1. Procedure Consider table Student (Roll, Name, Attendance ,Status).Write a PL/SQL block for following requirement and handle the exceptions. Roll no. of student will be entered by user. Attendance of roll no. entered by user will be checked in Stud table. If attendance is less than 75% then display the message “Term not granted” and set the status in stud table as “Detained”. Otherwise display message “Term granted” and set the status in stud table as “Not Detained”

➔ create table stud(
roll int,
name varchar2(30),
att int,
status varchar2(30) null);

SQL> insert into stud values(1,'a',78,null);

1 row created.

SQL> insert into stud values(2,'b',50,null);

1 row created.

SQL> insert into stud values(3,'c',12,null);

1 row created.

SQL> insert into stud values(4,'d',98,null);

1 row created.

SQL> select * from stud;

```
set serveroutput on;
create or procedure check_att(rollin integer) as
 2 roll integer :=rollin;
 3  att integer;
 4  begin
 5  select att into att from stud where rollin=roll;
 6   if (att<75) then
 7  dbms_output.put_line('Term not granted');
 8    update stud set status='Detained' where rollin=roll;
 9   else
10  dbms_output.put_line('Term  granted');
11   update stud set status='Not Detained' where rollin=roll;
12  end if;
13  end;
14  /
```

→call check_att(1);

2. Account(Acc_no, branch_name,balance)

branch(branch_name,branch_city,assets)

customer(cust_name,cust_street,cust_city)

Depositor(cust_name,acc_no)

Loan(loan_no,branch_name,amount)

Borrower(cust_name,loan_no)

Solve following query:

Create above tables with appropriate constraints like primary key, foreign key, check constrains, not null etc.

a) Find the names of all branches in loan relation.

b) Find all loan numbers for loans made at 'Shivaji Nagar' Branch with loan amount > 12000.

c) Find all customers who have a loan from bank. Find their names,loan_no and loan amount.

d) List all customers in alphabetical order who have loan from Akurdi branch.

e) Find all customers who have an account or loan or both at bank.

f) Find all customers who have both account and loan at bank.

g) Find all customers who have account but no loan at the bank.

h) Find average account balance at Akurdi branch.

i) Find the branches where average account balance > 12000.

j) Find number of tuples in customer relation.

k) Calculate total loan amount given by bank.

l) Delete all loans with loan amount between 1300 and 1500.

m) Delete all tuples at every branch located in Nigdi.

n) Create sequence roll_seq and use in student table for roll_no column.

➔ **Answer GENERATED BY CHATGPT AND SENT BY SWAPNIL**

Sure, based on the given information, here are the SQL statements for creating the tables with appropriate constraints and solving the provided queries:

Creating Tables with Constraints:

sql

```
CREATE TABLE Branch (
   branch_name VARCHAR(50) PRIMARY KEY,
   branch_city VARCHAR(50),
   assets DECIMAL(10, 2)
);

CREATE TABLE Account (
   Acc_no INT PRIMARY KEY,
   branch_name VARCHAR(50),
   balance DECIMAL(10, 2),
   FOREIGN KEY (branch_name) REFERENCES Branch(branch_name)
);

CREATE TABLE Customer (
   cust_name VARCHAR(50) PRIMARY KEY,
   cust_street VARCHAR(100),
   cust_city VARCHAR(50)
);

CREATE TABLE Depositor (
   cust_name VARCHAR(50),
   acc_no INT,
   PRIMARY KEY (cust_name, acc_no),
   FOREIGN KEY (cust_name) REFERENCES Customer(cust_name),
   FOREIGN KEY (acc_no) REFERENCES Account(Acc_no)
);

CREATE TABLE Loan (
   loan_no INT PRIMARY KEY,
   branch_name VARCHAR(50),
   amount DECIMAL(10, 2),
   FOREIGN KEY (branch_name) REFERENCES Branch(branch_name)
);

CREATE TABLE Borrower (
   cust_name VARCHAR(50),
   loan_no INT,
   PRIMARY KEY (cust_name, loan_no),
   FOREIGN KEY (cust_name) REFERENCES Customer(cust_name),
   FOREIGN KEY (loan_no) REFERENCES Loan(loan_no)
);
```

Solving Queries:

a) Find the names of all branches in loan relation.

sql

```
SELECT DISTINCT branch_name FROM Loan;
```

b) Find all loan numbers for loans made at 'Shivaji Nagar' Branch with loan amount > 12000.
sql
SELECT loan_no FROM Loan WHERE branch_name = 'Shivaji Nagar' AND amount > 12000;

c) Find all customers who have a loan from the bank. Find their names, loan_no, and loan amount.
sql
SELECT c.cust_name, b.loan_no, l.amount
FROM Customer c
JOIN Borrower b ON c.cust_name = b.cust_name
JOIN Loan l ON b.loan_no = l.loan_no;

d) List all customers in alphabetical order who have a loan from Akurdi branch.
sql
SELECT c.cust_name
FROM Customer c
JOIN Borrower b ON c.cust_name = b.cust_name
JOIN Loan l ON b.loan_no = l.loan_no
WHERE l.branch_name = 'Akurdi'
ORDER BY c.cust_name;

e) Find all customers who have an account or loan or both at the bank.
sql
SELECT DISTINCT cust_name
FROM (
  SELECT cust_name FROM Depositor
  UNION
  SELECT cust_name FROM Borrower
) AS CombinedCustomers;

f) Find all customers who have both account and loan at the bank.
sql
SELECT cust_name
FROM Depositor
WHERE cust_name IN (SELECT cust_name FROM Borrower);

g) Find all customers who have an account but no loan at the bank.
sql
SELECT cust_name
FROM Depositor
WHERE cust_name NOT IN (SELECT cust_name FROM Borrower);

h) Find average account balance at Akurdi branch.
sql
SELECT AVG(balance)
FROM Account

```
WHERE branch_name = 'Akurdi';
```

i) Find the branches where average account balance > 12000.

```
sql
SELECT branch_name
FROM Account
GROUP BY branch_name
HAVING AVG(balance) > 12000;
```

j) Find the number of tuples in the customer relation.

```
sql
SELECT COUNT(*) FROM Customer;
```

k) Calculate total loan amount given by the bank.

```
sql
SELECT SUM(amount) FROM Loan;
```

l) Delete all loans with loan amount between 1300 and 1500.

```
sql
DELETE FROM Loan WHERE amount BETWEEN 1300 AND 1500;
```

m) Delete all tuples at every branch located in Nigdi.

```
sql
DELETE FROM Branch WHERE branch_city = 'Nigdi';
```

n) Create sequence roll_seq and use it in the student table for the roll no column.

```
sql
CREATE SEQUENCE roll_seq START WITH 1;

CREATE TABLE Student (
   roll_no INT DEFAULT nextval('roll_seq') PRIMARY KEY,
   student_name VARCHAR(50),
   student_city VARCHAR(50)
);
```

Please note that these SQL statements assume appropriate relationships between the tables based on the given attributes. Adjustments might be needed if there are additional constraints or requirements not specified in the given schema.

3. Trigger

Write a after trigger for Insert, update and delete event considering following requirement:

Emp(Emp_no, Emp_name, Emp_salary)

a) Trigger should be initiated when salary tried to be inserted is less than Rs.

50,000/-

b) Trigger should be initiated when salary tried to be updated for value less

than Rs. 50,000/-

c) Also the new values expected to be inserted will be stored in new table

Tracking(Emp_no, Emp_salary).

➔ **INSTRUCTIONS UNCLEAR HENCE THIS TRIGGER STOP USER WHEN INSERTING LESS SALARY**

```
create table Emp(
Emp_no NUMBER PRIMARY KEY,
Emp_name VARCHAR(30)NOT NULL,
Emp_salary NUMBER);

create table Tracking(
Emp_no NUMBER PRIMARY KEY,
Emp_salary NUMBER);

SET SERVEROUTPUT ON;
CREATE OR REPLACE TRIGGER Track_Emp
BEFORE DELETE OR INSERT OR UPDATE ON Emp
FOR EACH ROW
BEGIN
IF(:NEW.emp_salary<50000) THEN
RAISE_APPLICATION_ERROR( -20001, 'LOW SALARY ERROR! Salary is less than 50000, try
again' );
ELSE
INSERT INTO Tracking VALUES(:NEW.Emp_no,:NEW.Emp_salary);
END IF;
END Track_Emp;
/
```

4. Write Unnamed PL/SQL block of code for the following requirements:-

Exception handling is mandatory.

Borrower(Rollin, Name, DateofIssue, NameofBook, Status)

Fine(Roll_no,Date,Amount_Fine)

Accept Roll_no and Name of book from user. Check the number of days (from date of issue), if days are between 15 to 30 then the fine amounts will be Rs 5 per 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.

➔ create table borrower(

rollin integer not null primary key,

name varchar2(20),

dateofissue date,

nameofbook varchar2(20),

status char);

create table fine(

roll_number integer not null primary key,

fdate date,

amt integer);

insert into borrower values(1,'ROSHANI','03-september-20','Advance Java','I');

insert into borrower values(2,'NEHA','30-september-20','Database Management','I');

insert into borrower values(3,'SHRUTIKA','03-october-20','Python','I');

insert into borrower values(4,'CHAITRA','10-august-20','Computer Networks','I');

insert into borrower values(5,'SAI','21-september-20','Data Structure','I');

```
SQL> DECLARE
  2  roll_no integer := &rollin;
  3  issuedate date;
  4  amount integer := 0;
```

```
 5  name_of_book varchar2(20) := &nameofbook;
 6  days integer := 0;
 7
 8  BEGIN
 9  select dateofissue into issuedate from borrower where rollin = roll_no and
10  nameofbook = name_of_book;
11  days := sysdate-issuedate;
12  if days<30 then
13  if days>15 then
14  amount:= 5*days;
15  insert into fine values(roll_no,sysdate,amount);
16  update borrower set status = 'R' where rollin = roll_no
17  and nameofbook = name_of_book;
18  commit;
19  end if;
20  elsif days>30 then
21  amount:= 50*days;
22  insert into fine values(roll_no,sysdate,amount);
23  update borrower set status = 'R' where rollin = roll_no
24  and nameofbook = name_of_book;
25  commit;
26  end if;
27  update borrower set status = 'R' where rollin = roll_no
28  and nameofbook = name_of_book;
29  commit;
30  EXCEPTION
31  when no_data_found then
32  dbms_output.put_line('Record not found');
33  END;
34  /
Enter value for rollin: 1
```

```
old   2: roll_no integer := &rollin;
new   2: roll_no integer := 1;
Enter value for nameofbook: 'Advance Java'
old   5: name_of_book varchar2(20) := &nameofbook;
new   5: name_of_book varchar2(20) := 'Advance Java';

PL/SQL procedure successfully completed.

SQL> select * from borrower;

   ROLLIN NAME                 DATEOFISS NAMEOFBOOK          S
---------- -------------------- --------- -------------------- -
         1 ROSHANI              03-SEP-20 Advance Java         R
         2 NEHA                 30-SEP-20 Database Management  I
         3 SHRUTIKA             03-OCT-20 Python               R
         4 CHAITRA              10-AUG-20 Computer Networks    I
         5 SAI                  21-SEP-20 Data Structure       I

SQL> select * from fine;

ROLL_NUMBER FDATE           AMT
----------- --------- ----------
          3 23-OCT-23      55750
         69 23-OCT-23     363350
          1 03-NOV-23      57850
```

5. 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 for using procedure created with above requirement. Stud_Marks(Roll, Name, Total_marks) Result(Roll, Name, Class)

➔ **GCLASSROOM APPROACH : (USES CURSOR)**

```
SQL> create table Stud_Marks(
2 rollno integer,
3 name varchar2(30),
4 total_marks integer)
5 ;

Table created.

SQL> describe Stud_Marks;
Name Null? Type
----------------------------------------- -------- ----------------------------
ROLLNO NUMBER(38)
NAME VARCHAR2(30)
TOTAL_MARKS NUMBER(38)

SQL> create table Result(
2 rollno integer,
3 name varchar2(20),
4 marks integer,
5 class varchar2(30));

Table created.

SQL> desc Result;
Name Null? Type
----------------------------------------- -------- ----------------------------
ROLLNO NUMBER(38)
NAME VARCHAR2(20)
MARKS NUMBER(38)
CLASS VARCHAR2(30)

SQL> insert into Stud_Marks values(1,'ROSHANI',1200);

1 row created.

SQL> insert into Stud_Marks values(2,'NEHA',900);

1 row created.

SQL> insert into Stud_Marks values(3,'SHRUTIKA',890);
```

1 row created.
SQL>select * from Stud_Marks;
ROLLNO NAME MARKS
---------- -------------------- ----------
1 ROSHANI 1200
2 NEHA 900
3 SHRUTIKA 890
4 CHAITRA 840

A)STORED PROCEDURE WITHOUT PARAMETER

CODE:

```
--stored procedure without parameter

create or replace procedure proc_Grade
as
cursor c1 is SELECT*FROM Stud_Marks;
rowx Stud_Marks%ROWTYPE;

begin
open c1;
LOOP
FETCH c1 INTO rowx;
EXIT WHEN c1%NOTFOUND;
if(rowx.total_marks<=1500 and rowx.total_marks>=990)then
insert into Result
values(rowx.rollno,rowx.name,rowx.total_marks,'Distinction');
elsif(rowx.total_marks<=989 and rowx.total_marks>=900)then
insert into Result values(rowx.rollno,rowx.name,rowx.total_marks,'First');
elsif(rowx.total_marks<=899 and rowx.total_marks>=825)then
insert into Result
values(rowx.rollno,rowx.name,rowx.total_marks,'Higher Second');
end if;
END LOOP;
close c1;
end;
/
```

B)STORED PROCEDURE WITH PARAMETER

CODE:

```
--stored procedure with parameters
create or replace procedure proc_Grade1(trno in int,tname in
Stud_Marks.name%type,tmarks in int)
as
begin
if(tmarks<=1500 and tmarks>=990)then
insert into Result values(trno,tname,tmarks,'Distinction');
```

```
elsif(tmarks<=989 and tmarks>=900)then
insert into Result values(trno,tname,tmarks,'First Class');
elsif(tmarks<=899 and tmarks>=825)then
insert into Result values(trno,tname,tmarks,'Higher Second');
end if;
end;

/
```

EXECUTION:

```
SQL> @stored_para;

Procedure created.

SQL> EXEC proc_Grade1(10,'SAI',1100);
```

SAM'S APPROACH

```
create table student(
Rno number PRIMARY KEY,
Name varchar(20) NOT NULL,
TotMarks integer,
class varchar(20));

insert into student values(1,'Piyush',1160,NULL);
insert into student values(56,'Shreya',969,NULL);
insert into student values(6,'Mohit',1499,NULL);
insert into student values(62,'Neel',875,NULL);
insert into student values(52,'Sid',680,NULL);
insert into student values(69,'Varad',669,NULL);
insert into student values(21,'Sameer',599,NULL);

CREATE or REPLACE PROCEDURE set_class AS
BEGIN
for stud in (SELECT Rno,TotMarks FROM student)
LOOP
IF stud.TotMarks <= 1500 AND stud.TotMarks >= 990 THEN
UPDATE student
SET class = 'Distinction' WHERE Rno=stud.Rno;
ELSIF stud.TotMarks <= 989 AND stud.TotMarks >= 900 THEN
UPDATE student
SET class = 'First Class' WHERE Rno=stud.Rno;
ELSIF stud.TotMarks <= 899 AND stud.TotMarks >= 825 THEN
UPDATE student
SET class = 'Higher Second Class' WHERE Rno=stud.Rno;
ELSIF stud.TotMarks <= 824 AND stud.TotMarks >= 600 THEN
```

```
UPDATE student
SET class = 'Second Class' WHERE Rno=stud.Rno;
ELSIF stud.TotMarks < 600 THEN
UPDATE student
SET class = 'Fail' WHERE Rno=stud.Rno;

End IF;
END LOOP;
END set_class;
/
```

PIUS APPROACH

```
CREATE or REPLACE PROCEDURE assign_class AS
BEGIN
update student set class = 'Distinction' where Totmarks between 990 and 1500;
update student set class = 'First Class' where Totmarks between 900 and 989;
update student set class = 'Higher Second Class' where Totmarks between 825 and 899;
update student set class = 'Second Class' where Totmarks between 600 and 824;
update student set class = 'Fail' where Totmarks<600;
END assign_class;
/
```

6. Trigger Consider CUSTOMER (ID, Name, Age, Address, Salary) create a row level trigger for the CUSTOMERS table that would fire for INSERT or UPDATE or DELETE operations performed on the CUSTOMERS table. This trigger will display the salary difference between the old values and new values.

➔ create table customers(

```
ID INTEGER PRIMARY KEY,
name VARCHAR(20),
address VARCHAR(30),
salary NUMBER);


set serveroutput on;
CREATE or REPLACE TRIGGER cal_saldiff
BEFORE INSERT or UPDATE or DELETE on customers
for EACH ROW
DECLARE
sal_diff NUMBER;
BEGIN
sal_diff := :NEW.salary - :OLD.salary ;
dbms_output.put_line('Salary difference is : ' || sal_diff);
END cal_saldiff;
/
```

7. Trigger Create the customer_Master table(cust_no, Cust_name, DOB, Addr). Create row level trigger when an update, insert or delete operation is performed on the Customer_Master table. Depending on the operation being performed, a variable is assigned the value update, insert or delete. The previous and new values of the modified records should be inserted in the Customer_Audit table.

8 Basic SQL queries

Create table Cust_Master(Cust_no, Cust_name, Qty_Ordered, Order_date, Cust_addr)

Cust_no is defined as primary key,

Insert ten records in the table.

List names of customers having 'a' as second letter in their name.

List customers who stay in city whose first letter is 'M'

Display order from Customer no C1002,C1005,C1007 and C1008

List Clients who stay in either 'Banglore or 'Manglore'

Create view Customer_View consisting of Cust_no, Qty_ordered and Order_date

9 MongoDB CRUD operations, Create collection Employee consisting of Emp_id, Emp_Name, Emp_salary, Emp_Dept.

Insert 10 Documents in the collection.

Find the employees whose salary is greater than 50000 Rs.

Increase the salary of Smith by 5000 Rs

Display the information of employees working in Marketing department.with less than 45000 salary .

Display first five highest paid employees

Delete Employee with Id 'E1007'

Create an Index on Emp_Id field , compare the time require to search Emp_id 'E10008' before and after creating an index. (Hint Add at least 10000 Documents)

➔

```
use EmployeeDB;

db.createCollection("employees")

//insert 10 Documents in the collection.
db.employees.insertMany([
{"Emp_id":"E1001","Emp_Name":"Pius","Emp_salary": 59000, "Emp_Dept":"HR"},
{"Emp_id":"E1002","Emp_Name":"Shreya","Emp_salary": 6900, "Emp_Dept":"Marketing"},
{"Emp_id":"E1003","Emp_Name":"Moit","Emp_salary": 85000, "Emp_Dept":"Documentation"},
{"Emp_id":"E1004","Emp_Name":"Sid","Emp_salary": 66000, "Emp_Dept":"Bakchodi"},
{"Emp_id":"E1005","Emp_Name":"Neel","Emp_salary": 90000, "Emp_Dept":"Design"},
{"Emp_id":"E1006","Emp_Name":"Alexa","Emp_salary": 70000, "Emp_Dept":"Publicity"},
{"Emp_id":"E1007","Emp_Name":"Venom","Emp_salary": 60000, "Emp_Dept":"Chaos"},
{"Emp_id":"E1008","Emp_Name":"Varad","Emp_salary": 85000, "Emp_Dept":"Play"},
{"Emp_id":"E1009","Emp_Name":"Smith","Emp_salary": 50000, "Emp_Dept":"Testing"},
{"Emp_id":"E1010","Emp_Name":"John","Emp_salary": 30000, "Emp_Dept":"Marketing"}]);

//Find the employees whose salary is greater than 50000 Rs.
```

```
db.employees.find({Emp_salary:{$gt:50000}})

// Increase the salary of Smith by 5000 Rs

db.employees.updateOne({Emp_Name:'Smith'}, [{$set:{Emp_salary:{$add:["$Emp_salary",5000]}}}])

// Display the information of employees working in Marketing department.with less than 45000 salary .

db.employees.find({Emp_Dept:"Marketing",Emp_salary:{$lt:45000}}).pretty()

// Display first five highest paid employees

db.employees.aggregate([{$sort:{Emp_salary:-1}},{$limit:5}])

// Delete Employee with Id ‘E1007’

db.employees.deleteOne({Emp_id:"E1007"})

// Create an Index on Emp_Id field , compare the time require to search Emp_id ‘E10008’ before and after creating an index. (Hint Add at least 10000 Documents)

db.employees.find({Emp_id:"E1008"}).explain("executionStats")

// here no. of documents  need to be examined

db.employees.createIndex({"Emp_id":1})

db.employees.find({Emp_id:"E1008"}).explain("executionStats")

 // here direct key is access hence only 1 document examined
```

10 Basic SQL Create following tables with suitable constraints. Make suitable use of AUTO_INCREMENT.

Insert data and solve the following queries:

PROPERTIES(Pno, Type, Sq_Ft_Area, Rent, Address, Status, Owner No)

OWNERS (OwnerNo, OwnerName, Phno)

1. Display all 1BHK apartments in Kothrud which are not rented
2. Display all properties owned by “Gopal”
3. Write a query to display the smallest property of each owner
4. Display all properties in Kothrud in Descending order of rent
5. Create a view which shows OwnerName along with his Pno, type, Address and Rent
6. Display the names of all Owners whose name has “ee”
7. Display Pno, Type, Address, Rent and status of all properties with rent greater than 15000/- and less than 22000/-
8. Display different property types registered with the real estate agency

11 Basic SQL Create following tables with suitable constraints. Make suitable use of AUTO_INCREMENT.

Insert data and solve the following queries:

PROPERTIES(Pno, Type, Sq_Ft_Area, Rent, Address, Status, Owner No)

CLIENTS(ClientNo, ClientName, Phno, Requirement)

RENTAL(Pno, ClientNo, FromDate, ToDate)

1. Write a query that produces a list of all available properties
2. Write a query to list all properties which will be available in December 2017
3. Write a query to count the number of properties area wise
4. Display the names of all Owners whose name starts with 'R'

5. Create a view to show ClientName along with the Pno, Type, Address and Rent of the property which he has rented

6. Display the alphabetical list of Clients

7. Increase rent of Pno 1006 by 2000/-

8. Delete record of Pno=1007

14 MapReduce

Create a customer collection consisting of fields like name, email ID, profession, gender, bill amount

1. Write a MapReduce query for finding the count of male and female customers in the collection

2. Write a MapReduce query for finding the count of each profession in the collection

3. Display list of all customers with bill amounts greater than 5000/-

4. Update the bill amount of any one customer

5. Display all customers with name starting with 'B'

6. Display list of all customers with profession = “Business”

7. Display all customers in Descending order of Bill amount

8. Create an index on name field of customer collection. Also use the explain() function

15. Mongo Aggregation

Create a student collection consisting of fields like Roll No, name, class, marks, sports etc

1. Create an index on name field of employee collection. Also use the explain() function
2. Display the first 5 toppers of TE
3. Display marks of topper of each division
4. Display the average marks of each division
5. Display the rollcall of TEComp A
6. Display list of fail students from TE Comp A
7. Display Name, Class and Marks of all students
8. Display list of students who play football

➔

```
use studentDB;
db.createCollection("student");
db.student.insertMany([
{"roll":1,"name":"Pius","class":"A","marks": 59, "sports":"cricket"},
{"roll":2,"name":"Shreya","class":"A","marks": 69, "sports":"badminton"},
{"roll":3,"name":"Moit","class":"A","marks": 85, "sports":"tennis"},
{"roll":4,"name":"Sid","class":"A","marks": 66, "sports":"football"},
{"roll":5,"name":"Neel","class":"A","marks": 90, "sports":"CS2"},
{"roll":1,"name":"Armaan","class":"B","marks": 70, "sports":"basketball"},
{"roll":2,"name":"Varun","class":"B","marks": 60, "sports":"badminton"},
```

```
{"roll":3,"name":"Varad","class":"B","marks": 85, "sports":"cricket"},

{"roll":4,"name":"Aish","class":"B","marks": 50, "sports":"football"},

{"roll":5,"name":"Harsh","class":"B","marks": 75, "sports":"table tennis"}]);


// Display first 5 toppers of TE

db.student.aggregate([{$sort:{marks:-1}},{$limit:5}]);


// Display marks of topper of each division

db.student.aggregate([{$sort:{class:-1,marks:-
1}},{$group:{_id:'$class',toppermarks:{$max:'$marks'},toppername:{$first:"$name"}}}])


// Display average marks of each division

db.student.aggregate([{$group:{_id:'$class', avgMarks:{$avg:'$marks'}}}])


// roll call of A division

db.student.aggregate([{$sort:{roll:1}},{$match:{class:'A'}}])


// list of failed student from A division

db.student.aggregate([{$match:{class:'A',marks:{$lt:40}}}]) // if marks<40 is failing criteria


// Display name class marks of all student

db.student.aggregate([{$group:{_id:null,Student:{$push:{name:"$name",class:"$class",marks
:"$marks"}}}}])


// Display list of student who play football

db.student.aggregate([{$match:{sports:"football"}},{$group:{_id:"$sports",Student:{$push:{n
ame:"$name",class:"$class"}}}}])
```

16 PL/SQL procedure

Create the following tables with suitable constraints

MEMBERS(mem_id, mname, addr, phno)

BOOKS(Bno, bname, publisher, cost, DOP,status)

ISSUE_RETURN(mem_id, Bno, issue_date, return_date, fine)

Write a PL/SQL procedure which requires mem_id, Bno and issue _date as arguments. The procedure calculates the fine and performs the book returning operation

Assume suitable conditions for calculating fine.

17 PL/SQL procedure

Create the following table with suitable constraints. Put NULL value in the Class field while entering the data.

STUDENT(Rno, Name, TotMarks, Class)

Write a PL/SQL procedure to find the class of every student and update the Class field of the STUDENT table accordingly

if marks<=1500 and marks >=990 then Class = Distinction if marks<=989 and marks >=900 then Class = First Class

if marks<=899 and marks >=825 then Class = Higher Second Class if marks<=824 and marks >=600 then Class = Second Class if marks<600 then class = Fail