

EXPERIMENT 10**PRE-LAB****1. Analyze the code and tell your observation?**

```
DECLARE
a number(3) := 100;
BEGIN
IF (a = 50 ) THEN
dbms_output.put_line('Value of a is 10' );
ELSEIF ( a = 75 ) THEN
dbms_output.put_line('Value of a is 20' );
ELSE
dbms_output.put_line('None of the values is matching');
END IF;
dbms_output.put_line('Exact value of a is: ' || a );
END;
```

Ans) The Output is
None of the values is matching
Exact value of a is 100

2. What will be the output of the following code?

```
DECLARE
lines dbms_output.chararr;
num_lines number;
BEGIN
Dbms_output.enable;
dbms_output.put_line('Hello!');
dbms_output.put_line('Hope you are doing well!');
num_lines := 2;
dbms_output.get_lines(lines, num_lines);
FOR i IN 1..num_lines LOOP
dbms_output.put_line(lines(i));
END LOOP;
END;
```

Ans) Hello Reader
Hope you have enjoyed doing well
2

3. Consider the following code :-

```
DECLARE
-- Global variables
num number := 95;
BEGIN
dbms_output.put_line('num: ' || num1);
DECLARE
```

```
-- Local variables
num number := 195;
BEGIN
  dbms_output.put_line('num: ' || num1);
END;
END;
```

What will happen when the code is executed?

Ans) Not executed , because syntax error.

4. What would be printed when the following code is executed?

```
DECLARE
  x NUMBER;
BEGIN
  x := 5;
  x := 10;
  dbms_output.put_line(-x);
  dbms_output.put_line(+x);
  x := -10;
  dbms_output.put_line(-x);
  dbms_output.put_line(+x);
END;
```

Ans) -10
10
10
-10

5. What will be printed by the following PL/SQL block?

```
DECLARE
  a number;
  b number;
  c number;
PROCEDURE findMin(x IN number, y IN number, z OUT number) IS
BEGIN
  IF x < y THEN
    z:= x;
  ELSE
    z:= y;
  END IF;
END;
BEGIN
  a:= 2;
  b:= 5;
  findMin(a, b, c);
  dbms_output.put_line(c);
END;
```

Ans) -5
-10
-25

6. What will be printed by the following PL/SQL block?

```
DECLARE
  a number;
```

```
PROCEDURE squareNum(x IN OUT number) IS
BEGIN
  x := x * x;
END;
BEGIN
  a:= 5;
  squareNum(a);
  dbms_output.put_line(a);
END;
```

Ans) -5
-10
-25

7. When is the pre-defined exception “CASE_NOT_FOUND” raised?

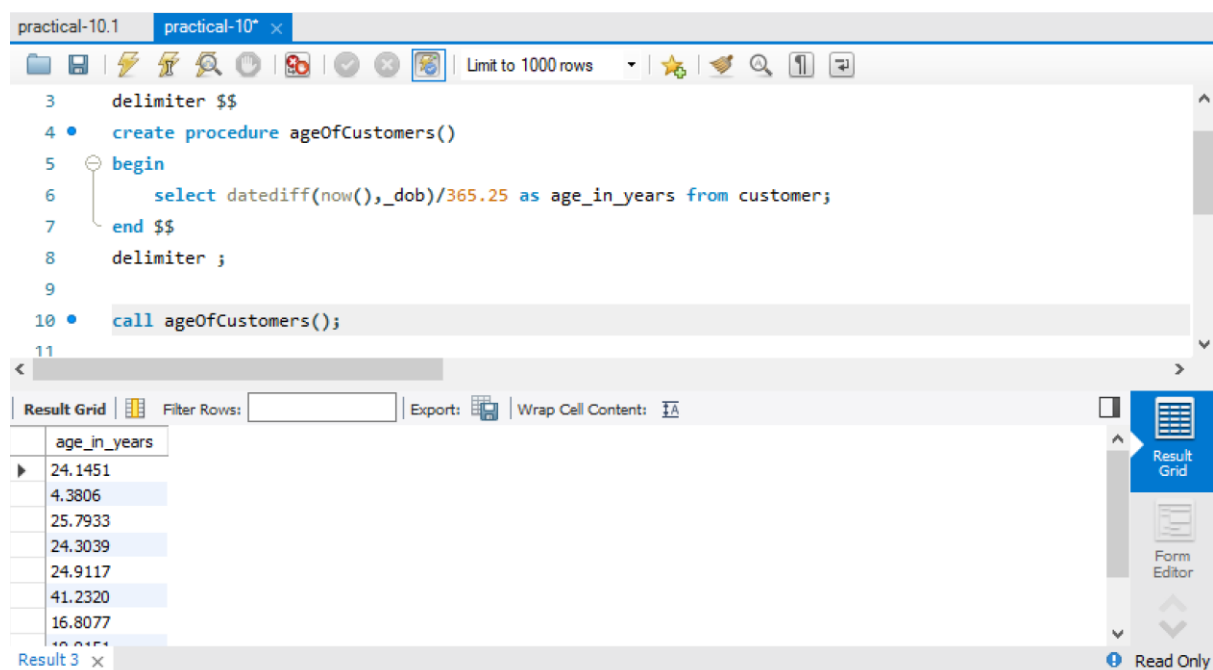
None of the choices in the when clauses of a case statement is selected , and there is no ELSE clause.

IN-LAB**Case Study 1 : TRANSPORT DEPARTMENT**

1. Write a PL/SQL stored procedure to know the current age of customers who are associated with AP transport department.

```
delimiter $$
create procedure ageOfCustomers()
begin
    select datediff(now(),_dob)/365.25 as age_in_years from customer;
end $$
delimiter ;

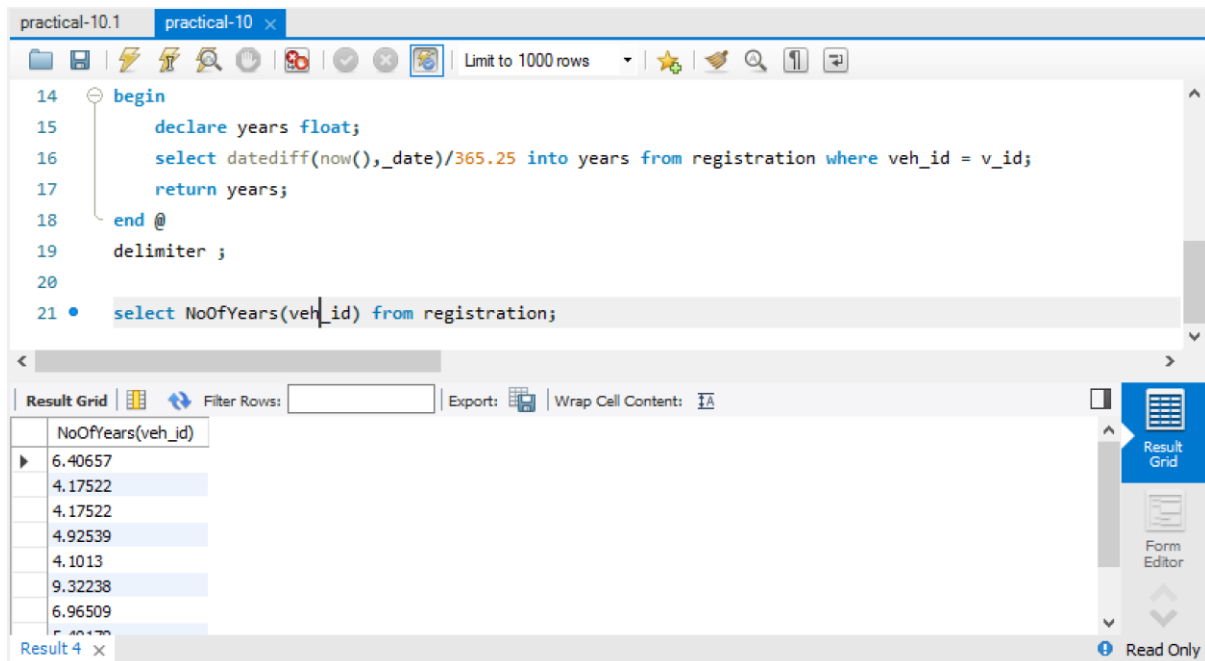
call ageOfCustomers();
```



2. Write a PL/SQL stored function to know that, from how many years vehicles are registered with the AP transport department.

```
delimiter @
create function NoOfyears(v_id int) returns float
begin
    declare years float;
    select datediff(now(),_date)/365.25 into years from registration where veh_id = v_id;
    return years;
end @
delimiter ;

select NoOfYears(veh_id) from registration;
```



3. Create a trigger before insert to maintain the summary of DealerCenter table into DealerCenterstats. Whenever the capacity of DealerCenters is increased or decreased then the total statistics should be reflected in DealerCenterstats

```

create table dealerCenterStats(new_deal_id int,new_deal_name varchar(25),new_city
varchar(25),new_street varchar(25), new_state varchar(25),new_pincode int,new_dno
int,new_phno bigint);
  
```

delimiter \$\$

create trigger new_dealer before insert on dealer for each row

begin

```

    insert into dealerCenterStats values(NEW.deal_id, NEW.deal_name, NEW.city,
NEW.street, NEW.state,NEW.pincode,NEW.d_no,NEW.ph_no);
  
```

end \$\$

delimiter ;

```

insert into dealer values(61,'RK','Vijayawada','BenzCircle','AndhraPradesh',500023,
112,7286009239);
  
```

```

select * from dealercenterstats;
  
```

The screenshot shows a SQL IDE with a script editor and a result grid. The script editor contains the following SQL code:

```

27 • create trigger new_dealer before insert on dealer for each row
28 • begin
29 •     insert into dealerCenterStats values(NEW.deal_id,NEW.deal_name,NEW.city,NEW.street,NEW.state,NEW.pi
30 • end $$
31 • delimiter ;
32
33 • insert into dealer values(61,'RK','Vijayawada','Benz Circle','AndhraPradesh',500023,112,7286009239);
34 • select * from dealercenterstats;
  
```

The result grid shows the following data:

| new_deal_id | new_deal_name | new_city | new_street | new_state | new_pincode | new_dno | new_phno |
|-------------|---------------|------------|-------------|---------------|-------------|---------|------------|
| 61 | RK | Vijayawada | Benz Circle | AndhraPradesh | 500023 | 112 | 7286009239 |

The IDE interface includes a toolbar with icons for file operations, a "Limit to 1000 rows" dropdown, and a "Result Grid" button on the right. The status bar at the bottom indicates "dealercenterstats 5" and "Read Only".

4. Create trigger after insert in members table , a trigger should check the value of attribute name and if it is updated then show the message for updating on name in reminder table.

```
create table remainder(before_name varchar(50),after_name varchar(50));
```

```
delimiter $$
```

```
create trigger Customer_log after update on customer for each row
```

```
begin
```

```
    insert into remainder values(OLD.cust_name,NEW.cust_name);
```

```
end $$
```

```
delimiter ;
```

```
update customer set cust_name='RK' where cust_id=41;
```

```
select * from remainder;
```

The screenshot shows a SQL IDE with a script editor and a result grid. The script editor contains the following SQL code:

```

39 • create trigger Customer_log after update on customer for each row
40 • begin
41 •     insert into remainder values(OLD.cust_name,NEW.cust_name);
42 • end $$
43 • delimiter ;
44
45 • update customer set cust_name='RK' where cust_id=41;
46 • select * from remainder;
  
```

The result grid shows the following data:

| before_name | after_name |
|-------------|------------|
| raju | RK |

The IDE interface includes a toolbar with icons for file operations, a "Limit to 1000 rows" dropdown, and a "Result Grid" button on the right. The status bar at the bottom indicates "Form Editor".

Case Study 4 : KL UNIVERSITY ERP

1. Write a Program to create a row level trigger that would fire for INSERT or UPDATE or DELETE operations performed on the Faculty table. The program has to print the salary difference of faculty along with Old salary and New salary

```
create table faculty_Log(operation_id int primary key auto_increment,FID int,
FNAME varchar(10),Designation varchar(10),Salary int,FMOBILE bigint,
FMAIL varchar(20),FADD varchar(10),Branch varchar(10),changed_at DATETIME NOT
NULL,
operation varchar(3) NOT NULL,CHECK(operation = 'INS' or operation = 'DEL'));
```

```
delimiter $$
```

```
create trigger trig_faculty_insert after insert on faculty
```

```
for each row
```

```
begin
```

```
    insert into
```

```
faculty_Log(FID,FNAME,Designation,Salary,FMOBILE,FMAIL,FADD,Branch,changed_at,ope
ration)
```

```
values(NEW.FID,NEW.FNAME,NEW.Designation,NEW.Salary,NEW.FMOBILE,NEW.FMAIL,N
EW.FADD,NEW.BRANCH,current_timestamp,'INS');
```

```
end $$
```

```
delimiter ;
```

```
insert into faculty
```

```
values(5005,'Surya','Assoc.Prof',50000,8328130161,'klu@kluniversity.in','Mumbai','CSE');
```

```
select *from faculty_Log;
```

The screenshot shows a SQL IDE window titled 'practical-10.1' and 'practical-10'. The SQL editor contains the following code:

```

9  begin
10      insert into faculty_Log(FID,FNAME,Designation,Salary,FMOBILE,FMAIL,FADD,Branch,changed_at,operation)
11      values(NEW.FID,NEW.FNAME,NEW.Designation,NEW.Salary,NEW.FMOBILE,NEW.FMAIL,NEW.FADD,NEW.BRANCH,current_timestamp,'INS');
12  end $$
13  delimiter ;
14  insert into faculty values(5005,'Surya','Assoc.Prof',50000,8328130161,'klu@kluniversity.in','Mumbai','CSE');
15
16  select *from faculty_Log;
17

```

The 'Result Grid' shows the output of the query. It has columns: operation_id, FID, FNAME, Designation, Salary, FMOBILE, FMAIL, FADD, Branch, changed_at, operation. The first row shows the result of the insertion:

| operation_id | FID | FNAME | Designation | Salary | FMOBILE | FMAIL | FADD | Branch | changed_at | operation |
|--------------|------|-------|-------------|--------|------------|---------------------|--------|--------|---------------------|-----------|
| 1 | 5005 | Surya | Assoc.Prof | 50000 | 8328130161 | klu@kluniversity.in | Mumbai | CSE | 2020-11-05 09:48:15 | INS |

The IDE also shows a 'Form Editor' and 'Apply'/'Revert' buttons at the bottom.

```

delimiter $$
create trigger trig_faulty_delete after delete on faculty
for each row
begin
    insert into
    faculty_Log(FID,FNAME,Designation,Salary,FMOBILE,FMAIL,FADD,Branch,changed_at,operation)
    values(OLD.FID,OLD.FNAME,OLD.Designation,OLD.Salary,OLD.FMOBILE,OLD.FMAIL,OLD.FADD,OLD.BRANCH,current_timestamp,'DEL');
end $$
delimiter ;

```

```

delete from faculty where FID = 5005;
select *from faculty_Log;

```

The screenshot shows a database management tool interface. The top pane displays SQL code for creating a trigger and deleting a record. The bottom pane shows the 'Result Grid' with two rows of data from the 'faculty_Log' table.

| operation_id | FID | FNAME | Designation | Salary | FMOBILE | FMAIL | FADD | Branch | changed_at | operation |
|--------------|------|-------|-------------|--------|------------|---------------------|--------|--------|---------------------|-----------|
| 1 | 5005 | Surya | Assoc.Prof | 50000 | 8328130161 | klu@kluniversity.in | Mumbai | CSE | 2020-11-05 09:48:15 | INS |
| 2 | 5005 | Surya | Assoc.Prof | 50000 | 8328130161 | klu@kluniversity.in | Mumbai | CSE | 2020-11-05 09:53:05 | DEL |

- Write a Program to create a row level trigger that would fire for INSERT or UPDATE or DELETE operations performed on the LIBRARYBooks table. The program has to print the status of the DML operations(Like Insert, Update and delete) performed

```

create table library_Log(operation_id int primary key auto_increment,ACCNO int,
updated_accno int default NULL,BTITLE varchar(30),updated_bttitle varchar(30) default
'No Updation',
AUTHOR varchar(30),updated_author varchar(30) default 'No Updation',PUBLISHER
varchar(25),
updated_publisher varchar(25) default 'No updation',EDITION int,updated_edition int
default null,
PRICE int,updated_price int default null,No_of_Copies int,updated_copies int default null,
changed_at DATETIME NOT NULL,operation varchar(20) NOT NULL,

```



```
CHECK(operation = 'Inserted' or operation = 'Deleted' or operation = 'Updated'));
```

```
delimiter $$
```

```
create trigger trig_library_insert after insert on library_Books
```

```
for each row
```

```
begin
```

```
    insert into
```

```
library_Log(ACCNO,BTITLE,AUTHOR,PUBLISHER,EDITION,PRICE,No_of_Copies,changed_at,operation)
```

```
values(new.ACCNO,NEW.BTITLE,NEW.AUTHOR,NEW.PUBLISHER,NEW.EDITION,NEW.PRICE,NEW.No_of_Copies,current_timestamp,'Inserted');
```

```
end $$
```

```
delimiter ;
```

```
insert into library_Books values(105,'MSWD','Radha','Krishna',10,1000,50);
```

```
select *from library_Log;
```

The screenshot shows a database management tool interface. The top toolbar includes icons for file operations, search, and execution. The main area displays SQL code with line numbers 58 to 66. The code defines a trigger and performs an insert and select operation. Below the code, the 'Result Grid' is visible, showing the results of the SQL execution. The grid has columns for operation_id, ACCNO, updated_accno, BTITLE, updated_btitle, AUTHOR, updated_author, PUBLISHER, updated_publisher, EDITION, and updated_edition. The first row shows the results of the insert operation.

| operation_id | ACCNO | updated_accno | BTITLE | updated_btitle | AUTHOR | updated_author | PUBLISHER | updated_publisher | EDITION | updated_edition |
|--------------|-------|---------------|--------|----------------|--------|----------------|-----------|-------------------|---------|-----------------|
| 1 | 105 | NULL | MSWD | No Updation | Radha | No Updation | Krishna | No updation | 10 | NULL |

```
delimiter $$
```

```
create trigger trig_library_update after update on library_Books
```

```
for each row
```

```
begin
```

```
    insert into library_Log(ACCNO,updated_accno,BTITLE, updated_btitle, AUTHOR, updated_author,PUBLISHER,updated_publisher,EDITION,updated_edition,PRICE,updated_price,No_of_Copies,updated_copies,changed_at,operation) values (old.ACCNO, new.ACCNO,old.BTITLE,new.BTITLE,old.AUTHOR,new.AUTHOR,old.PUBLISHER,new.PUBLISHER,old.EDITION,new.EDITION,old.PRICE,new.PRICE,old.No_of_Copies,new.No_of_Copies,current_timestamp,'Updated');
```

```
end $$
```

```
delimiter ;
```

update library_Books set BTITLE = 'OSD' where ACCNO = 105;
select *from library_Log;

The screenshot shows a SQL IDE window with a script editor and a result grid. The script editor contains the following SQL code:

```

81 begin
82     insert into library_Log(ACCNO,updated_accno,BTITLE,updated_btitle,AUTHOR,updated_author,PUBLISHER,u
83 end $$
84 delimiter ;
85
86 • update library_Books set BTITLE = 'OSD' where ACCNO = 105;
87
88 • select *from library_Log;
89

```

The result grid displays the output of the SQL script. It has columns: operation_id, ACCNO, updated_accno, BTITLE, updated_btitle, AUTHOR, updated_author, PUBLISHER, updated_publisher, and EDITOR. The data is as follows:

| operation_id | ACCNO | updated_accno | BTITLE | updated_btitle | AUTHOR | updated_author | PUBLISHER | updated_publisher | EDITOR |
|--------------|-------|---------------|--------|----------------|--------|----------------|-----------|-------------------|--------|
| 1 | 105 | NULL | MSWD | No Updation | Radha | No Updation | Krishna | No updation | 10 |
| 2 | 105 | 105 | MSWD | OSD | Radha | Radha | Krishna | Krishna | 10 |
| * | NULL | NULL | NULL | NULL | NULL | NULL | NULL | NULL | NULL |

3. Write a PL/SQL Program to calculate the tax of a faculty based on the below conditions using Functions.
 - a. If the salary of a faculty is between 0 and 30000 then tax should be 10%
 - b. If the salary of a faculty is between 30001 and 50000 then tax should be 15%
 - c. If the salary of a faculty is above 50001 then tax should be 25%

4. Write a PL/SQL Program to create a package that contains the following functions:
 - a. Function for computing Annual salary of a faculty

```

delimiter $$
create function annual_salary(Salary int) returns integer
deterministic
begin
    declare anu_sal int;
    set anu_sal=Salary*12;
    return anu_sal;
end $$
delimiter ;

```

```
select annual_salary(Salary) from faculty;
```

The screenshot shows a SQL IDE window titled 'practical-10.1' and 'practical-10'. The code editor contains the following SQL code:

```

110 begin
111     declare anu_sal int;
112     set anu_sal=Salary*12;
113     return anu_sal;
114 end $$
115 delimiter ;
116
117 • select annual_salary(Salary) from faculty;
118

```

Below the code editor, the 'Result Grid' is displayed, showing the output of the query:

| annual_salary(Salary) |
|-----------------------|
| 474000 |
| 954000 |
| 534000 |
| 414000 |

The IDE interface includes a toolbar at the top with various icons for file operations, a 'Limit to 1000 rows' dropdown, and a 'Filter Rows' input field. The bottom status bar indicates 'Result 8' and 'Read Only'.

b. Function to calculate the tax of a faculty based on the conditions in Q3 above.

delimiter \$\$

create function faculty_tax(Salary int) returns integer

deterministic

begin

declare tax int;

if Salary >= 0 and Salary < 30000 then

set tax = 10/100 * Salary;

elseif Salary >= 30001 and Salary < 50000 then

set tax = 15/100 * Salary;

elseif Salary >= 50001 then

set tax = 25/100 * Salary;

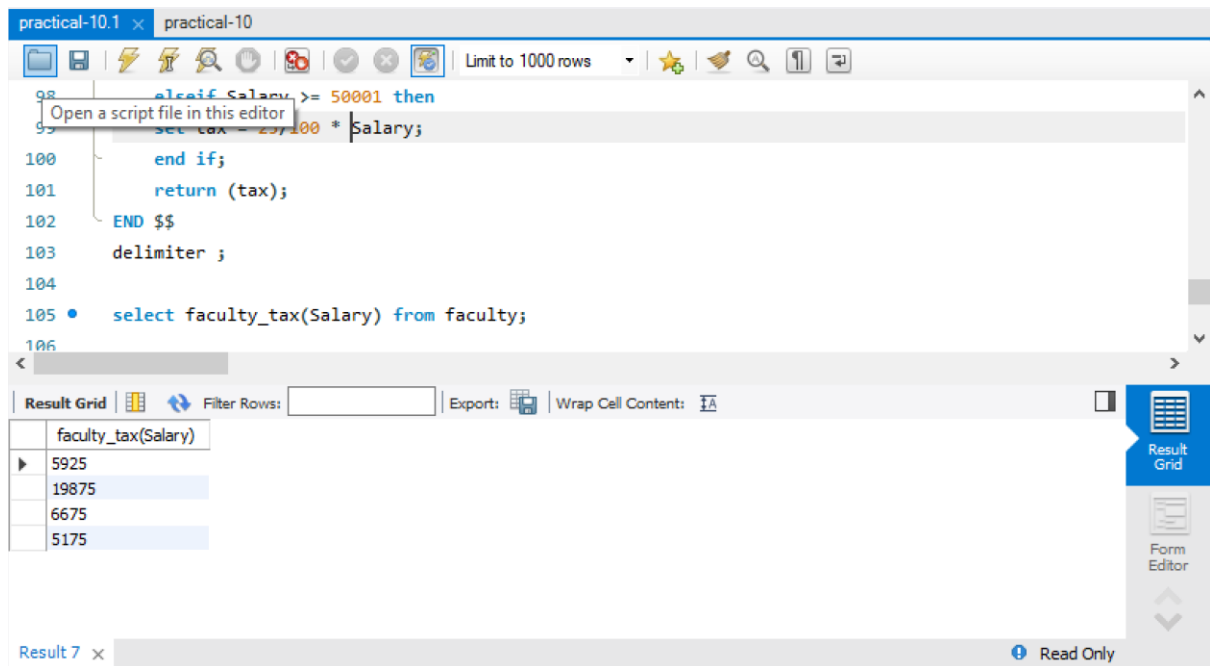
end if;

return (tax);

END \$\$

delimiter ;

select faculty_tax(Salary) from faculty;



The screenshot displays a SQL IDE interface with a script editor and a result grid. The script calculates a tax based on salary using a CASE statement. The result grid shows the output of the query.

```
98      elseif Salary >= 50001 then
99      set tax = 25,000 * Salary;
100    end if;
101    return (tax);
102  END $$
103  delimiter ;
104
105 • select faculty_tax(Salary) from faculty;
106
```

Result Grid

| faculty_tax(Salary) |
|---------------------|
| 5925 |
| 19875 |
| 6675 |
| 5175 |

Result 7 x Read Only

Java Database Connectivity with MySQL

To connect Java application with the MySQL database, we need to follow 5 following steps. In this example we are using MySQL as the database. So we need to know following informations for the mysql database:

1. **Driver class:** The driver class for the mysql database is **com.mysql.jdbc.Driver**.
2. **Connection URL:** The connection URL for the mysql database is **jdbc:mysql://localhost:3306/sonoo** where jdbc is the API, mysql is the database, localhost is the server name on which mysql is running, we may also use IP address, 3306 is the port number and sonoo is the database name. We may use any database, in such case, we need to replace the sonoo with our database name.
3. **Username:** The default username for the mysql database is **root**.
4. **Password:** It is the password given by the user at the time of installing the mysql database. In this example, we are going to use root as the password.

Let's first create a table in the mysql database, but before creating table, we need to create database first.

create database transport;

use transport;

create table emp(id int(10),name varchar(40),age int(3));

In this example, transport is the database name, root is the username and password both.

```
import java.sql.*;
class MysqlCon{
    public static void main(String args[]){
    try{
    Class.forName("com.mysql.jdbc.Driver");
    Connection con=DriverManager.getConnection(
    "jdbc:mysql://localhost:3306/sonoo","root","root");
    //here sonoo is database name, root is username and password
    Statement stmt=con.createStatement();
    ResultSet rs=stmt.executeQuery("select * from emp");
    while(rs.next())
    System.out.println(rs.getInt(1)+" "+rs.getString(2)+" "+rs.getString(3));
    con.close();
    }catch(Exception e){ System.out.println(e);}
    }
}
```

The above example will fetch all the records of emp table.

connect java application with the mysql database, **mysqlconnector.jar** file is required to be loaded.

[download the jar file mysql-connector.jar](#)

Two ways to load the jar file:

1. Paste the mysqlconnector.jar file in jre/lib/ext folder
 2. Set classpath
- 1) Paste the mysqlconnector.jar file in JRE/lib/ext folder:

Download the mysqlconnector.jar file. Go to jre/lib/ext folder and paste the jar file here.

2) Set classpath:

There are two ways to set the classpath:

- temporary
- permanent

How to set the temporary classpath

open command prompt and write:

C:>set classpath=c:\folder\mysql-connector-java-5.0.8-bin.jar;.

How to set the permanent classpath

Go to environment variable then click on new tab. In variable name write **classpath** and in variable value paste the path to the mysqlconnector.jar file by appending mysqlconnector.jar;. as C:\folder\mysql-connector-java-5.0.8-bin.jar;.

POST-LAB

Queries using aggregate functions(COUNT,AVG,MIN,MAX,SUM),Group by, Order by, Having.

| E_id | E_name | Age | Salary |
|------|---------|-----|--------|
| 101 | AREEB | 22 | 9000 |
| 102 | DHEERAJ | 29 | 8000 |
| 103 | RAHUL | 34 | 6000 |
| 104 | MANOJ | 44 | 10000 |
| 105 | THARUN | 35 | 8000 |
| 106 | ANAND | 27 | 7000 |
| 107 | SAI | 29 | 8000 |

(i) Create Employee table containing all Records.

```

1  create schema practical10;
2  create table practical10.Employee_table(
3      E_id int not null,
4      E_name varchar(20) not null,
5      Age int not null,
6      Salary int not null,
7      primary key (E_id)
8  );
9  insert into practical10.Employee_table
10 values(101,'Areeb',22,9000),
11        (102,'Dheeraj',29,8000),
12        (103,'Rahul',34,6000),
13        (104,'Manoj',44,10000),
14        (105,'Tharun',35,8000),
15        (106,'Anand',27,7000),
16        (107,'Sai',29,8000);
  
```

(ii) Count number of employee names from employee table

```

1  select count(*) from practical10.Employee_table;
  
```

Result Grid

| count(*) |
|----------|
| 7 |

Result 1 x Read Only

(iii) Find the Maximum age from employee table.

SQL File 2

```
1 • select max(age) from practical10.Employee_table;
```

Result Grid

| max(age) |
|----------|
| 44 |

Result 2 x Read Only

(iv) Find the Minimum age from employee table

SQL File 2

```
1 • select min(age) from practical10.Employee_table;
```

Result Grid

| min(age) |
|----------|
| 22 |

Result 3 x Read Only

(v) Display the Sum of age employee table

SQL File 2

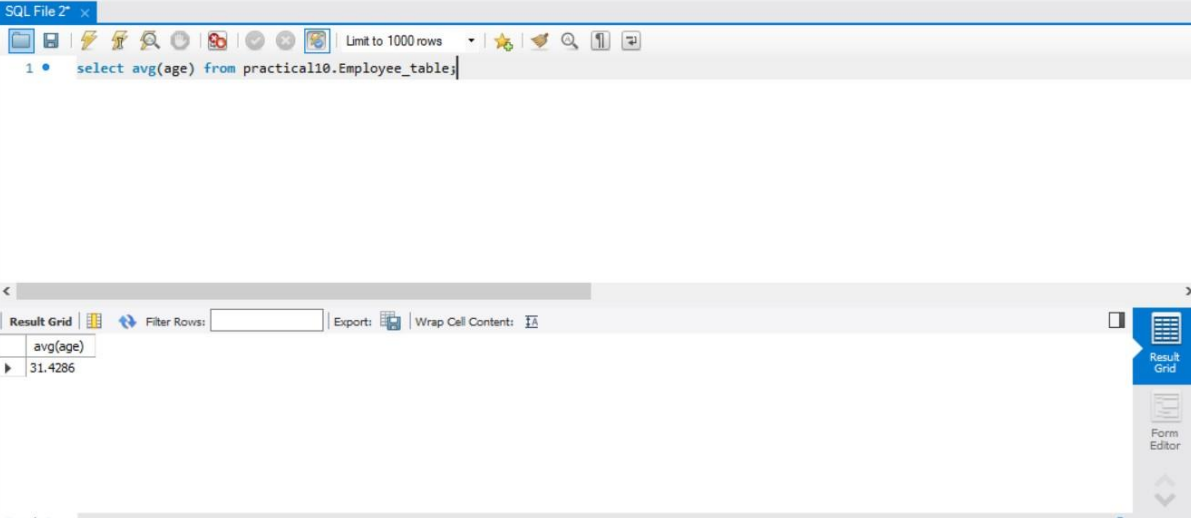
```
1 • select sum(age) from practical10.Employee_table;
```

Result Grid

| sum(age) |
|----------|
| 220 |

Result 4 x Read Only

(vi) Display the Average of age from Employee table.



The screenshot shows a SQL IDE window titled "SQL File 2". The query editor contains the following SQL statement:

```
1 • select avg(age) from practical10.Employee_table;
```

The results pane below the query editor displays the output of the query. It shows a single row with the column name "avg(age)" and the value "31.4286".

| avg(age) |
|----------|
| 31.4286 |

The interface includes a toolbar with various icons, a "Limit to 1000 rows" dropdown, and a "Filter Rows" input field. The results pane also has an "Export" button and a "Wrap Cell Content" checkbox. The status bar at the bottom indicates "Result 5" and "Read Only".