DBMS Lab

Assignmnet 1

```
CREATE DATABASE assignment1;
USE assignment1;

-- Q1: Create Tables with Proper Primary and Foreign
Keys
CREATE TABLE employee (
    employee_name VARCHAR(50) PRIMARY KEY,
    street VARCHAR(50),
    city VARCHAR(50)
);

CREATE TABLE company (
    company_name VARCHAR(50) PRIMARY KEY,
    city VARCHAR(50)
);

CREATE TABLE works (
    employee_name VARCHAR(50),
    company_name VARCHAR(50),
    salary DECIMAL(10, 2),
    PRIMARY KEY (employee_name, company_name),
    FOREIGN KEY (employee_name) REFERENCES
employee(employee_name),
    FOREIGN KEY (company_name) REFERENCES
company(company_name)
);

CREATE TABLE manages (
    employee_name VARCHAR(50),
    manager_name VARCHAR(50),
    PRIMARY KEY (employee_name, manager_name),
    FOREIGN KEY (employee_name) REFERENCES
employee(employee_name),
    FOREIGN KEY (manager_name) REFERENCES
employee(employee_name)
);

-- Q2: Insert Records into Each Table
INSERT INTO employee (employee_name, street, city)
VALUES

('Raj Shukla', 'Street A', 'Pune'),
('Sandeep Patil', 'Street B', 'Mumbai'),
('Shital Sonje', 'Street C', 'Nashik'),
('Anita Sharma', 'Street D', 'Delhi'),
('Ravi Verma', 'Street E', 'Kolkata'),
('Pooja Singh', 'Street F', 'Pune'),
('Vikas Gupta', 'Street G', 'Mumbai'),
('Amit Kumar', 'Street H', 'Nashik'),
('Sunita Desai', 'Street I', 'Delhi'),
('Manoj Mehta', 'Street J', 'Kolkata');

INSERT INTO company (company_name, city) VALUES
('First Bank Corporation', 'Pune'),
('Small Bank Corporation', 'Mumbai'),
('Special Bank Corporation', 'Nashik');

INSERT INTO works (employee_name, company_name,
salary) VALUES
('Raj Shukla', 'First Bank Corporation', 60000),
('Sandeep Patil', 'Small Bank Corporation', 45000),
('Shital Sonje', 'Special Bank Corporation', 50000),
('Anita Sharma', 'First Bank Corporation', 55000),
```

```sql
('Ravi Verma', 'First Bank Corporation', 65000),
('Pooja Singh', 'Small Bank Corporation', 48000),
('Vikas Gupta', 'Special Bank Corporation', 52000),
('Amit Kumar', 'First Bank Corporation', 58000),
('Sunita Desai', 'Small Bank Corporation', 47000),
('Manoj Mehta', 'Special Bank Corporation', 53000);

INSERT INTO manages (employee_name, manager_name)
VALUES
('Raj Shukla', 'Anita Sharma'),
('Sandeep Patil', 'Ravi Verma'),
('Shital Sonje', 'Pooja Singh'),
('Anita Sharma', 'Ravi Verma'),
('Ravi Verma', 'Anita Sharma');

-- Q3: Add Birthdate Column to Employee Table
ALTER TABLE employee ADD COLUMN birthdate DATE;

-- Q4: Update Birthdate of All Employees
UPDATE employee SET birthdate = '1980-01-01' WHERE
employee_name = 'Raj Shukla';
UPDATE employee SET birthdate = '1982-02-15' WHERE
employee_name = 'Sandeep Patil';
UPDATE employee SET birthdate = '1985-03-20' WHERE
employee_name = 'Shital Sonje';
UPDATE employee SET birthdate = '1990-04-10' WHERE
employee_name = 'Anita Sharma';
UPDATE employee SET birthdate = '1983-05-25' WHERE
employee_name = 'Ravi Verma';
UPDATE employee SET birthdate = '1988-06-30' WHERE
employee_name = 'Pooja Singh';
UPDATE employee SET birthdate = '1979-07-18' WHERE
employee_name = 'Vikas Gupta';
```

```sql
UPDATE employee SET birthdate = '1984-08-12' WHERE
employee_name = 'Amit Kumar';
UPDATE employee SET birthdate = '1987-09-22' WHERE
employee_name = 'Sunita Desai';
UPDATE employee SET birthdate = '1992-10-05' WHERE
employee_name = 'Manoj Mehta';

-- Q5: Delete Specified Employees
DELETE FROM works WHERE employee_name IN ('Raj
Shukla', 'Sandeep Patil', 'Shital Sonje');
DELETE FROM manages WHERE employee_name IN ('Raj
Shukla', 'Sandeep Patil', 'Shital Sonje');
DELETE FROM employee WHERE employee_name IN ('Raj
Shukla', 'Sandeep Patil', 'Shital Sonje');
-- Ensure the employees to be deleted do not have
foreign key dependencies in other tables

-- Q6: Rename Salary to Monthly Salary
ALTER TABLE works CHANGE COLUMN salary monthly_salary
DECIMAL(10, 2);

-- Q7: Create Dependants Table
CREATE TABLE dependant (
    employee_name VARCHAR(50),
    dependant_name VARCHAR(50),
    relation VARCHAR(50),
    PRIMARY KEY (employee_name, dependant_name),
    FOREIGN KEY (employee_name) REFERENCES
employee(employee_name)
);

-- Q8: Add Rows to Dependants Table
INSERT INTO dependant (employee_name, dependant_name,
relation) VALUES
```

```
('Anita Sharma', 'Amit Sharma', 'Spouse'),
('Ravi Verma', 'Rekha Verma', 'Spouse'),
('Pooja Singh', 'Anil Singh', 'Spouse'),
('Vikas Gupta', 'Vandana Gupta', 'Spouse'),
('Amit Kumar', 'Neha Kumar', 'Spouse');

-- Q9: Difference Between TRUNCATE and DELETE
-- No SQL code, just explanation:
-- TRUNCATE: Removes all rows from a table without
logging individual row deletions. It's faster and
uses fewer system resources.
-- DELETE: Removes rows one at a time and logs each
row deletion. It can be used with a WHERE clause to
delete specific rows.

-- Q10: Single-Select Queries
-- a. Find the names of employees starting with "S"
SELECT employee_name FROM employee WHERE
employee_name LIKE 'S%';

-- b. Find the names of employees who work for First
Bank Corporation
SELECT employee_name FROM works WHERE company_name =
'First Bank Corporation';

-- c. Find the names of the companies located in
"Pune"
SELECT company_name FROM company WHERE city = 'Pune';

-- d. Find the age of employees
SELECT employee_name, YEAR(CURDATE()) -
YEAR(birthdate) AS age FROM employee;
```

```sql
-- e. Find the names and cities of residence of all
employees who work for First Bank Corporation
SELECT e.employee_name, e.city FROM employee e JOIN
works w ON e.employee_name = w.employee_name WHERE
w.company_name = 'First Bank Corporation';

-- f. Find the salaries of the employees whose salary
is greater than 50000
SELECT employee_name, monthly_salary FROM works WHERE
monthly_salary > 50000;

-- g. Find the employee names, street addresses,
cities of residence, and company names of all
employees
SELECT e.employee_name, e.street, e.city,
w.company_name FROM employee e JOIN works w ON
e.employee_name = w.employee_name;

-- h. Find the manager of "Sandeep Patil"
SELECT manager_name FROM manages WHERE employee_name
= 'Sandeep Patil';

-- i. Find the age of all employees
SELECT employee_name, YEAR(CURDATE()) -
YEAR(birthdate) AS age FROM employee;

-- j. Find the names of the employees having their
birthday in January
SELECT employee_name FROM employee WHERE
MONTH(birthdate) = 1;

-- Q11: Five More Queries with Date, String, and
Other Functions
-- a. Find employees whose names contain 'a'
```

```
SELECT employee_name FROM employee WHERE
employee_name LIKE '%a%';

-- b. Calculate the average salary of employees
SELECT AVG(monthly_salary) AS avg_salary FROM works;

-- c. Count the number of employees in each company
SELECT company_name, COUNT(employee_name) AS
num_employees FROM works GROUP BY company_name;

-- d. Find the length of each employee's name
SELECT employee_name, LENGTH(employee_name) AS
name_length FROM employee;

-- e. Find employees hired before a specific date
-- Assuming there is a hire_date column in the works
table
ALTER TABLE works ADD COLUMN hire_date DATE;

UPDATE works SET hire_date = '2020-01-01' WHERE
employee_name = 'Anita Sharma';
-- Repeat the update for other employees as needed

SELECT employee_name FROM works WHERE hire_date <
'2021-01-01';
```

Assignment 2

```
tee assignment2.txt
CREATE DATABASE assignment2;
USE assignment2;

-- Create tables
```

```sql
CREATE TABLE instructor (
    instructor_id VARCHAR(5),
    instructor_name VARCHAR(90),
    instructor_city VARCHAR(25),
    specialization VARCHAR(30),
    PRIMARY KEY (instructor_id)
);

CREATE TABLE student (
    student_id INT,
    student_name VARCHAR(60),
    instructor_id VARCHAR(5),
    student_city VARCHAR(35),
    FOREIGN KEY (instructor_id) REFERENCES
instructor(instructor_id)
);

-- Insert data into instructor table
INSERT INTO instructor VALUES
('1001', 'Ashwin', 'Maharashtra', 'Science'),
('1002', 'Nina', 'Maharashtra', 'Arts'),
('1003', 'Ravi', 'Tamil Nadu', 'Commerce'),
('1004', 'Anil', 'Uttar Pradesh', 'Science'),
('1005', 'Sita', 'Telangana', 'Arts'),
('1006', 'Sanjay', 'Gujarat', 'Commerce'),
('1007', 'Arun', 'Punjab', 'Science'),
('1008', 'Meera', 'Kerala', 'Arts'),
('1009', 'Rajesh', 'Rajasthan', 'Commerce'),
('1010', 'Kumar', 'Madhya Pradesh', 'Science'),
('1011', 'Neelam', 'Odisha', 'Arts'),
('1012', 'Deepak', 'West Bengal', 'Commerce'),
('1013', 'Sunil', 'Haryana', 'Science'),
('1014', 'Pooja', 'Bihar', 'Arts'),
('1015', 'Suresh', 'Assam', 'Commerce'),
('1016', 'Gaurav', 'Chhattisgarh', 'Science'),
('1017', 'Ritika', 'Jharkhand', 'Arts'),
('1018', 'Kiran', 'Goa', 'Commerce'),
('1019', 'Vikram', 'Himachal Pradesh', 'Science'),
('1020', 'Asha', 'Uttarakhand', 'Arts');

-- Insert data into student table
INSERT INTO student VALUES
(2, 'Asma', '1001', 'Karnataka'),
(3, 'Rahul', '1002', 'Maharashtra'),
(4, 'Priya', '1003', 'Tamil Nadu'),
(5, 'Amit', '1004', 'Uttar Pradesh'),
(6, 'Sneha', '1005', 'Telangana'),
(7, 'Vikram', '1006', 'Gujarat'),
(8, 'Neha', '1007', 'Punjab'),
(9, 'Sanjay', '1008', 'Kerala'),
(10, 'Meera', '1009', 'Rajasthan'),
(11, 'Ravi', '1010', 'Madhya Pradesh'),
(12, 'Pooja', '1011', 'Odisha'),
(13, 'Kiran', '1012', 'West Bengal'),
(14, 'Deepak', '1013', 'Haryana'),
(15, 'Anjali', '1014', 'Bihar'),
(16, 'Rahul', '1015', 'Assam'),
(17, 'Jaya', '1016', 'Chhattisgarh'),
(18, 'Vinay', '1017', 'Jharkhand'),
(19, 'Swati', '1018', 'Goa'),
(20, 'Raj', '1019', 'Himachal Pradesh'),
(21, 'Asha', '1020', 'Uttarakhand');

-- 1. Find the instructor of each student.
SELECT student.student_name,
instructor.instructor_name
FROM student
```

```sql
JOIN instructor ON student.instructor_id =
instructor.instructor_id;

-- 2. Find the student who is not having any
instructor.
SELECT student_name
FROM student
WHERE instructor_id IS NULL;

-- 3. Find the student who is not having any
instructor as well as the instructor who is not
having a student.
-- Students without an instructor
SELECT student_name
FROM student
WHERE instructor_id IS NULL;

-- Instructors without students
SELECT instructor_name
FROM instructor
WHERE instructor_id NOT IN (SELECT instructor_id FROM
student);

-- 4. List students and their instructors for a
specific specialization (e.g., 'Science').
SELECT student.student_name,
instructor.instructor_name
FROM student
JOIN instructor ON student.instructor_id =
instructor.instructor_id
WHERE instructor.specialization = 'Science';

-- 5. Develop SQL query to find students whose city
does not match with their instructor.
```

```
SELECT student.student_name, student.student_city,
instructor.instructor_name,
instructor.instructor_city
FROM student
JOIN instructor ON student.instructor_id =
instructor.instructor_id
WHERE student.student_city <>
instructor.instructor_city;

-- 6. List All Instructors and Their Students
(Including Without Students).
SELECT instructor.instructor_name,
student.student_name
FROM instructor
LEFT JOIN student ON instructor.instructor_id =
student.instructor_id
ORDER BY instructor.instructor_name;

-- 7. List of students of each instructor who stays
in Pune (Instructor).
SELECT student.student_name,
instructor.instructor_name
FROM student
JOIN instructor ON student.instructor_id =
instructor.instructor_id
WHERE instructor.instructor_city = 'Pune';

-- 8. Calculate the Average Number of Students per
Instructor in Nashik.
SELECT AVG(student_count)
FROM (
    SELECT COUNT(student.student_id) AS student_count
    FROM instructor
    LEFT JOIN student ON instructor.instructor_id =
student.instructor_id
    WHERE instructor.instructor_city = 'Nashik'
    GROUP BY instructor.instructor_id
) AS avg_students;

-- 9. Create a view containing the total number of
students whose instructor belongs to ‘Pune’.
CREATE VIEW StudentsWithPuneInstructors AS
SELECT COUNT(student.student_id) AS total_students
FROM student
JOIN instructor ON student.instructor_id =
instructor.instructor_id
WHERE instructor.instructor_city = 'Pune';

-- 10. Write the following queries on the view:
-- a. List All Specializations of Instructors in Pune
SELECT DISTINCT specialization
FROM instructor
WHERE instructor_city = 'Pune';

-- b. Count Students for Each Instructor in Pune
SELECT instructor.instructor_name,
COUNT(student.student_id) AS student_count
FROM instructor
LEFT JOIN student ON instructor.instructor_id =
student.instructor_id
WHERE instructor.instructor_city = 'Pune'
GROUP BY instructor.instructor_name;

-- c. List the Number of Students Grouped by
Specialization of Instructors in Pune
SELECT instructor.specialization,
COUNT(student.student_id) AS student_count
```

```sql
FROM instructor
LEFT JOIN student ON instructor.instructor_id =
student.instructor_id
WHERE instructor.instructor_city = 'Pune'
GROUP BY instructor.specialization;

-- d. List Instructors from Pune with No Students
SELECT instructor.instructor_name
FROM instructor
LEFT JOIN student ON instructor.instructor_id =
student.instructor_id
WHERE instructor.instructor_city = 'Pune' AND
student.student_id IS NULL;

-- Design 5 more queries having join of 3 tables and
aggregate functions as per your choice:
-- Note: Since there are only two tables given
(student and instructor), the "join of 3 tables" part
cannot be fulfilled.
-- However, I'll provide aggregate functions
involving these two tables.

-- 1. Find the number of students each instructor has
in each specialization.
SELECT instructor.instructor_name,
instructor.specialization, COUNT(student.student_id)
AS student_count
FROM instructor
LEFT JOIN student ON instructor.instructor_id =
student.instructor_id
GROUP BY instructor.instructor_name,
instructor.specialization;
```

```sql
-- 2. List the instructors who have more than 5
students and their specialization.
SELECT instructor.instructor_name,
instructor.specialization, COUNT(student.student_id)
AS student_count
FROM instructor
LEFT JOIN student ON instructor.instructor_id =
student.instructor_id
GROUP BY instructor.instructor_name,
instructor.specialization
HAVING student_count > 5;

-- 3. Find the total number of students in each city
for each specialization.
SELECT instructor.specialization,
student.student_city, COUNT(student.student_id) AS
total_students
FROM instructor
JOIN student ON instructor.instructor_id =
student.instructor_id
GROUP BY instructor.specialization,
student.student_city;

-- 4. List instructors along with the total number of
students in each city.
SELECT instructor.instructor_name,
student.student_city, COUNT(student.student_id) AS
total_students
FROM instructor
JOIN student ON instructor.instructor_id =
student.instructor_id
GROUP BY instructor.instructor_name,
student.student_city;
```

```sql
-- 5. Find the city with the highest number of
students for each specialization.
SELECT instructor.specialization,
student.student_city, COUNT(student.student_id) AS
total_students
FROM instructor
JOIN student ON instructor.instructor_id =
student.instructor_id
GROUP BY instructor.specialization,
student.student_city
ORDER BY instructor.specialization, total_students
DESC;
```

Assignment 3

```sql
CREATE DATABASE assignment3;
USE assignment3;

CREATE TABLE employee (
    employee_name VARCHAR(50) PRIMARY KEY,
    street VARCHAR(50),
    city VARCHAR(50)
);

CREATE TABLE company (
    company_name VARCHAR(50) PRIMARY KEY,
    city VARCHAR(50)
);

CREATE TABLE works (
    employee_name VARCHAR(50),
    company_name VARCHAR(50),
    salary DECIMAL(10, 2),
    PRIMARY KEY (employee_name, company_name),
    FOREIGN KEY (employee_name) REFERENCES
employee(employee_name),
    FOREIGN KEY (company_name) REFERENCES
company(company_name)
);

CREATE TABLE manages (
    employee_name VARCHAR(50),
    manager_name VARCHAR(50),
    PRIMARY KEY (employee_name, manager_name),
    FOREIGN KEY (employee_name) REFERENCES
employee(employee_name),
    FOREIGN KEY (manager_name) REFERENCES
employee(employee_name)
);

INSERT INTO employee (employee_name, street, city)
VALUES
('Raj Shukla', 'Street A', 'Pune'),
('Sandeep Patil', 'Street B', 'Mumbai'),
('Shital Sonje', 'Street C', 'Nashik'),
('Anita Sharma', 'Street D', 'Delhi'),
('Ravi Verma', 'Street E', 'Kolkata'),
('Pooja Singh', 'Street F', 'Pune'),
('Vikas Gupta', 'Street G', 'Mumbai'),
('Amit Kumar', 'Street H', 'Nashik'),
('Sunita Desai', 'Street I', 'Delhi'),
('Manoj Mehta', 'Street J', 'Kolkata');

INSERT INTO company (company_name, city) VALUES
('First Bank Corporation', 'Pune'),
('Small Bank Corporation', 'Mumbai'),
('Special Bank Corporation', 'Nashik');

INSERT INTO works (employee_name, company_name,
salary) VALUES
('Raj Shukla', 'First Bank Corporation', 60000),
('Sandeep Patil', 'Small Bank Corporation', 45000),
('Shital Sonje', 'Special Bank Corporation', 50000),
('Anita Sharma', 'First Bank Corporation', 55000),
('Ravi Verma', 'First Bank Corporation', 65000),
('Pooja Singh', 'Small Bank Corporation', 48000),
('Vikas Gupta', 'Special Bank Corporation', 52000),
('Amit Kumar', 'First Bank Corporation', 58000),
('Sunita Desai', 'Small Bank Corporation', 47000),
('Manoj Mehta', 'Special Bank Corporation', 53000);

INSERT INTO manages (employee_name, manager_name)
VALUES
('Raj Shukla', 'Anita Sharma'),
('Sandeep Patil', 'Ravi Verma'),
('Shital Sonje', 'Pooja Singh'),
('Anita Sharma', 'Ravi Verma'),
('Ravi Verma', 'Anita Sharma');

-- Write SQL queries for the following

-- 1. Find the names of all employees who work for
First Bank Corporation.
SELECT employee_name
FROM works
```

```sql
WHERE company_name = 'First Bank Corporation';

-- 2. Find the names and cities of residence of all
employees who work for First Bank Corporation.
SELECT e.employee_name, e.city
FROM employee e
JOIN works w ON e.employee_name = w.employee_name
WHERE w.company_name = 'First Bank Corporation';

-- 3. Find the names, street addresses, and cities of
residence of all employees who work for First Bank
Corporation and earn more than Rs.10,000.
SELECT e.employee_name, e.street, e.city
FROM employee e
JOIN works w ON e.employee_name = w.employee_name
WHERE w.company_name = 'First Bank Corporation' AND
w.salary > 10000;

-- 4. Find all employees in the database who live in
the same cities as the companies for which they work.
SELECT e.employee_name
FROM employee e
JOIN works w ON e.employee_name = w.employee_name
JOIN company c ON w.company_name = c.company_name
WHERE e.city = c.city;

-- 5. Find all employees in the database who live in
the same cities and on the same streets as do their
managers.
SELECT e1.employee_name
FROM manages m
JOIN employee e1 ON m.employee_name =
e1.employee_name
JOIN employee e2 ON m.manager_name = e2.employee_name
```

```
WHERE e1.city = e2.city AND e1.street = e2.street;

-- 6. Find all employees in the database who do not
work for First Bank Corporation.
SELECT e.employee_name
FROM employee e
LEFT JOIN works w ON e.employee_name =
w.employee_name AND w.company_name = 'First Bank
Corporation'
WHERE w.company_name IS NULL;

-- 7. Find all employees in the database who earn
more than each employee of Small Bank Corporation.
SELECT e.employee_name
FROM employee e
WHERE EXISTS (
    SELECT *
    FROM works w
    WHERE w.company_name = 'Small Bank Corporation'
    AND e.employee_name != w.employee_name
    AND e.employee_name IN (
        SELECT w2.employee_name
        FROM works w2
        WHERE w2.salary > (SELECT MAX(w3.salary) FROM
works w3 WHERE w3.company_name = 'Small Bank
Corporation')
    )
);

-- 8. Find all companies located in every city in
which Small Bank Corporation is located.
SELECT c.company_name
FROM company c
WHERE NOT EXISTS (
    SELECT *
    FROM company s
    WHERE s.company_name = 'Small Bank Corporation'
    AND s.city NOT IN (SELECT city FROM company WHERE
company_name = c.company_name)
);

-- 9. Find all employees who earn more than the
average salary of all employees of their company.
SELECT e.employee_name
FROM employee e
JOIN works w ON e.employee_name = w.employee_name
WHERE w.salary > (
    SELECT AVG(w2.salary)
    FROM works w2
    WHERE w2.company_name = w.company_name
);

-- 10. Find the company that has the most employees.
SELECT company_name
FROM works
GROUP BY company_name
ORDER BY COUNT(employee_name) DESC
LIMIT 1;

-- 11. Find the company that has the smallest
payroll.
SELECT company_name
FROM works
GROUP BY company_name
ORDER BY SUM(salary) ASC
LIMIT 1;
```

```sql
-- 12. Find those companies whose employees earn a
higher salary, on average, than the average salary at
First Bank Corporation.
SELECT c.company_name
FROM company c
JOIN works w ON c.company_name = w.company_name
GROUP BY c.company_name
HAVING AVG(w.salary) > (SELECT AVG(salary) FROM works
WHERE company_name = 'First Bank Corporation');
```

Assignmnet 4

-- Step 1: Create the borrow table

```
CREATE TABLE borrow (
    rollin INT,
    name VARCHAR2(20),
    dt_issue DATE,
    nameofbook VARCHAR2(50),
    status CHAR(1)
);
```

-- Step 2: Insert data into borrow table

INSERT INTO borrow VALUES (1, 'aa', TO_DATE('20-Aug-2024', 'DD-Mon-YYYY'), 'dbms', 'I');

```
INSERT INTO borrow VALUES (2, 'cc', TO_DATE('20-Jul-2024', 'DD-Mon-
YYYY'), 'db', 'I');
INSERT INTO borrow VALUES (3, 'bb', TO_DATE('30-Jul-2024', 'DD-Mon-
YYYY'), 'db1', 'I');

-- Step 3: Create the fine table
CREATE TABLE fine (
   roll INT,
   rdate DATE,
   fineamt INT
);

-- Step 4: PL/SQL Block to calculate fine
DECLARE
   roll INT := :r; -- Bind variable or assign a roll number value here
   famount INT := 0;
   idate DATE;
   days INT;
   sta CHAR(1);
BEGIN
   -- Retrieve dt_issue and status for the specified roll number
   SELECT dt_issue, status INTO idate, sta FROM borrow WHERE rollin =
roll;
```

```
-- Check if the book is issued
IF sta = 'I' THEN
   -- Calculate the number of days since the issue date
   days := SYSDATE - idate;
   DBMS_OUTPUT.PUT_LINE('No of days: ' || days);

   -- Fine calculation based on number of overdue days
   IF days >= 15 AND days <= 30 THEN
      famount := (days - 15) * 5;
   ELSIF days > 30 THEN
      famount := (15 * 5) + ((days - 30) * 50);
   ELSE
      famount := 0;
   END IF;

   DBMS_OUTPUT.PUT_LINE('Fine amount: ' || famount);

   -- Insert the calculated fine into the fine table
   INSERT INTO fine VALUES (roll, SYSDATE, famount);

   -- Update the status to 'R' in the borrow table
```

```
        UPDATE borrow SET status = 'R' WHERE rollin = roll;
    ELSE
        DBMS_OUTPUT.PUT_LINE('Status is already R');
    END IF;
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.PUT_LINE('Roll number does not exist');
END;
```

Assignment 5

```
-- Step 1: Create the employee table
CREATE TABLE employee (
    id INT,
    name VARCHAR2(20),
    salary NUMBER,
    hire_date DATE
);

-- Step 2: Insert data into employee table
```

```
INSERT INTO employee VALUES (1, 'aaa', 20000, TO_DATE('2-Jan-2000',
'DD-Mon-YYYY'));
INSERT INTO employee VALUES (2, 'bbb', 30000, TO_DATE('2-Jan-2020',
'DD-Mon-YYYY'));
INSERT INTO employee VALUES (3, 'ccc', 20000, TO_DATE('2-Jul-2015',
'DD-Mon-YYYY'));

-- Step 3: PL/SQL Block to loop through employees and print details
DECLARE
   CURSOR ecur IS SELECT * FROM employee;
   erec ecur%ROWTYPE;
   yr NUMBER;
   incentive NUMBER := 0;
BEGIN
   OPEN ecur;
   LOOP
      FETCH ecur INTO erec;
      EXIT WHEN ecur%NOTFOUND;

      -- Calculate years of service
      yr := (SYSDATE - erec.hire_date) / 365;

      -- Calculate incentive based on years of service
```

```
);

      IF yr > 10 THEN
         incentive := erec.salary * 0.1;
      ELSIF yr > 5 THEN
         incentive := erec.salary * 0.05;
      ELSE
         incentive := 0;
      END IF;

      -- Display employee details and incentive
      DBMS_OUTPUT.PUT_LINE(erec.id || ' ' || erec.name || ' ' ||
erec.salary);
      DBMS_OUTPUT.PUT_LINE('Incentive: ' || incentive);
   END LOOP;
   CLOSE ecur;
END;
```

Assignment 6

```
-- Creating the student table
CREATE TABLE student (
   roll NUMBER,
   name VARCHAR2(20)
);
```

```
-- Inserting data into student table
INSERT INTO student VALUES (1, 'aaa');
INSERT INTO student VALUES (2, 'bbb');
INSERT INTO student VALUES (3, 'ccc');

-- Creating the student_audit table
CREATE TABLE student_audit (
    roll NUMBER,
    name VARCHAR2(20),
    action VARCHAR2(20),
    changedate DATE
);

-- Creating the before_student_update trigger to log changes
CREATE OR REPLACE TRIGGER before_student_update
BEFORE UPDATE ON student
FOR EACH ROW
BEGIN
    INSERT INTO student_audit (roll, name, action, changedate)
    VALUES (:OLD.roll, :OLD.name, 'update', SYSDATE);
END;
```

-- Updating student table to trigger the audit

UPDATE student

SET name = 'XYZ'

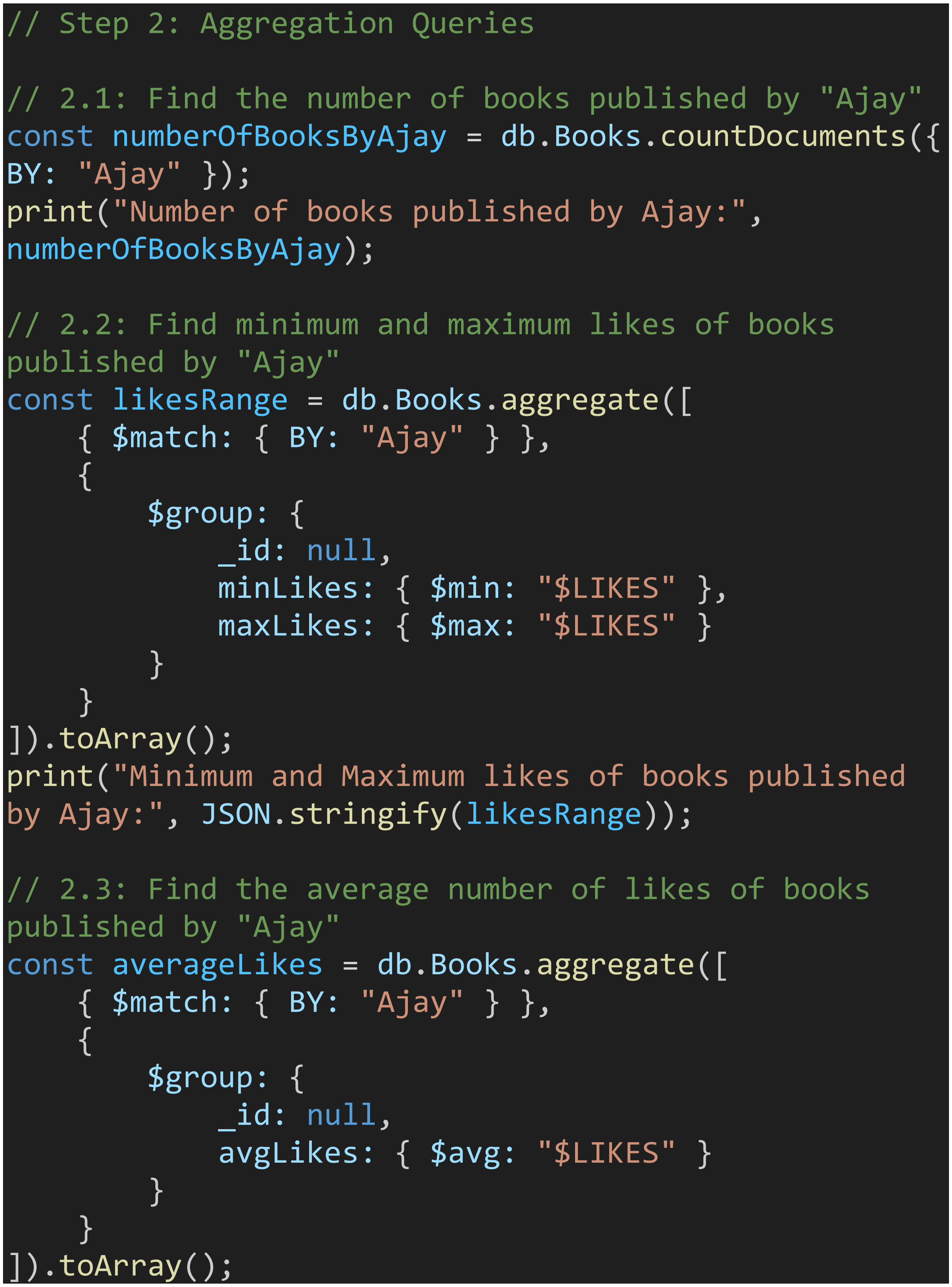
WHERE roll = 2;

Assignment 9

```
// Step 1: Insert multiple documents into the books
collection
db.books.insertMany([{
        TITLE: "mongodb",
        DESCRIPTION: "A NoSQL database.",
        BY: "Ajay",
        URL: "http://mongodb.com",
        TAGS: ["database", "nosql"],
        LIKES: 15
    },
    {
        TITLE: "Introduction to MongoDB",
        DESCRIPTION: "Learn the basics of MongoDB.",
        BY: "Ajay",
        URL: "http://mongodb-intro.com",
        TAGS: ["mongodb", "nosql"],
        LIKES: 25
    },
    {
        TITLE: "Advanced MongoDB",
        DESCRIPTION: "Deep dive into MongoDB.",
        BY: "Rahul",
        URL: "http://mongodb-advanced.com",
        TAGS: ["mongodb", "database"],
        LIKES: 5
    },
    {
        TITLE: "NoSQL Overview",
        DESCRIPTION: "Understanding NoSQL 
databases.",
        BY: "Priya",
        URL: "http://nosql-overview.com",
        TAGS: ["nosql", "database"],
        LIKES: 50
    },
    {
        TITLE: "Database Fundamentals",
        DESCRIPTION: "Basics of databases.",
        BY: "Ajay",
        URL: "http://db-fundamentals.com",
        TAGS: ["database"],
        LIKES: 10
    }
]);

// Step 2: Insert a single document with additional 
fields
db.books.insertOne({
    TITLE: "mongodb",
    DESCRIPTION: "A NoSQL database.",
    BY: "Ajay",
    URL: "http://mongodb.com",
    TAGS: ["database", "nosql"],
    LIKES: 15,
    user: "User1",
    comments: "Great resource!"
});

// Step 3: Find documents with specific criteria
db.books.find({ TITLE: "mongodb" }); // Find by title
db.books.find({ $or: [{ BY: "Ajay" }, { TITLE:
"mongodb" }] }); // Find by author or title
db.books.find({ TITLE: "mongodb", BY: "Ajay" }); //
Find by title and author
db.books.find({ LIKES: { $gt: 10 } }); // Find with
likes greater than 10
db.books.find({ LIKES: { $gt: 100 }, $or: [{ TITLE:
"mongodb" }, { BY: "Ajay" }] }); // Find with complex
condition

// Step 4: Update a document's title
db.books.updateOne({ TITLE: "mongodb" }, { $set: {
TITLE: "mongodb overview" } });

// Step 5: Delete a document by title
db.books.deleteOne({ TITLE: "NoSQL Overview" });

// Step 6: Pagination and Sorting
db.books.find({ BY: "Ajay" }).limit(2); // Limit
results
db.books.find({ BY: "Ajay" }).skip(1).limit(1); //
Skip and limit results
db.books.find().sort({ TITLE: 1 }); // Sort by title
in ascending order

// Step 7: Save (or insert) a new document
db.books.save({
    TITLE: "New MongoDB Book",
    DESCRIPTION: "A new perspective on MongoDB.",
    BY: "Ajay",
    URL: "http://new-mongodb-book.com",
    TAGS: ["mongodb", "database"],
    LIKES: 30
});
```

Assignment 10

```
// Connect to the MongoDB server and select the
database
use db10; // Replace 'db10' with your actual database
name

// Step 1: Create the Books collection and insert
sample data
db.Books.insertMany([
    { TITLE: "Book 1", DESCRIPTION: "Description 1",
BY: "Ajay", URL: "http://example.com/book1", TAGS:
["tag1", "tag2"], LIKES: 10 },
    { TITLE: "Book 2", DESCRIPTION: "Description 2",
BY: "Ajay", URL: "http://example.com/book2", TAGS:
["tag1"], LIKES: 5 },
    { TITLE: "Book 3", DESCRIPTION: "Description 3",
BY: "Ajay", URL: "http://example.com/book3", TAGS:
["tag2"], LIKES: 15 },
    { TITLE: "Book 4", DESCRIPTION: "Description 4",
BY: "Someone Else", URL: "http://example.com/book4",
TAGS: ["tag3"], LIKES: 20 },
    { TITLE: "Book 5", DESCRIPTION: "Description 5",
BY: "Ajay", URL: "http://example.com/book5", TAGS:
["tag2", "tag3"], LIKES: 8 }
]);
```



```
// Step 2: Aggregation Queries

// 2.1: Find the number of books published by "Ajay"
const numberOfBooksByAjay = db.Books.countDocuments({
BY: "Ajay" });
print("Number of books published by Ajay:",
numberOfBooksByAjay);

// 2.2: Find minimum and maximum likes of books
published by "Ajay"
const likesRange = db.Books.aggregate([
    { $match: { BY: "Ajay" } },
    {
        $group: {
            _id: null,
            minLikes: { $min: "$LIKES" },
            maxLikes: { $max: "$LIKES" }
        }
    }
]).toArray();
print("Minimum and Maximum likes of books published
by Ajay:", JSON.stringify(likesRange));

// 2.3: Find the average number of likes of books
published by "Ajay"
const averageLikes = db.Books.aggregate([
    { $match: { BY: "Ajay" } },
    {
        $group: {
            _id: null,
            avgLikes: { $avg: "$LIKES" }
        }
    }
]).toArray();
```

```
print("Average number of likes of books published by
Ajay:", JSON.stringify(averageLikes));

// 2.4: Find the first and last book (alphabetically
by title) published by "Ajay"
const firstAndLastBook = db.Books.aggregate([
    { $match: { BY: "Ajay" } },
    { $sort: { TITLE: 1 } }, // Sort by title in
ascending order
    {
        $group: {
            _id: null,
            firstBook: { $first: "$TITLE" },
            lastBook: { $last: "$TITLE" }
        }
    }
]).toArray();
print("First and Last book published by Ajay:",
JSON.stringify(firstAndLastBook));

// Step 3: Create an index on the author name
db.Books.createIndex({ BY: 1 });

// Step 4: Display books published by "Ajay" and
check index usage
const ajayBooks = db.Books.find({ BY: "Ajay"
}).hint({ BY: 1 }).toArray();
print("Books published by Ajay:",
JSON.stringify(ajayBooks));
```