15CSE302 - Database Management Systems

Lab Manual for Database Management System Practises

Sl. No	DBMS Topics
1	Overview of SQL Basic Data Types Basic Schema Definition Modification of the database • Insertions • Deletion • Updates Altering the Schema Definition • Add a column • Drop a column • Modify the data type of a column
2	Queries on Single Relations Queries on Multiple Relations – Cartesian Product and natural Join
3	Additional Basic Operations • Rename Operations • String Operations • Attribute Specifications in select clause • Where clause predicates
4	Set Operations
6	Aggregate Functions • Aggregate Functions with grouping • The having clause
7	Nested Subqueries • Set membership • Set comparison – some, all • Test for empty relations • Test for absence of duplicate tables
8	Nested Subqueries • Subqueries in from clause • The with clause • Scalar subqueries
9	Join expressions Views • Definition • Updation • Using views in queries Indexing, Sequences
11	PL/SQL block Procedures Functions
12	Cursors Exceptions Trigger

List of Exercises and Evaluation Questions

Exercise I - Basic Schema Definition

Create table department(dept_name varchar(20), building varchar(15),budget numeric(12,2), primary key(dept_name);

```
create table department
    (dept_name      varchar (20),
     building       varchar (15),
     budget         numeric (12,2),
     primary key (dept_name));
```

```
create table course
    (course_id      varchar (7),
     title          varchar (50),
     dept_name      varchar (20),
     credits        numeric (2,0),
     primary key (course_id),
     foreign key (dept_name) references department);

create table instructor
    (ID             varchar (5),
     name           varchar (20) not null,
     dept_name      varchar (20),
     salary         numeric (8,2),
     primary key (ID),
     foreign key (dept_name) references department);
```

```
create table section
    (course_id      varchar (8),
     sec_id         varchar (8),
     semester       varchar (6),
     year           numeric (4,0),
     building       varchar (15),
     room_number    varchar (7),
     time_slot_id   varchar (4),
     primary key (course_id, sec_id, semester, year),
     foreign key (course_id) references course);

create table teaches
    (ID             varchar (5),
     course_id      varchar (8),
     sec_id         varchar (8),
     semester       varchar (6),
     year           numeric (4,0),
     primary key (ID, course_id, sec_id, semester, year),
     foreign key (course_id, sec_id, semester, year) references section,
     foreign key (ID) references instructor);
```

Figure 3.1 SQL data definition for part of the university database.

Exercise II - Insertion of Values.

SQL >insert into instructor values (‘22222’ ,’ Einsteing’ ,’ Physics’ ,95000)

(or)

SQL >insert into instructor values (‘&ID’ ,’ &name’ ,’ &dept_name’ ,&salary)

ID	*name*	*dept_name*	*salary*
22222	Einstein	Physics	95000
12121	Wu	Finance	90000
32343	El Said	History	60000
45565	Katz	Comp. Sci.	75000
98345	Kim	Elec. Eng.	80000
76766	Crick	Biology	72000
10101	Srinivasan	Comp. Sci.	65000
58583	Califieri	History	62000
83821	Brandt	Comp. Sci.	92000
15151	Mozart	Music	40000
33456	Gold	Physics	87000
76543	Singh	Finance	80000

(a) The *instructor* table

dept_name	*building*	*budget*
Comp. Sci.	Taylor	100000
Biology	Watson	90000
Elec. Eng.	Taylor	85000
Music	Packard	80000
Finance	Painter	120000
History	Painter	50000
Physics	Watson	70000

(b) The *department* table

Figure 1.2 A sample relational database.

insert into instructor values (‘22222’ ,’ Einsteing’ ,’ Physics’ ,95000)

insert into department values(‘Comp. Sci.’ ,’ Taylor’ ,100000)
insert into department values(‘&dept_name’ ’&building’ , &budget)

ID	*name*	*dept_name*	*salary*
22222	Einstein	Physics	95000
12121	Wu	Finance	90000
32343	El Said	History	60000
45565	Katz	Comp. Sci.	75000
98345	Kim	Elec. Eng.	80000
76766	Crick	Biology	72000
10101	Srinivasan	Comp. Sci.	65000
58583	Califieri	History	62000
83821	Brandt	Comp. Sci.	92000
15151	Mozart	Music	40000
33456	Gold	Physics	87000
76543	Singh	Finance	80000

(a) The *instructor* table

dept_name	*building*	*budget*
Comp. Sci.	Taylor	100000
Biology	Watson	90000
Elec. Eng.	Taylor	85000
Music	Packard	80000
Finance	Painter	120000
History	Painter	50000
Physics	Watson	70000

(b) The *department* table

Figure 1.2 A sample relational database.

course_id	*title*	*dept_name*	*credits*
BIO-101	Intro. to Biology	Biology	4
BIO-301	Genetics	Biology	4
BIO-399	Computational Biology	Biology	3
CS-101	Intro. to Computer Science	Comp. Sci.	4
CS-190	Game Design	Comp. Sci.	4
CS-315	Robotics	Comp. Sci.	3
CS-319	Image Processing	Comp. Sci.	3
CS-347	Database System Concepts	Comp. Sci.	3
EE-181	Intro. to Digital Systems	Elec. Eng.	3
FIN-201	Investment Banking	Finance	3
HIS-351	World History	History	3
MU-199	Music Video Production	Music	3
PHY-101	Physical Principles	Physics	4

Figure 2.2 The *course* relation.

course_id	prereq_id
BIO-301	BIO-101
BIO-399	BIO-101
CS-190	CS-101
CS-315	CS-101
CS-319	CS-101
CS-347	CS-101
EE-181	PHY-101

Figure 2.3 The *prereq* relation.

ID	course_id	sec_id	semester	year
10101	CS-101	1	Fall	2009
10101	CS-315	1	Spring	2010
10101	CS-347	1	Fall	2009
12121	FIN-201	1	Spring	2010
15151	MU-199	1	Spring	2010
22222	PHY-101	1	Fall	2009
32343	HIS-351	1	Spring	2010
45565	CS-101	1	Spring	2010
45565	CS-319	1	Spring	2010
76766	BIO-101	1	Summer	2009
76766	BIO-301	1	Summer	2010
83821	CS-190	1	Spring	2009
83821	CS-190	2	Spring	2009
83821	CS-319	2	Spring	2010
98345	EE-181	1	Spring	2009

Figure 2.7 The *teaches* relation.

course_id	sec_id	semester	year	building	room_number	time_slot_id
BIO-101	1	Summer	2009	Painter	514	B
BIO-301	1	Summer	2010	Painter	514	A
CS-101	1	Fall	2009	Packard	101	H
CS-101	1	Spring	2010	Packard	101	F
CS-190	1	Spring	2009	Taylor	3128	E
CS-190	2	Spring	2009	Taylor	3128	A
CS-315	1	Spring	2010	Watson	120	D
CS-319	1	Spring	2010	Watson	100	B
CS-319	2	Spring	2010	Taylor	3128	C
CS-347	1	Fall	2009	Taylor	3128	A
EE-181	1	Spring	2009	Taylor	3128	C
FIN-201	1	Spring	2010	Packard	101	B
HIS-351	1	Spring	2010	Painter	514	C
MU-199	1	Spring	2010	Packard	101	D
PHY-101	1	Fall	2009	Watson	100	A

Figure 2.6 The *section* relation.

Exercise III - Modification of the database

Deletion:

```
SQL >delete from instructor where dept_name =  'Finance' ;
SQL >delete from instructor where salary between 1300 and 2000;
```

Updates:

```
SQL >update instructor set salary = salary + 1000 where salary<7000;
SQL >update instructor set salary=1000 where name =' Raj' ;
```

Exercise IV　　- Altering the Schema Definition

```
SQL> create table AAA(a varchar(10));
SQL> desc AAA;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 A                                                  VARCHAR2(10)
```

To add a column in a table:

```
SQL> alter table AAA add b varchar(10);

SQL> desc AAA;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 A                                                  VARCHAR2(10)
 B                                                  VARCHAR2(10)
SQL> alter table AAA add c numeric(10,3);

Table altered.

SQL> desc AAA;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 A                                                  VARCHAR2(10)
 B                                                  VARCHAR2(10)
 C                                                  NUMBER(10,3)
```

To delete a column in a table

```
SQL> alter table AAA drop column b;

Table altered.

SQL> desc AAA;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 A                                                  VARCHAR2(10)
 C                                                  NUMBER(10,3)
```

To change the data type of a column in a table,

```
SQL>alter table AAA modify c varchar(10);

SQL> desc aaa;
 Name                                      Null?    Type
 ----------------------------------------- -------- ------------------------
 A                                                  VARCHAR2(10)
 C                                                  VARCHAR2(10)
```

Exercise V - Queries

Queries on Single Relations

1) Find the name of all the instructors

   SQL >select name from instructors;

2) Find the department name of all the instructors

   SQL >select dept_name from instructors;

3) Select clause with arithmetic expression

   SQL >select name,salary*1.1 from instructors;

4) Where clause with predicates

   SQL >select name from instructor where dept_name=' Comp.Sci.' and salary>7000;

Queries on Multiple Relations

1) Retrieve the names of all instructors along with their department names and department building name.

   SQL >select name, instructors.dept_name,building from instructors, department where instructors.dept_name=department.dept_name;

2) Understanding Cartesian product.

   Create the following tables and insert the records as given

Table 1

ID	Name
a	abc
b	xyz
c	def
e	ghi

Table 2

ID	Course_Id
a	c1
b	c2
c	c3

SQL >select Table1.id, Name, Table2.id, Course_id from Table1, Table2.

SQL >select Table1.id, Name, Table2.id, Course_id from Table1, Table2 where Table1.ID=Table2.ID

SQL >select Name, Course_id from Table1, Table2 where Table1.ID=Table2.ID

3) Using Cartesian Product

1) To find the list of instructors in Comp. Sci dept with their course

SQL >select name, course_id from instructor, teaches where instructor.id=teaches.id and instructor.dept_name=' Comp. Sci.' ;

4) Understanding Natural Joins.

SQL >select * from Table1 natural join Table2;

5) Using Natural Joins.

SQL >select name, instructor from instructor natural join teaches;

6) Using Cartesian product and Natural Joins.

To list the names of instructors along with the titles of the course that they teach.

Q1 SQL >select name, title from instructor natural join teaches, course where teaches.course_id=course.course_id.

Q2 SQL >select name, title from instructor natural join teaches natural join course;

Compare the result of Q1 and Q2 and understand the difference.

Q3 SQL > select name, title from (instructor natural join teaches) join course using (course_id);

Exercise VI - Additional Basic Operations

(a) Rename operations

1. "Select the name of all the instructors"

   ```
   SQL >Select name as instructor_name from instructor;
   ```

2. "For all the instructor in the university who have taught some course find their names and the course_id of all the course they taught"

   ```
   SQL >Select T.name, S.course_id from instructor T, teaches S where T.id = S.id;
   ```

3. "Find the names of all the instructor whose salary is greater than at least one instructor in the biology department"

   (to compare the tuples in the same relations)

   ```
   SQL >Select distinct T.name from instructor S, instructor T where T.salary > S.salary and S.dept_name='Biology' ;
   ```

(b) String operations

1. concatenation - ||

   ```
   SQL >select 'hai'||'welcome' from dual;
   SQL >select '--'||dept_name||'--'||building||'--' from department;
   ```

2. extracting substring - substr(coumn_name,position,length)

   ```
   SQL >select substr(dept_name,3,4) from department;
   ```

3. finding length - length(coumn_name)

   ```
   SQL >select length(dept_name) from department;
   ```

4. uppercase to lowercase - lower(coumn_name)

   ```
   SQL >select lower(dept_name) from department;
   ```

5. lowercase to uppercase - upper(coumn_name)

   ```
   SQL >select upper(dept_name) from department;
   ```

6. removing space at end - trim(coumn_name)

SQL > select trim(dept_name) from department;

7. pattern matching - like operator

Patterns

% - matches any substring
_ - matches any character

Examples
(1) 'Intro%' - matches any string beginning with 'Intro'
(2) '%Comp%' - matches any string that has Comp as substring
(3) '___' - matches any string with exactly three characters
(4) '___%' - matches any string with at least three characters
Additional Examples (using escape character)
(5) 'ab¥%cd%' - matches strings begin with *ab%cd*
(6) 'ab¥¥cd%' - matches strings begin with *ab¥cd*

Example Queries

1. "Find the names of all departments whose building name includes the substring 'Watson';
SQL >select dept_name from department where building like '%watson%';

2. SQL >select name from instructor where name like 'K%';

3. SQL >select name from instructor where name like 'K_';

(c) Attribute Specifications in select clause

SQL > select * from instructor;
SQL > select instructor.* from instructor,teaches where instructor.id =teaches.id;

(d) Where clause predicates

1. SQL >Select name from instructor where dept_name =' physics';

2. SQL >select dept_name from department where building=' Taylor' or building=' Watson';

3. SQL >select name from instructor where salary <=10000 and salary >=20000;

4. SQL >select name, course_id from instructor, teaches where instructor.id=salry.id and dept_name=' Biology';
(or)

SQL >select name,course_id from instructor, teaches where (instructor.id, dept_name) =(teaches.id,' Biology');

5. SQL >select name from instructor where salary between 10000 and 20000;

6. SQL >select name from instructor where salary in (60000,80000,40000);

7. SQL >select name from instructor where dept_name is null;

8. SQL >select name from instructor where dept_name is not null;

Exercise VII - Set Operations

Three operations

- Union
- Intersection
- Minus

A Simple Example

```
>create table AAA(a varchar(10));

>create table BBB(a varchar(10));

>insert into AAA values ( ‘&a’ );
a
b
c
d
e
e

>insert into BBB values ( ‘&a’ );
a
b
f
a

>(select * from AAA) union(select * from BBB);
```

Result:

```
a
b
c
d
e
f

>(select * from AAA) union all(select * from BBB);
```

Result:

```
a
b
c
d
e
e
a
b
f
```

a

>(select * from AAA) intersect (select * from BBB);

Result:

a
b

(select * from AAA) minus(select * from BBB);

Result:

c
d
e

Example for set operations based on the university database

(a) Union

Example

To find Set of all courses taught in Fall 2009 or Spring 2010 semesters.

```
(select course_id from section where semester=' Fall'  and
Year=2009)union(select course_id from section where semester='
Spring'  and year=2010);
```

- The union operation eliminate duplicates, to retain all the duplicates we must use *union all*.

```
(select course_id from section where semester=' Fall'  and
Year=2009)union all(select course_id from section where semester='
Spring'  and year=2010);
```

(b) Intersection

Example

To find Set of all courses taught in Fall 2009 and Spring 2010 semesters.

```
(select course_id from section where semester='Fall' and
Year=2009)intersect(select course_id from section where
semester='Spring' and year=2010);
```

(c) Minus

Example

To find Set of all courses taught in Fall 2009 but not in Spring 2010 semesters.

```
(select course_id from section where semester=' Fall'  and
Year=2009)minus (select course_id from section where semester='
Spring'  and year=2010);
```

Exercise VIII - Aggregate Functions

These functions take a collection of values as input and return a single value.
(a) avg (b) min (c) max (d) sum (e) count

(a) Basic aggregation

a. SQL>select avg(salary) from instructor;

b. SQL>select avg(salary) as avg_salary from instructor;

c. SQL>select count(distinct ID) from teaches where semester=' spring' and year = 2010;

d. SQL>select count(*) from course

e. SQL>select min(salary) from instructor;

f. SQL>select max(salary) from instructor;

g. SQL>select sum(salary) from instructor;

(b) Aggregation with Grouping

a. "To find the average salary of each department;

SQL>select dept_name, avg(salary)group_by dept_name;

b. "To find the number of instructors in each department who teach a course in the Spring 2010 semester"

SQL> select dept_name, count(distinct ID) from instructor natural join teaches where semester=' Spring' and year =2010 group by dept_name;

Note : When using grouping the attribute that appear in the select clause outside the aggregate function should present in the group by clause.

Example: (understand the error in the following query)
SQL>select dept_name, ID, avg(salary) from instructor group by dept_name;

(c) The having clause

a. To find the average instructors salary of the department with the average greater than 40,000.

SQL> select dept_name, avg(salary) from instructor group by dept_name having avg(salary)>40000;

Note: as in select clause any attribute that appear in the having clause outside the aggregate function should present in the group by clause.

Exercise IX _ Nested Subqueries - Part I

(a) Set membership

- 'in' connective test
- 'not in' connective test

"Find all the courses taught in both Fall 2009 and Spring 2010 Semesters"

SQL > 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);

"Find the courses taught in Fall 2009 but not in Spring 2010 semester"

SQL > 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);

SQL> select count(ID) from takes where course_id in (select course_id from section where semester=' Fall' and year=2009)

(b) Set comparison - some, all

1. (Recall the query)
   "Find the names of all the instructor whose salary is greater than at least one instructor in the biology department"

   SQL >Select distinct T.name from instructor S, instructor T where T.salary > S.salary and S.dept_name='Biology' ;

   This can be written as below

   SQL> select name from instructor where salary > some(select salary from instructor where dept_name=' Biology' ;

   Also can use : < some, <=some, >=some, =some, <> some
   =some is similar to 'in'
   <> some is similar to 'not in'

2. "Find the names of all the instructors who have a salary value greater than that of each instructor in biology department"

   SQL> select name from instructor where salary > all (select salary from instructor where dept_name=' Biology' ;

3. Find the department that has the highest average salary

```
SQL > select dept_name
      from instructor
      group by dept_name
      having avg(salary)>=all
      (select avg(salary)
      from instructor
      group by dept_name);
```

(c) Test for empty relations

To test whether a subquery has any tuple in the results.

1. Find all courses taught in both Fall 2009 and Spring 2010

```
SQL > 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)
```

The *exists* construct return true if the result of the subquery is not empty. The *not exists* construct also available.

(d) Test for absence of duplicate tables

The unique construct return true in the subquery contains no duplicate record

1. "Find all the course that were offered at most once in 2009"

```
SQL> select T.course_id
     from course as T
     where unique (select R.course_id
     from section as R
     where T.course_id=R.course_id and R.year=2009)
```

for a course not offered in 2009 the subquery return empty result, and the unique construct will return true for the empty result.

The below query is equivalent to the above query.

```
SQL> select T.course_id
      from course as T
      where 1 < = ( select count(R.course_id)
      from section as R
      where T.course_id = R.course_id and R.year=2009);
```

2. “Find all courses that were offered at least twice in 2009”

```
SQL> select T.course_id
   from course as T
   where not unique
   (select R.course_id
   from section as R
   where T.course_id = R.course_id
   and R.year=2009);
```

Exercise X - Nested Subqueries - Part II

(a) Subqueries in from clause

1. Find the average instructors salaries of those department where the average salary is greater than 42000

```
SQL> select dept_name, avg_salary
      from (select dept_name, avg(salary) as avg_salary
      from instructor group by dept_name)
      where avg_salary>42000;
```

Note: the attribute in the subquery can be used in the outer query Eg. avg_salary

2. Find the maximum across all departments of the total salary at each department.

```
SQL > select max(tot_salary)
      from (select dept_name, sum(salary) as tot_salary from
instructor group by dept_name);
```

(b) The with clause

1. To find the department with maximum budget

```
SQL> with max_budget(value) as
     (select max(budget) from department)
     select budget from department, max_budget where
     deparment.budget=max_budget.value;
```

2. To find all department where total salary is greater than the average of the total salary of all the departments

```
SQL> with dept_total(dept_name, value) as (select dept_name,
sum(salary) from instructor group by dept_name),
dept_total_avg(value) as
(select avg(value) from dept_total)
 select dept_name
 from dept_total, dept_total_avg
 where dept_total.value > = dept_total_avg.value;
```

(c) Scalar subqueries

1. To list all departments with the number of instructors in each deparments.

```
SQL > select dept_name, (select count(*) from insructor where
department.dept_name=instructor.dept_name) as num_instructors from
department;
```

Exercise XI - Join Expressions

(a) Inner Join

SQL > select * from instructor natural join teaches

But, if an instructor has offered no course his details would not be displayed in the result. Thus, some tuples in either both of the relations being joined may be lost.

(b) Outer Joins

The outer join preserve those tuples that would be lost in a join, by creating tuples in the result containing null values.

Simple example

Table1

Name	Age
A	20
B	30
C	25

Table2

Name	Salary
A	2000
B	3000
D	2500

SQL > select * from Table1 natural join Table2

Result

Name	Age	Salary
A	20	2000
B	30	3000

a. Left outer join

SQL> select * from Table1 natural left outer join Table2

Result

Name	Age	Salary
A	20	2000
B	30	3000
C	25	null

b. Right outer join

SQL> select * from Table1 natural right outer join Table2

Result

Name	Age	Salary
A	20	2000
B	30	3000
D	Null	2500

c. Full outer join

SQL> select * from Table1 natural full outer join Table2

Result

Name	Age	Salary
A	20	2000
B	30	3000
C	25	Null
D	Null	2500

Example from University Database

1) "Find the names of the instructors who have not offered any course"

SQL> select name from instructor natural left outer join teaches where course_id is null.

Exercise XII - Creating and Using Views

It is often needed to hide certain part of a database from certain user. We can use view for that purpose.

(a) Creating Views
(b) Inserting through views
(c) Using views.

(a) Creating views

Examples

1. 'To create a view named faculty to hide the instructor detail of the instructor'

   SQL>Create view faculty as select id, name,dept_name from instructor

2. 'To create a view lists all the courses offered by physics department in the Fall 2009 semester'

   SQL> 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=209.

3. 'The attribute name of the view can be specified explicitly'
   Create view dept_tot_sal(dept_name,tot_salary) as select dept_name, sum(salary) from instructor group by dept_name;

(b) Inserting through views

1. SQL> insert into faculty values('12121' ,' Ram' ,' Music');
   For salary null value would be inserted.

(c) Using views

1. SQL > select * from faculty;

2. SQL > select course_id from physics_fall_2009 where building =' Watson' ;

3. 'Can use existing views to create another view'
   SQL > create view physics_watson as select course_id from physics_fall_2009 where building =' Watson'

Exercise XIII - Indexing and Sequencing

1. Indexing

An indexing is an ordered list of contents of a column or group of columns in a table.

a. Creating

i. Simple Index

SQL>create index indexfile_name on table_name(column_name)

ii. Composite index

SQL>create index indexfile_name on table_name(column_name1, column_name2)

Unique index

SQL> create unique index indexfile_name on table_name(column_name)

SQL>create unique index indexfile_name on table_name(column_name1, column_name2)

b. Dropping

SQL>drop index indexfile_name

2. Sequence

Most applications require the automatic generation of a numeric value. Oracle provides an automatic sequence generator of numeric values.

a. Creating

To create a sequence order_seq which will start generating numbers from 1 to 9999 in ascending order with an interval of 1.

```
SQL>create sequence order_seq
increment by 1
start with 1
maxvalue 9999
cycle;
```

b. Referencing a sequence

This can be done by using select statement

To refer to the next value

SQL>select order_seq.nextval from dual

To refer to the current value

SQL>select order_seq.currval from dual

c. Using a sequence

Insert values in the sales_order table, the s_order_no must be generated by using the order_seq sequence

SQL>insert into sales_order(s_order_no,s_order_date,client_no) values(order_seq.nextval,sysdate,' c0001');

d. Altering a sequence

SQL>alter sequence order_seq increment by 2

e. Dropping

SQL>drop sequence order_seq

Exercise XIV - PL SQL block

While the SQL is the natural language of the DBA, it does not have any procedural capabilities such as looping and branching. For all this, oracle provides PL/SQL, it adds power to SQL and provides the user with all the facilities of a programming environment. It bridges the gap between database technology and procedural programming languages.

Execute the following command first

```
SQL>set serveroutput on;
```

Example 1: (Simple Example)

```
SQL> DECLARE
A varchar2(20);
BEGIN
   select dept_name into A from department where budget = 80000;
   dbms_output.put_line(A);
END;
```

Example 2: (To use if...then...else...endif)

```
DECLARE
B number(12,2);
BEGIN
   select budget into B from department where dept_name =
'Music';
   if B > 5000 then
        dbms_output.put_line('Good');
   else
       dbms_output.put_line('bad');
   end if;
END;
```

Example 3: (To use while loop)

```
Declare
name varchar2(20);
counter number(2):=5;
BEGIN
  select dept_name into name from department where budget=80000;
  while counter>0
  loop
```

```
            dbms_output.put_line(name);
            counter:=counter-1;
        end loop;
      END;
```

Example 4: (To use while loop)

```
      /* counter variable need not be declared
Declare
name varchar2(20);
BEGIN
  select dept_name into name from department where budget=80000;
  for counter in 1..5
  loop
       dbms_output.put_line(counter||'.'||name);
   end loop;
END;


We can also use for counter in reverse 1..5
```

Example 5: (To use goto statement)

```
DECLARE
B number(12,2);
BEGIN
   select budget into B from department where dept_name = 'Music';
   if B > 79000 then
        goto good;
   else
      goto bad;
   end if;
<<good>>
      dbms_output.put_line('Good');
<<bad>>
      dbms_output.put_line('Bad');

END;
```

Exercise XV - Procedures and Functions

1. Procedures

Procedures are named PL/SQL blocks that can take parameters, perform an action and can be invoked.

a. Creating

```
SQL> create or replace procedure s1 as
temp varchar2(10);
begin
select name into temp from instructor where id ='10101';
dbms_output.put_line(temp);
end;
```

To call the procedure, use the following command

```
SQL>exec s1
```

To see the errors use

```
SQL>show errors procedure s1
```

b. Dropping

```
SQL>drop procedure s1
```

2. Functions

Functions are named PL/SQL blocks that can take parameters, perform an action, can be invoked and return a value to the host environment. A function can return only one value.

a. Creating

```
SQL>CREATE FUNCTION f_itemcheck(itemno IN number) RETURN number IS
dummyitem number(4)
BEGIN
      Select itemid into dummyitem from item_master where
itemid=itemno;
            .........
            return  1
```

```
            --------
            return 0
END;
```

The PL/SQL block to call the function

```
DECLARE
  -----
BEGIN
      ----
        Val=f_itemcheck;
      if val = 0 then
       -----
      elseif val = 1
      ------
      end if
END
```

Example for Function

```
create or replace function avg_sal(n string) return number is
res number(5);
begin
 select avg(salary) into res from instructor where dept_name =
n;
return(res);
end;
```

Function Call

```
select id, name from instructor where salary >
avg_sal('Physics');
```

b. Dropping

```
drop function f_itemchecm
```

Exercise XVI - Cursors

When a query is executed by oracle, it uses a work area for the internal processing related to that query. This work area is private to the SQL's operations and is called ***cursor***. The data that is available in the cursor is called ***active data set***. Oracle has a pre-defined area in main memory with in which it opens the cursors.

When a query like ***'select emp_no, salary from emp'*** returns multiple rows, in addition to the data held in the cursor, Oracle also maintain a row pointer. Depending on the user requests to view the data the row pointer will be relocated within the cursor's active data set. Additionally Oracle also maintains cursor variables loaded with the value of the total number of rows fetched from the active data set.

In PL/SQL block, if the records created by a query are to be evaluated and processed once at a time, then the only method available is by using Explicit cursor.

Explicit Cursor

A cursor declared by the user is called explicit cursor. For queries that return more than one row, you must declare a cursor explicitly. We can use it to process the rows individually.

The steps involved are

a. Declare a cursor
b. Open a cursor
c. Fetch one row at a time
d. Close the cursor

Example 1 (Simple example):

Assume there are two tables AAA(A varchar2(20)); and BBB(B varchar2(20)); and AAA has the records a, b, c, d and e. You want to read the values of the record and store it in the table BBB. It can be done with cursor as follows.

```
declare
cursor c1 is select A from AAA;
dum varchar(10);
begin
      open c1;
      loop
         fetch c1 into dum;
         exit when c1%notfound;
```

```
                Insert into BBB values(dum);
            end loop;
            commit;
            close c1;
        end;
```

Example 2 (Simple example):

```
/* To read and display the names of the instructor using cursor */

    DECLARE
        cursor c2 is select name from instructor;
        str_name instructor.name%type;
    BEGIN
        open c2;
        loop
            fetch c2 into str_name;
            exit when c2%notfound;
            dbms_output.put_line(str_name);
        end loop;
        commit;
        close c2;
    END;
```

Example 3 (From university database):

/* To increase the salary of the instructors of the Music department and store the details in instructor_raise table */

```
SQL>create table instructor_raise(id varchar2(10),date_raise
date,salary_raise numeric(12,2));

DECLARE
        cursor c3 is select id, salary from instructor where
dept_name='Music';
        str_id instructor.id%type;
        str_salary instructor.salary%type;
    BEGIN
        open c3;
        loop
```

```
            fetch c3 into str_id, str_salary;
            exit when c3%notfound;
            update instructor set salary=str_salary+(str_salary*0.5)
                  where id = str_id;
            insert into instructor_raise values
                  (str_id,sysdate,str_salary*0.05);
      end loop;
      commit;
      close c3;
END;
```

Example 4:

Consider: employee(emp_code, ename, deptno, job, salary) and emp_raise(emp_code, raise_date, raise_amt)

The HR manager has decided to raise the salary for all the employee in department no 20 by 0.05. Whenever any such raise is given to the employee the date when the raise was given and the amount is maintained in the emp_raise table. Write a PL/QL block to update the salary of the employee and insert a record in the emp_raise table.

```
      DECLARE
            cursor c_emp is select emp_code, salary from employee
where deptno=20
            str_emp_code employee.emp_code%type;
            num_salary employee.salary%type;
      BEGIN
            open c_emp;
            loop
                  fetch c_emp into str_emp_code, num_salary;
                  update employee set salary = num_salary +
(num_salary*0.5)
                        where emp_code=str_emp_code;
                  insert into emp_raise values
                        (str_emp_code, sysdate, num_salary*0.05)
            end loop;
            commit;
            close c_emp;
      END;
```

Exercise XVII - Exceptions and Trigger

1. Exception

When a SQL statement is executed, if it result into an error condition, Oracle returns an error number and message. PL/SQL can deal with these errors. They have number of error conditions, called as internally-defined exceptions. We can also program from user-defined exceptions.

User Defined exception

Example 1 (Simple Example)

```
/* To raise an exception when a student has more than ten arrears, else add
his name for scholarship */

SQL>create table studen_info(rollno varchar(10),name
varchar(10),no_of_arrears numeric(2,0));

SQL>insert into studen_info values('&rollno','&name',&no_of_arrears);

DECLARE
      more_arrear exception;
      arrear_count studen_info.no_of_arrears%type;
      str_rollno studen_info.rollno%type;
      BEGIN
            select rollno,no_of_arrears into str_rollno,arrear_count from
studen_info where rollno='&rollno';
            if arrear_count>10 then
                  raise more_arrear;
            else
                  insert into scholarship values (rollno, name,
arrear_count);
            end if;
      EXCEPTION
            when more_arrear then
                  dbms_output.put_line('Student :'||str_rollno||' has got
more than ten arrears');
END;
```

Example 2 (from university database)

```
/* To add commission to the instructors based on the number of subjects
they have offered till now */

SQL> create table inst_commission(id varchar(10), commission numeric(12,2));

DECLARE
      no_subject exception;
      subject_count number(2);
      str_id instructor.id%type;
      BEGIN
           select count(*) into subject_count from teaches where id=&str_id;
            if subject_count=0 then
                  raise no_subject;
            else
                  insert into inst_commission values
(str_id,subject_count*1000);
            end if;
      EXCEPTION
            when no_subject then
                  dbms_output.put_line('The instructor with id '||str_id||'
has not offered any course');
END;
```

Example 3

```
DECLARE
      less_than_target exception
      s_no salesman_master.salesman_no%type;
      s_com salesman_master.comm%type;
      s_target salesman_master.target_sales%type;
      s_actual salesman_master.actual_sales%type
      BEGIN
            select saleman_no, comm, target_sales, actual_sales
            into s_no, s_com, s_target, s_actual from salesman_master
            where salesman_no=&s_no;
            if s_actual<s_target
                  raise less_than_targer;
            else
                  insert into commission_payable values (s_no,
s_actual*s_com/100);
            end if;
      EXCEPTION
            when less_than_target then
                  dbms_output.put_lines( 'Salesman No' ||s_no||' is not
```

```
entitles to get commission' );
END;
```

Internal exceptions

DUP_VAL_ON_INDEX, LOGIN_DENIES, NO_DATA_FOUNT, NOT_LOGGED_ON, PROGRAM_ERROR, TIMEOUT_ON_RESOURCE, TOO_MANY_ROYS and VALUE_ERROR.

3. Trigger

Triggers are the procedures that are stored in the database and are implicitly executed when the contents of a table are changed. They can not be called by the user explicitlty.

Types of triggers

Row triggers
Statement Trigger
Before trigger
After Trigger

Syntax

```
CREATE OR REPLACE TRIGGER [schema.]triggername
      {BEFORE, AFTER}
      {DELETE, INSERT, UPDATE [ OF column,…]}
      ON [schema.]tablename
      [REFERENCING {OLD AS old, NEW AS new}]
      [FOR EACH ROW [WHEN condition]]
DECLARE
      Variable declarations;
      Constant declaration
BEGIN
      PL/SQL body
EXCEPTION
      Exception PL/SQL block
END;
```

Example 1 (based on university database)

/* To create a trigger to store the average salary in inst_avg table, after each update on the instructor relation */

```
SQL> create table inst_avg(avg_sal numeric(12,2));

            create or replace trigger sal_avg after update on instructor
            declare
            s number(5);
            begin
            select avg(salary) into s from instructor;
            insert into inst_avg values(s);
            end;
```

Example 2 (based on university database)

```
            create or replace trigger sal_update before insert on employee
            declare
            s number(5);
            begin
            select avg(salary) into s from employee;
            insert into t values(s);
            end;
```