE relation_1.field-x=relation_2.field-y GROUP BY field-z;`

Example: `SQL> SELECT empno,SUM(SALARY) FROM emp,deptWHERE emp.deptno =20 GROUP BY empno;`

Output:

8. UNION: This query is used to display the combined rows of two different queries, which are having the same structure, without duplicate rows.

Syntax: `SELECT field_1,field_2,..... FROM relation_1 WHERE (Condition) UNION SELECT field_1,field_2,..... FROM relation_2 WHERE (Condition);`

Example:

`SQL> SELECT * FROM STUDENT;`

Output:

```
SQL> SELECT * FROM STD;
```

Output:

```
SQL> SELECT * FROM student UNION SELECT * FROM std;
```

Output:

9. INTERSET: This query is used to display the common rows of two different queries, which are having the same structure, and to display a selected set of fields out of them.

Syntax: `SELECT field_1,field_2,.. FROM relation_1 WHERE`

```
(Condition) INTERSECT SELECT field_1,field_2,.. FROM relation_2
WHERE(Condition);
```

Example :`SQL> SELECT * FROM student INTERSECT SELECT * FROM std;`

Output:

10. MINUS: This query is used to display all the rows in relation_1,which are not having in the relation_2.

Syntax: `SELECT field_1,field_2,......FROM relation_1`

```
WHERE(Condition) MINUS SELECT field_1,field_2,.....
FROM relation_2 WHERE(Conditon);
```

SQL> `SELECT * FROM student MINUS SELECT * FROM std;`

Output:

TRANSATIONAL CONTROL LANGUAGE (T.C.L):

A transaction is a logical unit of work. All changes made to the database can be referred to as a transaction. Transaction changes can be mode permanent to the database only if they are committed a transaction begins with an executable SQL statement & ends explicitly with either role back or commit statement.

1. COMMIT: This command is used to end a transaction only with the help of the commit command transaction changes can be made permanent to the database.

Syntax:`SQL>COMMIT;`

Example: `SQL>COMMIT;`

Output:

2. SAVE POINT: Save points are like marks to divide a very lengthy transaction to smaller once. They are used to identify a point in a transaction to which we can latter role back. Thus, save point is used in conjunction with role back.

Syntax: `SQL>SAVE POINT ID;`

Example: `SQL>SAVE POINT xyz;`

Output:

3. ROLE BACK: A role back command is used to undo the current transactions. We can role back the entire transaction so that all changes made by SQL statements are undo (or) role back a transaction to a save point so that the SQL statements after the save point are role back.

Syntax: `ROLE BACK( current transaction can be role back)`

`ROLE BACK to save point ID;`

Example: SQL>ROLE BACK;

SQL>ROLE BACK TO SAVE POINT xyz;

Output:

DATA CONTROL LANGUAGE (D.C.L):

DCL provides uses with privilege commands the owner of database objects (tables), has the soul authority ollas them. The owner (data base administrators) can allow other data base uses to access the objects as per their requirement

1. GRANT: The GRANT command allows granting various privileges to other users and allowing them to perform operations with in their privileges

For Example, if a uses is granted as 'SELECT' privilege then he/she can only view data but cannot perform any other DML operations on the data base object GRANTED privileges can also be withdrawn by the DBA at any time

Syntax: SQL>GRANT PRIVILEGES on object_name To user_name;

Example: SQL>GRANT SELECT, UPDATE on empTohemanth;

Output:

2. REVOKE: To with draw the privileges that has been GRANTED to a uses, we use the REVOKE command

Syntax: SQL>REVOKE PRIVILEGES ON object-name FROM user_name;

Example: SQL>REVOKE SELECT, UPDATE ONemp FROM ravi;

Output:

1. Creation, altering and dropping of tables and inserting rows into a table (use constraints while creating tables) examples using SELECT command.

1. CREATE:

(a)CREATE TABLE: This is used to create a new relation

Syntax: CREATE TABLE relation_name

(field_1 data_type(Size),field_2 data_type(Size), .. .);

Example:

SQL>CREATE TABLE Student (snoNUMBER(3)**PRIMARY KEY**,sname

CHAR(10),classCHAR(5));

Output:

2. ALTER:

(a)ALTER TABLE ...ADD...: This is used to add some extra fields into existing relation.

Syntax:ALTER TABLE relation_name ADD(new field_1 data_type(size), new field_2 data_type(size),..);

Example :SQL>ALTER TABLE std ADD(Address CHAR(10));

Output:

(b)ALTER TABLE...MODIFY...: This is used to change the width as well as data type of fields of existing relations.

Syntax:
```
ALTER TABLE relation_nameMODIFY (field_1 newdata_type(Size),
          field_2 newdata_type(Size),....field_newdata_type(Size));
```

Example:
```
SQL>ALTER TABLE student MODIFY(snameVARCHAR(10),classVARCHAR(5));
```

Output:

3. DROP TABLE: This is used to delete the structure of a relation. It permanently deletes the records in the table.

Syntax: `DROP TABLE relation_name;`

Example: `SQL>DROP TABLE std;`

Output:

4. INSERT:

Syntax:
```
INSERT INTO relation_namefield_1,field_2,.....field_n) VALUES
     (&data_1,&data_2,........&data_n);
```

Example:
```
SQL>INSERT INTO student(sno,sname,class,address)
     VALUES(&sno,'&sname','&class','&address');
```

Output:

5. SELECT FROM: To display all fields for all records.

Syntax : SELECT * FROM relation_name;

Example : `SQL> select * from student;`

Output:

2. SELECT FROM: To display a set of fields for all records of relation.

Syntax: `SELECT a set of fields FROM relation_name;`

Example: `SQL> select sno, sname from student;`

Output:

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

Syntax: `SELECT a set of fields FROMrelation_nameWHERE  condition;`

Example: `SQL> select * FROM student WHERE class='CSE';`

Output:

There are 5 constraints available in ORACLE:

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

Syntax:

CREATE TABLE Table_Name(column_namedata_type(*size*) **NOT NULL,**);

Example:

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

Output:

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

Syntax:

```
CREATE TABLETable_Name(column_namedata_type(size) UNIQUE, ….);
```

Example:

```
CREATE TABLE student (snoNUMBER(3) UNIQUE, name CHAR(10));
```

Output:

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

Syntax:

```
CREATE TABLETable_Name(column_namedata_type(size) CHECK(logical
                        expression), ….);
```

Example: `CREATE TABLE student (snoNUMBER (3), nameCHAR(10),class CHAR(5),CHECK(class IN('CSE','CAD','VLSI'));`

Output:

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

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

Syntax:

```
CREATE TABLETable_Name(column_namedata_type(size) PRIMARY KEY, ….);
```

Example:

```
CREATE TABLE faculty (fcodeNUMBER(3) PRIMARY KEY,fname CHAR(10));
```

Output:

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

Syntax: **CREATE TABLE**Table_Name(column_namedata_type(*size*)

FOREIGN KEY(column_name) **REFERENCES**table_name);

Example:

CREATE TABLE subject (scode**NUMBER (3) PRIMARY KEY,**

subname **CHAR**(**10**),fcode**NUMBER(3),**

FOREIGN KEY(fcode) **REFERENCE** faculty);

Output:

Defining integrity constraints in the alter table command:

Syntax: **ALTER TABLE**Table_Name**ADDPRIMARY KEY** (column_name);

Example: **ALTER TABLE** student **ADDPRIMARY KEY** (sno);

(Or)

Syntax: **ALTER TABLE** table_name **ADD CONSTRAINT** constraint_name

PRIMARY KEY(colname)

Example: **ALTER TABLE** student **ADD CONSTRAINT** SN **PRIMARY KEY(**SNO**)**

Output:

Dropping integrity constraints in the alter table command:

Syntax: **ALTER TABLE**Table_Name**DROP**constraint_name;

Example: **ALTER TABLE** student **DROPPRIMARY KEY;**

(or)

Syntax: **ALTER TABLE** student **DROP CONSTRAINT** constraint_name;

Example: **ALTER TABLE** student **DROP CONSTRAINT** SN;

Output:

Week-3:QUERIES USING AGGREGATE FUNCTIONS

Exp 9:

Aim:To Practice Aggregate functions using oracle.

Aggregative operators: In addition to simply retrieving data, we often want to perform some computation or summarization. SQL allows the use of arithmetic expressions. We now consider a powerful class of constructs for computing aggregate values such as MIN and SUM.

1. Count: COUNT following by a column name returns the count of tuple in that column. If DISTINCT keyword is used then it will return only the count of unique tuple in the column. Otherwise, it will return count of all the tuples (including duplicates) count (*) indicates all the tuples of the column.

Syntax: `COUNT (Column name)`

Example:`SELECT COUNT (Sal) FROM emp;`

Output:

2. SUM: SUM followed by a column name returns the sum of all the values in that column.

Syntax:`SUM (Column name)`

Example: `SELECT SUM (Sal) From emp;`

Output:

3. AVG: AVG followed by a column name returns the average value of that column values.

Syntax:`AVG (n1,n2..)`

Example: `Select AVG(10, 15, 30) FROM DUAL;`

Output:

4. MAX: MAX followed by a column name returns the maximum value of that column.

Syntax: `MAX (Column name)`

Example: `SELECT MAX (Sal) FROM emp;`

Output:

```
SQL> select deptno,max(sal) from emp group by deptno;
```

Output:

```
SQL> select deptno,max(sal) from emp group by deptno having max(sal)<3000;
```

Output:

5. MIN: MIN followed by column name returns the minimum value of that column.

Syntax: `MIN (Column name)`

Example : `SELECT MIN (Sal) FROM emp;`

```
SQL>select deptno,min(sal) from emp group by deptno having min(sal)>1000;
```

Output:

VIEW: In SQL, a view is a virtual table based on the result-set of an SQL statement.

A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database.

You can add SQL functions, WHERE, and JOIN statements to a view and present the data as if the data were coming from one single table.

A view is a virtual table, which consists of a set of columns from one or more tables. It is similar to a table but it doest not store in the database. View is a query stored as an object.

Syntax: CREATE VIEW view_name AS SELECT set of fields FROM relation_name
WHERE (Condition)

```
SQL>CREATE VIEW employee AS SELECT empno,ename,jobFROM EMP
     WHERE job = 'clerk';
```

OUTPUT:

```
SQL> SELECT * FROM EMPLOYEE;
```

OUTPUT:

2.Example:

```
CREATE VIEW [Current Product List] AS
SELECT ProductID,ProductName
FROM Products
WHERE Discontinued=No
```

DROP VIEW:This query is used to delete a view , which has been already created.

Syntax: DROP VIEW View_name;

Example : SQL> DROP VIEW EMPLOYEE;

Output:

Exp 10:

Aim: To practice String & Character functions using sql

CONVERSION FUNCTIONS:

To_char: TO_CHAR (number) converts n to a value of VARCHAR2 data type, using the optional number format fmt. The value n can be of type NUMBER, BINARY_FLOAT, or BINARY_DOUBLE.

SQL>select to_char(65,'RN')from dual; LXV

To_number : TO_NUMBER converts expr to a value of NUMBER data type.

SQL>Select to_number ('1234.64') from Dual;

Output:

To_date:TO_DATE converts char of CHAR, VARCHAR2, NCHAR, or NVARCHAR2 data type to a value of DATE data type.

SQL>SELECT TO_DATE('January 15, 1989, 11:00 A.M.')FROM DUAL;

Output:

STRING FUNCTIONS:

Concat: CONCAT returns char1 concatenated with char2. Both char1 and char2 can be any of the datatypes

SQL>SELECT CONCAT('ORACLE','CORPORATION')FROM DUAL;
Output:

Lpad: LPAD returns expr1, left-padded to length n characters with the sequence of characters in expr2.

SQL>SELECT LPAD('ORACLE',15,'*')FROM DUAL;

Output:

Rpad: RPAD returns expr1, right-padded to length n characters with expr2, replicated as many times as necessary.

SQL>SELECT RPAD ('ORACLE',15,'*')FROM DUAL;
Output:

Ltrim: Returns a character expression after removing leading blanks.

SQL>SELECT LTRIM('SSMITHSS','S')FROM DUAL;
Output:

Rtrim: Returns a character string after truncating all trailing blanks

SQL>SELECT RTRIM(‘SSMITHSS’,’S’)FROM DUAL;
Output:

Lower: Returns a character expression after converting uppercase character data to lowercase.

SQL>SELECT LOWER(‘DBMS’)FROM DUAL;
Output:

Upper: Returns a character expression with lowercase character data converted to uppercase

SQL>SELECT UPPER(‘dbms’)FROM DUAL;
Output:

Length: Returns the number of characters, rather than the number of bytes, of the given string expression, excluding trailing blanks.

SQL>SELECT LENGTH(‘DATABASE’)FROM DUAL;
Output:

Substr: Returns part of a character, binary, text, or image expression.

SQL>SELECT SUBSTR(‘ABCDEFGHIJ’3,4) FROM DUAL;
Output:

Instr: The INSTR functions search string for substring. The function returns an integer indicating the position of the character in string that is the first character of this occurrence.

SQL>SELECT INSTR('CORPORATE FLOOR','OR',3,2) FROM DUAL;
Output:
DATE FUNCTIONS:

Sysdate:

SQL>SELECT SYSDATE FROM DUAL; 29-DEC-08

Output:

next_day:

SQL>SELECT NEXT_DAY(SYSDATE,’WED’)FROM DUAL;

Output:

add_months:

SQL>SELECT ADD_MONTHS(SYSDATE,2)FROM DUAL;

Output:

last_day:

SQL>SELECT LAST_DAY(SYSDATE)FROM DUAL;

Output:

months_between:

SQL>SELECT MONTHS_BETWEEN(SYSDATE,HIREDATE)FROM EMP;

Output:

Least:

SQL>SELECT LEAST('10-JAN-07','12-OCT-07')FROM DUAL;

Output:

Greatest:

SQL>SELECT GREATEST('10-JAN-07','12-OCT-07')FROM DUAL;

Output:

Trunc:

SQL>SELECT TRUNC(SYSDATE,'DAY')FROM DUAL;

Output:

Round:

SQL>SELECT ROUND(SYSDATE,'DAY')FROM DUAL;

Output:

to_char:

SQL> select to_char(sysdate, "dd\mm\yy") from dual;

Output:

to_date:

SQL> select to date (sysdate, "dd\mm\yy") from dual;

Output:

Truncate:
SQL>SELECT TRUNC(SYSDATE,'DAY')FROM DUAL;
Output:

Round:
SQL>SELECT ROUND(SYSDATE,'DAY')FROM DUAL;
Output:

to_char:
SQL> select to_char(sysdate, "dd\mm\yy") from dual;
Output:

to_date:
SQL> select to date (sysdate, "dd\mm\yy") from dual;
Output:

Exp 11:

Aim: Consider the following schema:

Sailors (sid,sname, rating, age)

Boats (bid, bname, color)

Reserves (sid, bid,day(date))

Write subquery statement for the following queries**.**

```
create table sailors (
sidint primary key,
sname varchar(38),
   rating int,
   age float check (age > 16 and age < 110)
);

create table boats (
   bid int primary key,
bname varchar(25),
   color varchar(21)
);
```

```
create table reserves (
sidint,
   bid int,
   day date,
   foreign key (sid) references sailors (sid),
   foreign key (bid) references boats (bid)
);
```

1. Find the names of the sailors who have reserved both a red or a yellow boat.

```
SQL> select s.sname from sailors s, boats b, reserves r where s.sid=r.sid and
b.bid=r.bid and (b.color='red' or b.color='yellow');
```

Output:

2. Find all sids of sailors who have a rating of 10 or have reserved boat 111.

```
SQL> select s.sid from sailors s where s.rating = 10  union  select r.sid from
       reserves r where r.bid = 111;
```

Output:

3. Find all sids of sailors who have reserved red boats but not yellow boats

```
SQL>  select s.sid from sailors s, boats b, reserves r where s.sid = r.sid and r.bid =
b.bid and b.color = 'red'
         minus
         select s2.sid from sailors s2, boats b2, reserves r2 where s2.sid = r2.sid and
         r2.bid = b2.bid and b2.color = 'yellow';
```

Output:

4.Find the names of the sailors who have reserved both a red and a yellow boat.

```
SQL> select s.sname  from sailors s, boats b, reserves r where s.sid = r.sid and
r.bid = b.bid and b.color = 'red'
        intersect
        select s2.sname from sailors s2, boats b2, reserves r2 where s2.sid = r2.sid
        and r2.bid = b2.bid and b2.color = 'yellow';
```

Output:

5.Find the names of sailors who have reserved a red boat, and list in the order of age.

```
SQL>SELECT S.sname, S.ageFROM Sailors S, Reserves R, Boats B
WHERE S.sid = R.sid AND R.bid = B.bid AND B.color = ‘red’ORDER BY S.age;
```

Output:

6.Find the names of sailors who have reserved at least one boat.

```
SQL>SELECT snameFROM Sailors S, Reserves RWHERE S.sid = R.sid;
```

Output:

7.Find the ids and names of sailors who have reserved two different boats on the same day.

```
SQL>SELECT DISTINCT S.sid, S.snameFROM Sailors S, Reserves R1, Reserves
R2WHERE S.sid = R1.sid AND S.sid = R2.sidAND R1.day = R2.day AND R1.bid <>
R2.bid;
```

Output:

8.Using Expressions and Strings in the SELECT Command.

```
SQL>ELECT sname, age, rating + 1 as sth
```

FROM Sailors
WHERE 2* rating – 1 < 10 AND sname like ‘B_%b’;

Output:

9.Nested Query
Find the names of sailors who have reserved boat 103.

SQL>SELECT S.snameFROM Sailors SWHERE S.sid
IN (SELECT R.sidFROM Reserves RWHERE R.bid = 103)
Output:

SQL>SELECT S.snameFROM Sailors SWHERE
EXISTS (SELECT *FROM Reserves RWHERE R.bid = 103AND R.sid
= S.sid)

Output:

10.Find the name and the age of the youngest sailor.
SQL>SELECT S.sname, S.ageFROM Sailors SWHERE S.age<= ALL (
SELECT ageFROM Sailors)

Output:

Week-4: PROGRAMS ON PL/SQL

Exp 12:

PL/SQL Introduction

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

Oracle uses a PL/SQL engine to processes the PL/SQL statements. A PL/SQL code can be stored in the client system (client-side) or in the database (server-side).

Advantages of PL/SQL:

- ***Block Structures*:** PL SQL consists of blocks of code, which can be nested within each other. Each block forms a unit of a task or a logical module. PL/SQL Blocks can be stored in the database and reused.
- ***Procedural Language Capability*:** PL SQL consists of procedural language constructs such as conditional statements (if else statements) and loops like (FOR loops).
- ***Better Performance*:** PLSQL engine processes multiple SQL statements simultaneously as a single block, thereby reducing network traffic.
- ***Error Handling*:** PL/SQL handles errors or exceptions effectively during the execution of a PL/SQL program. Once an exception is caught, specific actions can be taken depending upon the type of the exception or it can be displayed to the user with a message.

Syntax of PL/SQL program:

```
Declare
      Variable declaration;
Begin
      Executable statements;
end;
```

Conditional Statements in PL/SQL

As the name implies, PL/SQL supports programming language features like conditional statements, iterative statements.The programming constructs are similar to how you use in programming languages like Java and C++. In this section I will provide you syntax of how to use conditional statements in PL/SQL programming.

IF THEN ELSE STATEMENT:

```
IF condition THEN
      statement 1;
ELSE
```

```
        statement 2;
    END IF;

    IF condition 1 THEN
        statement 1;
        statement 2;
    ELSIF condtion2 THEN
        statement 3;
    ELSE
        statement 4;
    END IF
```

1. Write a PL/SQL program to find the total and average of 4 subjects and display the grade

```
declare
      java number(10);
      dbms number(10);
      co number(10);
      mfcs number(10);
      total number(10);
      avgs number(10);
      per number(10);
begin
      dbms_output.put_line('ENTER THE MARKS');
      java:=&java;
      dbms:=&dbms;
      co:=&co;
      mfcs:=&mfcsl;
      total:=(java+dbms+co+mfcs);
      per:=(total/600)*100;
      if java<40 or dbms<40 or co<40 or mfcs<40 then
            dbms_output.put_line('FAIL');
```

```
      if per>75 then
            dbms_output.put_line('GRADE A');
      elsif per>65 and per<75 then
      dbms_output.put_line('GRADE B');
      elsif per>55 and per<65 then
            dbms_output.put_line('GRADE C');
            else
                  dbms_output.put_line('INVALID INPUT');
      end if;
      dbms_output.put_line('PERCENTAGE IS '||per);
end;
/
```

OUTPUT:

2.Write a PL/SQL program to find the largest of three numbers

```
declare
      a number;
      b number;
```

```
      c number;
begin
      a:=&a;
      b:=&b;
      c:=&c;
      if a=b and b=c and c=a then
            dbms_output.put_line('ALL ARE EQUAL');
      elsif a>b and a>c then
dbms_output.put_line('A IS GREATER');
elsif b>c then
dbms_output.put_line('B IS GREATER');
            else
dbms_output.put_line('C IS GREATER');
      end if;
end;
/
```

OUTPUT:

Loops in PL/SQL

There are three types of loops in PL/SQL:

1. Simple Loop

2. While Loop
3. For Loop

1. Simple Loop:A Simple Loop is used when a set of statements is to be executed at least once before the loop terminates. An EXIT condition must be specified in the loop, otherwise the loop will get into an infinite number of iterations. When the EXIT condition is satisfied the process exits from the loop.

Syntax:

LOOP

statements;

EXIT;

{or EXIT WHEN condition ;}

END LOOP;

2. While Loop:A WHILE LOOP is used when a set of statements has to be executed as long as a condition is true. The condition is evaluated at the beginning of each iteration. The iteration continues until the condition becomes false.

Syntax:

WHILE <condition>

LOOP statements;

END LOOP;

3. FOR Loop:A FOR LOOP is used to execute a set of statements for a predetermined number of times. Iteration occurs between the start and end integer values given. The counter is always incremented by 1. The loop exits when the counter reaches the value of the end integer.

Syntax:

FOR counter IN val1..val2

LOOP statements;

END LOOP;

3.Write a PL/SQL program to generate Fibonacci series

```
declare
      a number;
      b number;
```

```
      c number;
      n number;
      i number;
begin
      n:=&n;
      a:=0;
      b:=1;
      dbms_output.put_line(a);
      dbms_output.put_line(b);
      for i in 1..n-2
      loop
            c:=a+b;
            dbms_output.put_line(c);
            a:=b;
            b:=c;
      end loop;
end;
/
```

OUTPUT:

4. Write a PL/SQL Program to display the number in Reverse Order

```
declare
```

```
      a number;
      rev number;
      d number;
begin
      a:=&a;
      rev:=0;
      while a>0
      loop
            d:=mod(a,10);
            rev:=(rev*10)+d;
            a:=trunc(a/10);
      end loop;
      dbms_output.put_line('no is'|| rev);
end;
/
```

OUTPUT:

Week-5: PROCEDURES AND FUNCTIONS

Exp 13:

A procedures or function is a group or set of SQL and PL/SQL statements that perform a specific task. A function and procedure are a named PL/SQL Block which is similar. The major difference between a procedure and a function is, a function must always return a value, but a procedure may or may not return a value.

Procedures:

A procedure is a named PL/SQL block which performs one or more specific task. This is similar to a procedure in other programming languages. A procedure has a header and a body. The header consists of the name of the procedure and the parameters or variables passed to the procedure.

The body consists or declaration section, execution section and exception section similar to a general PL/SQL Block. A procedure is similar to an anonymous PL/SQL Block but it is named for repeated usage.

We can pass parameters to procedures in three ways:

Parameters	**Description**
IN type	These types of parameters are used to send values to stored procedures.
OUT type	These types of parameters are used to get values from stored procedures. This is similar to a return type in functions.
IN OUT type	These types of parameters are used to send values and get values from stored procedures.

"A procedure may or may not return any value."

Syntax:

```
CREATE [OR REPLACE] PROCEDURE procedure_name (<Argument> {IN,
OUT, IN OUT}    <Datatype>,…)
IS
  Declaration section<variable, constant> ;
BEGIN
  Execution section
EXCEPTION
  Exception section
END
```

or

```
CREATE OR REPLACE PROCEDURE <procedure_name>
        (
        <parameterl IN/OUT <datatype>
        ..
        .
        )
[ IS | AS ]
        <declaration_part>
BEGIN
        <execution part>
EXCEPTION
        <exception handling part>
END;
```

Example:

```
CREATE OR REPLACE PROCEDURE welcome_msg (p_name IN VARCHAR2)
IS
BEGIN
dbms_output.put_line (‘Welcome '|| p_name);
END;
/
```

Output:EXEC welcome_msg (‘Guru99’);

1. create table named emp have two column id and salary with number datatype.

```
CREATE OR REPLACE PROCEDURE p1(id IN NUMBER, sal IN NUMBER)
AS
BEGIN
  INSERT INTO emp VALUES(id, sal);
  DBMD_OUTPUT.PUT_LINE('VALUE INSERTED.');
END;
/
```

Output:

Functions:

A function is a named PL/SQL Block which is similar to a procedure. The major difference between a procedure and a function is, a function must always return a value, but a procedure may or may not return a value.

Syntax:

```
CREATE [OR REPLACE] FUNCTION function_name [parameters]
 RETURN return_datatype;  {IS, AS}
Declaration_section<variable,constant> ;
 BEGIN
Execution_section
    Return return_variable;
 EXCEPTION
   exception section
    Return return_variable;
 END;
```

Example:

```
create or replace function getsal (no IN number) return number
is
sal number(5);
begin
 select salary into sal from emp where id=no;
 return sal;
end;
/
```

Output:

Week-6: TRIGGERS

Exp 13:

1.Create a row level trigger for the customers table that would fire for INSERT orUPDATE or DELETE operations performed on the CUSTOMERS table. This triggerwill display the salary

difference between the old values and new values:

CUSTOMERS table:

ID	NAME	AGE	ADDRESS	SALARY
1	Alive	24	Khammam	2000
2	Bob	27	Kadappa	3000
3	Catri	25	Guntur	4000
4	Dena	28	Hyderabad	5000
5	Eeshwar	27	Kurnool	6000
6	Farooq	28	Nellur	7000

Output:

2.Creation of insert trigger, delete trigger, update trigger practice triggers using the emp database.

```
SQL> create table emp (name varchar(10),empno number(3),age number(3));
```

```
Table
created.
SQL>
```

```
create or replace trigger t2 before insert onemp
for eachrow
when(new.age>100)
begin
RAISE_APPLICATION_ERROR(-20998,'INVALID ERROR'); 6*
end;
```

Output:

3.The following program creates a row-level trigger for the customers table that would fire forINSERT or UPDATE or DELETE operations performed on the CUSTOMERS table. This trigger will display the salary difference between the old values and new values.

CREATE OR REPLACE TRIGGER display_salary_changes

```
BEFORE DELETE OR INSERT OR UPDATE ON customers
FOR EACH ROW
WHEN (NEW.ID > 0)
DECLARE
sal_diff number;
BEGIN
sal_diff := :NEW.salary  - :OLD.salary;
dbms_output.put_line('Old salary: ' || :OLD.salary);
dbms_output.put_line('New salary: ' || :NEW.salary);
dbms_output.put_line('Salary difference: ' || sal_diff);
END;
/
```

Trigger is created

Output:

Week-7: PROCEDURES

Exp 14:

1. Create the procedure for palindrome of given number.

```
DECLARE
```

```
  n number;
  m number;
  temp number:=0;
  rem number;
BEGIN
  n :=12321;
  m :=n;
  while n>0
  loop
    rem := mod(n,10);
    temp := (temp*10)+rem;
    n := trunc(n/10);
  end loop;
  if m = temp
  then
dbms_output.put_line('Palindrome');
  else
dbms_output.put_line('Not Palindrome');
  end if;
END;
```

Output:

2. Create the procedure for GCD: Program should load two registers with two Numbers and then apply the logic for GCD of two numbers.

```
DECLARE

   -- declare variable num1, num2 and t
   -- and these three variables datatype are integer
   num1 INTEGER;
   num2 INTEGER;
   t    INTEGER;
BEGIN
   num1 := 8;

   num2 := 48;

   WHILE MOD(num2, num1) != 0 LOOP
      t := MOD(num2, num1);

      num2 := num1;
```

```
      num1 := t;
   END LOOP;

dbms_output.Put_line('GCD of ' ||num1 ||' and ' ||num2  ||' is ' ||num1);
END;
```

Output:

3. Write the PL/SQL programs to create the procedure for factorial of givennumber.

```
Delimiter //
 CREATE PROCEDURE fact(IN x INT)
BEGIN
DECLARE result INT;
DECLARE i INT;
SET result = 1;
SET i = 1;
WHILE i<= x DO
SET result = result * i;
SET i = i + 1;
END WHILE;
SELECT x AS Number, result as Factorial;
    END//
Query OK, 0 rows affected (0.17 sec)
```

Output:

4. Write the PL/SQL programs to create the procedure to find sum of N naturalnumber.

```
SQL> Declare
i number:=0;
 n number;
```

```
 sum1 number:=0;
 Begin
 n:=&n;
 while i
 loop
 sum1:=sum1+i;
dbms_output.put_line(i);
 i:=i+1;
 end loop;
dbms_output.put_line('The sum is:'||sum1);
 End;
 /
```

Output:

5.Write the PL/SQL programs to create the procedure to find Fibonacciseries.

```
eclare
    first number:=0;
    second number:=1;
    third number;
    n number:=&n;
    i number;

begin
    dbms_output.put_line('Fibonacci series is:');
    dbms_output.put_line(first);
    dbms_output.put_line(second);

    for i in 2..n
    loop
        third:=first+second;
        first:=second;
```

```
        second:=third;
        dbms_output.put_line(third);
    end loop;
end;
/
```

Output:

6.Write the PL/SQL programs to create the procedure to check the given number is perfect ornot

```
declare
j number(10);
n number(10);
sum number(10):= 0;
begin
j:=&num;
for n in 1..j-1 loop
if mod(j,n)=0 then
sum=sum+n;
end if;
end loop;
if sum=j then
dbms_output.put_line("perfect no.");
else
dbma_output.put_line("not perfect");
end if;
end;/
```

Output:

Week-8: CURSORS

Exp 15:

1.Write a PL/SQL block that will display the name, dept no, salary of fist highest paid employees.

```
DECLARE
   CURSOR dpt_cur IS
     SELECT d.department_id      id,
department_namedptname,
           city,
Nvl(first_name, '...') manager
     FROM   departments d
           left outer join employees e
                    ON ( d.manager_id = e.employee_id )
           join locations l USING(location_id)
     ORDER  BY 2;
emp_nameemployees.first_name%TYPE;
emp_max_salaryemployees.salary%TYPE;
BEGIN
   FOR dept_all IN dpt_cur LOOP
      SELECT Max(salary)
      INTO   emp_max_salary
      FROM   employees
      WHERE  department_id = dept_all.id;

      IF emp_max_salary IS NULL THEN
emp_name := '...';
      ELSE
        SELECT first_name
        INTO   emp_name
        FROM   employees
        WHERE  department_id = dept_all.id
             AND salary = emp_max_salary;
      END IF;

dbms_output.Put_line(Rpad(dept_all.dptname, 20)
                     || Rpad(dept_all.manager, 15)
                     || Rpad(dept_all.city, 20)
                     || Rpad(emp_name, 20));
   END LOOP;
END;
/
```

Output:

2.Write a PL/SQL block that will display the employee details along with salary using Cursors

```
DECLARE
z_employeeemployees%ROWTYPE;
BEGIN
   SELECT *
   INTO   z_employee -- INTO clause always notifies only single row can be fetch
   FROM   employees
   WHERE  employee_id = 149;

dbms_output.Put_line('Employee Details :   ID:'
                    ||z_employee.employee_id
                    ||'  Name: '
                    ||z_employee.first_name
                    ||' '
                    ||z_employee.last_name
                    ||'  Salary:  '
                    ||z_employee.salary);
END; /
```

Output:

Lab Exercise:

3.To write a Cursor to display the list of employees who are working as a Managers or Analyst.

4. To write a Cursor to find employee with given job and dept no.

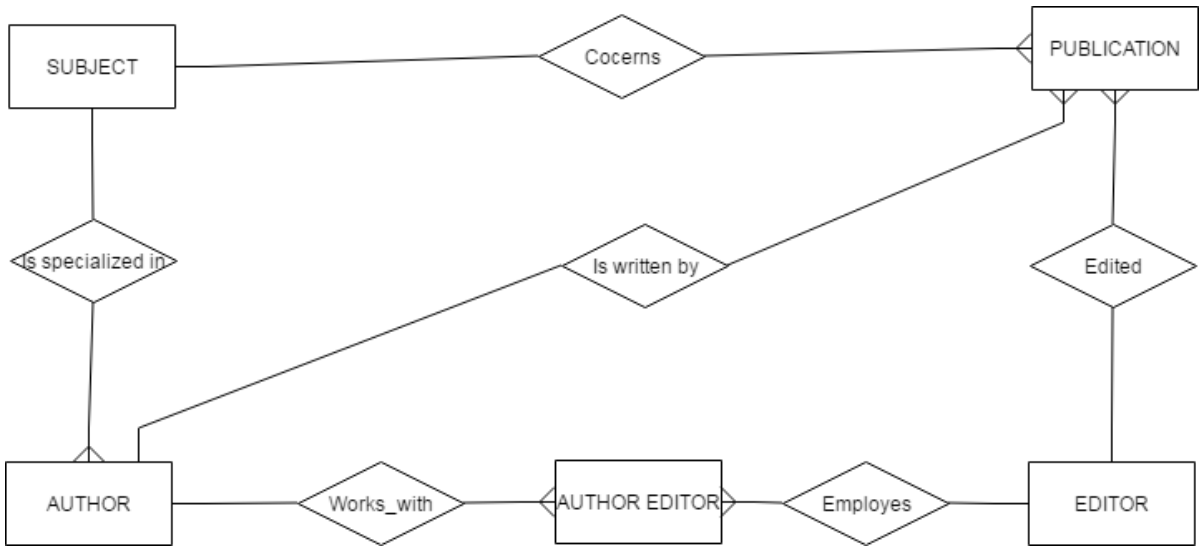
5. Write a PL/SQL block using implicit cursor that will display message, the salaries of allthe employees in the ‘employee’ table are updated. If none of the employee’s salary areupdated we geta message 'None of the salaries were updated'. Else we get a message likefor example, 'Salaries for 1000 employees are updated' if there are 1000 rows in‘employee’ table

WEEK-9

CASE STUDY: BOOK PUBLISHING COMPANY

AIM: A publishing company produces scientific books on various subjects. The books are written by authors who specialize in one particular subject. The company employs editors who, not necessarily being specialists in a particular area, each take sole responsibility for editing one or more publications.

A publication covers essentially one of the specialist subjects and is normally written by a single author. When writing a particular book, each author works with on editor, but may submit another work for publication to be supervised by other editors. To improve their competitiveness, the company tries to employ a variety of authors, more than one author being a specialist in a particular subject.



LAB ASSIGNMENT:

1. Analyze the datarequired.
2. Normalize theattributes.
3. Create the logical data model using E-Rdiagrams

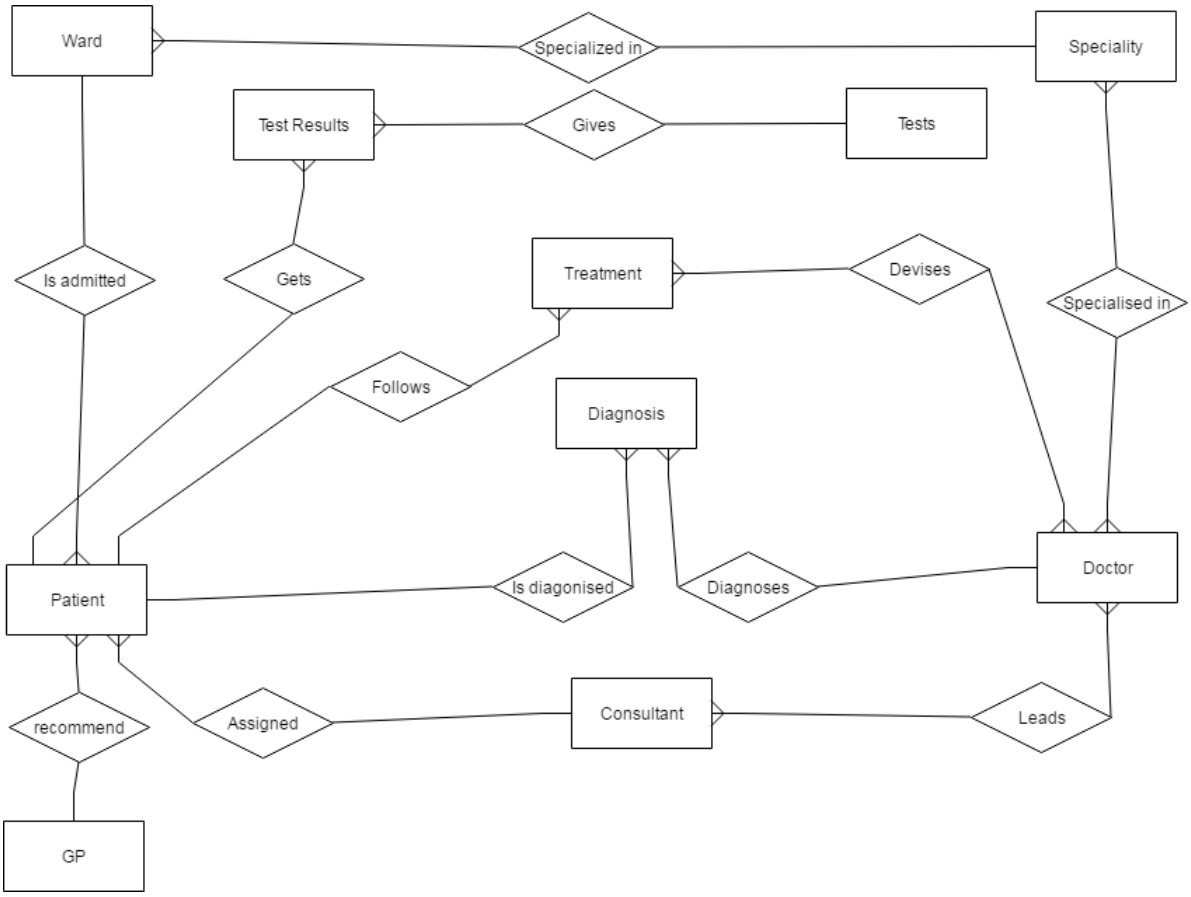
WEEK -10

CASE STUDY: GENERAL HOSPITAL

AIM: A General Hospital consists of a number of specialized wards (such as Maternity, Paediatry, Oncology, etc). Each ward hosts a number of patients, who were admitted on the

recommendation of their own GP and confirmed by a consultant employed by the Hospital. On admission, the personal details of every patient are recorded.

A separate register is to be held to store the information of the tests undertaken and the results of a prescribed treatment. A number of tests may be conducted for each patient. Each patient is assigned to one leading consultant but may be examined by another doctor, if required. Doctors are specialists in some branch of medicine and may be leading consultants for a number of patients, not necessarily from the sameward.



LAB ASSIGNMENT:

1. Analyze the datarequired.
2. Normalize theattributes.
3. Create the logical data model using E-Rdiagrams

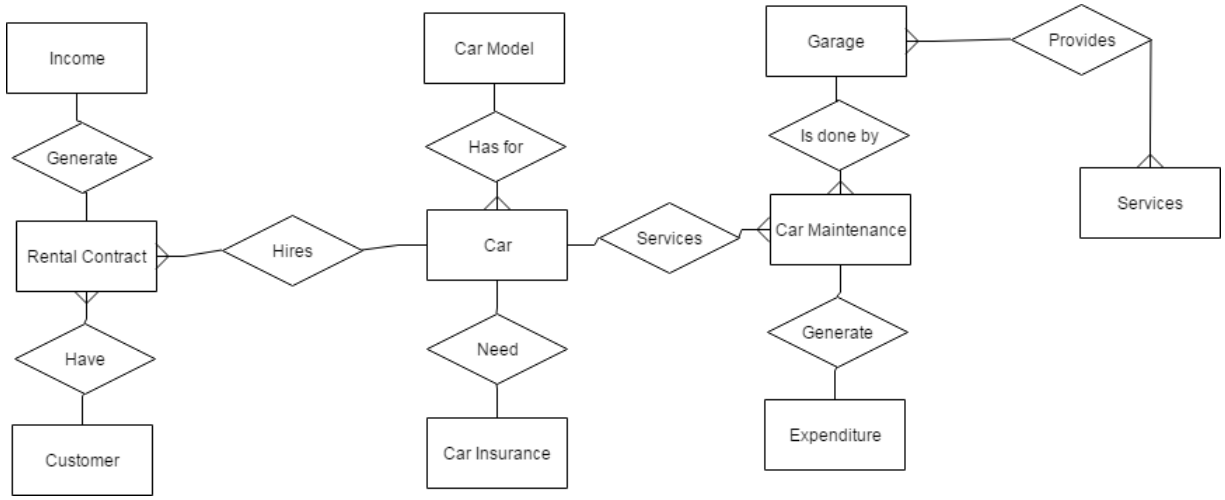
WEEK -11

CASE STUDY: CAR RENTAL COMPANY

AIM: A database is to be designed for a Car Rental Co. (CRC). The information required includes a description of cars, subcontractors (i.e. garages), company expenditures, company revenues and customers. Cars are to be described by such data as: make, model, year of production, engine size, and fuel type, number of passengers, registration number, purchase price, purchase date, rent price and insurance details. It is the company policy not to keep any car for a period exceeding one year.

All major repairs and maintenance are done by subcontractors (i.e. franchised garages), with whom CRC has long-term agreements. Therefore the data about garages to be kept in the database includes garage names, addresses, range of services and the like. Some garages require payments immediately after a repair has been made; with others CRC has made arrangements for credit facilities. Company expenditures are to be registered for all outgoings connected with purchases, repairs, maintenance, insurance etc.

Similarly the cash inflow coming from all sources - car hire, car sales, insurance claims - must be kept of file.CRCmaintains a reasonably stable client base. For this privileged category of customers special credit card facilities are provided. These customers may also book in advance a particular car. These reservations can be made for any period of time up to one month. Casual customers must pay a deposit for an estimated time of rental, unless they wish to pay by credit card. All major credit cards are accepted. Personal details (such as name, address, telephone number, driving license, number) about each customer are kept in thedatabase.



LAB ASSIGNMENT:

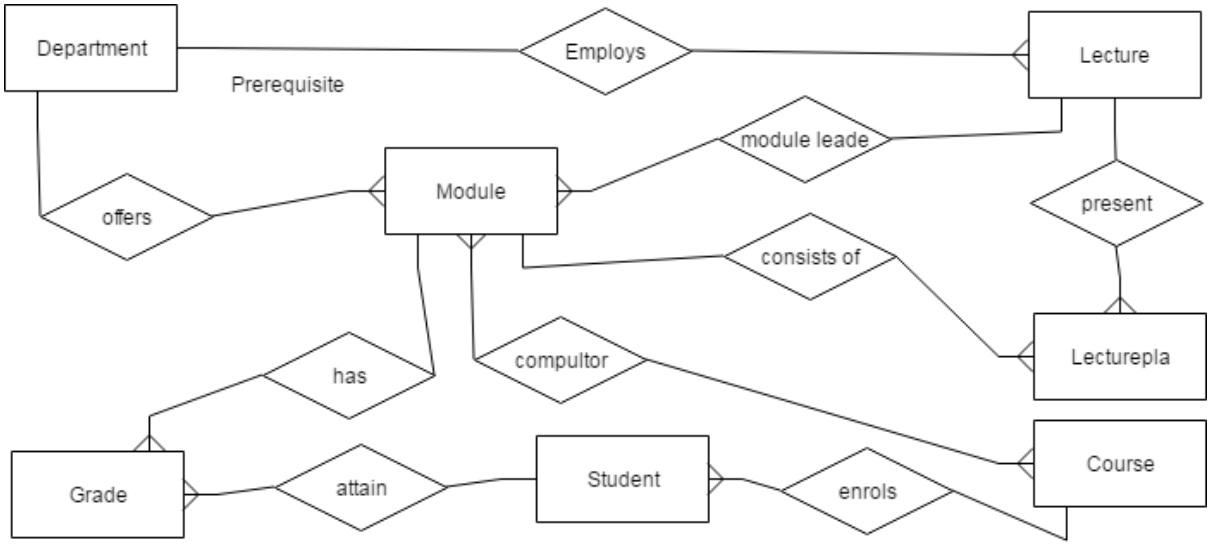
1. Analyze the datarequired.
2. Normalize theattributes.
3. Create the logical data model using E-Rdiagrams

WEEK-12

CASE STUDY: STUDENT PROGRESS MONITORING SYSTEM

AIM: A database is to be designed for a college to monitor students' progress throughout their course of study. The students are reading for a degree (such as BA, BA(Hons) MSc, etc) within the framework of the modular system. The college provides a number of module, each being characterised by its code, title, credit value, module leader, teaching staff and the department they come from. A module is co- ordinated by a module leader who shares teaching duties with one or more lecturers.

A lecturer may teach (and be a module leader for) more than one module. Students are free to choose any module they wish but the following rules must be observed: some modules require pre-requisites modules and some degree programmes have compulsory modules. The database is also to contain some information about students including their numbers, names, addresses, degrees they read for, and their past performance (i.e. modules taken and examination results).



LAB ASSIGNMENT:

1. Analyze the datarequired.
2. Normalize theattributes.
3. Create the logical data model using E-Rdiagrams

DBMS LAB Viva Questions

1) Define Database.

A prearranged collection of figures known as data is called database.

2) What is DBMS?

Database Management Systems (DBMS) are applications designed especially which enable userinteraction with other applications.

3) What are the various kinds of interactions catered by DBMS?

The various kind of interactions catered by DBMS are:

- Data definition
- Update
- Retrieval
- Administration

4) Segregate database technology's development.

The development of database technology is divided into:

- Structure or data model
- Navigational model
- SQL/ relational model

5) Who proposed the relational model?

Edgar F. Codd proposed the relational model in 1970.

6) What are the features of Database language?

A database language may also incorporate features like:

DBMS-specific Configuration and management of storage engine

Computations to modification of query results by computations, like summing, counting, averaging, grouping, sorting and cross-referencing Constraint enforcement Application Programming Interface

7) What do database languages do?

As special-purpose languages, they have:

- Data definition language
- Data manipulation language
- Query language

8) Define database model.

A data model determining fundamentally how data can be stored, manipulated and organisedandthe structure of the database logically is called database model.

9) What is SQL?

Structured Query Language (SQL) being ANSI standard language updates database and commandsfor accessing.

10) Enlist the various relationships of database.

The various relationships of database are:

- One-to-one: Single table having drawn relationship with another table having similar kind
- of columns.
- One-to-many: Two tables having primary and foreign key relation.
- Many-to-many: Junction table having many tables related to many tables.

11) Define Normalization.

Organized data void of inconsistent dependency and redundancy within a database is called normalization.

12) Enlist the advantages of normalizing database.

Advantages of normalizing database are:

- No duplicate entries
- Saves storage space
- Boasts the query performances.

13) Define Denormalization.

Boosting up database performance, adding of redundant data which in turn helps rid of complexdata is called denormalization.

14) Define DDL and DML.

Managing properties and attributes of database is called Data Definition Language(DDL).

Manipulating data in a database such as inserting, updating, deleting is defined as Data Manipulation Language. (DML)

15) Enlist some commands of DDL.

They are:

CREATE:

Create is used in the CREATE TABLE statement. Syntax is:

CREATE TABLE [column name] ([column definitions]) [table parameters]

ALTER:

It helps in modification of an existing object of database. Its syntax is:

ALTER objecttypeobjectname parameters.

DROP:

It destroys an existing database, index, table or view. Its syntax is:

DROP objecttypeobjectname.

16) Define Union All operator and Union.

Full recordings of two tables is Union All operator.

A distinct recording of two tables is Union.

17) Define cursor.

A database object which helps in manipulating data row by row representing a result set is calledcursor.

18) Enlist the cursor types.

They are:

- Dynamic: it reflects changes while scrolling.
- Static: doesn't reflect changes while scrolling and works on recording of snapshot.
- Keyset: data modification without reflection of new data is seen.

19) Enlist the types of cursor.

They types of cursor are:

- Implicit cursor: Declared automatically as soon as the execution of SQL takes place withoutthe awareness of the user.
- Explicit cursor: Defined by PL/ SQL which handles query in more than one row.

20) Define sub-query.

A query contained by a query is called Sub-query.

21) Why is group-clause used?

Group-clause uses aggregate values to be derived by collecting similar data.

22) Compare Non-clustered and clustered index

Both having B-tree structure, non-clustered index has data pointers enabling one table many nonclustered indexes while clustered index is distinct for every table.

23) Define Aggregate functions.

Functions which operate against a collection of values and returning single value is called aggregate functions

24) Define Scalar functions.

Scalar function is depended on the argument given and returns sole value.

25) What restrictions can you apply when you are creating views?

Restrictions that are applied are:

- Only the current database can have views.
- You are not liable to change any computed value in any particular view.
- Integrity constants decide the functionality of INSERT and DELETE.
- Full-text index definitions cannot be applied.
- Temporary views cannot be created.
- Temporary tables cannot contain views.
- No association with DEFAULT definitions.
- Triggers such as INSTEAD OF is associated with views.

26) Define "correlated subqueries".

A 'correlated subquery' is a sort of sub query but correlated subquery is reliant on another queryfor a value that is returned. In case of execution, the sub query is executed first and then thecorrelated query.

27) Define Data Warehousing.

Storage and access of data from the central location in order to take some strategic decision iscalled Data Warehousing. Enterprise management is used for managing the information whoseframework is known as Data Warehousing.

28) Define Join and enlist its types.

Joins help in explaining the relation between different tables. They also enable you to select datawith relation to data in another table.

The various types are:

- INNER JOINs: Blank rows are left in the middle while more than equal to two tables arejoined.
- OUTER JOINs: Divided into Left Outer Join and Right Outer Join. Blank rows are left at thespecified side by joining tables in other side.

  Other joins are CROSS JOINs, NATURAL JOINs, EQUI JOIN and NON-EQUI JOIN.

29) What do you mean by Index hunting?

Indexes help in improving the speed as well as the query performance of database. The procedureof boosting the collection of indexes is named as Index hunting.