SRINIVASA RAMANUJAN INSTITUTE OF TECHNOLOGY
(Autonomous)
Accredited by NAAC with 'A' Grade & Accredited by NBA (EEE, ECE, CSE)
Affiliated to JNTUA & Approved by AICTE
Rotarypuram Village, B K Samudram Mandal, Ananthapuramu - 515701

II B.Tech II Sem

Regulation: SRIT- R19

Database Management Systems

Lab Manual

Prepared by

Faculty Editors: Mr. M. Narasimhulu, Asst. Professor, Dept. of CSE.

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

SRINIVASA RAMANUJAN INSTITUTE OF TECHNOLOGY

B.Tech. II - II Sem.

L	T	P	C
0	0	3	1.5

Database Management systems Lab (194GA05304)

Objectives:

1. To understand the fundamentals of SQL and PL/SQL.
2. To use DDL, DML statements for design of database Schemas
3. To use features of Query language like aggregate functions, group-by clause
4. To use set operations and set comparison operators for evaluating complex queries.
5. To use PL/SQL language constructs to implement triggers, stored procedures, stored functions and cursors.

List of Experiments:

1. Write SQL queries to CREATE TABLES for various databases using DDL commands (i.e. CREATE, ALTER, DROP, TRUNCATE).
2. Write SQL queries to MANIPULATE TABLES for various databases using DML commands (i.e. INSERT, SELECT, UPDATE, DELETE,).
3. Write SQL queries to create VIEWS for various databases (i.e. CREATE VIEW, UPDATE VIEW, ALTER VIEW, and DELETE VIEW).
4. Write SQL queries to perform RELATIONAL SET OPERATIONS (i.e. UNION, UNION ALL, INTERSECT, MINUS, CROSS JOIN, NATURAL JOIN).
5. Write SQL queries to perform SPECIAL OPERATIONS (i.e. ISNULL, BETWEEN, LIKE, IN, EXISTS).
6. Write SQL queries to perform JOIN OPERATIONS (i.e. CONDITIONAL JOIN, EQUI JOIN, LEFT OUTER JOIN, RIGHT OUTER JOIN, FULL OUTER JOIN)
7. Write SQL queries to perform AGGREGATE OPERATIONS (i.e. SUM, COUNT, AVG, MIN, MAX).
8. Write SQL queries to perform ORACLE BUILT-IN FUNCTIONS (i.e. DATE, TIME).
9. Write SQL queries to perform KEY CONSTRAINTS (i.e. PRIMARY KEY, FOREIGN KEY, UNIQUE NOT NULL, CHECK, DEFAULT).
10. Write a PL/SQL program for calculating the factorial of a given number.
11. Write a PL/SQL program for finding the given number is prime number or not.
12. Write a PL/SQL program for displaying the Fibonacci series up to an integer.
13. Write PL/SQL program to implement Stored Procedure on table.
14. Write PL/SQL program to implement Stored Function on table.
15. Write PL/SQL program to implement Trigger on table.
16. Write PL/SQL program to implement Cursor on table.

Course Outcomes:

At the end of the lab, students will be able to

1. Implement a database schema for a given specifications.
2. Insert, alter and modify the database schema and its instances.
3. Write SQL query for a given requirement.
4. Evaluate equivalent SQL queries for a given requirement.
5. Develop PL/SQL triggers, stored procedures, stored functions for a database.

References:

1. "Database System Concepts", 6th Edition by Abraham Silberschatz, Henry F. Korth, S. Sudarshan, McGraw-Hill, 2011.
2. Steven Feuerstein. OraclePL/SQL Programming, 2014.

1. Write SQL queries to CREATE TABLES for various databases using DDL commands (i.e. CREATE, ALTER, DROP, TRUNCATE).

Description:

In SQL DDL commands are used to define the structure of the table. The DDL Commands as shown in the following Tables.

CREATE TABLE:

Creates a table with specified constraints

SYNTAX:

```
CREATE TABLE tablename (
column1 data_ type [constraint] [,
column2 data_ type [constraint] ] [,
PRIMARY KEY (column1 [, column2]) ] [,
FOREIGN KEY (column1 [, column2]) REFERENCES tablename] [,CONSTRAINT constraint]);
```

EXAMPLES:

```
CREATE TABLE Emp (
EmpNo short CONSTRAINT PKey PRIMARY KEY,
EName VarChar(15),
Job Char(10) CONSTRAINT Unik1 UNIQUE,
Mgr short CONSTRAINT FKey1 REFERENCES EMP (EmpNo),
Hiredate Date,
DeptNo short CONSTRAINT FKey2 REFERENCES
DEPT(DeptNo));
```

```
CREATE TABLE prog20 (
pname varchar2(20) NOT NULL,
doj date NOT NULL,
dob date NOT NULL,
sex varchar(1) NOT NULL,
prof1 varchar(20),
prof2 varchar(20),
salary number(7,2) NOT NULL);

CREATE TABLE businfo AS SELECT * FROM bus;
```

ALTER TABLE:

Used to add or modify table details like column names and data types, column constraints.

SYNTAXES:

```
ALTER TABLE tablename
{ADD | MODIFY} (column_name data_type [ { ADD|MODIFY }
Column_name data_type]);

ALTER TABLE tablename
ADD constraint [ADD constraint];

ALTER TABLE tablename
DROP { PRIMARY KEY | COLUMN column_name | CONSTRAINT constraint_name);

ALTER TABLE tablename
ENABLE CONSTRAINT constraint_name;
```

EXAMPLES:

```
ALTER TABLE EMP ADD CONSTRAINT Pkey1 PRIMARY KEY (EmpNo);
ALTER TABLE EMP DROP CONSTRAINT Pkey1;
```


DROP TABLE:

Deletes the specified table.

SYNTAX:

```
DROP TABLE table_name;
```

EXAMPLES:

DROP TABLE EMP;

TRUNCATE TABLE:

To remove all rows in a specified table.

SYNTAX:

TRUNCATE TABLE table_name;

EXAMPLE:

TRUNCATE TABLE department;

RENAME TABLE:

To rename table_name, column_name

SYNTAXES:

RENAME new_table_name TO old_table_name;

ALTER TABLE talbe_name RENAME TO new_table_name;

ALTER TABLE talbe_name RENAME COLUMN old_column TO new_column;

EXAMPLES:

ALTER TABLE emp1
RENAME COLUMN eid TO emp_num;

ALTER TABLE employee
RENAME to emp;

2. **Write SQL queries to MANIPULATE TABLES for various databases using DML commands (i.e. INSERT, SELECT, UPDATE, DELETE,).**

INSERT COMMAND:

It is used to add values to a table.

SYNTAX:

INSERT INTO tablename
VALUES (value1,value2,...,valuen);

INSERT INTO tablename (column1, column2,…,column)
VALUES (value1, value2,...,valuen);

SELECT COMMAND:

The SELECT command used to list the contents of a table.

SYNTAX:

SELECT columnlist
FROM tablelist
[WHERE conditionlist]
[GROUP BY columnlist]
[HAVING conditionlist]
[ORDER BY columnlist [ASC|DESC]];

UPDATE COMMAND:

The update command used to modify the contents of specified table.

SYNTAX:

UPDATE tablename
SET column_name = value[,
Column_name = value]
[WHERE condition_lsit];

DELETE COMMAND:

To delete all rows or specified rows in a table.

SYNTAX:

DELETE FROM tablename [WHERE condition_list];

Examples on INSERT:

INSERT INTO S (SNO, SNAME) VALUES ('S1', 'Smith');

INSERT INTO NEW_SUPPLIER (SNO, SNAME)
SELECT SNO, SNAME FROM S
WHERE CITY IN ('BLORE','MADRAS');

INSERT INTO prog VALUES ('kkk','05-may-56');

INSERT INTO prog VALUES ('Hema','25-sept-01'28-jan-85','f','c','c++','25000');

INSERT INTO prog VALUES ('&pname','&doj');

Examples on UPDATE:

UPDATE S SET CITY = 'KANPUR' WHERE SNO='S1';

UPDATE EMP SET SAL = 1.10 * SAL;

UPDATE emp SET sal=20000 WHERE empno=7369;

Examples on DELETE:

DELETE FROM SP WHERE PNO= 'P1';

DELETE FROM SP;

DELETE FROM emp WHERE empno=7369;

Examples on SELECT:

SELECT * FROM instructor where dept_name='history';

SELECT * From instructor;

SELECT * FROM History_instructor;

3. **Write SQL queries to create VIEWS for various databases (i.e. CREATE VIEW, UPDATE VIEW, ALTER VIEW, and DELETE VIEW).**

What is view?

A relation that is not part of a logical model, but it is made visible to a user as a virtual relation, is called a view. Therefore, view is referred as virtual relation.

View syntax:

CREATE VIEW VIEW_NAME AS <QUERY EXPRESSION>

Example-1:

Create a view for clerk to check instructor information with out salary visibility.

```
CREATE VIEW FACULTY AS
SELECT ID,NAME,DEPT_NAME
FROM INSTRUCTOR;
```

Example-2:

To create a view that lists all course sections offered by the Physics department in the Fall 2009 semester with the building and room number of each section

```
CREATE VIEW PHYSICS_FALL_2009 AS
SELECT COURSE.COURSE_ID, SEC_ID, BUILDING, ROOM_NUMBER
FROM COURSE, SECTION
WHERE COURSE.COURSE_ID = SECTION.COURSE_ID
AND COURSE.DEPT_NAME = 'PHYSICS'
AND SECTION.SEMESTER = 'FALL'
AND SECTION.YEAR = '2009';
```

Can we use views in SQL Queries?

Yes, we can use views it as normal relations in SQL.

Using the view physics fall 2009, we can find all Physics courses offered in the Fall 2009 semester in the Watson building

```
SELECT COURSE_ID
FROM PHYSICS_FALL_2009
WHERE BUILDING= 'WATSON';
```

Example-3:

Create a view that hold department names and Total salary of that department.

```
CREATE VIEW DEPARTMENTS_TOTAL_SALARY(DEPT_NAME, TOTAL_SALARY) AS
SELECT DEPT_NAME, SUM(SALARY)
FROM INSTRUCTOR
GROUP BY DEPT_NAME;
```


Can we define a view from another view?

Yes, By Using a view we can create another view.

Example-4:

We can define a view physics fall 2009 watson that lists the course ID and room number of all Physics courses offered in the Fall 2009 semester in the Watson building.

```
CREATE VIEW PHYSICS_FALL_2009 WATSON AS
SELECT COURSE_ID, ROOM_NUMBER
FROM PHYSICS_FALL 2009
WHERE BUILDING= 'WATSON';
```

An equivalent relation of view with out using view as original relation

```
CREATE VIEW PHYSICS_FALL_2009_WATSON AS
(SELECT COURSE_ID, ROOM_NUMBER
FROM (SELECT COURSE.COURSE_ID, BUILDING, ROOM_NUMBER
FROM COURSE, SECTION
WHERE COURSE.COURSE_ID = SECTION.COURSE_ID
AND COURSE.DEPT_NAME = 'PHYSICS'
AND SECTION.SEMESTER = 'FALL'
AND SECTION.YEAR = '2009')
WHERE BUILDING= 'WATSON';
```

We can perform insert,delete and update sql commands on views if the following conditions are met:

1. The from clause has only one database relation.
2. The select clause contains only attribute names of the relation, and does not have any expressions, aggregates, or distinct specification.
3. Any attribute not listed in the select clause can be set to null; that is, it does not have a not null constraint and is not part of a primary key.
4. The query does not have a group by or having clause**.**

Example-4:

Create a view that able to satify the above constraints to do insertion, deletion, and Update.


```
CREATE VIEW HISTORY_INSTRUCTORS AS
SELECT *
FROM INSTRUCTOR
WHERE DEPT_NAME= 'HISTORY';
```

To verify the output of view use the following statement.

```
SELECT * FROM history_instructors;
```

Commands to insert, Delete and update view

```
INSERT INTO History_instructors VALUES('58584','Dacid Coy','History',34000);
Update History_instructors SET name='james robert' where Id='58584';
Delete FROM History_instructors where id='58584';

SELECT * FROM instructor where dept_name='history';
SELECT * From instructor;
SELECT * FROM History_instructor;
```

To delete a view or truncate view.

```
TRUNCATE view History_instructors;
SELECT * FROM instrucor;
SELECT * FROM History_instructor;
```

Verify that content from the instructor is deleted or not for both delete command and Truncate command

```
DROP view History_instructors;
```

4. **Write SQL queries to perform RELATIONAL SET OPERATIONS (i.e. UNION, UNION ALL, INTERSECT, MINUS, CROSS JOIN, NATURAL JOIN).**

Query for Set Operations.

Section table

```
CREATE TABLE SECTION
(COURSE_ID    VARCHAR2(8),    SEC_ID                VARCHAR2(8),
 SEMESTER          VARCHAR2(6)  CHECK (SEMESTER IN ('FALL', 'WINTER',
'SPRING', 'SUMMER')),
  YEAR              NUMERIC(4,0) CHECK (YEAR > 1701 AND YEAR < 2100),
  BUILDING          VARCHAR2(15),
  ROOM_NUMBER             VARCHAR2(7),
  TIME_SLOT_ID      VARCHAR2(4),
```

```
        PRIMARY KEY (COURSE_ID, SEC_ID, SEMESTER, YEAR),
        FOREIGN KEY (COURSE_ID) REFERENCES COURSE(COURSE_ID)
                ON DELETE CASCADE,
        FOREIGN KEY (BUILDING, ROOM_NUMBER) REFERENCES CLASSROOM(BUILDING,
ROOM_NUMBER)
                ON DELETE SET NULL
      );
```

Classroom Table

```
      CREATE TABLE CLASSROOM
      (BUILDING             VARCHAR2(15),
       ROOM_NUMBER                VARCHAR2(7),
       CAPACITY             NUMERIC(4,0),
       PRIMARY KEY (BUILDING, ROOM_NUMBER)
      );
```

Insertion of VALUES in Classroom Table

```
      INSERT INTO classroom VALUES ('Packard', '101', '500');
      INSERT INTO classroom VALUES ('Painter', '514', '10');
      INSERT INTO classroom VALUES ('Taylor', '3128', '70');
      INSERT INTO classroom VALUES ('Watson', '100', '30');
      INSERT INTO classroom VALUES ('Watson', '120', '50');
```

Insert of Values in Section Table

```
      INSERT INTO section VALUES ('BIO-101', '1', 'Summer', '2009', 'Painter', '514', 'B');
      INSERT INTO section VALUES ('BIO-301', '1', 'Summer', '2010', 'Painter', '514', 'A');
      INSERT INTO section VALUES ('CS-101', '1', 'Fall', '2009', 'Packard', '101', 'H');
      INSERT INTO section VALUES ('CS-101', '1', 'Spring', '2010', 'Packard', '101', 'F');
      INSERT INTO section VALUES ('CS-190', '1', 'Spring', '2009', 'Taylor', '3128', 'E');
      INSERT INTO section VALUES ('CS-190', '2', 'Spring', '2009', 'Taylor', '3128', 'A');
      INSERT INTO section VALUES ('CS-315', '1', 'Spring', '2010', 'Watson', '120', 'D');
      INSERT INTO section VALUES ('CS-319', '1', 'Spring', '2010', 'Watson', '100', 'B');
      INSERT INTO section VALUES ('CS-319', '2', 'Spring', '2010', 'Taylor', '3128', 'C');
      INSERT INTO section VALUES ('CS-347', '1', 'Fall', '2009', 'Taylor', '3128', 'A');
      INSERT INTO section VALUES ('EE-181', '1', 'Spring', '2009', 'Taylor', '3128', 'C');
```

```
INSERT INTO section VALUES ('FIN-201', '1', 'Spring', '2010', 'Packard', '101', 'B');
INSERT INTO section VALUES ('HIS-351', '1', 'Spring', '2010', 'Painter', '514', 'C');
INSERT INTO section VALUES ('MU-199', '1', 'Spring', '2010', 'Packard', '101', 'D');
INSERT INTO section VALUES ('PHY-101', '1', 'Fall', '2009', 'Watson', '100', 'A');
```

Union operation

```
SELECT course_id
FROM section
where semester = 'Fall' AND year= 2009)
UNION
(SELECT course_id
FROM section
WHERE semester = 'Spring' AND year= 2010);
```

Union all Operation

```
select course id
from section
where semester = 'Fall' and year= 2009)
UNION ALL
select course id
from section
where semester = 'Spring' and year= 2010;
```

Intersect Operation

```
(select course id
from section
where semester = 'Fall' and year= 2009)
INTERSECT
(select course id
from section
where semester = 'Spring' and year= 2010);
```

Intersect all operation

```
(select course id
from section
where semester = 'fall' and year= 2009)
INTERSECT ALL
(select course id
from section
where semester = 'spring' and year= 2010);
```

except or minus operation

```
(select course id
from section
where semester = 'Fall' and year= 2009)
EXCEPT
(select course id
from section
where semester = 'Spring' and year= 2010);
```

except all or minus all operations

```
(select course id
from section
where semester = 'Fall' and year= 2009)
EXCEPT ALL
(select course id
from section     where semester = 'Spring' and year= 2010);
```

5. **Write SQL queries to perform SPECIAL OPERATIONS (i.e. ISNULL, BETWEEN, LIKE, IN, EXISTS).**

Instructor Table

```
CREATE TABLE INSTRUCTOR
(ID             VARCHAR2(5),
 NAME           VARCHAR2(20) NOT NULL,
 DEPT_NAME      VARCHAR2(20),
 SALARY                 NUMERIC(8,2) CHECK (SALARY > 29000),
 PRIMARY KEY (ID),
 FOREIGN KEY (DEPT_NAME) REFERENCES DEPARTMENT(DEPT_NAME)
        ON DELETE SET NULL
);
```

instances of instructor table

```
INSERT INTO instructor VALUES ('10101', 'Srinivasan', 'Comp. Sci.', '65000');
INSERT INTO instructor VALUES ('12121', 'Wu', 'Finance', '90000');
INSERT INTO instructor VALUES ('15151', 'Mozart', 'Music', '40000');
INSERT INTO instructor VALUES ('22222', 'Einstein', 'Physics', '95000');
INSERT INTO instructor VALUES ('32343', 'El Said', 'History', '60000');
INSERT INTO instructor VALUES ('33456', 'Gold', 'Physics', '87000');
INSERT INTO instructor VALUES ('45565', 'Katz', 'Comp. Sci.', '75000');
INSERT INTO instructor VALUES ('58583', 'Califieri', 'History', '62000');
INSERT INTO instructor VALUES ('76543', 'Singh', 'Finance', '80000');
INSERT INTO instructor VALUES ('76766', 'Crick', 'Biology', '72000');
INSERT INTO instructor VALUES ('83821', 'Brandt', 'Comp. Sci.', '92000');
INSERT INTO instructor VALUES ('98345', 'Kim', 'Elec. Eng.', '80000');
```

department table

```
CREATE TABLE DEPARTMENT
(DEPT_NAME   VARCHAR2(20),
 BUILDING          VARCHAR2(15),
 BUDGET            NUMERIC(12,2) CHECK (BUDGET > 0),
 PRIMARY KEY (DEPT_NAME)
);
```

instances of department table

```
INSERT INTO department VALUES ('Biology', 'Watson', '90000');
INSERT INTO department VALUES ('Comp. Sci.', 'Taylor', '100000');
INSERT INTO department VALUES ('Elec. Eng.', 'Taylor', '85000');
INSERT INTO department VALUES ('Finance', 'Painter', '120000');
INSERT INTO department VALUES ('History', 'Painter', '50000');
INSERT INTO department VALUES ('Music', 'Packard', '80000');
INSERT INTO department VALUES ('Physics', 'Watson', '70000');
```

Course table

```
CREATE TABLE COURSE
```

```
(COURSE_ID          VARCHAR2(8),
 TITLE              VARCHAR2(50),
 DEPT_NAME          VARCHAR2(20),
 CREDITS                    NUMERIC(2,0) CHECK (CREDITS > 0),
 PRIMARY KEY (COURSE_ID),
 FOREIGN KEY (DEPT_NAME) REFERENCES DEPARTMENT(DEPT_NAME)
        ON DELETE SET NULL
);
```

instances of course table

```
INSERT INTO course VALUES ('BIO-101', 'Intro. to Biology', 'Biology', '4');
INSERT INTO course VALUES ('BIO-301', 'Genetics', 'Biology', '4');
INSERT INTO course VALUES ('BIO-399', 'Computational Biology', 'Biology', '3');
INSERT INTO course VALUES ('CS-101', 'Intro. to Computer Science', 'Comp. Sci.', '4');
INSERT INTO course VALUES ('CS-190', 'Game Design', 'Comp. Sci.', '4');
INSERT INTO course VALUES ('CS-315', 'Robotics', 'Comp. Sci.', '3');
INSERT INTO course VALUES ('CS-319', 'Image Processing', 'Comp. Sci.', '3');
INSERT INTO course VALUES ('CS-347', 'Database System Concepts', 'Comp. Sci.', '3');
INSERT INTO course VALUES ('EE-181', 'Intro. to Digital Systems', 'Elec. Eng.', '3');
INSERT INTO course VALUES ('FIN-201', 'Investment Banking', 'Finance', '3');
INSERT INTO course VALUES ('HIS-351', 'World History', 'History', '3');
INSERT INTO course VALUES ('MU-199', 'Music Video Production', 'Music', '3');
INSERT INTO course VALUES ('PHY-101', 'Physical Principles', 'Physics', '4');
```

teaches table

```
CREATE TABLE TEACHES
(ID             VARCHAR2(5),
 COURSE_ID      VARCHAR2(8),
 SEC_ID         VARCHAR2(8),
 SEMESTER       VARCHAR2(6),
 YEAR           NUMERIC(4,0),
 PRIMARY KEY (ID, COURSE_ID, SEC_ID, SEMESTER, YEAR),
 FOREIGN KEY (COURSE_ID,SEC_ID, SEMESTER, YEAR) REFERENCES
SECTION(COURSE_ID,SEC_ID, SEMESTER, YEAR)
        ON DELETE CASCADE,
```

```
    FOREIGN KEY (ID) REFERENCES INSTRUCTOR(ID)
        ON DELETE CASCADE
);
```

instances of teach table

```
INSERT INTO teaches VALUES ('10101', 'CS-101', '1', 'Fall', '2009');
INSERT INTO teaches VALUES ('10101', 'CS-315', '1', 'Spring', '2010');
INSERT INTO teaches VALUES ('10101', 'CS-347', '1', 'Fall', '2009');
INSERT INTO teaches VALUES ('12121', 'FIN-201', '1', 'Spring', '2010');
INSERT INTO teaches VALUES ('15151', 'MU-199', '1', 'Spring', '2010');
INSERT INTO teaches VALUES ('22222', 'PHY-101', '1', 'Fall', '2009');
INSERT INTO teaches VALUES ('32343', 'HIS-351', '1', 'Spring', '2010');
INSERT INTO teaches VALUES ('45565', 'CS-101', '1', 'Spring', '2010');
INSERT INTO teaches VALUES ('45565', 'CS-319', '1', 'Spring', '2010');
INSERT INTO teaches VALUES ('76766', 'BIO-101', '1', 'Summer', '2009');
INSERT INTO teaches VALUES ('76766', 'BIO-301', '1', 'Summer', '2010');
INSERT INTO teaches VALUES ('83821', 'CS-190', '1', 'Spring', '2009');
INSERT INTO teaches VALUES ('83821', 'CS-190', '2', 'Spring', '2009');
INSERT INTO teaches VALUES ('83821', 'CS-319', '2', 'Spring', '2010');
INSERT INTO teaches VALUES ('98345', 'EE-181', '1', 'Spring', '2009');
```

student table

```
CREATE TABLE STUDENT
(ID             VARCHAR2(5),
 NAME           VARCHAR2(20) NOT NULL,
 DEPT_NAME      VARCHAR2(20),
 TOT_CRED       NUMERIC2(3,0) CHECK (TOT_CRED >= 0),
 PRIMARY KEY (ID),
 FOREIGN KEY (DEPT_NAME) REFERENCES DEPARTMENT(DEPT_NAME)
        ON DELETE SET NULL
);
```

instances of student table

```
INSERT INTO student VALUES ('00128', 'Zhang', 'Comp. Sci.', '102');
INSERT INTO student VALUES ('12345', 'Shankar', 'Comp. Sci.', '32');
INSERT INTO student VALUES ('19991', 'Brandt', 'History', '80');
INSERT INTO student VALUES ('23121', 'Chavez', 'Finance', '110');
INSERT INTO student VALUES ('44553', 'Peltier', 'Physics', '56');
INSERT INTO student VALUES ('45678', 'Levy', 'Physics', '46');
INSERT INTO student VALUES ('54321', 'Williams', 'Comp. Sci.', '54');
INSERT INTO student VALUES ('55739', 'Sanchez', 'Music', '38');
INSERT INTO student VALUES ('70557', 'Snow', 'Physics', '0');
INSERT INTO student VALUES ('76543', 'Brown', 'Comp. Sci.', '58');
INSERT INTO student VALUES ('76653', 'Aoi', 'Elec. Eng.', '60');
INSERT INTO student VALUES ('98765', 'Bourikas', 'Elec. Eng.', '98');
INSERT INTO student VALUES ('98988', 'Tanaka', 'Biology', '120');
```

section table

```
CREATE TABLE SECTION
(COURSE_ID              VARCHAR2(8),
SEC_ID                  VARCHAR2(8),
SEMESTER                VARCHAR2(6)
        CHECK (SEMESTER IN ('FALL', 'WINTER', 'SPRING', 'SUMMER')),
YEAR                    NUMERIC(4,0) CHECK (YEAR > 1701 AND YEAR < 2100),
BUILDING                VARCHAR2(15),
ROOM_NUMBER             VARCHAR2(7),
TIME_SLOT_ID            VARCHAR2(4),
PRIMARY KEY (COURSE_ID, SEC_ID, SEMESTER, YEAR),
FOREIGN KEY (COURSE_ID) REFERENCES COURSE(COURSE_ID)
        ON DELETE CASCADE,
FOREIGN KEY (BUILDING, ROOM_NUMBER) REFERENCES CLASSROOM(BUILDING, ROOM_NUMBER)
        ON DELETE SET NULL
);
```

instances of course table

```
INSERT INTO course VALUES ('BIO-101', 'Intro. to Biology', 'Biology', '4');
INSERT INTO course VALUES ('BIO-301', 'Genetics', 'Biology', '4');
INSERT INTO course VALUES ('BIO-399', 'Computational Biology', 'Biology', '3');
INSERT INTO course VALUES ('CS-101', 'Intro. to Computer Science', 'Comp. Sci.', '4');
INSERT INTO course VALUES ('CS-190', 'Game Design', 'Comp. Sci.', '4');
```

```
INSERT INTO course VALUES ('CS-315', 'Robotics', 'Comp. Sci.', '3');
INSERT INTO course VALUES ('CS-319', 'Image Processing', 'Comp. Sci.', '3');
INSERT INTO course VALUES ('CS-347', 'Database System Concepts', 'Comp. Sci.', '3');
INSERT INTO course VALUES ('EE-181', 'Intro. to Digital Systems', 'Elec. Eng.', '3');
INSERT INTO course VALUES ('FIN-201', 'Investment Banking', 'Finance', '3');
INSERT INTO course VALUES ('HIS-351', 'World History', 'History', '3');
INSERT INTO course VALUES ('MU-199', 'Music Video Production', 'Music', '3');
INSERT INTO course VALUES ('PHY-101', 'Physical Principles', 'Physics', '4');
```

Course table

```
CREATE TABLE COURSE
(COURSE_ID          VARCHAR2(8),
 TITLE              VARCHAR2(50),
 DEPT_NAME          VARCHAR2(20),
 CREDITS            NUMERIC(2,0) CHECK (CREDITS > 0),
 PRIMARY KEY (COURSE_ID),
 FOREIGN KEY (DEPT_NAME) REFERENCES DEPARTMENT(DEPT_NAME)
        ON DELETE SET NULL
);
```

instances of Course table

```
INSERT INTO course VALUES ('BIO-101', 'Intro. to Biology', 'Biology', '4');
INSERT INTO course VALUES ('BIO-301', 'Genetics', 'Biology', '4');
INSERT INTO course VALUES ('BIO-399', 'Computational Biology', 'Biology', '3');
INSERT INTO course VALUES ('CS-101', 'Intro. to Computer Science', 'Comp. Sci.', '4');
INSERT INTO course VALUES ('CS-190', 'Game Design', 'Comp. Sci.', '4');
INSERT INTO course VALUES ('CS-315', 'Robotics', 'Comp. Sci.', '3');
INSERT INTO course VALUES ('CS-319', 'Image Processing', 'Comp. Sci.', '3');
INSERT INTO course VALUES ('CS-347', 'Database System Concepts', 'Comp. Sci.', '3');
INSERT INTO course VALUES ('EE-181', 'Intro. to Digital Systems', 'Elec. Eng.', '3');
INSERT INTO course VALUES ('FIN-201', 'Investment Banking', 'Finance', '3');
INSERT INTO course VALUES ('HIS-351', 'World History', 'History', '3');
INSERT INTO course VALUES ('MU-199', 'Music Video Production', 'Music', '3');
INSERT INTO course VALUES ('PHY-101', 'Physical Principles', 'Physics', '4');
```

Special operations: IS NULL, IS NOT NULL, BETWEEN, IN, LIKE, EXISTS.

To Understand the usage of IN Operation and NOT IN.

IN act as a membership function.

```
select distinct course_id
from section
where semester = 'Fall' and year= 2009 and
course id in (select course_id
from section
where semester = 'Spring' and year= 2010);

select distinct course_id
from section
where semester = 'Fall' and year= 2009 and
course_id not in (select course_id
from section
where semester = 'Spring' and year= 2010);
```

It can also used on enumerated sets.

```
select distinct name
from instructor
where name not in ('Mozart', 'Einstein');
```

It is also possible to test for membership in an arbitary relation in SQL.
find the total number of (distinct) students who have taken course sections taught by the instructor with ID 110011"

```
select count (distinct ID )
from takes
where (course id, sec id, semester, year) in (select course id, sec id, semester, year
from teaches
where teaches. ID = 10101);
```


Exist keyword used to test empty relation

```
select course_id
from section as S
where semester = 'Fall' and year= 2009 and
exists   (select *
```

```
        from section as T
        where semester = 'Spring' and year= 2010 and
        S.course_id= T.course_id);

select distinct S.ID , S.name
from student as S
where not exists ((select course_id
        from course
        where dept name = 'Biology')
        minus
        (select T.course_id
        from takes as T
        where S.ID = T.ID ));
```

keyword null is used to test a null value

```
select name
from instructor
where salary is null;

select name
from instructor
where salary is not null;
```

keyword Between is used in where clause, it used to test a value is be less than or equal to some value and greater than or equal to some other value.

```
select name
from instructor
where salary between 90000 and 100000;
```


an equivalent query as follows

```
select name
from instructor
where salary <= 100000 and salary >= 90000;
```

- Use 'not between' keyword to the above between statement and justify your answer.

LIKE keyword is used to perform pattern matching on the strings.

To perform pattern matching, SQL describes two special characters.

1. Percant(%): The % character matches any substring.

2. Underscore(_): The _ character matches any character.

"Find the names of all departments whose building name includes the substring 'Watson'."

```
select dept_name
from department
where building like '%Watson%';
```

6. Write SQL queries to perform JOIN OPERATIONS (i.e. CONDITIONAL JOIN, EQUI JOIN, LEFT OUTER JOIN, RIGHT OUTER JOIN, FULL OUTER JOIN)

Natural JOIN

For all instructors in the university who have taught some course, find their names and the course ID of all courses they taught"

```
select name, course id
from instructor natural join teaches;
```

An Equivalent Query

```
select name, course id
from instructor, teaches
where instructor.ID = teaches.ID;
```

List the names of instructors along with the the titles of courses that they teach.

```
select name, title
from (instructor natural join teaches) join course using (course id);
```

An Equivalent Query

```
select name, title
from instructor natural join teaches, course
where teaches.course_id= course.course_id;
```

Did the Following Query works fine or not Justify your Answer.

```
select name, title
from instructor natural join teaches natural join course;
```

CONDITIONAL JOIN

JOIN with ON keyworkd refers it as Conditional JOIN

```
select *
from student join takes on student.ID = takes.ID ;

select student.ID as ID , name, dept_name, tot_cred,
course_id, sec_id, semester, year, grade
from student join takes on student.ID = takes.ID ;
```

OUTER JOINS

LEFT OUTER JOIN preserves tuples only in the relation named before (to the left of) the left outer join operation.

```
select *
from student natural left outer join takes;
```

To Find all students who have not taken a course.

```
select ID
from student natural left outer join takes
where course id is null;
```

RIGHT OUTER JOIN preserves tuples only in the relation named after (to the right of) the right outer join operation.

```
select *
from takes natural right outer join student;
```

Full Outer JOIN preserves tuples in both relations.

Display a list of all students in the Comp. Sci. department, along with the course sections, if any, that they have taken in Spring 2009; all course sections from Spring 2009 must be displayed, even if no student from the Comp. Sci. department has taken the course section."

```
select *
from (select *
from student
where dept name= 'Comp. Sci')
natural full outer join
(select *
from takes
```

where semester = 'Spring' and year = 2009);

7. Write SQL queries to perform AGGREGATE OPERATIONS (i.e. SUM, COUNT, AVG, MIN, MAX).

Instructor Table

```
CREATE TABLE INSTRUCTOR
(ID             VARCHAR2(5),
 NAME           VARCHAR2(20) NOT NULL,
 DEPT_NAME      VARCHAR2(20),
 SALARY                 NUMERIC(8,2) CHECK (SALARY > 29000),
 PRIMARY KEY (ID),
 FOREIGN KEY (DEPT_NAME) REFERENCES DEPARTMENT(DEPT_NAME)
        ON DELETE SET NULL
);
```

instances of instructor table

```
insert into instructor values ('10101', 'Srinivasan', 'Comp. Sci.', '65000');
insert into instructor values ('12121', 'Wu', 'Finance', '90000');
insert into instructor values ('15151', 'Mozart', 'Music', '40000');
insert into instructor values ('22222', 'Einstein', 'Physics', '95000');
insert into instructor values ('32343', 'El Said', 'History', '60000');
insert into instructor values ('33456', 'Gold', 'Physics', '87000');
insert into instructor values ('45565', 'Katz', 'Comp. Sci.', '75000');
insert into instructor values ('58583', 'Califieri', 'History', '62000');
insert into instructor values ('76543', 'Singh', 'Finance', '80000');
insert into instructor values ('76766', 'Crick', 'Biology', '72000');
insert into instructor values ('83821', 'Brandt', 'Comp. Sci.', '92000');
insert into instructor values ('98345', 'Kim', 'Elec. Eng.', '80000');
```

department table

```
CREATE TABLE DEPARTMENT
(DEPT_NAME  VARCHAR2(20),
 BUILDING           VARCHAR2(15),
 BUDGET             NUMERIC(12,2) CHECK (BUDGET > 0),
 PRIMARY KEY (DEPT_NAME)
);
```

instances of department table

```
insert into department values ('Biology', 'Watson', '90000');
```

```
insert into department values ('Comp. Sci.', 'Taylor', '100000');
insert into department values ('Elec. Eng.', 'Taylor', '85000');
insert into department values ('Finance', 'Painter', '120000');
insert into department values ('History', 'Painter', '50000');
insert into department values ('Music', 'Packard', '80000');
insert into department values ('Physics', 'Watson', '70000');
```

Course table

```
CREATE TABLE COURSE
(COURSE_ID          VARCHAR2(8),
 TITLE              VARCHAR2(50),
 DEPT_NAME          VARCHAR2(20),
 CREDITS                    NUMERIC(2,0) CHECK (CREDITS > 0),
 PRIMARY KEY (COURSE_ID),
 FOREIGN KEY (DEPT_NAME) REFERENCES DEPARTMENT(DEPT_NAME)
        ON DELETE SET NULL
);
```

instances of course table

```
insert into course values ('BIO-101', 'Intro. to Biology', 'Biology', '4');
insert into course values ('BIO-301', 'Genetics', 'Biology', '4');
insert into course values ('BIO-399', 'Computational Biology', 'Biology', '3');
insert into course values ('CS-101', 'Intro. to Computer Science', 'Comp. Sci.', '4');
insert into course values ('CS-190', 'Game Design', 'Comp. Sci.', '4');
insert into course values ('CS-315', 'Robotics', 'Comp. Sci.', '3');
insert into course values ('CS-319', 'Image Processing', 'Comp. Sci.', '3');
insert into course values ('CS-347', 'Database System Concepts', 'Comp. Sci.', '3');
insert into course values ('EE-181', 'Intro. to Digital Systems', 'Elec. Eng.', '3');
insert into course values ('FIN-201', 'Investment Banking', 'Finance', '3');
insert into course values ('HIS-351', 'World History', 'History', '3');
insert into course values ('MU-199', 'Music Video Production', 'Music', '3');
insert into course values ('PHY-101', 'Physical Principles', 'Physics', '4');
```

teaches table

```
CREATE TABLE TEACHES
(ID                 VARCHAR2(5),
 COURSE_ID          VARCHAR2(8),
 SEC_ID             VARCHAR2(8),
 SEMESTER           VARCHAR2(6),
```

```
        YEAR                    NUMERIC(4,0),
        PRIMARY KEY (ID, COURSE_ID, SEC_ID, SEMESTER, YEAR),
        FOREIGN KEY (COURSE_ID,SEC_ID, SEMESTER, YEAR) REFERENCES
SECTION(COURSE_ID,SEC_ID, SEMESTER, YEAR)
            ON DELETE CASCADE,
        FOREIGN KEY (ID) REFERENCES INSTRUCTOR(ID)
            ON DELETE CASCADE
      );
```

instances of teach table

```
insert into teaches values ('10101', 'CS-101', '1', 'Fall', '2009');
insert into teaches values ('10101', 'CS-315', '1', 'Spring', '2010');
insert into teaches values ('10101', 'CS-347', '1', 'Fall', '2009');
insert into teaches values ('12121', 'FIN-201', '1', 'Spring', '2010');
insert into teaches values ('15151', 'MU-199', '1', 'Spring', '2010');
insert into teaches values ('22222', 'PHY-101', '1', 'Fall', '2009');
insert into teaches values ('32343', 'HIS-351', '1', 'Spring', '2010');
insert into teaches values ('45565', 'CS-101', '1', 'Spring', '2010');
insert into teaches values ('45565', 'CS-319', '1', 'Spring', '2010');
insert into teaches values ('76766', 'BIO-101', '1', 'Summer', '2009');
insert into teaches values ('76766', 'BIO-301', '1', 'Summer', '2010');
insert into teaches values ('83821', 'CS-190', '1', 'Spring', '2009');
insert into teaches values ('83821', 'CS-190', '2', 'Spring', '2009');
insert into teaches values ('83821', 'CS-319', '2', 'Spring', '2010');
insert into teaches values ('98345', 'EE-181', '1', 'Spring', '2009');
```

student table

```
CREATE TABLE STUDENT
(ID                 VARCHAR2(5),
NAME                VARCHAR2(20) NOT NULL,
DEPT_NAME           VARCHAR2(20),
TOT_CRED            NUMERIC2(3,0) CHECK (TOT_CRED >= 0),
PRIMARY KEY (ID),
FOREIGN KEY (DEPT_NAME) REFERENCES DEPARTMENT(DEPT_NAME)
    ON DELETE SET NULL
);
```

instances of student table

```
insert into student values ('00128', 'Zhang', 'Comp. Sci.', '102');
```

```
insert into student values ('12345', 'Shankar', 'Comp. Sci.', '32');
insert into student values ('19991', 'Brandt', 'History', '80');
insert into student values ('23121', 'Chavez', 'Finance', '110');
insert into student values ('44553', 'Peltier', 'Physics', '56');
insert into student values ('45678', 'Levy', 'Physics', '46');
insert into student values ('54321', 'Williams', 'Comp. Sci.', '54');
insert into student values ('55739', 'Sanchez', 'Music', '38');
insert into student values ('70557', 'Snow', 'Physics', '0');
insert into student values ('76543', 'Brown', 'Comp. Sci.', '58');
insert into student values ('76653', 'Aoi', 'Elec. Eng.', '60');
insert into student values ('98765', 'Bourikas', 'Elec. Eng.', '98');
insert into student values ('98988', 'Tanaka', 'Biology', '120');
```

section table

```
CREATE TABLE SECTION
(COURSE_ID          VARCHAR2(8),
SEC_ID              VARCHAR2(8),
SEMESTER            VARCHAR2(6)
        CHECK (SEMESTER IN ('FALL', 'WINTER', 'SPRING', 'SUMMER')),
YEAR                NUMERIC(4,0) CHECK (YEAR > 1701 AND YEAR < 2100),
BUILDING            VARCHAR2(15),
ROOM_NUMBER             VARCHAR2(7),
TIME_SLOT_ID        VARCHAR2(4),
PRIMARY KEY (COURSE_ID, SEC_ID, SEMESTER, YEAR),
FOREIGN KEY (COURSE_ID) REFERENCES COURSE(COURSE_ID)
        ON DELETE CASCADE,
FOREIGN KEY (BUILDING, ROOM_NUMBER) REFERENCES CLASSROOM(BUILDING,
ROOM_NUMBER)
        ON DELETE SET NULL
);
```

instances of course table

```
insert into course values ('BIO-101', 'Intro. to Biology', 'Biology', '4');
insert into course values ('BIO-301', 'Genetics', 'Biology', '4');
insert into course values ('BIO-399', 'Computational Biology', 'Biology', '3');
insert into course values ('CS-101', 'Intro. to Computer Science', 'Comp. Sci.', '4');
insert into course values ('CS-190', 'Game Design', 'Comp. Sci.', '4');
insert into course values ('CS-315', 'Robotics', 'Comp. Sci.', '3');
```

```
insert into course values ('CS-319', 'Image Processing', 'Comp. Sci.', '3');
insert into course values ('CS-347', 'Database System Concepts', 'Comp. Sci.', '3');
insert into course values ('EE-181', 'Intro. to Digital Systems', 'Elec. Eng.', '3');
insert into course values ('FIN-201', 'Investment Banking', 'Finance', '3');
insert into course values ('HIS-351', 'World History', 'History', '3');
insert into course values ('MU-199', 'Music Video Production', 'Music', '3');
insert into course values ('PHY-101', 'Physical Principles', 'Physics', '4');
```

Course table

```
CREATE TABLE COURSE
(COURSE_ID          VARCHAR2(8),
 TITLE              VARCHAR2(50),
 DEPT_NAME          VARCHAR2(20),
 CREDITS            NUMERIC(2,0) CHECK (CREDITS > 0),
 PRIMARY KEY (COURSE_ID),
 FOREIGN KEY (DEPT_NAME) REFERENCES DEPARTMENT(DEPT_NAME)
        ON DELETE SET NULL
);
```

instances of Course table

```
insert into course values ('BIO-101', 'Intro. to Biology', 'Biology', '4');
insert into course values ('BIO-301', 'Genetics', 'Biology', '4');
insert into course values ('BIO-399', 'Computational Biology', 'Biology', '3');
insert into course values ('CS-101', 'Intro. to Computer Science', 'Comp. Sci.', '4');
insert into course values ('CS-190', 'Game Design', 'Comp. Sci.', '4');
insert into course values ('CS-315', 'Robotics', 'Comp. Sci.', '3');
insert into course values ('CS-319', 'Image Processing', 'Comp. Sci.', '3');
insert into course values ('CS-347', 'Database System Concepts', 'Comp. Sci.', '3');
insert into course values ('EE-181', 'Intro. to Digital Systems', 'Elec. Eng.', '3');
insert into course values ('FIN-201', 'Investment Banking', 'Finance', '3');
insert into course values ('HIS-351', 'World History', 'History', '3');
insert into course values ('MU-199', 'Music Video Production', 'Music', '3');
insert into course values ('PHY-101', 'Physical Principles', 'Physics', '4');
```

Aggregate Functions: Count, Sum, Min, Max and Avg

To find avg salary of cse dept.

```
select avg (salary)
from instructor
```

```
where dept name= 'Comp. Sci.';

select avg (salary) as avg_salary
from instructor
where dept name= 'Comp. Sci.';
```

To count instructors who teaches in the year 2010 having Spring semester

```
select count (distinct ID )
from teaches
where semester = 'Spring' and year = 2010;
```

To find the number of courses taught in the university.

```
select count (*)
from course;
```

To find the avaerage salary of each department.

```
select dept name, avg (salary) as avg_salary
from instructor
group by dept name;
```

To find the number of instructors in each department who taught in the year 2010 and semester Spring.

```
select dept name, count (distinct ID ) as instr_count
from instructor natural join teaches
where semester = 'Spring' and year = 2010
group by dept name;
```

To find average salary of each department and also the average salary is greater than $42000

```
select dept name, avg (salary) as avg_salary
from instructor
group by dept_name
having avg (salary) > 42000;
```

For each course section offered in 2009, find the average total credits (tot cred) of all students enrolled in the section, if the section had at least 2 students."

```
select course id, semester, year, sec_id, avg (tot_cred)
from takes natural join student
where year = 2009
group by course id, semester, year, sec_id
having count ( ID ) >= 2;
```

8. Write SQL queries to perform ORACLE BUILT-IN FUNCTIONS (i.e. DATE, TIME).

Built-in Functions

- Character Functions
  - case-conversion functions
  - character manipulation funtions
- Number Functions
- DATE functions

case-conversion functions

```
SELECT LOWER('SQL Course')
FROM DUAL;

SELECT UPPER('SQL Course')
FROM DUAL;

SELECT INITCAP('SQL course')
FROM DUAL;
```

character manipulation functions

```
SELECT CONCAT('HELLO', 'WORLD')
FROM DUAL;

SELECT SUBSTR('HELLO WORLD',1,5)
FROM DUAL;

SELECT LENGTH('HELLO WORLD');
FROM DUAL;

SELECT INSTR('HELLO WORLD', 'WORLD')
FROM DUAL;
```

```
SELECT LPAD(SALARY, 10, '*')
FROM INSTRUCTOR;

SELECT RPAD(SALARY, 10, '*')
FROM INSTRUCTOR;

SELECT REPLACE('JACK and JUE','J','BL')
FROM DUAL;

SELECT TRIM('H' FROM 'HelloWorld'
FROM DUAL;
```

Number Functions

```
SELECT ROUND(45.626,2)
FROM DUAL;
SELECT ROUND(45.626,0)
FROM DUAL;
SELECT ROUND(45.626,-1)
FROM DUAL;
SELECT ROUND(45.626,-2)
FROM DUAL;

SELECT TRUNC(45.626, 2)
FROM DUAL;
SELECT TRUNC(45.626, 0)
FROM DUAL;
SELECT TRUNC(45.626, -1)
FROM DUAL;
SELECT TRUNC(45.626, -2)
FROM DUAL;
```



```
SELECT MOD(1600,300)
FROM DUAL;
```

Date Functions

Oracle database stores date in the internal numeric format of Century, year, month, day, hours, minutes and seconds;

The Default display format is DD-MON-RR

```
SELECT SYSDATE
FROM DUAL;


SELECT MONTHS_BETWEEN(SYSDATE,'15-FEB-20')
FROM DUAL;

SELECT ADD_MONTHS(SYSDATE, 2)
FROM DUAL;

SELECT NEXT_DAY(SYSDATE,'THRUSDAY')
FROM DUAL;

SELECT LAST_DAY(SYSDATE)
FROM DUAL;

SELECT ROUND('25-JUL-03','MONTH')
FROM DUAL;

SELECT ROUND('25-JUL-03','YEAR')
FROM DUAL;
SELECT TRUNC('25-JUL-03','YEAR')
FROM DUAL;

SELECT TRUNC('25-JUL-03','MONTH')
FROM DUAL;
```


9. Write SQL queries to perform KEY CONSTRAINTS (i.e. PRIMARY KEY, FOREIGN KEY, UNIQUE NOT NULL, CHECK, DEFAULT).

Constraints

SQL constraints are used to specify rules for the data in a table.

Types of SQL Constraints.

/*

1. NOT NULL - Ensures that a column cannot have a NULL value

2. UNIQUE - Ensures that all values in a column are different

3. PRIMARY KEY - A combination of a NOT NULL and UNIQUE. Uniquely identifies each row in a table

4. FOREIGN KEY - Uniquely identifies a row/record in another table

5. CHECK - Ensures that all values in a column satisfies a specific condition

6. DEFAULT - Sets a default value for a column when no value is specified

*/

First APPLY CONSTRIANT USING CREATE Statement.

Second, DROP the Constraint

third, Verfify the constraint is working or not using insert statement;

Fourth, Make a note of error occurred in observation

After Verification Drop the table to define other Constraints on the same relation-name

NOT NULL COnstraint Example:

```
CREATE TABLE student (
ID int NOT NULL,
LastName varchar(255) NOT NULL,
FirstName varchar(255) NOT NULL,
Age int
);
ALTER TABLE student
MODIFY Age int NOT NULL;
```

UNIQUE CONSTRAINT Example:

```
CREATE TABLE Students(
ID int NOT NULL,
LastName varchar(255) NOT NULL,
FirstName varchar(255),
Age int,
CONSTRAINT UC_Person UNIQUE (ID,LastName)
);
```

```
ALTER TABLE students
ADD CONSTRAINT UC_Person UNIQUE (ID,LastName);

ALTER TABLE students
DROP CONSTRAINT UC_Person;
```

PRIMARY KEY CONSTRAINT Example:

```
CREATE TABLE Persons (
ID int NOT NULL,
LastName varchar(255) NOT NULL,
FirstName varchar(255),
Age int,
CONSTRAINT PK_Person PRIMARY KEY (ID,LastName)
);

ALTER TABLE Persons
ADD CONSTRAINT PK_Person PRIMARY KEY (ID,LastName);

ALTER TABLE Persons
DROP CONSTRAINT PK_Person;
```

FORIEGN KEY CONSTRAINTS Example:

```
CREATE TABLE Orders (
OrderID int NOT NULL,
OrderNumber int NOT NULL,
PersonID int,
PRIMARY KEY (OrderID),
CONSTRAINT FK_PersonOrder FOREIGN KEY (PersonID)
REFERENCES Persons(PersonID)
);

ALTER TABLE Orders
ADD CONSTRAINT FK_PersonOrder
FOREIGN KEY (PersonID) REFERENCES Persons(PersonID);

ALTER TABLE Orders
```

```
DROP CONSTRAINT FK_PersonOrder;
```

CHECK CONSTRAINTS Example:

```
CREATE TABLE Persons (
ID int NOT NULL,
LastName varchar(255) NOT NULL,
FirstName varchar(255),
Age int,
City varchar(255),
CONSTRAINT CHK_Person CHECK (Age>=18 AND City='Sandnes')
);

ALTER TABLE Persons
ADD CONSTRAINT CHK_PersonAge CHECK (Age>=18 AND City='Sandnes');

ALTER TABLE persons
DROP CONSTRAINT chk_personAge;
```

DEFAULT CONSTRAINTS Example:

```
CREATE TABLE Persons(
ID int NOT NULL,
LastName varchar(255) NOT NULL,
FirstName varchar(255),
Age int,
City varchar(255) DEFAULT 'Sandnes'
);

CREATE TABLE Orders (
ID int NOT NULL,
OrderNumber int NOT NULL,
OrderDate date DEFAULT GETDATE()
);

ALTER TABLE Persons
MODIFY City DEFAULT 'Sandnes';
```

Drop cannot be dropped, So make it null value to remove default

```
ALTER TABLE Persons MODIFY city DEFAULT NULL;
```

10. Write a PL/SQL program for calculating the factorial of a given number.

```
DECLARE
fac NUMBER :=1;
n NUMBER := 10;
BEGIN
WHILE n > 0 LOOP
fac:=n*fac;
n:=n-1;
END LOOP;
DBMS_OUTPUT.PUT_LINE(FAC);
END;
```

11. Write a PL/SQL program for finding the given number is prime number or not.

```
DECLARE
n NUMBER;
i NUMBER;
temp NUMBER;
BEGIN
n := 13;
i := 2;
temp := 1;
FOR i IN 2..n/2
LOOP
IF MOD(n, i) = 0
THEN
temp := 0;
EXIT;
END IF;
END LOOP;
IF temp = 1
```

```
THEN
DBMS_OUTPUT.PUT_LINE(n||' is a prime number');
ELSE
DBMS_OUTPUT.PUT_LINE(n||' is not a prime number');
END IF;
END;
```

12. Write a PL/SQL program for displaying the Fibonacci series up to an integer.

```
DECLARE
FIRST NUMBER := 0;
SECOND NUMBER := 1;
TEMP NUMBER;
N NUMBER := 5;
I NUMBER;
BEGIN
DBMS_OUTPUT.PUT_LINE('SERIES:');
DBMS_OUTPUT.PUT_LINE(FIRST);
DBMS_OUTPUT.PUT_LINE(SECOND);
FOR I IN 2..N
LOOP
TEMP:=FIRST+SECOND;
FIRST := SECOND;
SECOND := TEMP;
DBMS_OUTPUT.PUT_LINE(TEMP);
END LOOP;
END;
```

13. Write PL/SQL program to implement Stored Procedure on table.

PL/SQL Procedure

The PL/SQL stored procedure or simply a procedure is a PL/SQL block which performs one or more specific tasks. It is just like procedures in other programming languages.

The procedure contains a header and a body.

Header: The header contains the name of the procedure and the parameters or variables passed to the procedure.

Body: The body contains a declaration section, execution section and exception section similar to a general PL/SQL block.

How to pass parameters in procedure:

IN parameters: The IN parameter can be referenced by the procedure or function. The value of the parameter cannot be overwritten by the procedure or the function.

OUT parameters: The OUT parameter cannot be referenced by the procedure or function, but the value of the parameter can be overwritten by the procedure or function.

INOUT parameters: The INOUT parameter can be referenced by the procedure or function and the value of the parameter can be overwritten by the procedure or function.

SYNTAX:

```
CREATE [OR REPLACE] PROCEDURE procedure_name
  [ (parameter [,parameter]) ]
(IS | AS)
  [declaration_section]
BEGIN
  executable_section
[EXCEPTION
  exception_section]
END [procedure_name];
```

Example:1

```
CREATE TABLE SAILOR(ID NUMBER(10) PRIMARY KEY,NAME VARCHAR2(100));
    CREATE OR REPLACE PROCEDURE INSERTUSER
(ID IN NUMBER,
NAME IN VARCHAR2)
IS
BEGIN
INSERT INTO SAILOR VALUES(ID,NAME);
DBMS_OUTPUT.PUT_LINE('RECORD INSERTED SUCCESSFULLY');
END;
```

Execution Procedure:

```
DECLARE
CNT NUMBER;
BEGIN
```

```
INSERTUSER(101,'NARASIMHA');
SELECT COUNT(*) INTO CNT FROM SAILOR;
DBMS_OUTPUT.PUT_LINE(CNT||' RECORD IS INSERTED SUCCESSFULLY');
END;
```

DROP PROCEDURE:

```
DROP PROCEDURE insertuser;
```

14. Write PL/SQL program to implement Stored Function on table.

PL/SQL FUnction

The PL/SQL Function is very similar to PL/SQL Procedure. The main difference between procedure and a function is, a function must always return a value, and on the other hand a procedure may or may not return a value. Except this, all the other things of PL/SQL procedure are true for PL/SQL function too.

SYNTAX:

```
CREATE [OR REPLACE] FUNCTION function_name
  [ (parameter [,parameter]) ]
RETURN return_datatype
(IS | AS)
  [declaration_section]
BEGIN
  executable_section
[EXCEPTION
  exception_section]
END [procedure_name];
```

Example:1

```
CREATE OR REPLACE FUNCTION ADDER(N1 IN NUMBER, N2 IN NUMBER)
RETURN NUMBER
IS
N3 NUMBER(8);
BEGIN
N3 :=N1+N2;
RETURN N3;
END;
```

Execution Procedure:

```
DECLARE
  N3 NUMBER(2);
```

```
BEGIN
  N3 := ADDER(11,22);
  DBMS_OUTPUT.PUT_LINE('ADDITION IS: ' || N3);
END;
/
```

DROP FUNCTION

SYNTAX:

```
DROP FUNCTION Func_name;
DROP FUNCTION Adder;
```

Example 2:Recursive Fuction

```
CREATE FUNCTION fact(x number)
RETURN number
IS
  f number;
BEGIN
  IF x=0 THEN
    f := 1;
  ELSE
    f := x * fact(x-1);
  END IF;
RETURN f;
END;
```

Execution Procedure:

```
DECLARE
num number;
factorial number;
BEGIN
  num:= 6;
  factorial := fact(num);
  dbms_output.put_line(' Factorial '|| num || ' is ' || factorial);
END;
```

DROP FUNCTION

```
DROP FUNCTION fact;
```

15. Write PL/SQL program to implement Trigger on table.

PL/SQL Trigger

What is trigger?

Trigger is invoked by Oracle engine automatically whenever a specified event occurs.Trigger is stored into database and invoked repeatedly, when specific condition match. Triggers are stored programs, which are automatically executed or fired when some event occurs. Triggers are written to be executed in response to any of the following events.

A database manipulation (DML) statement (DELETE, INSERT, or UPDATE).
A database definition (DDL) statement (CREATE, ALTER, or DROP).
A database operation (SERVERERROR, LOGON, LOGOFF, STARTUP, or SHUTDOWN).

Advantages of Triggers

These are the following advantages of Triggers:

Trigger generates some derived column values automatically
Enforces referential integrity
Event logging and storing information on table access
Auditing
Synchronous replication of tables
Imposing security authorizations
Preventing invalid transactions

Syntax

```
CREATE [OR REPLACE ] TRIGGER TRIGGER_NAME
{BEFORE | AFTER | INSTEAD OF }
{INSERT [OR] | UPDATE [OR] | DELETE}
[OF COL_NAME]
ON TABLE_NAME
[REFERENCING OLD AS O NEW AS N]
[FOR EACH ROW]
WHEN (CONDITION)
DECLARE
  DECLARATION-STATEMENTS
BEGIN
```

```
  EXECUTABLE-STATEMENTS
EXCEPTION
  EXCEPTION-HANDLING-STATEMENTS
END;
```

CREATE [OR REPLACE] TRIGGER trigger_name:

It creates or replaces an existing trigger with the trigger_name.

{BEFORE | AFTER | INSTEAD OF} :

This specifies when the trigger would be executed. The INSTEAD OF clause is used for creating trigger on a view.

{INSERT [OR] | UPDATE [OR] | DELETE}:

This specifies the DML operation.

[OF col_name]:

This specifies the column name that would be updated.

[ON table_name]:

This specifies the name of the table associated with the trigger.

[REFERENCING OLD AS o NEW AS n]:

This allows you to refer new and old values for various DML statements, like INSERT, UPDATE, and DELETE.

[FOR EACH ROW]:

This specifies a row level trigger, i.e., the trigger would be executed for each row being affected. Otherwise the trigger will execute just once when the SQL statement is executed, which is called a table level trigger.

WHEN (condition):

This provides a condition for rows for which the trigger would fire. This clause is valid only for row level triggers.

PL/SQl Trigger Example:

Instructor Table

```
CREATE TABLE INSTRUCTOR
(ID             VARCHAR2(5),
 NAME           VARCHAR2(20) NOT NULL,
 DEPT_NAME      VARCHAR2(20),
 SALARY                 NUMERIC(8,2) CHECK (SALARY > 29000),
 PRIMARY KEY (ID),
```

```
        FOREIGN KEY (DEPT_NAME) REFERENCES DEPARTMENT(DEPT_NAME)
            ON DELETE SET NULL
    );
```

instances of instructor table

```
    insert into instructor values ('10101', 'Srinivasan', 'Comp. Sci.', '65000');
    insert into instructor values ('12121', 'Wu', 'Finance', '90000');
    insert into instructor values ('15151', 'Mozart', 'Music', '40000');
    insert into instructor values ('22222', 'Einstein', 'Physics', '95000');
    insert into instructor values ('32343', 'El Said', 'History', '60000');
    insert into instructor values ('33456', 'Gold', 'Physics', '87000');
    insert into instructor values ('45565', 'Katz', 'Comp. Sci.', '75000');
    insert into instructor values ('58583', 'Califieri', 'History', '62000');
    insert into instructor values ('76543', 'Singh', 'Finance', '80000');
    insert into instructor values ('76766', 'Crick', 'Biology', '72000');
    insert into instructor values ('83821', 'Brandt', 'Comp. Sci.', '92000');
    insert into instructor values ('98345', 'Kim', 'Elec. Eng.', '80000');
```

department table

```
    CREATE TABLE DEPARTMENT
    (DEPT_NAME   VARCHAR2(20),
     BUILDING              VARCHAR2(15),
     BUDGET                NUMERIC(12,2) CHECK (BUDGET > 0),
     PRIMARY KEY (DEPT_NAME)
    );
```

instances of department table

```
    insert into department values ('Biology', 'Watson', '90000');
    insert into department values ('Comp. Sci.', 'Taylor', '100000');
    insert into department values ('Elec. Eng.', 'Taylor', '85000');
    insert into department values ('Finance', 'Painter', '120000');
    insert into department values ('History', 'Painter', '50000');
    insert into department values ('Music', 'Packard', '80000');
    insert into department values ('Physics', 'Watson', '70000');
```

An example to create Trigger

```
CREATE OR REPLACE TRIGGER display_salary_changes
```

```
BEFORE UPDATE ON instructor
FOR EACH ROW
WHEN (NEW.ID = OLD.ID)
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;
```

A PL/SQL Procedure to execute a trigger

```
DECLARE
  total_rows number(2);
BEGIN
  UPDATE  instructor
  SET salary = salary + 5000;
  IF sql%notfound THEN
    dbms_output.put_line('no instructors updated');
  ELSIF sql%found THEN
    total_rows := sql%rowcount;
    dbms_output.put_line( total_rows || ' instructors updated ');
  END IF;
END;
```

SQL%FOUND, SQL%NOTFOUND, and SQL%ROWCOUNT are PL/SQL attributes that can be used to determine the effect of an SQL statement.

The SQL%FOUND attribute has a Boolean value that returns TRUE if at least one row was affected by an INSERT, UPDATE, or DELETE statement, or if a SELECT INTO statement retrieved one row.

The SQL%NOTFOUND attribute has a Boolean value that returns TRUE if no rows were affected by an INSERT, UPDATE, or DELETE statement, or if a SELECT INTO statement did not retrieve a row.

The SQL%ROWCOUNT attribute has an integer value that represents the number of rows that were affected by an INSERT, UPDATE, or DELETE statement.

DROP the Trigger

Syntax: DROP trigger trigger_name;

DROP trigger;

16. Write PL/SQL program to implement Cursor on table.

PL/SQL Cursor

What is Cursor?

When an SQL statement is processed, Oracle creates a memory area known as context area. A cursor is a pointer to this context area. It contains all information needed for processing the statement. In PL/SQL, the context area is controlled by Cursor. A cursor contains information on a select statement and the rows of data accessed by it.

Types of Cursor

1. implicit cursor

2. Explicit cursor

The implicit cursors are automatically generated by Oracle while an SQL statement is executed, if you don't use an explicit cursor for the statement.

Create customers table and have records:

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	23	Allahabad	20000
2	Suresh	22	Kanpur	22000
3	Mahesh	24	Ghaziabad	24000
4	Chandan	25	Noida	26000
5	Alex	21	Paris	28000
6	Sunita	20	Delhi	30000

Create update procedure

Create procedure:

```
DECLARE
  total_rows number(2);
BEGIN
  UPDATE  customers
  SET salary = salary + 5000;
  IF sql%notfound THEN
```

```
    dbms_output.put_line('no customers updated');
  ELSIF sql%found THEN
    total_rows := sql%rowcount;
    dbms_output.put_line( total_rows || ' customers updated ');
  END IF;
END;
/
```

Output:

6 customers updated

PL/SQL procedure successfully completed.

check it with Select statements

Now, if you check the records in customer table, you will find that the rows are updated.

SELECT * FROM CUSTOMERS;

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	23	Allahabad	25000
2	Suresh	22	Kanpur	27000
3	Mahesh	24	Ghaziabad	29000
4	Chandan	25	Noida	31000
5	Alex	21	Paris	33000
6	Sunita	20	Delhi	35000

Table Creation

```
CREATE TABLE customers(
ID NUMBER PRIMARY KEY,
NAME VARCHAR2(20) NOT NULL,
AGE NUMBER,
ADDRESS VARCHAR2(20),
SALARY NUMERIC(20,2));
```

Instances of Customers:

```
INSERT INTO customers VALUES(1,'Ramesh',23,'Allabad',25000);
INSERT INTO customers VALUES(2, 'Suresh',22,'Kanpur',27000);
```

```
INSERT INTO customers VALUES(3, 'Mahesh',24,'Ghaziabad',29000);
INSERT INTO customers VALUES(4, 'chandhan',25,'Noida',31000);
INSERT INTO customers VALUES(5, 'Alex', 21, 'paris',33000);
INSERT INTO customers VALUES(6, 'Sunita',20,'delhi',35000);
```

PL/SQL Explicit Cursors

The Explicit cursors are defined by the programmers to gain more control over the context area. These cursors should be defined in the declaration section of the PL/SQL block. It is created on a SELECT statement which returns more than one row.

SYNTAX:

```
CURSOR cursor_name IS select_statement;
```

Steps:

You must follow these steps while working with an explicit cursor.

Declare the cursor to initialize in the memory.

Open the cursor to allocate memory.

Fetch the cursor to retrieve data.

Close the cursor to release allocated memory.

1) Declare the cursor:

SYNTAX:

```
CURSOR cursor_name IS select_statement;
```

2) Open the cursor:

It is used to allocate memory for the cursor and make it easy to fetch the rows returned by the SQL statements into it.

SYNTAX:

```
OPEN cursor_name;
```

3) Fetch the cursor:

It is used to access one row at a time. You can fetch rows from the aboveopened cursor as follows:

SYNTAX:

```
FETCH cursor_name INTO variable_list;
```

4) Close the cursor:

It is used to release the allocated memory. The following syntax is used to close the above opened cursors.

SYNTAX:

```
Close cursor_name;
```

PL/SQL Program using Explicit Cursors

```
DECLARE
  c_id customers.id%type;
  c_name customers.name%type;
  c_addr customers.address%type;
  CURSOR c_customers is
    SELECT id, name, address FROM customers;
BEGIN
  OPEN c_customers;
  LOOP
    FETCH c_customers into c_id, c_name, c_addr;
    EXIT WHEN c_customers%notfound;
    dbms_output.put_line(c_id || ' ' || c_name || ' ' || c_addr);
  END LOOP;
  CLOSE c_customers;
END;
/
```

Output:

```
1  Ramesh  Allahabad
2  Suresh  Kanpur
3  Mahesh  Ghaziabad
4  Chandan  Noida
5  Alex  Paris
6  Sunita  Delhi
PL/SQL procedure successfully completed.
```