//21.Create Database BSIOTR using
MongoDBCreate following Collections

**Teachers(Tname,dno,dname,experience,salary,date_of_jo
ining)Students(Sname,roll_no,class)**

```
->use BSIOTR

->db.createCollection("Teachers")

->db.Teachers.insertMany([
 {
  Tname: "Teacher 1",
  dno: 101,
  dname: "Department A",
  experience: 5,
  salary: 50000,
  date_of_joining: new Date("2020-01-15")
 },
 {
  Tname: "Teacher 2",
  dno: 102,
  dname: "Department B",
  experience: 8,
  salary: 60000,
  date_of_joining: new Date("2018-07-22")
 }
])
->db.createCollection("Students")

->db.Students.insertMany([
 {
  Sname: "Student 1",
```

```
  roll_no: 1001,
  class: "A"
 },
 {
  Sname: "Student 2",
  roll_no: 1002,
  class: "B"
 }
])
//1. Display the department wise average salary
->db.Teachers.aggregate([
 {
  $group: {
   _id: "$dname", // Group by department name
   averageSalary: { $avg: "$salary" } // Calculate the average salary for each department
  }
 }
])
//2. display the no. Of employees working in each department
->db.Teachers.aggregate([
 {
  $group: {
   _id: "$dname", // Group by department name
   numberOfEmployees: { $sum: 1 } // Count the number of employees in each department
  }
 }
])
```

//3. Display the department wise total salary of departments having total salary greater thanor equals to 50000/-

```
->db.Teachers.aggregate([
 {
  $group: {
   _id: "$dname", // Group by department name
   totalSalary: { $sum: "$salary" } // Calculate the total salary for each department
  }
 },
 {
  $match: {
   totalSalary: { $gte: 50000 } // Filter departments with total salary >= 50000
  }
 }
])
//4. Write the queries using the different operators like max, min. Etc.
->db.Teachers.aggregate([
 {
  $group: {
   _id: null,
   maxSalary: { $max: "$salary" }
  }
 }
])
//5. Create unique index on any field for above given collections
->db.Teachers.createIndex({ Tname: 1 }, { unique: true })
//6. Create compound index on any fields for above given collections
->db.Students.createIndex({ roll_no: 1 }, { unique: true })
```

//Create Database BSIOTR

//Create following Collections
//**Teachers(Tname,dno,dname,experience,salary,date_of_joining)**
//**Students(Sname,roll_no,class)**

```
->use BSIOTR

->db.createCollection("Teachers")


->db.Teachers.insertMany([
 {
  Tname: "Teacher 1",
  dno: "D001",
  dname: "Computer Science",
  experience: 5,
  salary: 60000,
  date_of_joining: ISODate("2022-10-15")
 },
 {
  Tname: "Teacher 2",
  dno: "D002",
  dname: "Electrical Engineering",
  experience: 8,
  salary: 70000,
  date_of_joining: ISODate("2021-09-30")
 },
 // Add more teacher documents as needed
])


->db.createCollection("Students")


db.Students.insertMany([
 {
```

```
  Sname: "Student 1",
  roll_no: 1001,
  class: "Class A"
 },
 {
  Sname: "Student 2",
  roll_no: 1002,
  class: "Class B"
 },
 // Add more student documents as needed
])
//1. Find the information about two teachers
db.Teachers.find({}).limit(2)
//2. Find the information about all teachers of computer department
db.Teachers.find({ dname: "Computer" })
//3. Find the information about all teachers of computer,IT,and e&TC department
db.Teachers.find({
 dname: { $in: ["Computer", "IT", "E&TC"] }
})
//4.. Find the information about all teachers of computer,IT,and E&TC department having salary greate than or equl to 25000/-
db.Teachers.find({
 dname: { $in: ["Computer", "IT", "E&TC"] },
 salary: { $gte: 25000 } // Salary greater than or equal to 25,000
})
//5. Find the student information having roll_no = 25 or Sname=xyz
db.Students.find({
 $or: [
  { roll_no: 25 },
  { Sname: "xyz" }
 ]
```

```
})
//6Update the experience of teacher-praveen to 10years, if the entry is not available in database consider the entry as new entry.db.Teachers.updateOne(
db.Teachers.updateOne(
  { Tname: "Praveen" },
  {
    $set: {
      experience: 10
    }
  },
  { upsert: true }
)
//7. Update the deparment of all the teachers working in IT deprtment to COMP
db.Teachers.updateMany(
  { dname: "IT" },
  {
    $set: {
      dname: "COMP"
    }
  }
)
//8. find the teachers name and their experience from teachers collection
db.Teachers.find({}, { Tname: 1, experience: 1, _id: 0 })
//9. Using Save() method insert one entry in department collection
var newDepartment = {
  dno: "D005",
  dname: "New Department",
  location: "New Location"
};
//10. Delete all the doccuments from teachers collection having IT dept.
db.Teachers.deleteMany({ dname: "IT" })
```

//19.Create Database BSIOTR using MongoDB. Create following Collections

//Teachers(Tname,dno,dname,experience,salary,date_of_j
//oining)Students(Sname,roll_no,class)

```
->use BSIOTR
->db.createCollection("Teachers")
->db.Teachers.insertMany([
 {
  Tname: "Teacher 1",
  dno: "D001",
  dname: "Department A",
  experience: 5,
  salary: 60000,
  date_of_joining: new Date("2023-11-07")
 },
 {
  Tname: "Teacher 2",
  dno: "D002",
  dname: "Department B",
  experience: 8,
  salary: 70000,
  date_of_joining: new Date("2022-09-15")
 },
 // Add more teacher documents as needed
])
->db.createCollection("Students")

db.Students.insertMany([
 {
  Sname: "Student 1",
  roll_no: 1001,
  class: "Class A"
```

```
  },
  {
    Sname: "Student 2",
    roll_no: 1002,
    class: "Class B"
  },
  // Add more student documents as needed
])

//1. Find the information about all teachers
db.Teachers.find({})
//2. Find the information about all teachers of computer department
db.Teachers.find({ dname: "Computer" })
//3. Find the information about all teachers of computer,IT,and E&TC department having salary greate than or equl to 10000
db.Teachers.find({
  dname: { $in: ["Computer", "IT", "E&TC"] },
  salary: { $gte: 10000 } // Salary greater than or equal to 10,000
})
//4Find the student information having roll_no = 2 or Sname=xyz
db.Students.find({
  $or: [
    { roll_no: 2 },
    { Sname: "xyz" }
  ]
})
//6. Update the experience of teacher-praveen to 10years, if the entry is not available in database consider the entry as new entry.
db.Teachers.updateOne(
  { Tname: "Praveen" },
  {
```

```
   $set: {
     experience: 10
   }
  },
  { upsert: true }
)
//77. Update the deparment of all the teachers working in IT deprtment to COMP
db.Teachers.updateMany(
  { dname: "IT" },
  {
    $set: {
      dname: "COMP"}})
//8. find the teachers name and their experience from teachers collection
db.Teachers.find({}, { Tname: 1, experience: 1, _id: 0 })
//9 Using Save() method insert one entry in department collection
db.department.save(newDepartment)
//10. Delete all the doccuments from teachers collection having IT dept.
db.Teachers.deleteMany({ dname: "IT" })
```

//a) Consider table Stud(Roll, Att,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 “D”. Otherwise display message “Term granted” and set the status in stud table as “ND”

```
DECLARE
  v_roll_number NUMBER;
  v_attendance NUMBER;
  v_status CHAR(1);
BEGIN
  -- Accept roll number from the user
  v_roll_number := &roll_number;

  -- Check attendance for the given roll number
  SELECT Att INTO v_attendance
  FROM Stud
  WHERE Roll = v_roll_number;

  -- Check if attendance is less than 75%
  IF v_attendance < 75 THEN
    -- Attendance is less than 75%, so set status to "D"
    v_status := 'D';
    DBMS_OUTPUT.PUT_LINE('Term not granted');
  ELSE
    -- Attendance is 75% or more, so set status to "ND"
    v_status := 'ND';
    DBMS_OUTPUT.PUT_LINE('Term granted');
  END IF;

  -- Update the status in the Stud table
  UPDATE Stud
  SET Status = v_status
  WHERE Roll = v_roll_number;

  COMMIT;
EXCEPTION
  WHEN NO_DATA_FOUND THEN
    DBMS_OUTPUT.PUT_LINE('Student with Roll ' || v_roll_number || ' not found');
  WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE('An error occurred: ' || SQLERRM);
END;
/
```

// b) Write a PL/SQL block for following requirement using user defined exception handling. The account_master table records the current balance for an account, which is updated whenever, any deposits or withdrawals takes place. If the withdrawal attempted is more than the current balance held in the account. The user defined exception is raised, displaying an appropriate message. Write a PL/SQL block for above requirement using user defined exception handling

```
-- Define a user-defined exception
DECLARE
  insufficient_funds EXCEPTION;
  PRAGMA EXCEPTION_INIT(insufficient_funds, -20001);

  -- Declare variables
  v_account_number NUMBER := &account_number; -- Replace with the desired account number
  v_withdrawal_amount NUMBER := &withdrawal_amount; -- Replace with the withdrawal amount
  v_current_balance NUMBER;

BEGIN
  -- Retrieve the current balance for the specified account
  SELECT balance
  INTO v_current_balance
  FROM account_master
  WHERE account_number = v_account_number;

  -- Check if withdrawal amount exceeds the current balance
  IF v_withdrawal_amount > v_current_balance THEN
    -- Raise the user-defined exception
    RAISE insufficient_funds;
  ELSE
    -- Deduct the withdrawal amount from the current balance
    v_current_balance := v_current_balance – v_withdrawal_amount;

    -- Update the balance in the account_master table
    UPDATE account_master
    SET balance = v_current_balance
    WHERE account_number = v_account_number;

    COMMIT;
    DBMS_OUTPUT.PUT_LINE('Withdrawal successful. New balance: ' || v_current_balance);
  END IF;

EXCEPTION
  WHEN insufficient_funds THEN
    DBMS_OUTPUT.PUT_LINE('Withdrawal amount exceeds the current balance. Transaction canceled.');
  WHEN NO_DATA_FOUND THEN
    DBMS_OUTPUT.PUT_LINE('Account not found.');
  WHEN OTHERS THEN
```

```
    DBMS_OUTPUT.PUT_LINE('An error occurred: ' || SQLERRM);
END;
```

//a)Write an SQL code block these raise a user defined exception where business rule is voilated. BR for client_ master table specifies when the value of bal_due field is less than 0 handle the exception.

```
-- Define a user-defined exception
DECLARE
  insufficient_balance EXCEPTION;
BEGIN
  -- Check if the business rule is violated
  IF (SELECT COUNT(*) FROM client_master WHERE bal_due < 0) > 0 THEN
    -- Raise the user-defined exception
    RAISE insufficient_balance;
  END IF;
EXCEPTION
  WHEN insufficient_balance THEN
    DBMS_OUTPUT.PUT_LINE('Business rule violation: Balance due is less than 0.');
  WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE('An error occurred: ' || SQLERRM);
END;
/
```

//b Write an SQL code block

Borrow(Roll_no, Name, DateofIssue, NameofBook, Status)Fine(Roll_no,Date,Amt)

Accept roll_no & name of book 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.

```
-- Define a named exception
DECLARE
  borrowed_after_returned EXCEPTION;
BEGIN
  -- Accept roll_no and name of book from the user
  DECLARE
```

```
 v_roll_no NUMBER := &roll_no;          -- Replace with user input
 v_name_of_book VARCHAR2(50) := '&name_of_book'; -- Replace with user input
 v_date_of_issue DATE;
 v_days_late NUMBER;
 v_fine_amount NUMBER;
BEGIN
 -- Retrieve the date of issue and status for the book
 SELECT DateofIssue, Status
 INTO v_date_of_issue, v_status
 FROM Borrow
 WHERE Roll_no = v_roll_no AND NameofBook = v_name_of_book;

 -- Check if the book has already been returned
 IF v_status = 'R' THEN
  RAISE borrowed_after_returned;
 ELSE
  -- Calculate the number of days late
  v_days_late := TRUNC(SYSDATE - v_date_of_issue);

  -- Calculate the fine amount
  IF v_days_late > 30 THEN
   v_fine_amount := v_days_late * 50;
  ELSE
   v_fine_amount := v_days_late * 5;
  END IF;

  -- Update the status to 'R' indicating the book has been returned
  UPDATE Borrow
  SET Status = 'R'
  WHERE Roll_no = v_roll_no AND NameofBook = v_name_of_book;
```

```
    -- Insert the fine details into the Fine table
    INSERT INTO Fine (Roll_no, Date, Amt)
    VALUES (v_roll_no, SYSDATE, v_fine_amount);

    COMMIT;

    DBMS_OUTPUT.PUT_LINE('Fine calculated and updated. Fine Amount: Rs ' || v_fine_amount);
   END IF;
  EXCEPTION
   WHEN NO_DATA_FOUND THEN
    DBMS_OUTPUT.PUT_LINE('Book not found for Roll Number ' || v_roll_no);
   WHEN borrowed_after_returned THEN
    DBMS_OUTPUT.PUT_LINE('The book has already been returned.');
   WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE('An error occurred: ' || SQLERRM);
  END;
END;
/
```

// **16Cursor (Any Two)**

a) The bank manager has decided to activate all those accounts which were previously marked asinactive for performing no transaction in last 365 days. Write a PL/SQ block (using implicit cursor) to update the status of account, display an approximate message based on the no. of rows affected by the update. (Use of %FOUND, %NOTFOUND, %ROWCOUNT)

```
DECLARE
  -- Declare a variable to count the number of updated rows
  v_rows_updated NUMBER := 0;

BEGIN
  -- Implicit cursor is used in the UPDATE statement
  UPDATE account
  SET status = 'Active'
  WHERE last_transaction_date < SYSDATE - 365;

  -- Get the number of updated rows
  v_rows_updated := SQL%ROWCOUNT;

  -- Check if any rows were updated
  IF v_rows_updated > 0 THEN
    -- Rows were updated, display a message
    DBMS_OUTPUT.PUT_LINE(v_rows_updated || ' accounts were reactivated.');
  ELSE
    -- No rows were updated, display a different message
    DBMS_OUTPUT.PUT_LINE('No accounts needed reactivation.');
  END IF;

  COMMIT;
END;
```

/

//

a) Organization has decided to increase the salary of employees by 10% of existing salary, who are having salary less than average salary of organization, Whenever such salary updates takes place, a record for the same is maintained in the increment_salary table.

```
DECLARE
  v_avg_salary NUMBER;
  v_incremented_salary NUMBER := 0;

BEGIN
  -- Calculate the average salary of the organization
  SELECT AVG(salary) INTO v_avg_salary FROM employees;

  -- Implicit cursor is used in the UPDATE statement
  UPDATE employees
  SET salary = salary * 1.10 -- Increase salary by 10%
  WHERE salary < v_avg_salary;

  -- Check if any rows were updated
  IF SQL%FOUND THEN
    -- Rows were updated, display a message
    DBMS_OUTPUT.PUT_LINE('Salary increment completed.');
    v_incremented_salary := SQL%ROWCOUNT;
    DBMS_OUTPUT.PUT_LINE(v_incremented_salary || ' employees received a salary increment.');

    -- Insert records into the increment_salary table
    INSERT INTO increment_salary (employee_id, increment_date, increment_amount)
    SELECT employee_id, SYSDATE, (salary * 0.10)
    FROM employees
```

```
    WHERE salary < v_avg_salary;
  ELSE
    -- No rows were updated, display a message
    DBMS_OUTPUT.PUT_LINE('No employees received a salary increment.');
  END IF;

  COMMIT;
END;
/
```

// **16Cursor (Any Two)**

a) The bank manager has decided to activate all those accounts which were previously marked asinactive for performing no transaction in last 365 days. Write a PL/SQ block (using implicit cursor) to update the status of account, display an approximate message based on the no. of rows affected by the update. (Use of %FOUND, %NOTFOUND, %ROWCOUNT)

```
DECLARE
  -- Declare a variable to count the number of updated rows
  v_rows_updated NUMBER := 0;

BEGIN
  -- Implicit cursor is used in the UPDATE statement
  UPDATE account
  SET status = 'Active'
  WHERE last_transaction_date < SYSDATE - 365;

  -- Get the number of updated rows
  v_rows_updated := SQL%ROWCOUNT;

  -- Check if any rows were updated
  IF v_rows_updated > 0 THEN
    -- Rows were updated, display a message
    DBMS_OUTPUT.PUT_LINE(v_rows_updated || ' accounts were reactivated.');
  ELSE
    -- No rows were updated, display a different message
    DBMS_OUTPUT.PUT_LINE('No accounts needed reactivation.');
  END IF;

  COMMIT;
END;
```

/

//

a) Organization has decided to increase the salary of employees by 10% of existing salary, who are having salary less than average salary of organization, Whenever such salary updates takes place, a record for the same is maintained in the increment_salary table.

```
DECLARE
  v_avg_salary NUMBER;
  v_incremented_salary NUMBER := 0;

BEGIN
  -- Calculate the average salary of the organization
  SELECT AVG(salary) INTO v_avg_salary FROM employees;

  -- Implicit cursor is used in the UPDATE statement
  UPDATE employees
  SET salary = salary * 1.10 -- Increase salary by 10%
  WHERE salary < v_avg_salary;

  -- Check if any rows were updated
  IF SQL%FOUND THEN
    -- Rows were updated, display a message
    DBMS_OUTPUT.PUT_LINE('Salary increment completed.');
    v_incremented_salary := SQL%ROWCOUNT;
    DBMS_OUTPUT.PUT_LINE(v_incremented_salary || ' employees received a salary increment.');

    -- Insert records into the increment_salary table
    INSERT INTO increment_salary (employee_id, increment_date, increment_amount)
    SELECT employee_id, SYSDATE, (salary * 0.10)
    FROM employees
```

```
    WHERE salary < v_avg_salary;
  ELSE
    -- No rows were updated, display a message
    DBMS_OUTPUT.PUT_LINE('No employees received a salary increment.');
  END IF;

  COMMIT;
END;
/
```