****
****

Assignment 2 : Design and develop SQL DDL statements which demonstrate the use of schema objects such as **Table, View, Index, Sequence, Synonym**, and different **Constraints**, etc.

****
****

PR NO 01

Enter password: ****
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 13
Server version: 8.0.37 MySQL Community Server - GPL

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> show databases;

mysql> use rajesh;

```
mysql> show database rajesh;
  mysql> create database ass1and2;

  mysql> use ass1and2;

  mysql> create table record(ID int primary key,name varchar(10) not null,Gender char);

  mysql> desc record;

  mysql>
  mysql> insert into record(ID,name,gender)VALUES('123','RAJ','m');

  mysql> select * from record;
  ********************************************************************************
  ******************************************
```

PR NO 02

```
  mysql> create table emp(emp_id int primary key auto_increment,emp_fname char(20) not
  null,emp_lname char(20) not null,emp_salary int,join_date date);

  mysql>  desc emp;
  mysql> insert into emp values(1,'ram','kappor',2000,'2015-02-23');

  mysql>  insert into emp values(2,'raj','kappor',4000,'2017-08-14');

  mysql> insert into emp values(3,'ramesh','kappor',45000,'2017-08-14');

  mysql> select *from emp;

  mysql>  update emp set emp_salary=25000 where emp_id=3;

  mysql> select*from emp;

  mysql> delete from emp where emp_id=3;
```

```
mysql> select *from emp;


mysql>  insert into emp  values (3,'ramesh','kapoor',25000,'2017-08-14');

mysql> select *from emp;

mysql>  select avg(emp_salary) from emp;
mysql> select concat(emp_fname,emp_lname) from emp where emp_salary=45000;

mysql> select concat(emp_fname,emp_lname) from emp where emp_salary=4000;

mysql> select concat(emp_fname,emp_lname) from emp where emp_salary=2000;
```

```
mysql> select * from emp where year(join_date)='2014';

mysql> select * from emp where year(join_date)='2015';

mysql> insert into emp values(null,'little','kapoor',null,'2017-08-14');

mysql> select *from emp;

mysql> select *from emp where emp_id in('1','2','3');

mysql> select * from emp where emp_fname like 'Ram%';
mysql> select *from emp where emp_id not  in('1','2','3');
mysql> select *from emp where year(join_date) between '2013'and '2019';
mysql> select *from emp where year(join_date) between '2014'and '2016';
```

****
****
Assignment 4 - Unnamed PL/SQL Code block
****
****
****

```
mysql> CREATE DATABASE LibraryDB;


mysql> USE LibraryDB;
mysql> CREATE TABLE borrower (
    ->     rollin INT(11) NOT NULL PRIMARY KEY,
    ->     name CHAR(20),
    ->     dateofIssue DATE,
    ->     bname CHAR(50),
    ->     status CHAR(1)
    -> );
mysql> CREATE TABLE fine (
    ->     rollno INT(11),
    ->     fdate DATE,
    ->     amt INT(11)
    -> );
mysql> INSERT INTO borrower (rollin, name, dateofIssue, bname, status) VALUES
    -> (1, 'a', '2020-10-01', 'Java Programming', 'I'),
    -> (2, 'b', '2020-10-15', 'Computer Network', 'I'),
    -> (3, 'c', '2020-10-01', 'DBMS', 'I'),
    -> (4, 'd', '2020-09-22', 'Data Structure', 'I');
mysql> desc borrower;
mysql> desc fine;
mysql> select*from borrower;
mysql> select*from fine;
mysql> DELIMITER //
mysql> CREATE PROCEDURE fine_calculation(IN p_rno INT, IN p_bname CHAR(50))
    -> BEGIN
    ->     DECLARE l_date DATE;
    ->     DECLARE fine_amt INT DEFAULT 0;
    ->     DECLARE days_diff INT;
    ->
```

```
    ->     -- Get date of issue
    ->     SELECT dateofIssue INTO l_date
    ->     FROM borrower
    ->     WHERE rollin = p_rno AND bname = p_bname;
    ->
    ->     -- Calculate difference in days
    ->     SET days_diff = DATEDIFF(CURDATE(), l_date);
    ->
    ->     -- Fine rules
    ->     IF days_diff > 15 AND days_diff <= 30 THEN
    ->         SET fine_amt = (days_diff - 15) * 5;
    ->     ELSEIF days_diff > 30 THEN
    ->         SET fine_amt = (15 * 5) + ((days_diff - 30) * 10);
    ->     ELSE
    ->         SET fine_amt = 0;
    ->     END IF;
    ->
    ->     -- Insert into fine table
    ->     INSERT INTO fine (rollno, fdate, amt)
    ->     VALUES (p_rno, CURDATE(), fine_amt);
    ->
    ->     -- Update borrower status to Returned (R)
    ->     UPDATE borrower
    ->     SET status = 'R'
    ->     WHERE rollin = p_rno AND bname = p_bname;
    -> END //
mysql> DELIMITER ;
mysql> CALL fine_calculation(3, 'DBMS');
mysql> SELECT * FROM fine;
mysql> SELECT * FROM borrower;
mysql>
```

****
****

Assignment 5 -Write a Stored Procedure namely proc_Grade for the categorization of student. If marks scoredby 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

****
****
****

```
mysql> create databaase Score;
mysql> CREATE DATABASE Score;
mysql> USE Score;
Database changed
mysql> create table stud_marks(name varchar(20),total_marks int(5));
mysql> create table Result(roll_no int(3) primary key,name varchar(20),class varchar(20));
mysql> insert into stud_marks values('Suresh',995);
mysql> insert into stud_marks values('Harish',865);
mysql> insert into stud_marks values('Samart',920);
mysql> insert into stud_marks values('Mohan',1000);
mysql> insert into stud_marks values('Soham',745);
mysql> select * from stud_marks;
mysql> insert into Result(roll_no,Name) values(1,'Suresh');
mysql> insert into Result(roll_no,Name) values(2,'Harish');
mysql> insert into Result(roll_no,Name) values(3,'Samart');
mysql> insert into Result(roll_no,Name) values(4,'Mohan');
mysql> insert into Result(roll_no,Name) values(5,'Soham');
mysql> select * from Result;
mysql> delimiter //
mysql> create procedure proc_Grade(in r int(2),out grade char(25))
   -> begin
   -> declare m int(4);
     -> select total_marks into m from stud_marks where name=(select name from Result
where roll_no=r);
   -> if m>=990 and m<=1500 then
   -> select 'Distinction' into grade;
   -> update Result set Class='Distinction' where Roll_no=r;
   -> elseif m>=900 and m<=989 then
   -> select 'FirstClass' into grade;
   -> update Result set Class='FirstClass' where Roll_no=r;
   -> elseif m>=825 and m<=899 then
   -> select 'SecondClass' into grade;
   -> update Result set Class='SecondClass' where Roll_no=r;
```

```
    -> else
    -> select '--' into grade;
    -> update Result set Class='--' where Roll_no=r;
    -> end if;
    -> end //
mysql> delimiter //
mysql> create function func_Grade(r int(2))
    -> returns varchar(25)
    -> deterministic
    -> begin
    -> declare grade varchar(25);
    -> call proc_Grade(r,grade);
    -> return grade;
    -> end //
 mysql> select func_Grade(1); //
mysql> select func_Grade(2); //
mysql> select func_Grade(3); //
mysql> select func_Grade(4); //
mysql> select func_Grade(5); //
mysql> select * from Result; //
mysql>
```

****
****

Assignment 6 -Write a PL/SQL block of code using parameterized Cursor that will merge the data availablein 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

****
****
****

```
mysql> CREATE DATABASE `cursor`;
mysql> use cursor;
mysql> create table o_rollcall(rno int(11) primary key, name varchar(20), addr varchar(30));
mysql> create table n_rollcall(rno int(11), name varchar(20), addr varchar(30));
mysql> insert into O_rollcall values(1, 'Sunny', 'Mumbai');
mysql> insert into O_rollcall values(2, 'harshit', 'Pune');
mysql> insert into O_rollcall values(3, 'Vivek', 'Surat');
mysql> insert into O_rollcall values(4, 'Sahil', 'Benglore');
mysql> insert into O_rollcall values(5, 'Sachin', 'Menglore');
mysql> insert into n_rollcall values(1, 'Sunny', 'Mumbai');
mysql> insert into n_rollcall values(2, 'harshit', 'Pune');
mysql> insert into n_rollcall values(3, 'Vivek', 'Surat');
mysql> delimiter //
```

```
mysql> create procedure n1(IN rno1 int)
    -> begin
    -> declare rno2 int;
    -> declare exit_cond boolean;
    -> declare c1 cursor for select rno from o_rollcall where rno>rno1;
    -> declare continue handler for not found set exit_cond=TRUE;
    -> open c1;
    ->  l1: loop
    -> fetch c1 into rno2;
    -> if not exists(select * from n_rollcall where rno=rno2) then
    ->  insert into n_rollcall select * from o_rollcall where rno=rno2;
    -> end if;
    ->  if exit_cond then
    -> close c1;
    -> leave l1;
    -> end if;
    -> end loop l1;
    -> end;
    -> //
mysql> DELIMITER //
mysql> CALL n1(3);
    -> //
mysql> select * from n_rollcall;
    -> //
mysql>
```

****
****

Assignment 7 -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 should be added in Library_Audit table.

****
****
****

```
mysql> use lib;
mysql> create table library1(bno int(5),bname varchar(40),author varchar(20),allowed_days
int(5));
mysql> create table library_audit(bno int(5),old_all_days int(5),new_all_days int(5));
mysql> insert into library1 values(1,'Database Systems','Connally T',10);
mysql> insert into library1 values(2,'System Programming','John Donovan',20);
mysql> insert into library1 values(3,'Computer Network & Internet','Douglas E. Comer',18);
mysql> insert into library1 values(4,'Agile Project Management','Ken Schwaber',24);
mysql> insert into library1 values(5,'Python for Data Analysis','Wes McKinney',12);
mysql> select * from library1;
mysql> delimiter //
mysql> create trigger tr1
   -> before update on library1
   -> for each row
   -> begin
   -> insert into library_audit values(new.bno,old.allowed_days,new.allowed_days);
   -> end //
mysql> update library1 set allowed_days=15 where bno=1; //
mysql> update library1 set allowed_days=25 where bno=2; //
mysql> update library1 set allowed_days=13 where bno=3; //
mysql> update library1 set allowed_days=19 where bno=4; //
mysql> update library1 set allowed_days=17 where bno=5; //
mysql> select * from library1; //
mysql> select * from library_audit; //
mysql>
```

****
****

Assignment 3 - Design at least 10 sql queries for suitable database application using sql DML statements all type of join ,sub-query and view.

****
****

```
mysql> CREATE DATABASE PRNO3;
 mysql> USE PRNO3;
mysql> create table student2 (roll_no int primary key,name varchar(20),address varchar(40),phone varchar(10),age varchar(2));
mysql> desc student2;
mysql> insert into student2 values (1,'sham','bihar','1111111111','18');
mysql> insert into student2 values (2,'ram','kolkata','2222222222','20');
mysql> insert into student2 values (3,'priyanka','pune','3333333333','19');
mysql> insert into student2 values (4,'sai','mumbai','4444444444','18');
mysql> select*from student2;
mysql> create table studentcourse (cID int(1),roll_no int, foreign key(roll_no) references student2(roll_no));
mysql> desc studentcourse;
mysql> insert into studentcourse values(1,1);
mysql> insert into studentcourse values(2,2);
mysql> insert into studentcourse values(2,3);
mysql> insert into studentcourse values (3,4);
mysql> select *from studentcourse;
mysql> select*from student2 inner join studentcourse on student2.roll_no =studentcourse.roll_no;
mysql> select*from student2 left join studentcourse on student2.roll_no =studentcourse.roll_no;
mysql> select*from student2 full join studentcourse on student2.roll_no =studentcourse.roll_no;
mysql> select*from student2 join studentcourse;
mysql> select* from student2;
mysql> select a.name,b.roll_no from student2 a,student2 b where a.age<b.age order by a.name;
mysql> select a.name,b.roll_no from student2 a,student2 b where a.age<b.age;
mysql> select*from student2 right join studentcourse on student2.roll_no =studentcourse.roll_no;
mysql> create view newView as select student2.name,student2.roll_no,studentcourse.cID from student2 left join studentcourse on student2.roll_no=studentcourse.roll_no;
mysql> select*from newView;
mysql> select *from student2 where roll_no in
    -> (select roll_no from studentcourse);
mysql> select*from student2 where roll_no in(select roll_no from studentcourse where student2.age<20);
mysql>
```

Assignment 8 : MySQL python connectivity

****

****

Program :

```
import mysql.connector
con = mysql.connector.connect(
      user='root',
      password='Tejal@123',
      host='localhost',
      database='rajDB')
#con = mysql.connector.connect(host="localhost", user="root", password="Expert123",
database="poonam_db")
print(con)
def insert(id,name, age, city):
  res = con.cursor()
  sql = "insert into users (id,name,age,city) values (%s,%s,%s,%s)"
  user = (id,name, age, city)
  res.execute(sql, user)
  con.commit()
  print("Data Insert Success")
def update(name, age, city,id):
  res = con.cursor()
  sql = "update users set name=%s,age=%s,city=%s where id=%s"
  user = (name, age, city,id)
  res.execute(sql, user)
  con.commit()
  print("Data Update Success")
def select():
  res = con.cursor()
  sql = "SELECT ID,NAME,AGE,CITY from users"
  res.execute(sql)
  # result=res.fetchone()
  # result=res.fetchmany(2)
  #result = res.fetchall()
  #print(tabulate(result, headers=["ID", "NAME", "AGE", "CITY"]))
  print(res.fetchall())
def delete(id):
  res = con.cursor()
  sql = "delete from users where id=%s"
  users = (id,)
  res.execute(sql, users)
  con.commit()
  print("Data Delete Success")
while True:
  print("1.Insert Data:")
  print("2.Update Data")
  print("3.Select Data")
  print("4.Delete Data")
  print("5.Exit")
  choice = int(input("Enter Your Choice : "))
  if choice == 1:
     id = input("Enter The Id : ")
     name = input("Enter Name : ")
```

```
    age = input("Enter Age : ")
    city = input("Enter City : ")
    insert(id,name, age, city)
elif choice == 2:
    id = input("Enter The Id : ")
    name = input("Enter Name : ")
    age = input("Enter Age : ")
    city = input("Enter City : ")
    update(name, age, city,id)
elif choice == 3:
    select()
elif choice == 4:
    id = input("Enter The Id to Delete : ")
    delete(id)
elif choice == 5:
    quit()
else:
    print("Invalid Selection . Please Try Again !")
```

Assignment 9 : MySQL python connectivity

****

****

```
Last login: Sat Oct 25 14:02:32 on ttys014
/Users/raj/.zshrc:2: bad assignment
raj@Sairajs-MacBook-Air ~ % mongosh
Current Mongosh Log ID:      68fc8b9997d84c68b8f231a7
Connecting to:
          mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000
&appName=mongosh+2.5.8
Using MongoDB:               7.0.25
Using Mongosh:               2.5.8
For mongosh info see: https://www.mongodb.com/docs/mongodb-shell/
------
   The server generated these startup warnings when booting
   2025-10-24T14:36:22.248+05:30: Access control is not enabled for the database. Read and
write access to data and configuration is unrestricted
------
test> show dbs
admin          40.00 KiB
config         96.00 KiB
hospital_db    232.00 KiB
local          80.00 KiB
mydatabase     72.00 KiB
restaurant_db  272.00 KiB
test> use book
switched to db book
book> show collections;
book> db.createCollection("Library");
{ ok: 1 }
book> db.library.insert({"bid":1,"name":"C++"});
DeprecationWarning: Collection.insert() is deprecated. Use insertOne, insertMany, or
bulkWrite.
{
  acknowledged: true,
  insertedIds: { '0': ObjectId('68fc8bee97d84c68b8f231a8') }
}
book> db.library.insert({"bid":2,"name":"SEPM","author":"Pressman"});
|
{
  acknowledged: true,
  insertedIds: { '0': ObjectId('68fc8bfb97d84c68b8f231a9') }
}
book> db.library.insert({"bid":3,"name":"CN","author":"Forouzan","cost":700});
|
{
  acknowledged: true,
  insertedIds: { '0': ObjectId('68fc8c0297d84c68b8f231aa') }
}
book> db.library.find().pretty();
|
[
  { _id: ObjectId('68fc8bee97d84c68b8f231a8'), bid: 1, name: 'C++' },
  {
```

```
    _id: ObjectId('68fc8bfb97d84c68b8f231a9'),
    bid: 2,
    name: 'SEPM',
    author: 'Pressman'
  },
  {
    _id: ObjectId('68fc8c0297d84c68b8f231aa'),
    bid: 3,
    name: 'CN',
    author: 'Forouzan',
    cost: 700
  }
]
book> db.library.remove({"bid":1});
|
DeprecationWarning: Collection.remove() is deprecated. Use deleteOne, deleteMany,
findOneAndDelete, or bulkWrite.
{ acknowledged: true, deletedCount: 1 }
book> db.library.count();
|
DeprecationWarning: Collection.count() is deprecated. Use countDocuments or
estimatedDocumentCount.
2
book> db.library.find().pretty();
|
[
  {
    _id: ObjectId('68fc8bfb97d84c68b8f231a9'),
    bid: 2,
    name: 'SEPM',
    author: 'Pressman'
  },
  {
    _id: ObjectId('68fc8c0297d84c68b8f231aa'),
    bid: 3,
    name: 'CN',
    author: 'Forouzan',
    cost: 700
  }
]
book> db.library.insert({"bid":1,"name":"C++"});
|
{
  acknowledged: true,
  insertedIds: { '0': ObjectId('68fc8c1d97d84c68b8f231ab') }
}
book> db.library.find().pretty();
|
[
  {
    _id: ObjectId('68fc8bfb97d84c68b8f231a9'),
    bid: 2,
    name: 'SEPM',
    author: 'Pressman'
  },
```

```
  {
    _id: ObjectId('68fc8c0297d84c68b8f231aa'),
    bid: 3,
    name: 'CN',
    author: 'Forouzan',
    cost: 700
  },
  { _id: ObjectId('68fc8c1d97d84c68b8f231ab'), bid: 1, name: 'C++' }
]
book> db.library.find().sort({"bid":1})
|
[
  { _id: ObjectId('68fc8c1d97d84c68b8f231ab'), bid: 1, name: 'C++' },
  {
    _id: ObjectId('68fc8bfb97d84c68b8f231a9'),
    bid: 2,
    name: 'SEPM',
    author: 'Pressman'
  },
  {
    _id: ObjectId('68fc8c0297d84c68b8f231aa'),
    bid: 3,
    name: 'CN',
    author: 'Forouzan',
    cost: 700
  }
]
book> db.library.insert({"bid":4,"name":"SPOS","author":"Pearson","cost":500});
|
{
  acknowledged: true,
  insertedIds: { '0': ObjectId('68fc8c2b97d84c68b8f231ac') }
}
book> db.library.find().pretty();
|
[
  {
    _id: ObjectId('68fc8bfb97d84c68b8f231a9'),
    bid: 2,
    name: 'SEPM',
    author: 'Pressman'
  },
  {
    _id: ObjectId('68fc8c0297d84c68b8f231aa'),
    bid: 3,
    name: 'CN',
    author: 'Forouzan',
    cost: 700
  },
  { _id: ObjectId('68fc8c1d97d84c68b8f231ab'), bid: 1, name: 'C++' },
  {
    _id: ObjectId('68fc8c2b97d84c68b8f231ac'),
    bid: 4,
    name: 'SPOS',
    author: 'Pearson',
```

```
    cost: 500
  }
]
book> db.library.find().sort({"bid":1})
|
[
  { _id: ObjectId('68fc8c1d97d84c68b8f231ab'), bid: 1, name: 'C++' },
  {
    _id: ObjectId('68fc8bfb97d84c68b8f231a9'),
    bid: 2,
    name: 'SEPM',
    author: 'Pressman'
  },
  {
    _id: ObjectId('68fc8c0297d84c68b8f231aa'),
    bid: 3,
    name: 'CN',
    author: 'Forouzan',
    cost: 700
  },
  {
    _id: ObjectId('68fc8c2b97d84c68b8f231ac'),
    bid: 4,
    name: 'SPOS',
    author: 'Pearson',
    cost: 500
  }
]
book> db.library.find({$and:[{"name":"CN"},{"cost":700}]}).pretty()
|
[
  {
    _id: ObjectId('68fc8c0297d84c68b8f231aa'),
    bid: 3,
    name: 'CN',
    author: 'Forouzan',
    cost: 700
  }
]
book> db.library.insert({"bid":5,"name":"TOC","author":"Addison-Wesley","cost":600});
|
{
  acknowledged: true,
  insertedIds: { '0': ObjectId('68fc8c4197d84c68b8f231ad') }
}
book> db.library.insert({"bid":6,"name":"AI","author":"McGraw Hill Education","cost":800});
|
{
  acknowledged: true,
  insertedIds: { '0': ObjectId('68fc8c4797d84c68b8f231ae') }
}
book> db.library.find().pretty();
|
[
  {
```

```
    _id: ObjectId('68fc8bfb97d84c68b8f231a9'),
    bid: 2,
    name: 'SEPM',
    author: 'Pressman'
  },
  {
    _id: ObjectId('68fc8c0297d84c68b8f231aa'),
    bid: 3,
    name: 'CN',
    author: 'Forouzan',
    cost: 700
  },
  { _id: ObjectId('68fc8c1d97d84c68b8f231ab'), bid: 1, name: 'C++' },
  {
    _id: ObjectId('68fc8c2b97d84c68b8f231ac'),
    bid: 4,
    name: 'SPOS',
    author: 'Pearson',
    cost: 500
  },
  {
    _id: ObjectId('68fc8c4197d84c68b8f231ad'),
    bid: 5,
    name: 'TOC',
    author: 'Addison-Wesley',
    cost: 600
  },
  {
    _id: ObjectId('68fc8c4797d84c68b8f231ae'),
    bid: 6,
    name: 'AI',
    author: 'McGraw Hill Education',
    cost: 800
  }
]
book> db.library.find({$or:[{"cost":500},{"cost":800}]}).pretty()
|
[
  {
    _id: ObjectId('68fc8c2b97d84c68b8f231ac'),
    bid: 4,
    name: 'SPOS',
    author: 'Pearson',
    cost: 500
  },
  {
    _id: ObjectId('68fc8c4797d84c68b8f231ae'),
    bid: 6,
    name: 'AI',
    author: 'McGraw Hill Education',
    cost: 800
  }
]
book> db.library.find({"cost":{$ne:500}})
|
```

```
[
  {
    _id: ObjectId('68fc8bfb97d84c68b8f231a9'),
    bid: 2,
    name: 'SEPM',
    author: 'Pressman'
  },
  {
    _id: ObjectId('68fc8c0297d84c68b8f231aa'),
    bid: 3,
    name: 'CN',
    author: 'Forouzan',
    cost: 700
  },
  { _id: ObjectId('68fc8c1d97d84c68b8f231ab'), bid: 1, name: 'C++' },
  {
    _id: ObjectId('68fc8c4197d84c68b8f231ad'),
    bid: 5,
    name: 'TOC',
    author: 'Addison-Wesley',
    cost: 600
  },
  {
    _id: ObjectId('68fc8c4797d84c68b8f231ae'),
    bid: 6,
    name: 'AI',
    author: 'McGraw Hill Education',
    cost: 800
  }
]
book> db.library.find({$nor:[{"cost":500},{"author":"Forouzan"}]})
|
[
  {
    _id: ObjectId('68fc8bfb97d84c68b8f231a9'),
    bid: 2,
    name: 'SEPM',
    author: 'Pressman'
  },
  { _id: ObjectId('68fc8c1d97d84c68b8f231ab'), bid: 1, name: 'C++' },
  {
    _id: ObjectId('68fc8c4197d84c68b8f231ad'),
    bid: 5,
    name: 'TOC',
    author: 'Addison-Wesley',
    cost: 600
  },
  {
    _id: ObjectId('68fc8c4797d84c68b8f231ae'),
    bid: 6,
    name: 'AI',
    author: 'McGraw Hill Education',
    cost: 800
  }
]
```

```
book> db.library.find({"cost":{$not:{$gt:800}}})
|
[
  {
    _id: ObjectId('68fc8bfb97d84c68b8f231a9'),
    bid: 2,
    name: 'SEPM',
    author: 'Pressman'
  },
  {
    _id: ObjectId('68fc8c0297d84c68b8f231aa'),
    bid: 3,
    name: 'CN',
    author: 'Forouzan',
    cost: 700
  },
  { _id: ObjectId('68fc8c1d97d84c68b8f231ab'), bid: 1, name: 'C++' },
  {
    _id: ObjectId('68fc8c2b97d84c68b8f231ac'),
    bid: 4,
    name: 'SPOS',
    author: 'Pearson',
    cost: 500
  },
  {
    _id: ObjectId('68fc8c4197d84c68b8f231ad'),
    bid: 5,
    name: 'TOC',
    author: 'Addison-Wesley',
    cost: 600
  },
  {
    _id: ObjectId('68fc8c4797d84c68b8f231ae'),
    bid: 6,
    name: 'AI',
    author: 'McGraw Hill Education',
    cost: 800
  }
]
book> db.library.insert({"bid":7,"name":"CC","author":"Wiley Publications","cost":400})
|
{
  acknowledged: true,
  insertedIds: { '0': ObjectId('68fc8c6397d84c68b8f231af') }
}
book> db.library.find()
|
[
  {
    _id: ObjectId('68fc8bfb97d84c68b8f231a9'),
    bid: 2,
    name: 'SEPM',
    author: 'Pressman'
  },
  {
```

```
    _id: ObjectId('68fc8c0297d84c68b8f231aa'),
    bid: 3,
    name: 'CN',
    author: 'Forouzan',
    cost: 700
  },
  { _id: ObjectId('68fc8c1d97d84c68b8f231ab'), bid: 1, name: 'C++' },
  {
    _id: ObjectId('68fc8c2b97d84c68b8f231ac'),
    bid: 4,
    name: 'SPOS',
    author: 'Pearson',
    cost: 500
  },
  {
    _id: ObjectId('68fc8c4197d84c68b8f231ad'),
    bid: 5,
    name: 'TOC',
    author: 'Addison-Wesley',
    cost: 600
  },
  {
    _id: ObjectId('68fc8c4797d84c68b8f231ae'),
    bid: 6,
    name: 'AI',
    author: 'McGraw Hill Education',
    cost: 800
  },
  {
    _id: ObjectId('68fc8c6397d84c68b8f231af'),
    bid: 7,
    name: 'CC',
    author: 'Wiley Publications',
    cost: 400
  }
]
book> db.library.update({'cost':400},{$set:{'cost':600}})
|
DeprecationWarning: Collection.update() is deprecated. Use updateOne, updateMany, or
bulkWrite.
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
book> db.library.update({'cost':800},{$set:{'cost':1200}})
|
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
```

```
}
book> db.library.find().pretty();
|
[
  {
    _id: ObjectId('68fc8bfb97d84c68b8f231a9'),
    bid: 2,
    name: 'SEPM',
    author: 'Pressman'
  },
  {
    _id: ObjectId('68fc8c0297d84c68b8f231aa'),
    bid: 3,
    name: 'CN',
    author: 'Forouzan',
    cost: 700
  },
  { _id: ObjectId('68fc8c1d97d84c68b8f231ab'), bid: 1, name: 'C++' },
  {
    _id: ObjectId('68fc8c2b97d84c68b8f231ac'),
    bid: 4,
    name: 'SPOS',
    author: 'Pearson',
    cost: 500
  },
  {
    _id: ObjectId('68fc8c4197d84c68b8f231ad'),
    bid: 5,
    name: 'TOC',
    author: 'Addison-Wesley',
    cost: 600
  },
  {
    _id: ObjectId('68fc8c4797d84c68b8f231ae'),
    bid: 6,
    name: 'AI',
    author: 'McGraw Hill Education',
    cost: 1200
  },
  {
    _id: ObjectId('68fc8c6397d84c68b8f231af'),
    bid: 7,
    name: 'CC',
    author: 'Wiley Publications',
    cost: 600
  }
]
book>
```

Assignment 10 : MongoDB indexing and aggregation

****

****

```
Last login: Sat Oct 25 14:10:45 on ttys014
/Users/raj/.zshrc:2: bad assignment
raj@Sairajs-MacBook-Air ~ % mongosh
Current Mongosh Log ID:	68fc8d3bcafb589085538da4
Connecting to:
	mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000
&appName=mongosh+2.5.8
Using MongoDB:		7.0.25
Using Mongosh:		2.5.8
For mongosh info see: https://www.mongodb.com/docs/mongodb-shell/
------
   The server generated these startup warnings when booting
   2025-10-24T14:36:22.248+05:30: Access control is not enabled for the database. Read and
write access to data and configuration is unrestricted
------
test> show dbs
|
admin          40.00 KiB
book           80.00 KiB
config         72.00 KiB
hospital_db    232.00 KiB
local          80.00 KiB
mydatabase     72.00 KiB
restaurant_db  272.00 KiB
test> use customer
|
switched to db customer
customer> db.cust_table.insert({Item_id:1,Cust_Name:"Ram",Product:"Milk",Amount:40});
|
DeprecationWarning: Collection.insert() is deprecated. Use insertOne, insertMany, or
bulkWrite.
{
  acknowledged: true,
  insertedIds: { '0': ObjectId('68fc8d60cafb589085538da5') }
}
customer> db.cust_table.insert({Item_id:2,Cust_Name:"Ram",Product:"Parle_G",Amount:50});
|
{
  acknowledged: true,
  insertedIds: { '0': ObjectId('68fc8d65cafb589085538da6') }
}
customer> db.cust_table.insert({Item_id:3,Cust_Name:"Mohan",Product:"Lays
Chips",Amount:40});
|
{
  acknowledged: true,
  insertedIds: { '0': ObjectId('68fc8d6ccafb589085538da7') }
}
customer>
db.cust_table.insert({Item_id:4,Cust_Name:"Shivam",Product:"Mentos",Amount:10});
|
```

```
{
  acknowledged: true,
  insertedIds: { '0': ObjectId('68fc8d72cafb589085538da8') }
}
customer>
db.cust_table.insert({Item_id:5,Cust_Name:"Mohan",Product:"Maggie",Amount:60});
|
{
  acknowledged: true,
  insertedIds: { '0': ObjectId('68fc8d78cafb589085538da9') }
}
customer> db.cust_table.aggregate({$group:{_id:"$Cust_Name","total":{$sum:"$Amount"}}});
|
[
  { _id: 'Ram', total: 90 },
  { _id: 'Mohan', total: 100 },
  { _id: 'Shivam', total: 10 }
]
customer> db.cust_table.aggregate({$group:{_id:"$Cust_Name","total":{$avg:"$Amount"}}});
|
[
  { _id: 'Ram', total: 45 },
  { _id: 'Shivam', total: 10 },
  { _id: 'Mohan', total: 50 }
]
customer> db.cust_table.aggregate({$group:{_id:"$Cust_Name","total":{$min:"$Amount"}}});
|
[
  { _id: 'Ram', total: 40 },
  { _id: 'Shivam', total: 10 },
  { _id: 'Mohan', total: 40 }
]
customer> db.cust_table.aggregate({$group:{_id:"$Cust_Name","total":{$max:"$Amount"}}});
|
[
  { _id: 'Mohan', total: 60 },
  { _id: 'Ram', total: 50 },
  { _id: 'Shivam', total: 10 }
]
customer> db.cust_table.aggregate({$group:{_id:"$Cust_Name","total":{$first:"$Amount"}}});
|
[
  { _id: 'Mohan', total: 40 },
  { _id: 'Ram', total: 40 },
  { _id: 'Shivam', total: 10 }
]
customer> db.cust_table.aggregate({$group:{_id:"$Cust_Name","total":{$last:"$Amount"}}});
|
[
  { _id: 'Mohan', total: 60 },
  { _id: 'Ram', total: 50 },
  { _id: 'Shivam', total: 10 }
]
customer>
db.cust_table.aggregate({$group:{_id:"$Cust_Name","total":{$psuh:"$Amount"}}});
```

```
|
MongoServerError[Location15952]: unknown group operator '$psuh'
customer> db.cust_table.aggregate({$group:{_id:"$Cust_Name","total":{$sum:1}}});
|
[
  { _id: 'Ram', total: 2 },
  { _id: 'Mohan', total: 2 },
  { _id: 'Shivam', total: 1 }
]
customer>
db.cust_table.aggregate({$group:{_id:"$Cust_Name","total":{$addToSet:"$Amount"}}});
|
[
  { _id: 'Mohan', total: [ 60, 40 ] },
  { _id: 'Ram', total: [ 40, 50 ] },
  { _id: 'Shivam', total: [ 10 ] }
]
customer> db.cust_table.createIndex({'Item_id':1})
|
Item_id_1
customer> db.cust_table.createIndex({'Item_id':2})
|
Item_id_2
customer> db.cust_table.createIndex({'Item_id':4})
|
Item_id_4
customer> db.cust_table.getIndexes()
|
[
  { v: 2, key: { _id: 1 }, name: '_id_' },
  { v: 2, key: { Item_id: 1 }, name: 'Item_id_1' },
  { v: 2, key: { Item_id: 2 }, name: 'Item_id_2' },
  { v: 2, key: { Item_id: 4 }, name: 'Item_id_4' }
]
customer> db.cust_table.dropIndex({'Item_id':4})
|
{ nIndexesWas: 4, ok: 1 }
customer> db.cust_table.dropIndex({'Item_id':1})
|
{ nIndexesWas: 3, ok: 1 }
customer> db.cust_table.getIndexes()
|
[
  { v: 2, key: { _id: 1 }, name: '_id_' },
  { v: 2, key: { Item_id: 2 }, name: 'Item_id_2' }
]
customer> db.cust_table.getIndexes()
[
  { v: 2, key: { _id: 1 }, name: '_id_' },
  { v: 2, key: { Item_id: 2 }, name: 'Item_id_2' }
]
customer>
```

Assignment 11 : Map Reduce

****

****

```
Last login: Sat Oct 25 14:10:45 on ttys014
/Users/raj/.zshrc:2: bad assignment
raj@Sairajs-MacBook-Air ~ % mongosh
Current Mongosh Log ID:      68fc8d3bcafb589085538da4
Connecting to:
          mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000
&appName=mongosh+2.5.8
Using MongoDB:               7.0.25
Using Mongosh:               2.5.8
For mongosh info see: https://www.mongodb.com/docs/mongodb-shell/
------
   The server generated these startup warnings when booting
   2025-10-24T14:36:22.248+05:30: Access control is not enabled for the database. Read and
write access to data and configuration is unrestricted
------
test> show dbs
|
admin          40.00 KiB
book           80.00 KiB
config         72.00 KiB
hospital_db    232.00 KiB
local          80.00 KiB
mydatabase     72.00 KiB
restaurant_db  272.00 KiB
test> use customer
|
switched to db customer
customer> db.cust_table.insert({Item_id:1,Cust_Name:"Ram",Product:"Milk",Amount:40});
|
DeprecationWarning: Collection.insert() is deprecated. Use insertOne, insertMany, or
bulkWrite.
{
  acknowledged: true,
  insertedIds: { '0': ObjectId('68fc8d60cafb589085538da5') }
}
customer> db.cust_table.insert({Item_id:2,Cust_Name:"Ram",Product:"Parle_G",Amount:50});
|
{
  acknowledged: true,
  insertedIds: { '0': ObjectId('68fc8d65cafb589085538da6') }
}
customer> db.cust_table.insert({Item_id:3,Cust_Name:"Mohan",Product:"Lays
Chips",Amount:40});
|
{
  acknowledged: true,
  insertedIds: { '0': ObjectId('68fc8d6ccafb589085538da7') }
}
customer>
db.cust_table.insert({Item_id:4,Cust_Name:"Shivam",Product:"Mentos",Amount:10});
|
```

```
{
  acknowledged: true,
  insertedIds: { '0': ObjectId('68fc8d72cafb589085538da8') }
}
customer>
db.cust_table.insert({Item_id:5,Cust_Name:"Mohan",Product:"Maggie",Amount:60});
|
{
  acknowledged: true,
  insertedIds: { '0': ObjectId('68fc8d78cafb589085538da9') }
}
customer> db.cust_table.aggregate({$group:{_id:"$Cust_Name","total":{$sum:"$Amount"}}});
|
[
  { _id: 'Ram', total: 90 },
  { _id: 'Mohan', total: 100 },
  { _id: 'Shivam', total: 10 }
]
customer> db.cust_table.aggregate({$group:{_id:"$Cust_Name","total":{$avg:"$Amount"}}});
|
[
  { _id: 'Ram', total: 45 },
  { _id: 'Shivam', total: 10 },
  { _id: 'Mohan', total: 50 }
]
customer> db.cust_table.aggregate({$group:{_id:"$Cust_Name","total":{$min:"$Amount"}}});
|
[
  { _id: 'Ram', total: 40 },
  { _id: 'Shivam', total: 10 },
  { _id: 'Mohan', total: 40 }
]
customer> db.cust_table.aggregate({$group:{_id:"$Cust_Name","total":{$max:"$Amount"}}});
|
[
  { _id: 'Mohan', total: 60 },
  { _id: 'Ram', total: 50 },
  { _id: 'Shivam', total: 10 }
]
customer> db.cust_table.aggregate({$group:{_id:"$Cust_Name","total":{$first:"$Amount"}}});
|
[
  { _id: 'Mohan', total: 40 },
  { _id: 'Ram', total: 40 },
  { _id: 'Shivam', total: 10 }
]
customer> db.cust_table.aggregate({$group:{_id:"$Cust_Name","total":{$last:"$Amount"}}});
|
[
  { _id: 'Mohan', total: 60 },
  { _id: 'Ram', total: 50 },
  { _id: 'Shivam', total: 10 }
]
customer>
db.cust_table.aggregate({$group:{_id:"$Cust_Name","total":{$psuh:"$Amount"}}});
```

```
|
MongoServerError[Location15952]: unknown group operator '$psuh'
customer> db.cust_table.aggregate({$group:{_id:"$Cust_Name","total":{$sum:1}}});
|
[
  { _id: 'Ram', total: 2 },
  { _id: 'Mohan', total: 2 },
  { _id: 'Shivam', total: 1 }
]
customer>
db.cust_table.aggregate({$group:{_id:"$Cust_Name","total":{$addToSet:"$Amount"}}});
|
[
  { _id: 'Mohan', total: [ 60, 40 ] },
  { _id: 'Ram', total: [ 40, 50 ] },
  { _id: 'Shivam', total: [ 10 ] }
]
customer> db.cust_table.createIndex({'Item_id':1})
|
Item_id_1
customer> db.cust_table.createIndex({'Item_id':2})
|
Item_id_2
customer> db.cust_table.createIndex({'Item_id':4})
|
Item_id_4
customer> db.cust_table.getIndexes()
|
[
  { v: 2, key: { _id: 1 }, name: '_id_' },
  { v: 2, key: { Item_id: 1 }, name: 'Item_id_1' },
  { v: 2, key: { Item_id: 2 }, name: 'Item_id_2' },
  { v: 2, key: { Item_id: 4 }, name: 'Item_id_4' }
]
customer> db.cust_table.dropIndex({'Item_id':4})
|
{ nIndexesWas: 4, ok: 1 }
customer> db.cust_table.dropIndex({'Item_id':1})
|
{ nIndexesWas: 3, ok: 1 }
customer> db.cust_table.getIndexes()
|
[
  { v: 2, key: { _id: 1 }, name: '_id_' },
  { v: 2, key: { Item_id: 2 }, name: 'Item_id_2' }
]
customer> db.cust_table.getIndexes()
[
  { v: 2, key: { _id: 1 }, name: '_id_' },
  { v: 2, key: { Item_id: 2 }, name: 'Item_id_2' }
]
customer>
```

Assignment 12 : MongoDB Connectivity

****

****

****

Program:

```
from pymongo import MongoClient

def show_data(collection):
  print("\n    Current Records:")
  for doc in collection.find():
     print(f"Name: {doc.get('Name', '')}\tAge: {doc.get('Age', '')}\tMobile: {doc.get('Mobile
Number', '')}")
  print()

def main():
  # Connect to MongoDB
  client = MongoClient("localhost", 27017)
  print("✓ Connected to MongoDB successfully")

  db = client["Info"]
  collection = db["Personal"]

  while True:
     # INSERT OPERATION
     n = int(input("\nEnter the number of records you want to insert: "))
     for i in range(n):
        print(f"\nEnter data for record {i + 1}:")
        name = input("Enter Name: ")
        age = int(input("Enter Age: "))
        mobile = input("Enter Mobile Number: ")
        collection.insert_one({
           "Name": name,
           "Age": age,
           "Mobile Number": mobile
        })
     print("\n✓ Insert Operation Done.")
     show_data(collection)

     # DELETE OPERATION
     n = int(input("\nEnter the number of records you want to delete: "))
     for i in range(n):
        name = input(f"Enter name to delete ({i + 1}): ")
        result = collection.delete_one({"Name": name})
        if result.deleted_count > 0:
           print(f"    Deleted record with name: {name}")
        else:
           print(f"   No record found with name: {name}")

     print("\n    Delete Operation Done.")
     show_data(collection)
```

```
        # DROP DATABASE OPTION
        ans = input("\nDo you want to drop the database (y/n)? ").lower()
        if ans == 'y':
            client.drop_database("Info")
            print("   Database Dropped.")
            break

        # CONTINUE OPTION
        ans1 = input("Do you want to continue (y/n)? ").lower()
        if ans1 != 'y':
            break

    client.close()
    print("    Disconnected from MongoDB.")

if __name__ == "__main__":
    main()
```