DATABASE MANAGEMENT SYSTEM LAB

(PCC- CSE-201G)

IV- SEMESTER

COMPUTER SCIENCE AND ENGINEERING

DPG Institute of Technology and Management,
Gurgaon Haryana

Submitted to:

Ms. Smriti Dwivedi
Asst. Professor
DPGITM

Submitted by:

Ankit Raj
B. Tech (CSE-DS)
4TH Semester
Roll No.: 22DS07

INDEX

EXPERIMENT 1

Creation of a database and writing SQL queries to retrieve information from the database.

To create a new database in MySQL, you use the CREATE DATABASE statement with the following syntax:

CREATE DATABASE[IF NOT EXISTS] database_name
[CHARACTER SET charset_name] [COLLATE collation_name] In this syntax:

- First, specify name of the database after the CREATE DATABASE keywords. The database name must be unique within a MySQL server instance. If you attempt to create a database with a name that already exists, MySQL will issue an error.
- Specify the character set and collation for the new database. If you skip the CHARACTER SET and COLLATE clauses, MySQL will the default character set and collation for the new database.

For Example-

CREATE DATABASE roadways_travels;

OutPut-

Following would be the result

Query OK, 1 row affected(0.02 sec)

Now, you can start creating tables and other databases objects within RoadwaysTravels database

Creation of Tables in database

SQL> create table Bus(Bus_No varchar(5), source varchar(20), destination varchar(20),CouchType varchar2(10),fair number);

Table Created.

Desc Command-

Desc <Table Name>

SQL> desc bus;

Name	Null?	Type
BUS_NO	NOT NULL	INTEGER(5),
SOURCE		VARCHAR2(10),
DESTINATION		VARCHAR2(10),
COUCH_TYPE		VARCHAR2(10);

Similarly all the tables of the database can be created and with the use of “desc” command these tables can be easily seen.

EXPERIMENT 2

Performing Insertion, Deletion, Modifying, Altering, Updating and Viewing records based on conditions.

Operation – Insertion

Syntax for insertion operation:

It is possible to write the INSERT INTO statement in two ways:

1. Specify both the column names and the values to be inserted:

```
INSERT INTO table_name (column1, column2..) VALUES
(value1, value2, value3, ...);
```

2. If you are adding values for all the columns of the table, you do not need to specify the column names in the SQL query.

```
INSERT INTO table_name
VALUES (value1, value2, value3, ...);
```

Example for Insert operation-

The following SQL statement inserts a new record in the "Bus" table:

```
INSERT INTO Customers(Bus_no,Source,Destination,CouchType) VALUES
('2345', ' gurugram'. 'delhi', 'general');
```

Operation- Deletion

The DELETE statement is used to delete existing records in a table.

Example of Deletion Operation

```
Delete from Bus where Bus_no='1024';
```

This query will delete record from the bus table with Bus_no equal to '1024'.

Operation- Alter

The ALTER TABLE statement is used to add, delete, or modify columns in an existing table.

The ALTER TABLE statement is also used to add and drop various constraints on an existing table.

Alter Table- Add Column

To add a column in a table, use the following syntax:

```
ALTER TABLE table_name
ADD column_name datatype;
```

Example

```
ALTER TABLE Bus
ADD Time varchar(40); Alter
```

Table- Drop column

To delete a column in a table, use the following syntax:

```
ALTER TABLE table_name
DROP COLUMN column_name;
```

Example

```
ALTER TABLE Bus
DROP COLUMN Time;
```

Alter table – Alter/Modify Column

To change the data type of a column in a table, use the following syntax:

```
ALTER TABLE table_name
MODIFY COLUMN column_name datatype;
```

Example

```
ALTER TABLE Bus
MODIFY COLUMN Bus_No Varchar(50);
```

The SQL UPDATE Statement

The UPDATE statement is used to modify the existing records in a table.

UPDATE Syntax

```
UPDATE table_name
SET column1 = value1, column2 = value2,..
WHERE condition;
```

Example

```
UPDATE Bus
SET Source= ‘Kullu’
WHERE Bus_No = ‘1024’;
```

This update statement will change source for BusNo 1024.

Operation- Viewing of Records

The SQL SELECT Statement

The SELECT statement is used to select data from a database.

The data returned is stored in a result table, called the result-set.

SELECT Syntax
SELECT column1, column2,...
FROM table_name;
Here, column1, column2, ... are the field names of the table you want to select data from. If you want to select all the fields available in the table, use the following syntax:

```
SELECT * FROM table_name;
```

Example

```
Select * from Bus;
```

This Statement will give you all records of the table Bus.

```
Select Bus_no,Source, Destination From Bus;
```

This query will give records from Bus with only three field values.

EXPERIMENT 3

Creation of Views, Synonyms, Sequence, Indexes, Save point

SQL CREATE VIEW Statement

A view is created with the CREATE VIEW statement.

Syntax

```
CREATE VIEW view_name AS
SELECT column1, column2,
FROM table_name
WHERE condition;
```

The following SQL creates a view that shows all Buses from Gurugram:

```
CREATE VIEW TouristBus AS
SELECT Bus_No, Couch_Type
FROM Bus
WHERE Source='Gurugram';
```

INDEXES

SQL CREATE INDEX Statement

The CREATE INDEX statement is used to create indexes in tables.

CREATE INDEX Syntax

Creates an index on a table. Duplicate values are allowed:

```
CREATE INDEX index_name
ON table_name (column1, column2, ...);
```

CREATE UNIQUE INDEX Syntax

Creates a unique index on a table. Duplicate values are not allowed:

```
CREATE UNIQUE INDEX index_name
ON table_name (column1, column2, ...);
```

CREATE INDEX Example

```
CREATE INDEX idx_lastname ON
```

Persons (LastName);

CREATE INDEX idx_pname
ON Persons (LastName, FirstName); DROP

INDEX Statement

The DROP INDEX statement is used to delete an index in a table.

ALTER TABLE table_name DROP INDEX index_name; Example:-

alter table Bus drop index lastname;

SYNONYM-

SQL Server CREATE SYNONYM statement syntax
To create a synonym, you use the CREATE SYNONYM statement as follows:

CREATE SYNONYM synonym_name
FOR object;

The object is in the following form:

[server_name.[database_name] . [schema_name_2]. object_name

SEQUENCES

Example of Usage

To create a sequence, the following general statement can be used:

CREATE SEQUENCE SEQUENCE_NAME

[MAXVALUE q |NOMAXVALUE]

[MINVALUE v |NOMINVALUE]

[START WITH n]

[INCREAMENT WITH N]

[CACHE x| NOCACHE]
[CYCLE|NOCYCLE];

Savepoint Command

The syntax for a SAVEPOINT command is as shown below.

```
SAVEPOINT SAVEPOINT_NAME;
```

The syntax for rolling back to a SAVEPOINT is as shown below.

```
ROLLBACK TO SAVEPOINT_NAME;
```

Example

Consider the CUSTOMERS table having the following records.

```
ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 | |
6 | Komal    |  22 | MP        |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

The following code block contains the series of operations.

```
SQL> SAVEPOINT SP1;
Savepoint created.
SQL> DELETE FROM CUSTOMERS WHERE ID=1;
1 row deleted. SQL> SAVEPOINT SP2;  Savepoint
created.
SQL> DELETE FROM CUSTOMERS WHERE ID=2;
1 row deleted. SQL> SAVEPOINT SP3;  Savepoint
created.
SQL> DELETE FROM CUSTOMERS WHERE ID=3;
1 row deleted.
SQL> ROLLBACK TO SP2;
Rollback complete.
```

Notice that only the first deletion took place since you rolled back to SP2.

```
SQL> SELECT * FROM CUSTOMERS;
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   |           SALARY   | +----+----------+-----+-----------+----------+
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
```

```
|  7 | Muffy    |  24 | Indore    | 10000.00 | +--
-+----------+-----+-----------+----------+       6
rows selected.
```

The RELEASE SAVEPOINT Command

The RELEASE SAVEPOINT command is used to remove a SAVEPOINT that you have created.

The syntax for a RELEASE SAVEPOINT command is as follows.

RELEASE SAVEPOINT SAVEPOINT_NAME;

Once a SAVEPOINT has been released, you can no longer use the ROLLBACK command to undo transactions performed since the last SAVEPOINT.

EXPERIMENT 4

Creating an Employee database to set various constraints

Creating a Database of EMPLOYEE

```
create database Employee;
Query OK, 1 row affected (0.01 sec)

mysql> use Employee;
Database changed
```

Creating three relations :- 1) Employee
2) Project
3) Manager

```
mysql> create table Project ( Pid int primary key, Pname varchar(20), P_duration varchar(20) );
Query OK, 0 rows affected (0.08 sec)

mysql> create table Employee ( Eid int primary key, Ename varchar(20), E_number int(20), Pid int , foreign key (Pid) references Project(Pid) );

mysql> create table Manager ( Mid int primary key,
    -> Mname varchar(20), Pid int , foreign key (Pid) references Project(Pid) );
Query OK, 0 rows affected (0.11 sec)
```

Describing all relations :-

```
mysql> describe Project;
+------------+-------------+------+-----+---------+-------+
| Field      | Type        | Null | Key | Default | Extra |
+------------+-------------+------+-----+---------+-------+  |
 Pid        | int         | NO   | PRI | NULL    |       |
| Pname      | varchar(20) | YES  |     | NULL    |       | |
 P_duration | varchar(20) | YES  |     | NULL    |       |
+------------+-------------+------+-----+---------+-------+
3 rows in set (0.06 sec)

mysql> describe Manager;
+-------+-------------+------+-----+---------+-------+
| Field | Type        | Null | Key | Default | Extra |  +-
------+-------------+------+-----+---------+-------+  |
 Mid   | int         | NO   | PRI | NULL    |       |
| Mname | varchar(20) | YES  |     | NULL    |       | |
 Pid   | int         | YES  | MUL | NULL    |       |
+-------+-------------+------+-----+---------+-------+
3 rows in set (0.00 sec)

mysql> describe Employee;
+----------+-------------+------+-----+---------+-------+
| Field    | Type        | Null | Key | Default | Extra |
+----------+-------------+------+-----+---------+-------+
| Eid      | int         | NO   | PRI | NULL    |       |
| Ename    | varchar(20) | YES  |     | NULL    |       |
| E_number | int         | YES  |     | NULL    |       |
```

```
| Pid      | int         | YES  | MUL | NULL    |       |
+----------+-------------+------+-----+---------+-------+
4 rows in set (0.00 sec)
```

Inserting values in all relations :-

```
mysql> insert into Project values  ( 21, 'Abcd', '3 Months');
Query OK, 1 row affected (0.04 sec)

mysql> insert into Project values  ( 22, 'Abc', '2 Months');
Query OK, 1 row affected (0.04 sec)

mysql> insert into Project values  ( 41, 'Ab', '1 Months');
Query OK, 1 row affected (0.04 sec)

mysql> insert into Project values  ( 42, 'Abfg', '1 Months');
Query OK, 1 row affected (0.04 sec)

mysql> select * from project; +-----+-------+------------+
| Pid | Pname | P_duration |
+-----+-------+------------+ |
21 | Abcd  | 3 Months   |
|  22 | Abc   | 2 Months   |
|  41 | Ab    | 1 Months   | |
 42 | Abfg  | 1 Months   |
+-----+-------+------------+
4 rows in set (0.04 sec)

mysql> insert into Employee values ( 1, 'Sachin', 844747, 21 );
Query OK, 1 row affected (0.04 sec)

mysql> insert into Employee values ( 2, 'Ritik', 98765, 22 );
Query OK, 1 row affected (0.04 sec)

mysql> insert into Employee values ( 3, 'August', 9870, 42 );
Query OK, 1 row affected (0.00 sec)

mysql> insert into Employee values ( 4, 'Abhinav', 67859, 41 );
Query OK, 1 row affected (0.04 sec)

mysql> select * from Employee;
+-----+---------+----------+------+  |
 Eid | Ename   | E_number | Pid  |
```

```
+-----+---------+----------+------+  |
1 | Sachin  |   844747 |   21 |
|   2 | Ritik   |    98765 |   22 |
|   3 | August  |     9870 |   42 |
|   4 | Abhinav |    67859 |   41 |  +-----+---------+----------+------+
4 rows in set (0.04 sec)
```

```
mysql> insert into Manager values ( 101, 'Nishant', 21);
Query OK, 1 row affected (0.04 sec)
```

```
mysql> insert into Manager values ( 102, 'Abhinav', 22);
Query OK, 1 row affected (0.04 sec)
```

```
mysql> insert into Manager values ( 103, 'Prateek', 41);
Query OK, 1 row affected (0.00 sec)
```

```
mysql> insert into Manager values ( 104, 'Prem', 42);
Query OK, 1 row affected (0.04 sec)
```

```
mysql> select * from Manager;
+-----+---------+------+  |
 Mid | Mname   | Pid  |
+-----+---------+------+  |
 101 | Nishant |   21 |
| 102 | Abhinav |   22 |
| 103 | Prateek |   41 |
| 104 | Prem    |   42 |
+-----+---------+------+
4 rows in set (0.00 sec)
```

CREATING VIEWS :-

```
mysql> create view Emp_details as select Eid , Ename from Employee where Pid = 42;
Query OK, 0 rows affected (0.05 sec)
```

```
mysql> select * from Emp_details;
+-----+--------+
| Eid | Ename  |
+-----+--------+
|   3 | August |
+-----+--------+
1 row in set (0.04 sec)
```

UPDATING VIEWS:-

```
mysql> select * from Emp_details;
+-----+--------+
| Eid | Ename  |
+-----+--------+
|   3 | August |
+-----+--------+
```

1 row in set (0.00 sec)

```
mysql> select * from Emp_detail;
+-----+--------+------+----------+
| Eid | Ename  | Pid  | E_number |
+-----+--------+------+----------+
|   3 | August |   42 |     9870 |
+-----+--------+------+----------+
1 row in set (0.00 sec)
```

DROPPING VIEW:-

```
mysql> drop view Emp_detail;  Query
OK, 0 rows affected (0.01 sec)

mysql> select * from Emp_detail;
ERROR 1146 (42S02): Table 'employee.emp_detail' doesn't exist
```

INTIGRITY CONSTRAINTS

-> Creating a relation named " college_student ".

```
mysql> create table college_student ( id int primary key, Name varchar(20) not null, roll_number int unique );
Query OK, 0 rows affected (0.12 sec)
```

Here id = Primary key
Name = not null
roll_number = unique

```
mysql> describe
college_student;
+-------------+-------------+------+-----+---------+-------+
| Field       | Type        | Null | Key | Default | Extra |
+-------------+-------------+------+-----+---------+-------+  |
id          | int         | NO   | PRI | NULL    |       |
| Name        | varchar(20) | NO   |     | NULL    |       | |
roll_number | int         | YES  | UNI | NULL    |       |
+-------------+-------------+------+-----+---------+-------+
```

-> Creating a relation named " college_class ".

```
mysql> create table college_class ( roll_number int primary key, semester int not null, foreign key(roll_number) references
college_student(roll_number) );
Query OK, 0 rows affected (0.09 sec)
```

Here roll_number = Primary key and foreign key semester

mysql> desc college_class;

```
+-------------+------+------+-----+---------+-------+
| Field       | Type | Null | Key | Default | Extra |  +-
------------+------+------+-----+---------+-------+  |
roll_number | int  | NO   | PRI | NULL    |       |
| semester    | int  | NO   |     | NULL    |       |
+-------------+------+------+-----+---------+-------+
```

-> Inserting values in the relations:-

```
mysql> insert into college_student values ( 21, 'Sachin', 1 );
Query OK, 1 row affected (0.01 sec)

mysql> insert into college_student values ( 22, 'August', 2 );
Query OK, 1 row affected (0.04 sec)

mysql> insert into college_student values ( 23, 'Vikas', 3 );
Query OK, 1 row affected (0.04 sec)

mysql> insert into college_student values ( 24, 'Raj', 4 );
Query OK, 1 row affected (0.00 sec)

mysql> select * from college_student;
+----+--------+-------------
+
| id | Name   | roll_number |
+----+--------+-------------+  |
21 | Sachin |           1 |
| 22 | August |           2 |
| 23 | Vikas  |           3 |
| 24 | Raj    |           4 |
+----+--------+-------------+

mysql> insert into college_class values ( 1, 3 );
Query OK, 1 row affected (0.01 sec)

mysql> insert into college_class values ( 2, 3 );
Query OK, 1 row affected (0.04 sec)

mysql> insert into college_class values ( 3, 3 );
Query OK, 1 row affected (0.04 sec)

mysql> insert into college_class values ( 4, 3 );
Query OK, 1 row affected (0.04 sec)

mysql> select * from college_class;
```

```
+-------------+----------+  |
roll_number | semester |
+-------------+----------+
|           1 |        3 |
|           2 |        3 |
|           3 |        3 |
|           4 |        3 |
+-------------+----------+
```

-> Implimenting Query / Query checking :-

```
mysql> select Name from college_class, college_student where (college_student.roll_number = college_class.roll_number );
+--------+
| Name   |
+--------+  |
 Sachin |
| August |
| Vikas  |
| Raj    |  +--------+
```

EXPERIMENT 5

Creating relationship In the databases.

we create a relationship between two tables when you want to associate rows of one table with rows of another.

Creating a new table with a foreign key requires CREATE TABLE permission in the database, and ALTER permission on the schema in which the table is being created.

Creating a foreign key in an existing table requires ALTER permission on the table

```
CREATE TABLE Sales.TempSalesReason
  (
    TempID int NOT NULL, Name nvarchar(50)
    , CONSTRAINT PK_TempSales PRIMARY KEY NONCLUSTERED (TempID)
    , CONSTRAINT FK_TempSales_SalesReason FOREIGN KEY (TempID)
      REFERENCES Sales.SalesReason (SalesReasonID)
      ON DELETE CASCADE
      ON UPDATE CASCADE
  )
;
```

Create a foreign key in an existing table

The following example creates a foreign key on the column TempID and references the column SalesReasonID in the Sales.SalesReason table in the AdventureWorks database.

```
ALTER TABLE Sales.TempSalesReason
  ADD CONSTRAINT FK_TempSales_SalesReason FOREIGN KEY (TempID)
    REFERENCES Sales.SalesReason (SalesReasonID)
    ON DELETE CASCADE
    ON UPDATE CASCADE
;
```

EXPERIMENT 6

Study of PL/SQL block.

PL/SQL Block Structure

In PL/SQL, as in most other procedural languages, the smallest meaningful grouping of code is known as a block. A block is a unit of code that provides execution and scoping boundaries for variable declarations and exception handling. PL/SQL allows you to create anonymous blocks (blocks of code that have no name) and named blocks, which may be packages, procedures, functions, triggers, or object types.

A PL/SQL block has up to four different sections, only one of which is mandatory:

Header

Used only for named blocks. The header determines the way the named block or program must be called. Optional. Declaration section

Identifies variables, cursors, and subblocks that are referenced in the execution and exception sections. Optional.

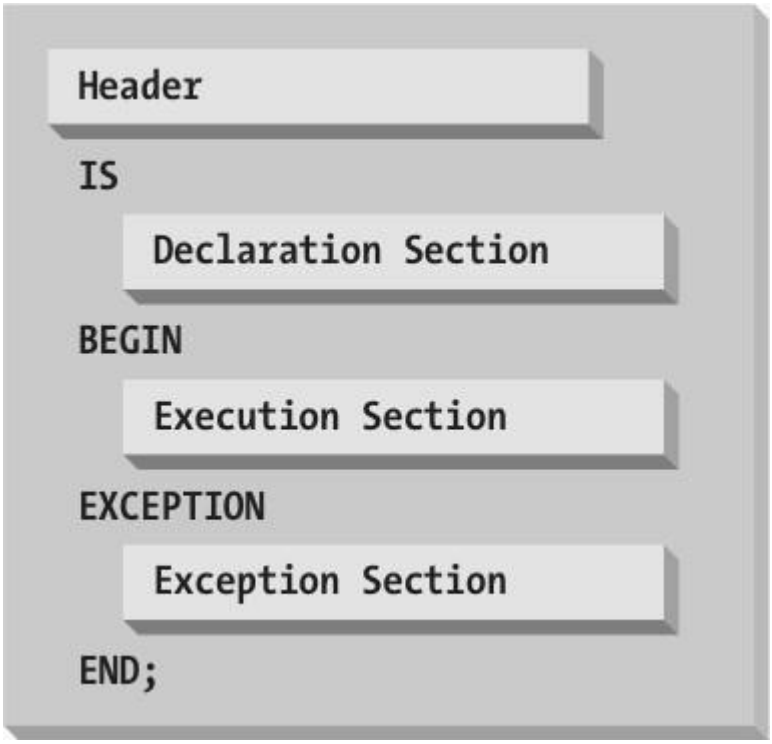
Execution section

Statements the PL/SQL runtime engine will execute at runtime. Mandatory.

Exception section

Handles exceptions to normal processing (warnings and error conditions). Optional.

Following figure shows the structure of the PL/SQL block for a procedure.



Below is a procedure containing all four sections of the elements of a block. This particular block begins with the keyword PROCEDURE, and, like all blocks, ends with the keyword END.

```
PROCEDURE get_happy (ename_in IN VARCHAR2)   <-- Header
IS
   l_hiredate DATE;                          <-- Declaration
BEGIN
   l_hiredate := SYSDATE - 2;
   INSERT INTO employee                      <-- Execution
      (emp_name, hiredate)
   VALUES (ename_in, l_hiredate);
EXCEPTION
   WHEN DUP_VAL_IN_INDEX
   THEN
      DBMS_OUTPUT.PUT_LINE                   <-- Exception
         ('Cannot insert.');
END;
```

EXPERIMENT 7

Write a PL/SQL block to satisfy some conditions by accepting input from the user.

Here presenting a PL/SQL block using different control (if, if else, for loop, while loop,...) statements.

To write a PL/SQL block using different control (if, if else, for loop, while loop,...) statements.

OBJECTIVE:

PL/SQL Control Structure provides conditional tests, loops, flow control and branches that let to produce well-structured programs. Addition of Two Numbers:

PL/ SQL General Syntax

```
SQL>  DECLARE
          <VARIABLE DECLARATION>;
      BEGIN
          <EXECUTABLE STATEMENT >;
      END;
```

PL/SQL CODING FOR ADDITION OF TWO NUMBERS :-

```
SQL> Set serveroutput on
SQL> declare
  2  a number;
  3  b number;
  4  c number;
  5  begin
  6  a := &a;
  7  b := &b;
  8  c := a+b;
  9  dbms_output.put_line('Sum of '||a||'and'||b||'is'||c);
 10  end;
 11  /
Enter value for a: 23
old   6: a := &a;
new   6: a := 23;
Enter value for b: 12
old   7: b := &b;
new   7: b := 12;
Sum of 23and12is35

PL/SQL procedure successfully completed.
```

PL/ SQL Program for IF Condition:

2. Write a PL/SQL Program using if condition:-

PROCEDURE

STEP 1: Start

STEP 2: Initialize the necessary variables.

STEP 3: invoke the if condition.

STEP 4: Execute the statements.

STEP 5: Stop.

PL/ SQL GENERAL SYNTAX FOR IF CONDITION: SQL>

DECLARE

<VARIABLE DECLARATION>;
BEGIN
IF(CONDITION)THEN
<EXECUTABLE STATEMENT >;
END; Coding

for If Statement:

```
SQL> Set serveroutput on
SQL> declare
  2  b number;
  3  c number;
  4  begin
  5  b :=10;
  6  c :=20;
  7  if (c>b) THEN
  8  dbms_output.put_line('c is maximum');
  9  end if;
 10  end;
 11  /
c is maximum

PL/SQL procedure successfully completed.
```

PL/ SQL GENERAL SYNTAX FOR IF AND ELSECONDITION:

SQL> DECLARE
<VARIABLE DECLARATION>;
BEGIN
IF (TEST CONDITION) THEN
<STATEMENTS>;
ELSE
<STATEMENTS>;
ENDIF;
END;

Coding for Less then or Greater Using IF ELSE :-

```
SQL> set serveroutput on
SQL> declare
  2  n number;
  3  begin
  4  dbms_output.put_line('Enter a number');
  5  n := &nnumber;
  6  if n<5 then
  7  dbms_output.put_line('Entered number is less than 5');
  8  else
  9  dbms_output.put_line('Entered number is greater than 5');
 10  end if;
 11  end;
 12  /
Enter value for nnumber: 2
old   5: n := &nnumber;
new   5: n := 2;
Enter a number
Entered number is less than 5

PL/SQL procedure successfully completed.
```

EXPERIMENT 8

Write a PL/SQL block that handles all types of exceptions.

Types of Exception
There are two types of Exceptions in Pl/SQL.

1. Predefined Exceptions
2. User-defined Exception Predefined Exceptions

```
DECLARE
<exception_name> EXCEPTION;
BEGIN
<Execution block>
EXCEPTION
WHEN <exception_name> THEN
<Handler>
END;
```

Example

```
DECLARE

Sample_exception EXCEPTION;

PROCEDURE nested_block
IS
BEGIN
Dbms_output.put_line('Inside nested block');
Dbms_output.put_line('Raising sample_exception from nested block');
RAISE sample_exception;
EXCEPTION
WHEN sample_exception THEN
Dbms_output.put_line ('Exception captured in nested block. Raising to main block');
RAISE,
END;
BEGIN
Dbms_output.put_line('Inside main block');
Dbms_output.put_line('Calling nested block');
Nested_block;
EXCEPTION
WHEN sample_exception THEN
                Dbms_output.put_line ('Exception captured in main block'); END:
```

EXPERIMENT 9

Creation of Procedures. Stored

Procedure

A stored procedure is a prepared SQL code that you can save, so the code can be reused over and over again.

So if you have an SQL query that you write over and over again, save it as a stored procedure, and then just call it to execute it.

You can also pass parameters to a stored procedure, so that the stored procedure can act based on the parameter value(s) that is passed.

Syntax

```
CREATE PROCEDURE procedure_name AS
sql_statement GO;

Execute a Stored Procedure  EXEC
procedure_name;
```

Example

The following SQL statement creates a stored procedure named "SelectAllCustomers" that selects all records from the "Customers" table:

```
CREATE PROCEDURE SelectAllCustomers
AS
SELECT * FROM Customers
GO;

Example
EXECl SelectAllCustomers;
```

EXPERIMENT 10

Creation of database triggers and functions

Syntax:

```
create trigger [trigger_name]  [before |
after]
{insert | update | delete}
on [table_name]   [for each
row]
[trigger_body]
```

BEFORE and AFTER of Trigger:

BEFORE triggers run the trigger action before the triggering statement is run. AFTER triggers run the trigger action after the triggering statement is run.

Example:

Given Student Report Database, in which student marks assessment is recorded. In such schema, create a trigger so that the total and average of specified marks is automatically inserted whenever a record is insert.

Here, as trigger will invoke before record is inserted so, BEFORE Tag can be used.

Suppose the database Schema –

```
mysql> desc Student;
+-------+-------------+------+-----+---------+----------------+
| Field | Type        | Null | Key | Default | Extra          |
+-------+-------------+------+-----+---------+----------------+
| tid   | int(4)      | NO   | PRI | NULL    | auto_increment |
| name  | varchar(30) | YES  |     | NULL    |                |
| subj1 | int(2)      | YES  |     | NULL    |                |
| subj2 | int(2)      | YES  |     | NULL    |                |
| subj3 | int(2)      | YES  |     | NULL    |                |
| total | int(3)      | YES  |     | NULL    |                |
| per   | int(3)      | YES  |     | NULL    |                |
+-------+-------------+------+-----+---------+----------------+
7 rows in set (0.00 sec)
```

SQL Trigger to problem statement.

```
create trigger stud_marks
before    INSERT        on
Student  for each row
set Student.total = Student.subj1 + Student.subj2 + Student.subj3, Student.per = Student.total * 60 /
100;
```

Above SQL statement will create a trigger in the student database in which whenever subjects marks are entered, before inserting this data into the database, trigger will compute those two values and insert with the entered values. i.e., mysql> insert into Student values(0, "ABCDE", 20, 20, 20, 0, 0); Query OK, 1 row affected (0.09 sec)

```
mysql> select * from Student;
+-----+-------+-------+-------+-------+-------+------+
| tid | name  | subj1 | subj2 | subj3 | total | per  |
+-----+-------+-------+-------+-------+-------+------+
| 100 | ABCDE |    20 |    20 |    20 |    60 |   36 |
+-----+-------+-------+-------+-------+-------+------+
1 row in set (0.00 sec)
```

In this way trigger can be creates and executed in the databases.

Functions

For user-defined functions, these syntaxes look as follows:

```
CREATE FUNCTION [database_name.]function_name (parameters)
RETURNS data_type AS
BEGIN
   SQL statements
   RETURN value
END;

ALTER FUNCTION [database_name.]function_name (parameters)
RETURNS data_type AS
BEGIN
   SQL statements
   RETURN value
END;

DROP FUNCTION [database_name.]function_name;
```

Example

We want to list all cities and write down are they east or west when compared to London (longitude = 0). Cities east of London will have positive city.long values, while those west of London will have this value negative.

```
CREATE FUNCTION east_or_west (
@long DECIMAL(9,6)
)
RETURNS CHAR(4) AS
BEGIN
DECLARE @return_value CHAR(4);
SET @return_value = 'same';
   IF (@long > 0.00) SET @return_value = 'east';
   IF (@long < 0.00) SET @return_value = 'west';

   RETURN @return_value
END;
```