Institute of Distance and Open Learning

(IDOL)

**University of Mumbai**



Certificate

This is to certify that Mr. Pralhad Shantaram pawar of **S.Y.MCA**  Semester 3 has completed the specified term work in the subject of Database Management System in satisfactorily manner within this institute as laid down by University of Mumbai during the academic year 2018 to 2019.

Subject In charge MCA Co-ordinator

(Prof. Mahendra Patil) (prof. Mahendra Patil)

Examiner

Date: College Seal

Institute of Distance and Open Learning (IDOL)

**University of Mumbai**



INDEX

Subject: Database Management System

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Sr. No.** | **Experiment Name** | **Pg. No.** | **Date** | **Grade** | **Sign.** | **Remarks** |
| **1** | SQL Program to create a table and perform DDL and DML operations. | 4-8 |  |  |  |  |
| **2** | SQL program to study and execute the aggregate functions. | 9-10 |  |  |  |  |
| **3** | SQL program to study constraints in SQL. | 11-14 |  |  |  |  |
| **4** | SQL program to study joins in SQL. | 15-18 |  |  |  |  |
| **5** | SQL program to study sub-query in SQL. | 19-20 |  |  |  |  |
| **6** | SQL program to study views in SQL. | 21-22 |  |  |  |  |
| **7** | SQL program to study triggers in SQL. | 23-25 |  |  |  |  |
| **8** | PL/SQL program to study procedure and function in PL/SQL. | 26-28 |  |  |  |  |
| **9** | Program to study loops/iterative statements in PL/SQL. | 29-31 |  |  |  |  |

Date: College Seal

**EXPERIMENT NO. 1**

**SQL Program to create a table and perform DDL and DML operations.**

**Problem Description:**

**Structured Query Language (SQL):** This language allows end users to extract information themselves, without using a system group for every little report.

Oracle is able to answer questions as like humans. The keywords used in a query to oracle are Select, From, Where and Order by.

**Tools of Oracle:** The tools provided by Oracle are so user friendly that a person with minimum skill in the field of computers can access them with ease. The main tools are:

1. SQL \* PLUS
2. PL/SQL
3. **SQL \* PLUS:** It is a Structured Query Language supported by Oracle. Through SQL \* PLUS we can store, retrieve, edit, enter and run the SQL commands and PL/SQL blocks. Using SQL \* PLUS we can perform calculations, list column definitions for any table and can also format query results in the form of a report.
4. **PL/SQL:** It is an extension of SQL. PL/SQL block can contain any number of SQL statements integrated with flow of control statements. Thus PL/SQL combines the data manipulating power of SQL with data processing power of procedural languages.

**Benefits of SQL:**

* Non-procedural language, because more than one record can be accessed rather than one record at a time.
* It is the common language for all relational databases. In other words it is portable and it requires very few modifications so that it can work on other databases.
* Very simple commands for querying, inserting, deleting and modifying data and objects.

In order to communicate with the database, SQL supports following commands:

**A) Data Definition Language (DDL):**

* Create
* Alter
* Drop
* Truncate

**B) Data Manipulation Language (DML):**

* Insert
* Update
* Delete
* Select

**C) Transaction Control Language (TCL):**

* Commit
* Rollback

**D) Data Control Language (DCL):**

* Grant
* Revoke

**Data types Available in SQL:**

1. Char
2. Varchar2
3. Date
4. Number
5. Float

* **Char and Varchar2** both are used for **character type of data**. By default the size of the char data type is 1. **Difference between char and varchar2 is:**
  1. **Maximum** length for **char** data type is **2000 characters** whereas for **varchar2**, it is **4000 characters.**
  2. store the date in any other format. In, Oracle along with every date value there **Char** is a **fixed length data type** whereas **varchar2** is a **variable length data type.**

For ex: Generally, for denoting the gender of a person we use char datatype because it is fixed.

* **Date data type:** It is used to store date type of data or values in a column. Date data type allows you to store the date as well as time.

There are only two formats of date, allowable in Oracle as follows:

1) dd-mon-yy for ex. 12-dec-98

2) dd-mon-yyyy for ex. 12-dec-1998 12:00:00 (Default time)

User can’t will be time, we can’t separate date and time.

* **Number data type:**  It is used to store numerical values. **Maximum length** of number data type is **38 digits.** Here, we can’t store the numbers with decimal places. For ex: sal number (5)

For storing the number with decimal places we have to change the definition of number data type.

For ex: sal number (5,2) i.e. Total number digits are 5 and 2 are for decimal places. (For ex: 132.72)

Note: sal number (2,2) then no integer values are allowed here. (for ex: .98)

Remember that the point is not calculated.

* **Float:** It is used to store the numbers with decimal places. Maximum length of float datatype is 38 digits.

For ex: sal float (5), then there are different combinations of sal as follows:

1.1111, 11.111, 111.11, 1111.1, 11111 etc.

* + 1. **Data Definition Language (DDL):**

This language commands deals with the structure of the table.

**Create**: Used to create the table (or SQL objects).

**Syntax:**

Create table [schema.]table name

(Colname1 datatype (width), colname2 datatype (width), colname3 datatype (width),……);

**Alter**: Used to modify the structure of the table.

1. Adding new columns to the existing table

Syntax:

alter table <table\_name> add (column1 datatype(width), column 2 datatype(width),........);

1. Dropping existing columns from the table

Syntax:

alter table <table\_name> drop column\_name;

1. Increase or decrease the length of the column

Syntax:

alter table <table\_name> modify (column1 datatype(width));

1. Change the data type of column

Syntax:

alter table <table\_name> modify (column1 new\_datatype(width));

1. Column name can’t be changed
2. Table name can be changed.

Syntax: rename old\_table\_name to new\_table\_name;

**Truncate**: Used to delete the content of a table. It deletes all the records but keeps the structure of the table for future insertion of records.

Syntax:

truncate table <table\_name> ;

**Drop**: Used to drop/delete the table. It not only deletes the records but also deletes the structure of the table.

Syntax:

drop table <table\_name> ;

**Describe command:**

This command is used to view the structure of the table.

Syntax:

desc <table\_name> ;

Note: All the DDL statements are auto-committed.

* + 1. **Data Manipulation Language (DML):**

**Insert:** Used to insert a record in a table.

Syntax:

insert into <table\_name> values (val1, val2,.........);

To insert more than one record we can use the macro of insert statement.

Syntax:

insert into <table\_name> values (&col1, &coll2,.........);

**Update:** Used to update/modify the content of the table.

Syntax:

Update table <table\_name> set col1=value, col2=value,... where condition;

**Delete:** Used to delete specific records or all the records.

Syntax:

delete from <table\_name>; // to delete all the records

or

delete from <table\_name> where condition;

**Select:** Used to display content of the table.

Syntax:

Select \* from <table\_name>; // to display all the records and columns

Or

Select \* from <table\_name> where condition; // to display specific records

**Note:** All DML statements are not auto-committed and we can perform rollback.

**EXPERIMENT NO.2**

**SQL program to study and execute the aggregate functions.**

**Problem Description:**

**Functions:** Functions are programs that take zero or more arguments or parameters and return a single value.

**Single Row Functions**

**Group Functions**

Single row function operate on every row of the table and return one row for each row. And the functions which operate on all rows in a table and returns only one row as a result is known as Group Function.

**Group Functions in Oracle:**

Frequently used group functions in Oracle are:

1) sum Operate only on numeric column

2) avg

3) max

4) min Operate on all data type columns

5) count

* SUM() - Returns the sum of all the values in a column
* MIN() - Returns the smallest value in a column
* MAX() - Returns the largest value in a column
* AVG() - Returns the average value
* COUNT() - Returns the number of rows

**Note:** All built-in functions are used with the select statement only.

For ex:

SQL> select sum (sal) from emp;

SUM(SAL)

--------

29565

SQL> select sum (sal) from emp where deptno=10;

SUM(SAL)

---------

8880

**We can select more than one group functions in one select statement.**

SQL> select max (sal), max (ename), max (hiredate) from emp;

MAX(SAL) MAX(ENAME) MAX(HIRED

--------- ---------- ---------

5000 WARD 23-MAY-87

SQL> select max (sal), min (ename), count (hiredate) from emp;

MAX(SAL) MIN(ENAME) COUNT(HIREDATE)

--------- ---------- ---------------

5000 ADAMS 14

**Note:**  With group function we can not select any other column of your table.

For ex:

SQL> select ename, sum (sal) from emp;

It gives error.

SQL> select count (ename) from emp;

COUNT (ENAME)

------------

14

SQL> select count (\*) from emp;

COUNT (\*)

---------

14

**Note:** We can pass \* as an argument to the count function. When we pass column name as an argument to the count function, it returns the values excluding the NULL values, but when we pass \* as an argument to the count function, it returns all the values including NULL.

For ex:

SQL> select count (ename), count (mgr), count (\*), count (comm) from emp;

COUNT(ENAME) COUNT(MGR) COUNT(\*) COUNT(COMM)

------------ ---------- ---------- -----------

14 13 14 4

**Conclusion:** We have studied all the aggregate functions in SQL and all the queries have been successfully executed.

**EXPERIMENT NO.3**

**SQL program to study constraints in SQL.**

**Problem Description:**

Constraints are basically used to maintain the integrity (data correctness) of data.

**Types of constraints:**

1. Domain Integrity constraints:
2. Not Null constraint
3. Check constraint
4. Entity Integrity Constraints:
5. Unique constraint
6. Primary key constraint
7. Referential Integrity Constraint
8. Foreign key constraint

**Constraint Naming:**You can give a name to any type constraint but if you have not given the name to the constraint then Oracle gives the system generated name to that constraint(for ex. SYS002).

Syntax for giving a name to the constraint:

***Constraint name constraint-type***

For ex:

SQL> create table stud\_master (roll\_no number (4) constraint uroll unique,

name varchar2 (20), age number (2) constraint chkage check(age between 18 and 25));

**Modification of Constraints:**

To change the constraint name or constraint type or to drop the constraint we use the ‘alter table’ command as follows:

SQL> alter table stud\_master drop constraint chkage;

**A) Domain Integrity Constraints:**

**I) Not Null Constraint:** It is used to not to enter the NULL values in the specified column. It is defined at the column level but can’t be defined at the table level.

**Syntax:**

[constraint <constraint name> ] [not] NULL

For ex:

SQL> create table orders(order\_no number(4) constraint nn\_order\_num NOT NULL, order\_date date NOT NULL, product\_id);

In the above query user has given a name for the constraint on the order\_no column, on the other hand no name is given for the constraint on the order\_date column so it will take the system generated name.

We use “alter table modify” to add or remove a NOT NULL constraint on the columns of an existing table.

For ex:

SQL> alter table orders modify order\_date NULL; Removing

SQL> alter table orders modify product\_id NOT NULL; Adding

**II) Check Constraint:** It used to allow only a particular range of values. It is used to check a particular condition on a specified column and this condition should evaluate to a boolean(True or False) result.

The condition can’t use:

i) queries

ii) environmental functions like sysdate, user, userenv and uid.

iii) pseudo-columns such as rownum, currval, nextval and level.

**Note:** One column can have more than one check constraint.

Check constraint can be defined at column level as well as table level.

Syntax:

[constraint <constraint name> ] check

For ex:

SQL> create table bonus(emp\_id varchar2(40) NOT NULL, salary number(9,2),

bonus number(9,2) constraint ck\_bonus check (bonus>0));

Use alter table command with ADD clause to add the constraint and use alter table command with DROP clause to delete or remove the constraint. For ex:

SQL> alter table orders add cust\_id number(5) constraint ck\_cust\_id check (cust\_id>0);

SQL> alter table orders drop constraint ck\_cust\_id;

**B) Entity Integrity Constraints**

**I) Unique Constraint:** It protects/prevents one or more columns in a table ensuring that no two rows contain duplicate data in the protected columns. No limits on columns to be unique. It allows NULL values.

**Note:** Unique constraint allows NULL values in the protected columns. (i.e. after inserting the first NULL any another Nulls are allowed in the unique column).

Syntax:

[constraint <constraint name> ] unique

SQL> create table stud\_master(roll\_no number(4) unique);

SQL> create table stud\_master(roll\_no number(4) constraint uq\_rollno unique);

Use alter table command with ADD clause to add the constraint and use alter table command with DROP clause to delete or remove the constraint.

SQL> alter table stud\_master add constraint uq\_rollno unique(roll\_no);

SQL> alter table stud\_master drop constraint uq\_rollno;

**II) Primary key Constraint:**

One can say it is the combination of UNIQUE and NOT NULL constraints.(i.e. all characteristics of the unique key are applicable to the primary key constraint except that NULL values are not allowed. A table can have only one primary key constraint.

**Note:** Primary key Constraint can’t be defined in an later table command when the table contains rows having NULL values.

It can be defined at column level as well as at table level.

Syntax:

[constraint <constraint name> ] primary key

Column Level:

SQL> create table stud\_master(roll\_no number(4) constraint pk\_rollno primary key, name varchar2(20) constraint nn\_name NOT NULL);

Table Level:

SQL> create table stud\_master(roll\_no number(4),

name varchar2(20) constraint nn\_name NOT NULL,

constraint pk\_rollno primary key);

Use alter table command with ADD clause to add the constraint and use alter table command with DROP clause to delete or remove the constraint.

SQL> alter table stud\_master add roll\_no number(4) constraint pk\_rollno primary key;

SQL> alter table stud\_master drop primary key;

OR

SQL> alter table stud\_master drop constraint pk\_rollno;

**C) Referential Integrity Constraints:**

**i) Foreign Key Constraint:** It protects one or more column in a table by ensuring that for each non-null value there is data available elsewhere in the database with a primary or unique key. The foreign key columns and referential integrity columns may be in the same table (self-referential integrity constraints).

To establish a parent-child or master-detail relationship between two tables having a common column, we make use of referential integrity constraints. To implement this, we should define the column in the parent table as a primary key and the same column in the child table as a foregin key referring to the corresponding parent entry.

For ex:

SQL> create table myemp (empno number(4), ename varchar2(20),

sal number(8,2), deptno number(2) references mydept (deptno));

**Conclusion:** We studied all the types of constraints in SQL and all the queries have been executed successfully.

**EXPERIMENT NO. 4**

**SQL program to study joins in SQL.**

**Problem Description:**

SQL joins are used to query data from two or more tables, based on a relationship between certain columns in these tables.

**Types of joins:**

There are three types of joins namely:

1. Simple join
2. Self join
3. Outer join
4. **Simple join**

There are two of simple join namely:

1) Equi Join

2) Non-Equi Join

1) Equi Join: When we compare the join condition with the equal to (=) operator then that join is known as equi-join.

For ex:

SQL> select empno,ename,sal,dname

from emp, dept

where emp.deptno=dept.deptno;

“OR” by using the table alias:

SQL> select e.empno, e.ename, e.sal, d.dname

from emp e, dept d

where e.deptno =d.deptno;

Here, the condition in the where clause is known as “Join Condition”.

2) Non-Equi Join: When we does not compare the join condition with equal to(=) operator but other than equal to (>, <, >=, <=, !=, etc) then that join is known as “Non-Equi Join.”

For ex:

SQL> select empno, ename, sal, grade

from emp, salgrade

where emp.sal>= salgrade.losal and emp.sal<=salgrade.hisal;

“OR” by using the table alias:

SQL> select e.empno, e.ename, e.sal, s.grade

from emp e, salgrade s

where e.sal>= s.losal and e.sal<=s.hisal;

“OR”

SQL> select empno, ename, sal, grade

from emp, salgrade

where sal between losal and hisal;

1. **Self Join:**

When join condition joins the table to itself then that join is known as self join. In this we have to use a table alias name to the table because we can’t write a statement as follows:

SQL> select empno, ename, sal

from emp, emp

wher emp.deptno=emp.deptno;

It raise an error for the ambiguity. So, to resolve this problem we use table alias name as follows:

SQL> select e1.empno, e1.ename, e1.sal

from emp e1, emp e2

where e1.deptno=e2.deptno;

This query will execute with no errors. Suppose we want deptno also then we can’t write the query as follows:

SQL> select e1.empno, e1.ename, deptno, e1.sal

from emp1, emp e2

where e1.deptno=e2.deptno;

This query raise an error because ambiguity arises due to the common column deptno. Oracle will be confused from which table to refer the deptno? To resolve this problem we use the alias name of the table with deptno column as follows:

SQL> select e1.empno, e1.ename, e1.deptno, e1.sal

from emp1, emp e2

where e1.deptno=e2.deptno;

1. **Outer Join:**

This join is nothing but getting the data which is not the part of that join (i.e. It retrieves the rows which satisfy the join condition and the rows which does not satisfy the join condition as well). The (+) sign indicates the outer join.

SQL> select empno, ename, sal, dname

from emp, dept

where emp.deptno=dept.deptno(+);

EMPNO ENAME SAL DNAME

------ ---------- ---------- --------------

7369 SMITH 800 RESEARCH

7499 ALLEN 1600 SALES

7521 WARD 1250 SALES

7566 JONES 2975 RESEARCH

7654 MARTIN 1250 SALES

7698 BLAKE 2850 SALES

7782 CLARK 2450 ACCOUNTING

7788 SCOTT 3300 RESEARCH

7839 KING 5000 ACCOUNTING

7844 TURNER 1500 SALES

7876 ADAMS 1210 RESEARCH

7900 JAMES 950 SALES

7902 FORD 3000 RESEARCH

7934 MILLER 1430 ACCOUNTING

14 rows selected.

The result of this query is all the rows that satisfy the join condition as well as the extra rows in emp table which does not satisfy the join condition.

SQL> select empno, ename, sal, dname

from emp, dept

where emp.deptno(+)=dept.deptno;

EMPNO ENAME SAL DNAME

------ ---------- ---------- --------------

7782 CLARK 2450 ACCOUNTING

7839 KING 5000 ACCOUNTING

7934 MILLER 1430 ACCOUNTING

7369 SMITH 800 RESEARCH

7876 ADAMS 1210 RESEARCH

7902 FORD 3000 RESEARCH

7788 SCOTT 3300 RESEARCH

7566 JONES 2975 RESEARCH

7499 ALLEN 1600 SALES

7698 BLAKE 2850 SALES

7654 MARTIN 1250 SALES

7900 JAMES 950 SALES

7844 TURNER 1500 SALES

7521 WARD 1250 SALES OPERATIONS

15 rows selected.

The result of this query is all the rows that satisfy the join condition as well as the extra rows in dept table which does not satisfy the join condition.

**Conclusion:** We have studied all the types of joins and all the queries have been executed successfully.

**EXPERIMENT NO.5**

**SQL program to study sub-query in SQL.**

**Problem Description:**

**Sub-query:** Query within the query is known as subquery.

It’s role is to select the value from one query and we can use it in our another query (Parent Query) expression where the output is needed for further processing.

A subquery can be defined as a group of nested SELECT statements inside a select, insert, update or delete statement. Subqueries can be used to retrieve data from multiple tables and can be used as an alternative to joins. A subquery can also be used inside the where or having clauses of the outer select, insert, update or delete statements. Select statements containing one or more subqueries are called nested queries.

For ex:

1. List of all employees whose salary is greater than the salary of employee Jones.

SQL> select \* from emp where sal> (select sal from emp where lower(ename)= ‘jones’);

EMPNO ENAME JOB MGR HIREDATE SAL COMM DEPTNO

------ ---------- --------- ---------- --------- -------- --------- ----------

7788 SCOTT ANALYST 7566 19-APR-87 3300 20

7839 KING PRESIDENT 17-NOV-81 5000 10

7902 FORD ANALYST 7566 03-DEC-81 3000 20

1. Display only the department number in which maximum number of employees are working.

SQL> select deptno, count(\*) from emp group by deptno having count(\*)= (select max(count(\*)) from emp group by deptno);

1. List of all employees whose salary is less than any one employee working in deptno 10.

SQL> select \* from emp where sal< (select min(sal) from emp where deptno=10);

EMPNO ENAME JOB MGR HIREDATE SAL COMM DEPTNO

------ ---------- --------- -------- --------- ---------- ---------- ----------

7369 SMITH CLERK 7902 17-DEC-80 800 20

7521 WARD SALESMAN 7698 22-FEB-81 1250 500 30

7654 MARTIN SALESMAN 7698 28-SEP-81 1250 1400 30

7876 ADAMS CLERK 7788 23-MAY-87 1210 20

7900 JAMES CLERK 7698 03-DEC-81 950 30

1. List of employees who are working in the research department.

SQL> select \* from emp where deptno =( select deptno from dept where lower(dname)=’research’);

1. List of employees who are working in New York City.

SQL> select \* from emp where job in ( select job from emp where deptno= (select deptno from dept where lower(loc)=’new york’));

**Conclusion:** We have studied the sub-query technique in SQL and all the queries have been executed successfully.

**EXPERIMENT NO. 6**

**SQL program to study views in SQL.**

**Problem Description:**

**View:** A view is a customized representation of data from one or more tables. A view takes the output of a query and treats it’s as a table; therefore a view can be thought of as a “Stored Query” or a “Virtual Table”. We can use views in most places where a table can be used. The tables upon which a view is based are called “Base Tables”. It is also called as derived table.

A view is a way of hiding the logic that is created by the joined table.

**Advantages of Views:**

1) Provide an additional level of table security by restricting access to a predetermined set of rows and/or columns of a table.

2) Hide data complexity. For ex: join view.

3) They simplify commands for the user because they allow them to select information from multiple tables without actually knowing how to perform a join.

4) They isolate applications from changes in definitions of base tables.

5) They provide data in a different perspective than that of a base table by renaming columns without affecting the base table.

6) Gives real-time data.

**Disadvantages of Views:**

1. Rows available through a view are not sorted and are not ordered either.

2. Cannot use DML operations on a View.

3. When table is dropped view becomes inactive, it depends on the table objects.

4. It affects performance, querying from view takes more time than directly querying from the table

**Syntax for creating View:**

Create [or replace] [[no] [force]] view <view name> [column alias name- -] as <query> [with [check option] [read only] [constraint]];

SQL> create view emp1 as select \* from emp;

SQL> create view emp2 as select ename, deptno from emp;

**Creating Views with errors:** To create views with errors, include the “FORCE” option in the CREATE VIEW command.

SQL> create force view emp3 as select \* from hallow;

**Join View:** The view is created by using the join is known as join view.

SQL> create view empdept as select e.empno, e.ename, d.dname, d.loc

From emp e, dept d where e.deptno=d.deptno;

**Partition Views:** With partition views the data resides in separate tables. These tables are then thought together at runtime using the relational operator UNION ALL.

Syntax:

Create view <view name> as

Select \* from <table name1>

Union all

Select \* from <table name2>

Union all

Select \* from <table name3>

For ex:

SQL> create view sales as select \* from sales94 union all select \* from sales95 union all select \* from sales96;

**Dropping Views:**

Syntax:

Drop view <view name>;

SQL> drop view sales;

**Conclusion:** We have studied the views in SQL and all the queries have been executed successfully.

**EXPERIMENT NO. 7**

**SQL program to study triggers in SQL.**

**Problem Description:**

**What is a trigger?**

A trigger is a statement that is executed automatically by the system as a side effect of a modification to the database.

To design a trigger mechanism, we must:

1) Specify the conditions under which the trigger is to be executed.

2) Specify the actions to be taken when the trigger executes.

3) Triggers introduced to SQL standard in SQL: 1999, but supported even earlier using non-standard syntax by most databases.

Examples:

Suppose that instead of allowing negative account balances, the bank deals with overdrafts by

1. Setting the account balance to zero

2) Creating a loan in the amount of the overdraft

3) Giving this loan a loan number identical to the account number of the overdrawn account

4) The condition for executing the trigger is an update to the account relation that results in a negative balance value.

EX:1)

create trigger overdraft-trigger after update on account

referencing new row as nrow

for each row

when nrow.balance< 0

begin atomic

insert into borrower

(select customer-name, account-number from depositor

where nrow.account-number = depositor.account-number);

insert into loan values(n.row.account-number, nrow.branch-name,–nrow.balance);

update account set balance = 0 where account.account-number = nrow.account-number;

end;

\*\*Triggering event can be insert, delete or update

* Triggers on update can be restricted to specific attributes
* E.g. create trigger overdraft-trigger after update of balance onaccount
* Values of attributes before and after an update can be referenced
* referencing old row as: for deletes and updates
* referencing new row as : for inserts and updates
* Triggers can be activated before an event, which can serve as extra constraints.

E.g. convert blanks to null.

create trigger setnull-trigger before update on r

referencing new row as nrow

for each row

when nrow.phone-number = ‘‘

set nrow.phone-number = null;

**\*\*Statement Level triggers:**

Instead of executing a separate action for each affected row, a single action can be executed for all rows affected by a transaction

* Use for each statement instead of for each row
* Use referencing old table or referencing new table to refer to temporary tables (called transition tables) containing the affected rows
* Can be more efficient when dealing with SQL statements that update a large number of rows.

**\*\*When not to use Triggers:**

Triggers were used earlier for tasks such as

-Maintaining summary data (e.g. total salary of each department)

-Replicating databases by recording changes to special relations (called change or delta-relations) and having a separate process that applies the changes over to a replica.

There are better ways of doing these now:

1) Databases today provide built in materialized view facilities to maintain summary data.

2) Databases provide built-in support for replication.

Encapsulation facilities can be used instead of triggers in many cases

* Define methods to update fields
* Carry out actions as part of the update methods instead of through a trigger.

EX:2) Create a trigger t1 which will not allow any modifications on Tuesday.

SQL> create or replace trigger t1

before insert or update or delete on emp

begin

if to\_char(sysdate,'dy')='tue' then

raise\_application\_error(-20001,'Can not Manipulate Today');

end if;

end;

Trigger Created.

SQL> update emp set sal=10000 where lower(ename)='jones';

update emp

\*

ERROR at line 1:

ORA-20001:Can not Manipulate Today

ORA-06512:at "SCOTT.t1", line 3

ORA-04088:error during execution of trigger 'SCOTT.t1'

**Conclusion:** We have studied the trigger in SQL and all the triggers have been executed successfully.

**EXPERIMENT NO. 8**

**PL/SQL program to study procedure and function in PL/SQL.**

**Problem Description:**

**What is a Stored Procedure?**

A **stored procedure** or in simple a **proc** is a named PL/SQL block which performs one or more specific task. This is similar to a procedure in other programming languages.

A procedure has a header and a body. The header consists of the name of the procedure and the parameters or variables passed to the procedure. The body consists or declaration section, execution section and exception section similar to a general PL/SQL Block.

A procedure is similar to an anonymous PL/SQL Block but it is named for repeated usage.

We can pass parameters to procedures in three ways.  
1) IN-parameters  
2) OUT-parameters  
3) IN OUT-parameters

A procedure may or may not return any value.

**General Syntax to create a procedure is:**

*CREATE [OR REPLACE] PROCEDURE proc\_name [list of parameters]*

*IS*

*Declaration section*

*BEGIN*

*Execution section*

*EXCEPTION*

*Exception section*

*END;*

**How to execute a Stored Procedure?**

There are two ways to execute a procedure.

1) From the SQL prompt.

*EXECUTE [or EXEC] procedure\_name;*

2) Within another procedure – simply use the procedure name.

*procedure\_name;*

**Example:**

Q.1) Create a procedure p1 which takes deptno as an argument and display its corresponding location

SQL> create or replace procedure p1(dno1 number)as dloc varchar2(15);

begin

select loc into dloc from dept where deptno=dno1;

dbms\_output.put\_line('The loc is ::'||dloc);

end;

Procedure created.

SQL> set serveroutput on;

SQL> exec p1(10);

The loc is ::NEW YORK

PL/SQL procedure successfully completed.

SQL> set serveroutput on;

SQL> exec p1(20);

The loc is ::DALLAS

PL/SQL procedure successfully completed.

**What is a Function in PL/SQL?**

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

***General Syntax to create a function is***

|  |  |
| --- | --- |
| *CREATE [OR REPLACE] FUNCTION function\_name [parameters]*  *RETURN return\_datatype;*  *IS*  *Declaration\_section*  *BEGIN*  *Execution\_section*  *Return return\_variable;*  *EXCEPTION*  *exception section*  *Return return\_variable;*  *END;* |  |

1) **Return Type:** The header section defines the return type of the function. The return data-type can be any of the oracle data-type like varchar, number etc.  
2) The execution and exception section both should return a value which is of the data-type defined in the header section.

**How to execute a PL/SQL Function?**

We can execute a function by calling it in some another program or in a print line.

**Examples:**

Q.2) Create function arith which performs addition of two numbers.

SQL> create or replace function arith(a number,b number) return number is begin

return(a+b);

end; /

Function created.

SQL> declare

add number;

begin

add:=arith(33,44);

dbms\_output.put\_line('Addition of given numbers is ='||add);

end; /

Addition of given numbers is =77

PL/SQL procedure successfully completed.

Q.3) Create function square which returns square of the number.

SQL> create or replace function square(a number) return number is begin

return(a\*a);

end;

Function created.

SQL> declare

sq number;

begin

sq:=square(35);

dbms\_output.put\_line('Square of given number is ='||sq);

end; /

Square of given number is =1225

PL/SQL procedure successfully completed.

**Conclusion:** We have studied the procedure and function in PL/SQL and all the procedures and functions have been executed successfully.

**EXPERIMENT NO. 9**

**Program to study loops/iterative statements in PL/SQL.**

**Problem Description:**

Loops/Iterative control Statements are used when we want to repeat the execution of one or more statements for specified number of times.

There are three types of loops in PL/SQL:

* Simple Loop
* While Loop
* For Loop

**1) Simple Loop**

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

### General Syntax to write a Simple Loop is:

*LOOP*

*statements;*

*EXIT;*

*{or EXIT WHEN condition;}*

*END LOOP;*

These are the important steps to be followed while using Simple Loop.  
1) Initialize a variable before the loop body.  
2) Increment the variable in the loop.  
3) Use an EXIT WHEN statement to exit from the Loop. If you use an EXIT statement without WHEN condition, the statements in the loop is executed only once.

### 2) While Loop

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

The General Syntax to write a WHILE LOOP is:

*WHILE <condition>*

*LOOP statements;*

*END LOOP;*

Important steps to follow when executing a while loop:   
1) Initialize a variable before the loop body.  
2) Increment the variable in the loop.  
3) EXIT WHEN statement and EXIT statements can be used in while loops but it's not done often.

### 3) FOR Loop

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

The General Syntax to write a FOR LOOP is:

*FOR counter IN val1..val2*

*LOOP statements;*

*END LOOP;*

* val1 - Start integer value.
* val2 - End integer value.

Important steps to follow when executing a while loop: 

1) The counter variable is implicitly declared in the declaration section, so it's not necessary to declare it explicitly.  
2) The counter variable is incremented by 1 and does not need to be incremented explicitly.  
3) EXIT WHEN statement and EXIT statements can be used in FOR loops but it's not done often.

**Conclusion:** We have studied the procedure and function in PL/SQL and all the procedures and functions have been executed successfully.

**Programs**

1. PL/SQL program to check whether a given number is EVEN/ODD.

declare

n number(2);

begin

n:=&n;

if n MOD 2=0 then

dbms\_output.put\_line('Even');

else

dbms\_output.put\_line('odd.');

end if;

end;

/

1. PL/SQL program to print table of given number.

declare

i integer; n integer;

begin

n:=&n;

for i in 1..10 loop

dbms\_output.put\_line (i||'\*'||n||'='||i\*n);

end loop;

end;

/

1. PL/SQL program to reverse a given number.

declare

n number(4);

rem number(4);

rev number(4);

begin

rev:=0;

n:=&n;

while(n>0) loop

rem:=n MOD 10;

rev:=rev\*10+rem;

n:=n/10;

end loop;

dbms\_output.put\_line('The Reverse Number='||rev);

end;/