Subject:-DBMS Lab

Class :-TE **Academic Year 2021-22**

Name of Faculty:-S.R.Nalamwar and M.M Phadtare

Experiment: A2

Aim: a. Design and Develop SQL DDL statements which demonstrate the use of SQL objects such as Table, View, Index, Sequence, Synonym, different constraints etc.
b. Write at least 10 SQL queries on the suitable database application using SQL DML statements.

Code and output:

Create Database :

create database college;

Query OK, 1 row affected (0.40 sec)

mysql> use college Database changed

Create Table :

SQL> create table student(roll_no int primary key,name varchar(15) not null,address varchar(20),email varchar(25));

Table created

Alter Table :

SQL> alter table student add mob_no int;

Table altered.

SQL> insert into student values(1001,'Snehal','Pune','snehal@gmail.com',3765371);

1 row created.

SQL> insert into student
values(1001,'Akshara','Pune','akshara@gmail.com',3765371);

1 row created.

SQL> insert into student
values(1002,'Akshara','Pune','akshara@gmail.com',3765371);

1 row created.

SQL> insert into student values(1003,'Tanvi','Pune','tanvi@gmail.com',84375371);

1 row created.

Truncate Table :

SQL> truncate table student;

Table truncated.

SQL> alter table student drop column address;

Table altered.

SQL> select * from student;

no rows selected

SQL> desc student

Name	Null?	Type
ROLL_NO	NOT NULL	NUMBER(38)
NAME	NOT NULL	VARCHAR2(15)
EMAIL		VARCHAR2(25)
MOB_NO		NUMBER(38)

Drop Table :

SQL> drop table student;

Table dropped.

Drop Database :

SQL> drop database college;

Query OK, 0 rows affected (0.26 sec)

Experiment: A3

Aim: Write at least 10 SQL queries for suitable database application using SQL DML statements.

Code and output:

```
SQL> CREATE TABLE employees (
  2 e_no INT PRIMARY KEY,
  3 e_name VARCHAR (20),
  4 address VARCHAR (20),
  5 basic_salary FLOAT,
  6 job_status VARCHAR (30)
  7 );

Table created.

SQL> INSERT INTO employees
VALUES(00005,'A.Ghosh','Kharagpur',04200.00,'Professor');
1 row created.

SQL> INSERT INTO employees
VALUES(00010,'G.Bhakta','Midnapur',02700.0,'Research fellow');
1 row created.

SQL> INSERT INTO employees
VALUES(00001,'P.Sen','Calcutta',05000.00,'Professor');
1 row created.

SQL> INSERT INTO employees
VALUES(00007,'D.Kundu','Kharagpur',08000.00,'Director');
1 row created.

SQL> INSERT INTO employees
VALUES(00003,'S.Prasad','Calcutta',05300.00,'Professor');
1 row created.
```

```
SQL> INSERT INTO employees
VALUES(00004,'P.Gupta','Midnapur',03000.0,'Research fellow');

1 row created.

SQL> INSERT INTO employees
VALUES(00011,'G.S.Bose','Kharagpur',02000.00,'Office Ass');

1 row created.

SQL> INSERT INTO employees
VALUES(00012,'L.Sen','Calcutta',01500.00,'Office Ass');

1 row created.

SQL> INSERT INTO employees
VALUES(00013,'K.Singh','Midnapur',01700.00,'Office Ass');

1 row created.

SQL> SELECT * FROM employees;

     E_NO E_NAME            ADDRESS           BASIC_SALARY
.....-.....
JOB_STATUS
--------------------
        5 A.Ghosh           Kharagpur              4200
Professor

     10 G.Bhakta           Midnapur               2700
Research fellow

        1 P.Sen            Calcutta               5000
Professor

     E_NO E_NAME            ADDRESS           BASIC_SALARY
```

……-……

JOB_STATUS

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

7 D.Kundu Kharagpur 8000

Director

3 S.Prasad Calcutta 5300

Professor

4 P.Gupta Midnapur 3000

Research fellow

E_NO E_NAME ADDRESS BASIC_SALARY

……-……

JOB_STATUS

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

11 G.S.Bose Kharagpur 2000

Office Ass

12 L.Sen Calcutta 1500

Office Ass

13 K.Singh Midnapur 1700

Office Ass

9 rows selected.

Q.1 Find the names of the employees whose basic salary is greater than 4000.00.

SQL> SELECT * FROM employees WHERE basic_salary>4000.0;

```
    E_NO E_NAME          ADDRESS          BASIC_SALARY
.............-.....................................
JOB_STATUS
--------------------
       5 A.Ghosh          Kharagpur              4200
Professor

       1 P.Sen            Calcutta               5000
Professor

       7 D.Kundu          Kharagpur              8000
Director


    E_NO E_NAME          ADDRESS          BASIC_SALARY
.............-.....................................
JOB_STATUS
--------------------
       3 S.Prasad         Calcutta               5300
Professor
```

Q.2 Find the names of the employees whose basic salary is greater than 40000.00 with job status professor.

SQL> SELECT * FROM employees WHERE basic_salary>4000.0 AND job_status='Professor';

```
    E_NO E_NAME          ADDRESS          BASIC_SALARY
.............-.....................................
JOB_STATUS
--------------------
       5 A.Ghosh          Kharagpur              4200
Professor

       1 P.Sen            Calcutta               5000
Professor

       3 S.Prasad         Calcutta               5300
```

Professor

Q.3 Find the name and employee number of the employee arranging employee number in descending order.

SQL> SELECT * FROM employees ORDER BY e_no DESC;

```
     E_NO E_NAME            ADDRESS           BASIC_SALARY
.............-.....................................................
JOB_STATUS
- - - - - - - - - - - - - - - - - - -
       13 K.Singh           Midnapur                1700
Office Ass

       12 L.Sen             Calcutta              1500
Office Ass

       11 G.S.Bose          Kharagpur               2000
Office Ass


     E_NO E_NAME            ADDRESS           BASIC_SALARY
.............-.....................................................
JOB_STATUS
- - - - - - - - - - - - - - - - - - -
       10 G.Bhakta          Midnapur                2700
Research fellow

        7 D.Kundu           Kharagpur               8000
Director

        5 A.Ghosh           Kharagpur               4200
Professor


     E_NO E_NAME            ADDRESS           BASIC_SALARY
.............-.....................................................
JOB_STATUS
- - - - - - - - - - - - - - - - - - -
```

4 P.Gupta Midnapur 3000
Research fellow

3 S.Prasad Calcutta 5300
Professor

1 P.Sen Calcutta 5000
Professor

9 rows selected.

Q. 4 Display names of employees whose job status begin with "Research"

SQL> SELECT * FROM employees WHERE job_status='Research fellow';

E_NO E_NAME ADDRESS BASIC_SALARY
JOB_STATUS
---

10 G.Bhakta Midnapur 2700
Research fellow

4 P.Gupta Midnapur 3000
Research fellow

Q.5 Display names of those employees whose employee is odd.

SQL> SELECT *FROM employees WHERE mod(e_no,2)=1;

E_NO E_NAME ADDRESS BASIC_SALARY
JOB_STATUS
---

5 A.Ghosh Kharagpur 4200
Professor

1 P.Sen Calcutta 5000
Professor

```
      7 D.Kundu           Kharagpur               8000
Director


    E_NO E_NAME           ADDRESS           BASIC_SALARY
-------- ----------------------------------------------------------
JOB_STATUS
--------------------
      3 S.Prasad          Calcutta                5300
Professor

     11 G.S.Bose          Kharagpur               2000
Office Ass

     13 K.Singh           Midnapur                1700
Office Ass


6 rows selected.
```

Q.6 Compute total basic salary of all the employees.

```
SQL> SELECT SUM(basic_salary) FROM employees;

SUM(BASIC_SALARY)
-----------
        33400
```

Q.7 Compute total basic salary of office Asst.

```
SQL> SELECT SUM(basic_salary) FROM employees WHERE job_status='Office Ass';

SUM(BASIC_SALARY)
-----------
        5200
```

Q.8 Find the number of employees who are having the same job status.

```
SQL> SELECT COUNT(e_no),job_status FROM employees GROUP BY job_status;

COUNT(E_NO) JOB_STATUS
------- ------------------
```

2 Research fellow
3 Professor
3 Office Ass
1 Director

Q.9 Determine average salary for the professors and research fellows.

```
SQL> SELECT AVG(basic_salary) FROM employees WHERE job_status
IN('Professor','Research fellow');

AVG(BASIC_SALARY)
-----------
        4040
```

Q.10 Find names of the employees who are working in the same department with P.Gupta

```
SQL> SELECT e_name FROM employees WHERE job_status=(SELECT job_status FROM
employees WHERE e_name='P.Gupta');

E_NAME
-------------
G.Bhakta
P.Gupta
```

Experiment: A4 (Part 1)

Aim: Consider Tables:

1. Borrower (Roll_no, Name, DateofIssue, NameofBook, Status)

2. Fine (Roll_no, Date, Amt)

Accept Roll_no & NameofBook from user.

- Check the number of days (from date of issue),
- If days are between 15 to 30 then fine amount will be Rs 5per day.
- If no. of days>30, per day fine will be Rs 50 per day & for days less than 30, Rs. 5 per day.
- After submitting the book, status will change from I to R.
- If condition of fine is true, then details will be stored into fine table.
- Also handles the exception by named exception handler or user define exception handler.

Code and output:

Table Creation:

create table borrower_detail(Rollin int, Name varchar(15), DateofIssue date, NameofBook varchar(20), Status varchar(10));

Table created.

SQL> create table fine_detail(Roll_no int,FineDate date,Amt int);

Table created.

PL/SQL Code:

SQL> edit fine

declare

roll_no number;

name_of_book varchar(10);

diff_date number;

p_date date;

```
i_date date;
fine_amt real;
begin
roll_no:=&roll_no;
name_of_book:='&name_of_book';
select dateofissue into i_date from Borrower_detail where rollin=roll_no; select
sysdate - i_date into diff_date from dual; dbms_output.put_line(diff_date);
if (diff_date>15 AND diff_date<30) then
fine_amt:=diff_date*5;
else
fine_amt:=diff_date*50;
end if;
dbms_output.put_line('fine amt='||fine_amt);
update Borrower_detail set status='r' where rollin=roll_no;
if (fine_amt>0) then

 insert into Fine_detail values(roll_no,i_date,fine_amt); end if;
end;
/
```

Data Insertion:

```
SQL> insert into borrower_detail values(1001,'Vasuki','03-NOV-20','CPP','I');
1 row created.
SQL> insert into borrower_detail values(1002,'Ria','03-OCT-20','SEPM','I');
1 row created.
SQL> insert into borrower_detail values(1003,'Madhuri','01-DEC-20','DBMS','I');
1 row created.
```

Output :

```
SQL> @fine
Enter value for roll_no: 1001
old 9: roll_no:=&roll_no;
new 9: roll_no:=1001;
Enter value for name_of_book: CPP

 old 10: name_of_book:='&name_of_book'; new 10: name_of_book:='CPP';
PL/SQL procedure successfully completed.
SQL> select * from fine_detail;

ROLL_NO FINEDATE AMT
.....-.....-.....
1001 03-NOV-20 148
SQL> @fine
Enter value for roll_no: 1002
old 9: roll_no:=&roll_no;
new 9: roll_no:=1002;
Enter value for name_of_book: SEPM
old 10: name_of_book:='&name_of_book'; new 10: name_of_book:='SEPM';
PL/SQL procedure successfully completed.
SQL> select * from fine_detail;


ROLL_NO FINEDATE AMT
.....-.....-.....
1001 03-NOV-20 148 1002 03-OCT-20 3025

 SQL> select * from borrower_detail;
```

ROLLIN	NAME	DATEOFISS	NAMEOFBOOK	STATUS
1001	Vasuki	03-NOV-20	CPP	r
1002	Ria	03-OCT-20	SEPM	r
1003	Madhuri	01-DEC-20	DBMS	l

Experiment: A4 (Part 2)

Aim: Write a PL/SQL code block to calculate the area of a circle for a value of radius varying from 5 to 9. Store the radius and the corresponding values of calculated area in an empty table named areas, consisting of two columns, radius and area.

Code and output:

PL/SQL Code:

```
declare
r1 number:=&r1;
a1 number(14,2);
value_out_of_bounds_exception exception;
begin
if r1>=5 and r1<=9 then
  a1 := 3.14*r1*r1;
  insert into areass values(r1,a1);
else
  raise value_out_of_bounds_exception;
end if;

exception
when value_out_of_bounds_exception then
    dbms_output.put_line('radius must be between 5 and 9.');
end;
/
```

Output:

```
SQL> set serverout on
```

```
SQL> @ circle assignment
Enter value for r1: 0
old 2: r1 number:=&r1;
new  2: r1 number:=0;
radius must be between 5 and 9.

PL/SQL procedure successfully completed.

SQL> @ circle assignment
Enter value for r1: 6
old 2: r1 number:=&r1;
new  2: r1 number:=6;

PL/SQL procedure successfully completed.
```

Experiment: A5

Aim: Write a Stored Procedure namely proc_Grade for the categorization of student. If marks scored by students in examination is <=1500 and marks>=990 then student will be placed in distinction category if marks scored are between 989 and 900 category is first class, if marks 899 and 825 category is Higher Second Class

Write a PL/SQL block to use procedure created with above requirement.
Stud_Marks(name, total_marks) Result(Roll,Name, Class)

Code and output:

```
CREATE TABLE stud_marks(
name VARCHAR(20),
total_marks NUMBER);

INSERT INTO stud_marks
VALUES('raj',89);
INSERT INTO stud_marks
VALUES('ram',94);
INSERT INTO stud_marks
VALUES('ramesh',95);

CREATE TABLE results(
roll_no NUMBER,
name VARCHAR(20),
class VARCHAR(20)
);

INSERT INTO stud_marks
```

```
VALUES(1,'raj','TE');
INSERT INTO stud_marks
VALUES(2,'ram','TE');
INSERT INTO stud_marks
VALUES(3,'ramesh','TE');
```

PL/SQL Code:

```
create or replace procedure proc_grade is marks number;
s_name stud_marks.name%type := &name;
s_marks stud_marks.total_marks%type;

begin
SELECT total_marks INTO s_marks FROM stud_marks WHERE name := s_name;

if((s_marks<1500) and (s_marks>=990))then
dbms_output.put_line('You have got distinction');
elsif((s_marks>=989) and (s_marks<=900))then
dbms_output.put_line('You have got firstclass');
elsif((s_marks>=899) and (s_marks<=825))then
dbms_output.put_line('You have got secondclass');
else
dbms_output.put_line('Indeterminate');
end if;
end;
/
```

Output:

SQL> exec proc_grade(94,90,96);

You have got Distinction with percent
93.333333333333333333333333333333333333

PL/SQL procedure successfully completed.

SQL> exec proc_grade (74,70,76);

You have got First class with percent
73.333333333333333333333333333333333333

PL/SQL procedure successfully completed.

Experiment: A6

Aim: Write a PL/SQL block of code using parameterized Cursor that will merge the data available in the newly created table N_RollCall with the data available in the table O_RollCall. If the data in the first table already exist in the second table then that data should be skipped.

Code and Output:

```
SQL> create table new_class(roll int,name varchar(20));

Table created.

SQL> create table old_class(roll int,name varchar(20));

Table created.

 SQL> edit c_class

declare cursor c_class is select * from old_class;

cursor c_chk(str_name varchar) is select roll from new_class where name=str_name;

str_roll new_class.roll%type;

str_name new_class.name%type;

v varchar(10);

begin open c_class;

 loop fetch c_class into str_roll,str_name;

 exit when c_class%notfound;

open c_chk(str_name);

fetch c_chk into v; if c_chk%found then dbms_output.put_line('Name '||str_name||'
is exist');

 else dbms_output.put_line('Name '||str_name||' does not exist. Inserting in new
class table'); insert into new_class values(str_roll,str_name);

end if;

close c_chk;
```

```
end loop;
close c_class;
end;
 /
 SQL> select * from old_class;


ROLL NAME
---------- ----------
101 AAA
102 BBB
103 CCC
104 DDD
 SQL> select * from new_class;
ROLL NAME
 ----------
101 AAA 102 BBB
 SQL> @c_class Name AAA is exist
Name BBB is exist
Name CCC does not exist.
 Inserting in new class table Name DDD does not exist.
 Inserting in new class table PL/SQL procedure successfully completed.
```

Experiment: A7

Aim: Write a database trigger on Library table. The System should keep track of the records that are being updated or deleted. The old value of updated or deleted records shouldbe added in Library_Audit table.

Code and Output:

```
SQL> set serverout on
SQL> select*from Library;

  ROLL_NO ISSUE_DAT BOOK_NAME
--------- --------- ------------------------------
       11 13-OCT-21 Wings of Fire
       21 23-SEP-21 India 2020
       34 05-MAR-21 Ignited Minds
       40 10-JUL-21 The Discovery of India


SQL> edit trigger_for_book

SQL> edit  Audit_book

SQL> edit library_audit

SQL> @library_audit
Enter value for roll: 21
old   8: roll :=&roll;
new  8: roll :=21;
Enter value for bname: 'Deception Point'
old  9: bname :=&bname;
new 9: bname :='Deception Point';
Name of the new book:Deception Point
Name of the old book:India 2020

PL/SQL procedure successfully completed.

SQL> select*from Library_Audit_table;
```

```
  ROLL_NO ISSUE_DAT OLD_BOOK              NEW_BOOK
---------- --------- ------------------------------
        21 20-OCT-21 India 2020            Deception Point

SQL> select*from Library;

  ROLL_NO ISSUE_DAT BOOK_NAME
---------- --------- ------------------------------
        11 13-OCT-21 Wings of Fire
        21 23-SEP-21 Deception Point
        34 05-MAR-21 Ignited Minds
        40 10-JUL-21 The Da Vinci Code

SQL>
```

Trigger

```
Declare
roll number;
new_date date;
bname varchar(25);
new_book varchar(25);
old_book varchar(25);
begin
roll  :=&roll;
bname :=&bname;
select book_name into old_book from Library where Roll_no =roll;
update Library set book_name=bname where Roll_no =roll;
select book_name into new_book from Library where Roll_no =roll;
select sysdate into new_date from dual;
insert into Library_Audit_table values(roll, new_date, old_book, new_book);
```

```
end;
/
```

Experiment: A8

Aim: Write a program to implement MYSQL/Oracle database connectivity with any front end language to implement Database navigation operations (add, delete, edit etc).

Code:

Terminal :

Installing MySQL-connector-python :

1).pip install mysql-connector-python

2).pip freeze

>>>mysql-connector-python==8.0.27

Editor (importing the connector in file):

```
import mysql.connector as connector

class DBHelper:

   def _init_(self):

self.con=connector.connect(host='localhost',port='3306',user='root',password='######',database='trial')

      cur=self.con.cursor()


      query='create table if not exists users(userId int primary key auto_increment, userName varchar(200), is_superuser tinyint(1) default 0, firstName varchar(200), lastName varchar(200), is_staff tinyint(1) default 0, password varchar(200) NOT NULL);)'

      cur.execute(query)

     print("Created")

     helper = DBHelper()
```

Experiment: B1

Aim: Design and Develop MongoDB Queries using CRUD operations. (Use CRUD operations, SAVE method, logical operators etc)

Code and output:

> use mydb

switched to db mydb

Create:

> db.createCollection ("Student");
{ "ok": 1 }

> db.Student.insert({roll_no:1, name: 'atharva', branch: 'computer' });

WriteResult({ "nInserted": 1 })

Read:

>db.Student.find().pretty();

"_id" : ObjectId("619c880144d046e368d58709"),
"roll_no": 1,
"name": "atharva",
"branch": "computer"

Update:

>db.Student.update({roll_no:1},{$set: {brach: 'civil"}});

WriteResult({ "nMatched": 1, "nupserted": 0, "nModified": 1 })

>db.Student.find();

{"_id" : ObjectId("619c880144d046e368d58709"), "roll_no": 1, "name": "atharva", "branch": "computer", "brach": "civil" }

Delete:

```
>db.Student.remove({roll_no:1});

WriteResult({ "nRemoved" : 1 })
```

Experiment: B2

Aim: Design and Develop MongoDB Queries using aggregation and indexing with suitable example using MongoDB.

Code and output:

Create the collection Books having the following fields TITLE, DESCRIPTION, BY, URL, TAGS AND LIKES. Implement the following Aggregation and Indexing Queries

1. Find the number of books published by kanetkar.

2. Find books which have minimum likes and maximum likes published by kanetkar.

3. Find the average number of likes of the books published by kanetkar.

4. Find the first and last book published by kanetkar.

5. Create an index on author name. Display the books published by john and check if it uses the index which we have created

```
/**********************************************************************
**************************************************/
```

FIND THE NUMBER OF BOOKS PUBLISHED BY kanetkar

```
>
db.BOOKS.aggregate([{$match:{by:"kanetkar"}},{$group:{_id:"$by",no_of_documents:{$sum:1}}}])
{ "_id" : "JOHN", "no_of_documents" : 2 }
```

```
/**********************************************************************
**************************************************/
```

FIND BOOKS WHICH HAVE MINIMUM LIKES AND MAXIMUM LIKES PUBLISHED BY JOHN

```
>
db.BOOKS.aggregate([{$match:{by:"kanetkar"}},{$group:{_id:"$by",minimum_likes:{$min:"$likes"}}}])
```

{ "_id" : "JOHN", "minimum_likes" : 50 }

>

db.BOOKS.aggregate([{$match:{by:"kanetkar"}},{$group:{_id:"$by",minimum_lik
es:{$max:"$likes"}}}])

{ "_id" : "JOHN", "minimum_likes" : 100 }

/**
**/

FIND THE AVERAGE NUMBER OF LIKES OF THE BOOKS PUBLISHED BY kanetkar

>

db.BOOKS.aggregate([{$match:{by:"kanetkar"}},{$group:{_id:"$by",avg_likes:{$a
vg:"$likes"}}}])

{ "_id" : "JOHN", "avg_likes" : 75 }

/**
**/

FIND THE FIRST AND LAST BOOK PUBLISHED BY JOHN

db.BOOKS.aggregate([{$match:{by:"kanetkar"}},{$group:{_id:"$by",first_book:{$f
irst:"$title"}}}])

{ "_id" : "kanetkar", "first_book" : "C OVERVIEW" }

>

db.BOOKS.aggregate([{$match:{by:"kanetkar"}},{$group:{_id:"$by",last_book:{$l
ast:"$title"}}}])

{ "_id" : "kanetkar", "last_book" : "Pointer in C" }

/**
**/

CREATE AN INDEX ON AUTHOR NAME

>

db.BOOKS.ensureIndex({"by":1})

{

"createdCollectionAutomatically" : false,

"numIndexesBefore" : 1, "numIndexesAfter" : 2,

```
"ok" : 1
}
>
db.BOOKS.createIndex({"by":1})
 {
 "createdCollectionAutomatically" : false,
"numIndexesBefore" : 1,
 "numIndexesAfter" : 2,
"ok" : 1
 }
```

Experiment: B3

Aim: Implement Map reduces operation with suitable example using MongoDB.

Code and output:

Create the following schema

Order(id,amount ,status)

Cus id	Amount	Status
A1	400	P
B1	300	D
A1	200	F
C1	200	F
B1	700	P
B1	800	P

Status: P=”Pending”, D= “Delivered”, F= “Failed”

Implement the following using Map Reduce function

1. Find the sum of amount of each customer whose status is P
2. Find the average amount of each customer
3. Find the min amount of each customer
4. Find the max amount of each customer whose status is F

==========

==========

#CREATION OF COLLECTION : ORDER

>

```
db.createCollection("Order")
```

```
{ "ok" : 1 }
>
db.Order.insert({Customer_id:'A1',Amount:400,Status:'P'})
WriteResult({ "nInserted" : 1 })
>
db.Order.insert({Customer_id:'B1',Amount:300,Status:'D'})
WriteResult({ "nInserted" : 1 })
>
db.Order.insert({Customer_id:'A1',Amount:200,Status:'F'})
WriteResult({ "nInserted" : 1 })
>
db.Order.insert({Customer_id:'C1',Amount:200,Status:'F'})
WriteResult({ "nInserted" : 1 })
>
db.Order.insert({Customer_id:'B1',Amount:700,Status:'P'})
WriteResult({ "nInserted" : 1 })
>
db.Order.insert({Customer_id:'B1',Amount:800,Status:'P'})
WriteResult({ "nInserted" : 1 })
> db.Order.find().pretty()
{
"_id" : ObjectId("5ba1dbe5691da4e812906374"),
"Customer_id" : "A1",
"Amount" : 400,
"Status" : "P"
}
{
"_id" : ObjectId("5ba1dbf8691da4e812906375"),
```

"Customer_id" : "B1",

"Amount" : 300,

"Status" : "D"

}

{

"_id" : ObjectId("5ba1dc06691da4e812906376"),

"Customer_id" : "A1",

"Amount" : 200,

"Status" : "F"

}

{

"_id" : ObjectId("5ba1dc0e691da4e812906377"),

"Customer_id" : "C1",

"Amount" : 200,

"Status" : "F"

}

{

"_id" : ObjectId("5ba1dc1d691da4e812906378"),

"Customer_id" : "B1",

"Amount" : 700,

"Status" : "P"

}

{

"_id" : ObjectId("5ba1dc24691da4e812906379"),

"Customer_id" : "B1",

"Amount" : 800,

"Status" : "P"

}

1. Find the sum of amount of each customer whose status is P

```
>
var
mapfunction=function(){if(this.Status=='P')emit(this.Customer_id,this.Amount)};
> var reducefunction=function(key,values){return Array.sum(values);};
 > db.Order.mapReduce(mapfunction,reducefunction,{'out':'Order_total'}) {
"result" :
 "Order_total",
 "timeMillis" : 558,
 "counts" :
{
"input" : 6, "emit" : 3,
"reduce" : 1,
"output" : 2
 }
, "ok" : 1 }
 > > db.Order_total.find() { "_id" : "A1", "value" : 400 }
 { "_id" : "B1", "value" : 1500 }
```

---

2. Find the average amount of each customer

```
> var
mapfunction=function()
{
if(this.Customer_id=='A1')emit(this.Customer_id,this.Amo unt)};
 > var
reducefunction=function(key,values){return Array.avg(values);
};
```

```
> db.Order.mapReduce(mapfunction,reducefunction,{'out':'Order_average_A1'})
 {
"result" :
"Order_average_A1",
"timeMillis" : 379, "counts" :
 {
  "input" : 6,
"emit" : 2,
"reduce" : 1,
"output" : 1 },
  "ok" : 1 }
> db.Order_average_A1.find()
{ "_id" : "A1", "value" : 300 }
.....
> var
 mapfunction=function()
{
if(this.Customer_id=='B1')emit(this.Customer_id,this.Amo unt)};
> var
reducefunction=function(key,values){return Array.avg(values);};
> db.Order.mapReduce(mapfunction,reducefunction,
{'out':'Order_average_B1'})
 { "result" :
  "Order_average_B1",
"timeMillis" : 443,
"counts" : { "input" : 6, "emit" : 3, "reduce" : 1, "output" : 1 },
"ok" : 1
 }
```

```
> db.Order_average_B1.find() { "_id" : "B1", "value" : 600 }
>
 ..........................................................................
> var
mapfunction=function()
{
if(this.Customer_id=='C1')emit(this.Customer_id,this.Amo unt)};
> var
reducefunction=function(key,values){return Array.avg(values);};
> db.Order.mapReduce(mapfunction,reducefunction,
{
'out':'Order_average_C1'})
 {
"result" : "Order_average_C1",
"timeMillis" : 413,
"counts" :
{
"input" : 6, "emit" : 1, "reduce" : 0, "output" : 1 },
 "ok" : 1 }
 > db.Order_average_C1.find()
 { "_id" : "C1", "value" : 200 }
 ----------------------------------------------------------
> var
 mapfunction=function(){emit(this.Customer_id,this.Amount)};
> var
 reducefunction=function(key,values){return Array.avg(values);};
> db.Order.mapReduce(mapfunction,reducefunction,
{'out':'Order_average'})
```

```
{
"result" :
"Order_average",
"timeMillis" : 422,
"counts" :
{
 "input" : 6, "emit" : 6, "reduce" : 2, "output" : 3
},
"ok" : 1 }
 > db.Order_average.find() { "_id" : "A1", "value" : 300 }
{ "_id" : "B1", "value" : 600 }
{ "_id" : "C1", "value" : 200 }
```

---

3. Find the min amount of each customer

```
> var
 mapfunction=function()
{
emit(this.Customer_id,this.Amount)};
 > var
reducefunction=function(key,values)
{
return Math.min.apply(Math,values);};
 > db.Order.mapReduce(mapfunction,reducefunction,
{'out':'Order_minimum'})
 {
 "result" :
"Order_minimum",
```

"timeMillis" : 458,

"counts" :

{

"input" : 6, "emit" : 6, "reduce" : 2, "output" : 3 }, "ok" : 1 }

> db.Order_minimum.find() { "_id" : "A1", "value" : 200 }

{ "_id" : "B1", "value" : 300 }

{ "_id" : "C1", "value" : 200 } >

---

4. Find the max amount of each customer whose status is F

> var

mapfunction=function()

{

if(this.Status=='F')emit(this.Customer_id,this.Amount)};

> var

reducefunction=function(key,values)

{

return Math.max.apply(Math,values);};

> db.Order.mapReduce(mapfunction,reducefunction,

{'out':'Order_Maximum_F'})

{

"result" :

"Order_Maximum_F",

"timeMillis" : 427,

"counts" :

{

"input" : 6, "emit" : 2, "reduce" : 0, "output" : 2 }, "ok" : 1 }

> db.Order_Maximum_F.find()

```
{
 "_id" : "A1", "value" : 200 }
{ "_id" : "C1", "value" : 200 }
.....
> var
mapfunction=function()
{
if(this.Status=='F')emit(this.Customer_id,this.Amount)};
> var
reducefunction=function(key,values)
{
return Math.max.apply(Math,values);};
> db.Order.mapReduce(mapfunction,reducefunction,{'out':'Order_Maximum_F'})
{
 "result" : "Order_Maximum_F",
 "timeMillis" : 419,
"counts" :
{
"input" : 8, "emit" : 4, "reduce" : 1, "output" : 3
},
 "ok" : 1
}
> db.Order_Maximum_F.find()
 { "_id" : "A1", "value" : 800 } { "_id" : "B1", "value" : 800 } { "_id" : "C1", "value" : 200
} >
```

Experiment: B4

Aim: Write a program to implement MongoDB database connectivity with any front end language to implement Database navigation operations (add, delete, edit etc)

Code and output:

Create employee array objects containing employee id,name,designation,salary using json and write a program in python to read and display employee information.

JSON.py

```
import json with open('dbms.json') as f:
data = json.load(f)
def main():
def displayAll():
for i in range(len(data["Employee"])):
 for j,k in data["Employee"][i].items():
print(j," : ",k)
print('\n')
return('Total Number of Records are {}'.format(len(data["Employee"])))
def displayById():
id_ = int(input('Enter the id'))
flag = 0
for i in range(len(data["Employee"])):
if data["Employee"][i]["id"] == id_:
flag = 1
for j,k in data["Employee"][i].items():
 print(j," : ",k)
print('\n')
```

```
return('Record Not Present \n\n ')
if flag == 0
else ('Record Present')
def insertNewRecord():
global data newRecord = {}
 listOfIds = [i["id"]
for i in data["Employee"]]
 id_= int(input('Enter the id number\n '))
 if id_ in listOfIds:
return('Id Already Present\n ')
else:
newRecord["id"] = id_
newRecord["name"] = input('Enter the name')
newRecord["designation"] = input('Enter the Designation')
newRecord["salary"] = int(input('Enter the Salary of the Employee'))
newRecord["hobbies"] = input('Enter the hobbies\n').split()
with open('dbms.json','w') as f1:
data['Employee'].append(newRecord)
json.dump(data,f1,indent=4)
return('Record Inserted Successfully\n')
boolvalue = True
while boolvalue:
print('1. To display all data')
print('2. To display by id')
print('3. To insert a new Data')
ch = input()
print('\n')
switcher = {
```

```
"1":displayAll,
"2":displayById,
"3":insertNewRecord
}
 try:
 fun = switcher[ch]
print(fun())
 except TypeError:
 print('Enter a valid choice\n ')
choice = input('Do you want to continue Y/N')
boolvalue = True if choice in ['y','Y'] else False
main()
dbms.json
 {
"Employee": [
{
 "id": 1,
"name": "vaibhav",
"designation": "Developer",
"salary": 45000,
"hobbies": [
 "playing Cricket",
 "chess" ]
},
{
"id": 56,
 "name": "suyash",
"designation": "android dev",
```

"salary": 56000,
"hobbies": [
"cricket,",
"meme"]
}