**INDIAN INSTITUTE OF INFORMATION TECHNOLOGY SURAT**



# LAB REPORT

### on

**ADVANCE DATABASE MANAGEMENT (CS 604)**

**Submitted by**

### [KALPAN BARIYA] (UI21CS28)

**Course Faculty MR. RISHI SHARMA DR. REEMA PATEL**

**Department of Computer Science and Engineering Indian Institute of Information Technology Surat Gujarat-394190, India**

**APRIL - 2024**

# Table of Contents

| **Exp. No** | **Name of the Experiments** | **Page number** | **Date of Experiment** | **Date of Submission** | **Sign** |
| --- | --- | --- | --- | --- | --- |
| **1** | Implement Basic commands of SQL | 03-13 | 21-01-2024 | 29-01-2024 |  |
| **2** | Implement PL/SQL commands | 14-19 | 21-01-2024 | 29-01-2024 |  |
| **3** | Write PL/SQL code as specified in tasks | 20-29 | 30-01-2024 | 5-02-2024 |  |
| **4** | Create TRIGGERS and implement given task | 30-33 | 06-02-2024 | 06-02-2024 |  |
| **5** | Analyze ecommerce Inventory | 34-37 | 13-02-2024 | 22-02-2024 |  |
| **6** | Object Oriented PL/SQL | 38-44 | 20-03-2024 | 16-04-2024 |  |
| **7** | PostgreSQL Practice | 45-47 | 10-04-2024 | 16-04-2024 |  |

**Assignment 1**

### Aim:

Create a Database for an Organization and create the following tables in the Organization Database:

Employee(EMP\_ID(PK), FIRST\_NAME, LAST\_NAME, SALARY, JOINING\_DATE, EPARTMENT)

Bonus (EMP\_REF\_ID(FK EMP\_ID), BONUS\_AMOUNT, BONUS\_DATE) Title (EMP\_REF\_ID(FKEMP\_ID), EMP\_TITLE, AFFECTED\_FROM)

Insert a minimum of 50 records in each table.

Retrieve the following information from the Organization database:

1. SQL query to print all Employee details from the Employee table order by FIRST\_NAME Ascending and DEPARTMENT Descending.
2. SQL query to fetch the count of employees working in the department ‘Admin’.
3. SQL query to fetch Employee names with salaries >= 50000 and <= 100000.
4. SQL query to print details of the Workers who are also Managers.
5. SQL query to fetch duplicate records having matching data in some fields of a table.
6. SQL query to show only even rows from a table.
7. SQL query to show records from one table that another table does not have. Find employees in employee table that do not exist in bonus table (i.e. who did not get bonus)
8. SQL query to show the to pn(say10) records of a table.
9. Find people who have the same salary
10. SQL query to fetch the first 50% records from a table.
11. Find the highest 2 salaries without LIMIT or TOP.
12. Create a trigger to ensure that no employee of age less than 18 can be inserted in the database.
13. Create a trigger which will work before deletion in employee table and create a duplicate copy of the record in another table employee\_backup.
14. Create a trigger to count number of new tupples inserted using each insert statement.

### MySQL Queries & Output :

CREATE DATABASE IF NOT EXISTS kp1;

USE kp1;

CREATE TABLE IF NOT EXISTS Employee (

    EMP\_ID INTEGER PRIMARY KEY,

    FIRST\_NAME VARCHAR(20),

    LAST\_NAME VARCHAR(20),

    SALARY INTEGER,

    JOINING\_DATE DATE,

    DEPARTMENT VARCHAR(50)

);

CREATE TABLE IF NOT EXISTS bonus(

    BONUS\_AMOUNT INTEGER,

    BONUS\_DATE DATE,

    EMP\_REF\_ID INTEGER,

    FOREIGN KEY (EMP\_REF\_ID) REFERENCES Employee(EMP\_ID)

);

CREATE TABLE IF NOT EXISTS title(

    EMP\_TITLE VARCHAR(50),

    AFFECTED\_FROM DATE,

    EMP\_REF\_ID INTEGER,

    FOREIGN KEY (EMP\_REF\_ID) REFERENCES Employee(EMP\_ID)

);

INSERT INTO Employee (EMP\_ID, FIRST\_NAME, LAST\_NAME, SALARY, JOINING\_DATE, DEPARTMENT)

VALUES

  (1, 'John', 'Doe', 50000, '2022-01-01', 'HR'),

  (2, 'Jane', 'Smith', 60000, '2022-02-01', 'IT'),

  (3, 'Alice', 'Johnson', 55000, '2022-03-01', 'Finance'),

  (4, 'Bob', 'Williams', 70000, '2022-04-01', 'Marketing'),

  (5, 'Eva', 'Jones', 48000, '2022-05-01', 'Sales'),

  (6, 'Mike', 'Brown', 52000, '2022-06-01', 'IT'),

  (7, 'Sara', 'Miller', 63000, '2022-07-01', 'Finance'),

  (8, 'Tom', 'Anderson', 55000, '2022-08-01', 'HR'),

  (9, 'Emily', 'Davis', 58000, '2022-09-01', 'Marketing'),

  (10, 'Chris', 'Taylor', 67000, '2022-10-01', 'Sales'),

  (11, 'David', 'Johnson', 59000, '2022-11-01', 'IT'),

  (12, 'Laura', 'White', 72000, '2022-12-01', 'HR'),

  (13, 'Alex', 'Turner', 60000, '2023-01-01', 'Sales'),

  (14, 'Grace', 'Smith', 55000, '2023-02-01', 'IT'),

  (15, 'Ryan', 'Williams', 68000, '2023-03-01', 'Finance'),

  (16, 'Jessica', 'Miller', 50000, '2023-04-01', 'Marketing'),

  (17, 'Eric', 'Brown', 75000, '2023-05-01', 'Sales'),

  (18, 'Olivia', 'Jones', 47000, '2023-06-01', 'IT'),

  (19, 'Michael', 'Anderson', 61000, '2023-07-01', 'HR'),

  (20, 'Sophia', 'Davis', 53000, '2023-08-01', 'Marketing'),

  (51, 'Laura', 'Adams', 58000, '2022-01-01', 'Marketing'),

  (52, 'Daniel', 'Perez', 70000, '2022-02-01', 'IT'),

  (53, 'Rachel', 'Smith', 52000, '2022-03-01', 'Sales'),

  (54, 'Mark', 'Johnson', 60000, '2022-04-01', 'Finance'),

  (55, 'Sophie', 'White', 48000, '2022-05-01', 'HR'),

  (56, 'Edward', 'Martinez', 67000, '2022-06-01', 'IT'),

  (57, 'Chloe', 'Turner', 55000, '2022-07-01', 'Finance'),

  (58, 'Oliver', 'Davis', 72000, '2022-08-01', 'HR'),

  (59, 'Mia', 'Walker', 63000, '2022-09-01', 'Marketing'),

  (60, 'Ethan', 'Hill', 59000, '2022-10-01', 'Sales'),

  (61, 'Emma', 'Garcia', 55000, '2022-11-01', 'IT'),

  (62, 'Liam', 'Clark', 60000, '2022-12-01', 'HR'),

  (63, 'Ava', 'Baker', 52000, '2023-01-01', 'Marketing'),

  (64, 'Noah', 'Ward', 65000, '2023-02-01', 'Finance'),

  (65, 'Isabella', 'Fisher', 53000, '2023-03-01', 'IT'),

  (66, 'Lucas', 'Harrison', 70000, '2023-04-01', 'Sales'),

  (67, 'Aria', 'Gomez', 48000, '2023-05-01', 'HR'),

  (68, 'Liam', 'Clark', 55000, '2023-06-01', 'Finance'),

  (69, 'Mila', 'Russell', 63000, '2023-07-01', 'Marketing'),

  (70, 'James', 'Gordon', 59000, '2023-08-01', 'Sales'),

  (71, 'Sophie', 'Thomas', 58000, '2023-01-01', 'Marketing'),

  (72, 'William', 'Moore', 70000, '2023-02-01', 'IT'),

  (73, 'Ava', 'Parker', 52000, '2023-03-01', 'Sales'),

  (74, 'Daniel', 'Barnes', 60000, '2023-04-01', 'Finance'),

  (75, 'Mia', 'Brown', 48000, '2023-05-01', 'HR'),

  (76, 'Liam', 'Ward', 67000, '2023-06-01', 'IT'),

  (77, 'Emma', 'Baker', 55000, '2023-07-01', 'Finance'),

  (78, 'Oliver', 'Taylor', 72000, '2023-08-01', 'HR'),

  (79, 'Isabella', 'Russell', 63000, '2023-09-01', 'Marketing'),

  (80, 'Lucas', 'Gomez', 59000, '2023-10-01', 'Sales');

INSERT INTO bonus (BONUS\_AMOUNT, BONUS\_DATE, EMP\_REF\_ID)

VALUES

  (1000, '2022-02-15', 1),

  (1500, '2022-03-01', 2),

  (1200, '2022-04-01', 3),

  (800, '2022-05-01', 4),

  (1300, '2022-06-01', 5),

  (900, '2022-07-01', 6),

  (1100, '2022-08-01', 7),

  (1000, '2022-09-01', 8),

  (1200, '2022-10-01', 9),

  (1400, '2022-11-01', 10),

  (950, '2022-12-01', 11),

  (1050, '2023-01-01', 12),

  (1150, '2023-02-01', 13),

  (1250, '2023-03-01', 14),

  (1350, '2023-04-01', 15),

  (1450, '2023-05-01', 16),

  (950, '2023-06-01', 17),

  (1050, '2023-07-01', 18),

  (1150, '2023-08-01', 19),

  (1250, '2023-09-01', 20),

  (1100, '2022-02-15', 51),

  (950, '2022-03-01', 52),

  (1200, '2022-04-01', 53),

  (850, '2022-05-01', 54),

  (1300, '2022-06-01', 55),

  (900, '2022-07-01', 56),

  (1000, '2022-08-01', 57),

  (1150, '2022-09-01', 58),

  (1050, '2022-10-01', 59),

  (1400, '2022-11-01', 60),

  (1200, '2022-12-01', 61),

  (1300, '2023-01-01', 62),

  (1000, '2023-02-01', 63),

  (1100, '2023-03-01', 64),

  (900, '2023-04-01', 65),

  (1250, '2023-05-01', 66),

  (850, '2023-06-01', 67),

  (950, '2023-07-01', 68),

  (1150, '2023-08-01', 69),

  (1050, '2023-09-01', 70),

  (1100, '2023-02-15', 71),

  (950, '2023-03-01', 72),

  (1200, '2023-04-01', 73),

  (850, '2023-05-01', 74),

  (1300, '2023-06-01', 75),

  (900, '2023-07-01', 76),

  (1000, '2023-08-01', 77),

  (1150, '2023-09-01', 78),

  (1050, '2023-10-01', 79),

  (1400, '2023-11-01', 80);

INSERT INTO title (EMP\_TITLE, AFFECTED\_FROM, EMP\_REF\_ID)

VALUES

  ('Manager', '2022-02-01', 1),

  ('Developer', '2022-03-01', 2),

  ('Analyst', '2022-04-01', 3),

  ('Coordinator', '2022-05-01', 4),

  ('Sales Representative', '2022-06-01', 5),

  ('Database Administrator', '2022-07-01', 6),

  ('Financial Analyst', '2022-08-01', 7),

  ('HR Specialist', '2022-09-01', 8),

  ('Marketing Manager', '2022-10-01', 9),

  ('Sales Manager', '2022-11-01', 10),

  ('IT Specialist', '2022-12-01', 11),

  ('Financial Planner', '2023-01-01', 12),

  ('Sales Analyst', '2023-02-01', 13),

  ('Software Engineer', '2023-03-01', 14),

  ('Marketing Coordinator', '2023-04-01', 15),

  ('HR Manager', '2023-05-01', 16),

  ('Sales Coordinator', '2023-06-01', 17),

  ('Database Analyst', '2023-07-01', 18),

  ('Financial Manager', '2023-08-01', 19),

  ('Marketing Analyst', '2023-09-01', 20),

  ('Manager', '2022-02-01', 51),

  ('Developer', '2022-03-01', 52),

  ('Analyst', '2022-04-01', 53),

  ('Coordinator', '2022-05-01', 54),

  ('Sales Representative', '2022-06-01', 55),

  ('Database Administrator', '2022-07-01', 56),

  ('Financial Analyst', '2022-08-01', 57),

  ('HR Specialist', '2022-09-01', 58),

  ('Marketing Manager', '2022-10-01', 59),

  ('Sales Manager', '2022-11-01', 60),

  ('IT Specialist', '2022-12-01', 61),

  ('Financial Planner', '2023-01-01', 62),

  ('Sales Analyst', '2023-02-01', 63),

  ('Software Engineer', '2023-03-01', 64),

  ('Marketing Coordinator', '2023-04-01', 65),

  ('HR Manager', '2023-05-01', 66),

  ('Sales Coordinator', '2023-06-01', 67),

  ('Database Analyst', '2023-07-01', 68),

  ('Financial Manager', '2023-08-01', 69),

  ('Marketing Analyst', '2023-09-01', 70),

  ('Manager', '2023-03-01', 71),

  ('Developer', '2023-04-01', 72),

  ('Analyst', '2023-05-01', 73),

  ('Coordinator', '2023-06-01', 74),

  ('Sales Representative', '2023-07-01', 75),

  ('Database Administrator', '2023-08-01', 76),

  ('Financial Analyst', '2023-09-01', 77),

  ('HR Specialist', '2023-10-01', 78),

  ('Marketing Manager', '2023-11-01', 79),

  ('Sales Manager', '2023-12-01', 80);

SELECT \* FROM Employee ORDER BY FIRST\_NAME;

SELECT \* FROM Employee ORDER BY FIRST\_NAME DESC;

SELECT COUNT(\*) AS C FROM EMPLOYEE WHERE DEPARTMENT='IT';

SELECT FIRST\_NAME, LAST\_NAME FROM Employee WHERE SALARY BETWEEN 50000 AND 100000;

SELECT FIRST\_NAME, LAST\_NAME, COUNT(\*) FROM Employee GROUP BY FIRST\_NAME, LAST\_NAME HAVING COUNT(\*) > 1;

SELECT \* FROM Employee WHERE MOD(EMP\_ID, 2) = 0;

SELECT Employee.\* FROM Employee LEFT JOIN bonus ON Employee.EMP\_ID = bonus.EMP\_REF\_ID WHERE bonus.EMP\_REF\_ID IS NULL;

SELECT \* FROM Employee ORDER BY EMP\_ID LIMIT 10;

SELECT FIRST\_NAME, LAST\_NAME, SALARY FROM Employee GROUP BY FIRST\_NAME, LAST\_NAME,SALARY HAVING COUNT(\*) > 1;

SELECT \* FROM Employee WHERE EMP\_ID <= (SELECT COUNT(\*) / 2 FROM Employee);

SELECT EMP\_ID, FIRST\_NAME, LAST\_NAME, SALARY

FROM (

    SELECT EMP\_ID, FIRST\_NAME, LAST\_NAME, SALARY,

           DENSE\_RANK() OVER (ORDER BY SALARY DESC) AS salary\_rank

    FROM Employee

) ranked\_salaries

WHERE salary\_rank <= 2;

DELIMITER //

CREATE TRIGGER check\_age BEFORE INSERT ON Employee

FOR EACH ROW

BEGIN

    IF (YEAR(CURRENT\_DATE) - YEAR(NEW.JOINING\_DATE)) < 18 THEN

        SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'Employee must be at least 18 years old';

    END IF;

END;

//

DELIMITER ;

DELIMITER //

CREATE TRIGGER backup\_employee BEFORE DELETE ON Employee

FOR EACH ROW

BEGIN

    INSERT INTO employee\_backup (EMP\_ID, FIRST\_NAME, LAST\_NAME, SALARY, JOINING\_DATE, DEPARTMENT)

    VALUES (OLD.EMP\_ID, OLD.FIRST\_NAME, OLD.LAST\_NAME, OLD.SALARY, OLD.JOINING\_DATE, OLD.DEPARTMENT);

END;

//

DELIMITER ;

DELIMITER //

CREATE TRIGGER count\_inserted\_tuples

BEFORE INSERT ON Employee

FOR EACH ROW

BEGIN

    -- Increment the counter for each new tuple insertion

    SET @inserted\_tuples\_count = @inserted\_tuples\_count + 1 ;

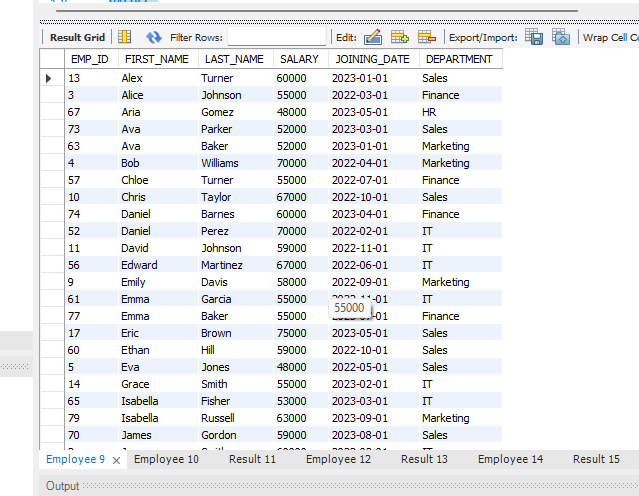
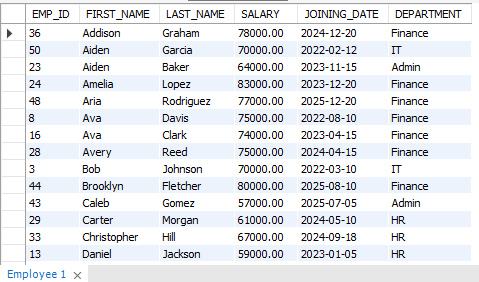
END;

SET @inserted\_tuples\_count = 0;

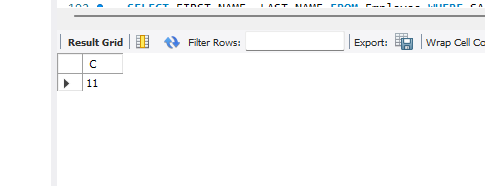
//

DELIMITER ;

**-- 1. SQL query to print all Employee details from the Employee table order by FIRST\_NAME Ascending and DEPARTMENT Descending.**

SELECT \* FROM Employee ORDER BY FIRST\_NAME ASC, DEPARTMENT DESC;

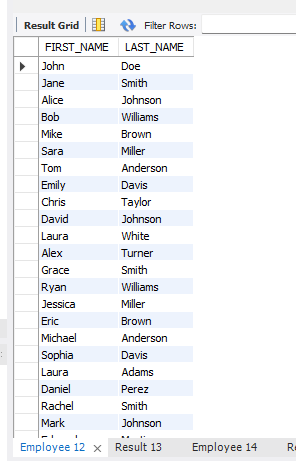
**-- 2 SQL query to fetch the count of employees working in the department ‘Admin’.**



SELECT COUNT(\*) FROM Employee WHERE DEPARTMENT = 'Admin';

**-- 3. SQL query to fetch Employee names with salaries >= 50000 and <= 100000.**

SELECT FIRST\_NAME, LAST\_NAME FROM Employee WHERE SALARY BETWEEN 50000 AND 100000;



**-- 4. SQL query to print details of the Workers who are also Managers.**

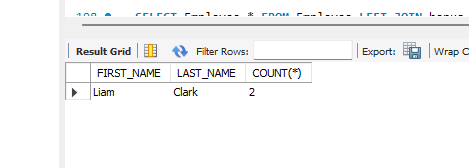
SELECT E.\* FROM Employee E

JOIN Title T ON E.EMP\_ID = T.EMP\_REF\_ID AND T.EMP\_TITLE = 'Manager



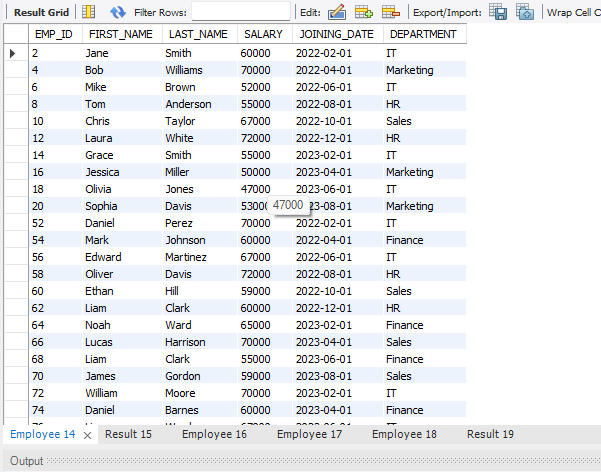
**-- 5. SQL query to fetch duplicate records having matching data in some fields of a table.**

SELECT EMP\_ID, COUNT(\*) FROM Employee GROUP BY EMP\_ID HAVING COUNT(\*) > 1;



**-- 6. SQL query to show only even rows from a table.**

SELECT \* FROM Employee WHERE MOD(EMP\_ID, 2) = 0;



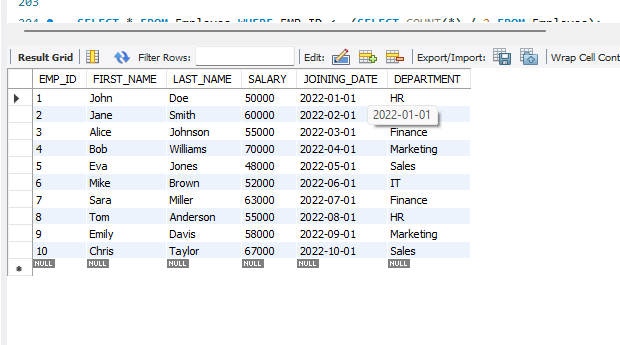
**-- 7. SQL query to show records from one table that another table does not have. Find employees in employee table that do not exist in bonus table.**

DELETE FROM Bonus WHERE EMP\_REF\_ID = 50;

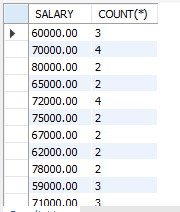
SELECT \* FROM Employee WHERE EMP\_ID NOT IN (SELECT EMP\_REF\_ID FROM Bonus);

**-- 8. SQL query to show the top n (say 10) records of a table.**

SELECT \* FROM Employee LIMIT 10;



**-- 9. Find people who have the same salary.**

SELECT SALARY, COUNT(\*) FROM Employee GROUP BY SALARY HAVING COUNT(\*) > 1;

**-- 10. SQL query to fetch the first 50% records from a table.**

SELECT \* FROM ( SELECT \*,

ROW\_NUMBER() OVER (ORDER BY EMP\_ID) AS rn

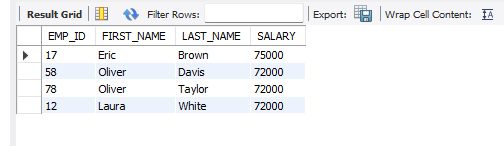
FROM Employee

) AS subquery

WHERE rn <= (SELECT COUNT(\*)/2 FROM Employee);

**-- 11. Find the highest 2 salaries without LIMIT or TOP.**

SELECT DISTINCT SALARY FROM Employee ORDER BY SALARY DESC LIMIT 2;



**-- 12. Create a trigger to ensure that no employee joining date less than current date can be inserted in the database.**

DELIMITER //

CREATE TRIGGER before\_insert\_employee BEFORE INSERT ON Employee

FOR EACH ROW BEGIN

IF NEW.JOINING\_DATE >= CURDATE() THEN SIGNAL SQLSTATE '45000'

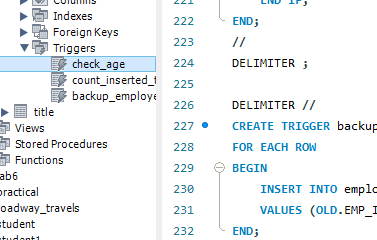
SET MESSAGE\_TEXT = 'Joining date must be less than the current date.'; END IF;

END;

// DELIMITER ;

-- TSETING 12

INSERT INTO Employee VALUES (51, 'PATEL', 'Fletcher', 80000.00, '2025-08-10', 'Finance');



**-- 13. Create a trigger which will work before deletion in employee table and create a duplicate copy of the record in another table employee\_backup.**

DELIMITER //

CREATE TRIGGER before\_delete\_employee BEFORE DELETE ON Employee

FOR EACH ROW BEGIN

INSERT INTO employee\_backup (EMP\_ID, FIRST\_NAME, LAST\_NAME, SALARY, JOINING\_DATE, DEPARTMENT)

VALUES (OLD.EMP\_ID, OLD.FIRST\_NAME, OLD.LAST\_NAME, OLD.SALARY, OLD.JOINING\_DATE, OLD.DEPARTMENT);

END;

// DELIMITER ;

-- TESING 13

DELETE FROM Title WHERE EMP\_REF\_ID = 50; DELETE FROM Bonus WHERE EMP\_REF\_ID = 50; DELETE FROM Employee WHERE EMP\_ID = 50;

select \* FROM employee\_backup;

**-- 14. Create a trigger to count the number of new tuples inserted using each insert statement.**

DELIMITER //

CREATE TRIGGER after\_insert\_employee AFTER INSERT ON Employee

FOR EACH ROW BEGIN

INSERT INTO insert\_count (table\_name, count) VALUES ('Employee', 1)

ON DUPLICATE KEY UPDATE count = count + 1; END;

// DELIMITER ;

-- TESTING 14

INSERT INTO Employee VALUES (52, 'PATELboss', 'Fletcher', 80000.00, '2021-08-10', 'Finance');

select \* from insert\_count;

### Conclusion:

Here I learned different basic MySQL queries form this assignment. Below attached database

images

# Assignment 2

### Aim:

Write a PL/SQL code block to find total and average of 6 subjects and display the grade.

### Queries & Output :

-- Create the database

CREATE DATABASE studentdata;

-- Use the studentdata database USE studentdata;

-- Create the student table CREATE TABLE student (

serial\_number INT PRIMARY KEY, student\_name VARCHAR(50), subject1 INT,

subject2 INT, subject3 INT, subject4 INT, subject5 INT, subject6 INT

);

-- Insert 10 sample student records

INSERT INTO student (serial\_number, student\_name, subject1, subject2, subject3, subject4, subject5, subject6)

##### VALUES

(1, 'John Doe', 85, 92, 78, 88, 94, 90),

(2, 'Jane Smith', 75, 80, 82, 88, 79, 85),

(3, 'Bob Johnson', 92, 88, 90, 87, 95, 91),

(4, 'Alice Brown', 78, 85, 76, 80, 82, 89),

(5, 'Charlie Davis', 93, 91, 89, 96, 87, 84),

(6, 'Eva White', 86, 92, 88, 75, 80, 92),

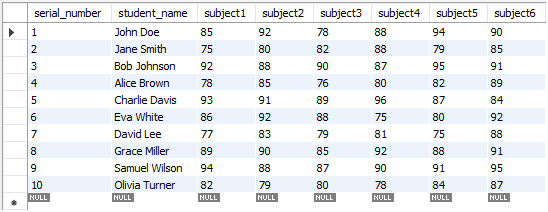
(7, 'David Lee', 77, 83, 79, 81, 75, 88),

(8, 'Grace Miller', 89, 90, 85, 92, 88, 91),

(9, 'Samuel Wilson', 94, 88, 87, 90, 91, 95),

(10, 'Olivia Turner', 82, 79, 80, 78, 84, 87);

-- Select all records from the student table SELECT \* FROM student;



#### -- Create a stored procedure to print "Hello World"

##### DELIMITER //

CREATE PROCEDURE temp() BEGIN

SELECT "Hello World"; END;

##### // DELIMITER ;

-- Call the temp stored procedure CALL temp();



#### -- Create a stored procedure to calculate factorial

##### DELIMITER //

CREATE PROCEDURE facto(IN n INT) BEGIN

DECLARE i INT DEFAULT 1; DECLARE fact INT DEFAULT 1;

factorial: LOOP SET fact = fact \* i; SET i = i + 1;

IF i <= n THEN

ITERATE factorial;

##### END IF;

LEAVE factorial;

##### END LOOP;

SELECT i, fact, n;

##### END;

// DELIMITER ;

call studentdata.facto(5);



#### -- Create a stored function to calculate the average grade for 6 subjects

##### DELIMITER //

CREATE PROCEDURE calculate\_average\_grade(IN score1 INT, IN score2 INT, IN score3 INT, IN score4 INT, IN score5 INT, IN score6 INT)

##### BEGIN

DECLARE average\_score INT; DECLARE total\_score INT; DECLARE avg\_grade VARCHAR(10);

-- Calculate total score

SET total\_score = score1 + score2 + score3 + score4 + score5 + score6;

-- Calculate average score

SET average\_score = total\_score / 6;

-- Calculate the average grade based on the average score IF average\_score >= 90 THEN

SET avg\_grade = 'A';

ELSEIF average\_score >= 70 THEN SET avg\_grade = 'B';

ELSEIF average\_score >= 60 THEN SET avg\_grade = 'C';

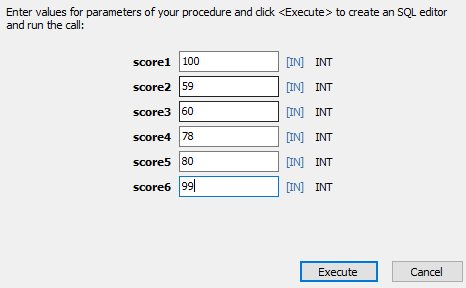
ELSEIF average\_score >= 50 THEN SET avg\_grade = 'D';

##### ELSE

SET avg\_grade = 'E'; END IF;

-- Return the average grade SELECT avg\_grade, average\_score; END;

// DELIMITER ;





#### -- Create a stored procedure to calculate average grade from student table

##### DELIMITER //

CREATE PROCEDURE calculate\_student\_average\_grade(IN student\_id INT) BEGIN

DECLARE score1 INT; DECLARE score2 INT; DECLARE score3 INT; DECLARE score4 INT; DECLARE score5 INT; DECLARE score6 INT;

DECLARE average\_score INT; DECLARE avg\_grade VARCHAR(10);

-- Fetch scores for the specified student\_id from the student table SELECT subject1, subject2, subject3, subject4, subject5, subject6 INTO score1, score2, score3, score4, score5, score6

FROM student

WHERE serial\_number = student\_id;

-- Calculate total score

SET average\_score = (score1 + score2 + score3 + score4 + score5 + score6) / 6;

-- Calculate the average grade based on the average score IF average\_score >= 90 THEN

SET avg\_grade = 'A';

ELSEIF average\_score >= 70 THEN SET avg\_grade = 'B';

ELSEIF average\_score >= 60 THEN SET avg\_grade = 'C';

ELSEIF average\_score >= 50 THEN SET avg\_grade = 'D';

##### ELSE

SET avg\_grade = 'E'; END IF;

-- Return the average grade and average score

SELECT avg\_grade AS grade, average\_score AS avg\_score; END;

##### // DELIMITER ;

call studentdata.calculate\_student\_average\_grade(2);



### Conclusion :

Here I learned about how to do a coding in PL/SQL and create procedure and alter it as well as find factorial, average grade from student database with MySQL query.

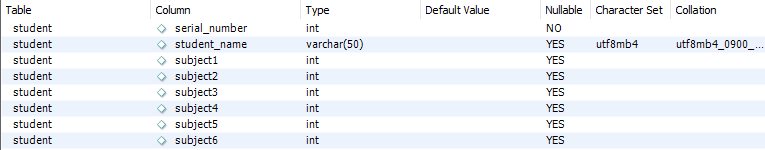


Figure - Database of student table

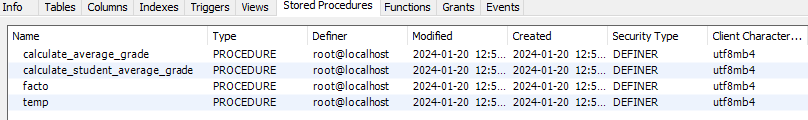


Figure - Procedure using PL/SQL

# Assignment 3

### Aim:

Consider the following table to write PL/SQL code as specified under

Teacher (t\_no, f\_name, l\_name, salary, supervisor, joining\_date, birth\_date, title) Class (class\_no, t\_no, room\_no)

Pay\_scale (Min\_limit, Max\_limit, grade)

1. Accept a range of salary and print the details of teachers from teacher table.
2. By using cursor - Calculate the bonus amount to be given to a teacher depending on the following conditions:
   1. if salary< 10000 then bonus is 10% of the salary.
   2. if salary is between 10000 and 20000 then bonus is 20% of the salary.
   3. if salary is between 20000 and 25000 then bonus is 25% of the salary.
   4. if salary exceeds 25000 then bonus is 30% of the salary.
3. Using a simple LOOP structure, list the first 10 records of the ‘teachers’ table.
4. Accept the room number and display the teacher details like t\_no, f\_name, l\_name, birth\_date, title from table Teacher.

### Queries & Output :

create DATABASE Teachers; use Teachers;

-- Creating Teacher table CREATE TABLE Teacher (

t\_no INT PRIMARY KEY,

f\_name VARCHAR(255), l\_name VARCHAR(255), salary DECIMAL(10, 2), supervisor INT, joining\_date DATE, birth\_date DATE,

title VARCHAR(50)

);

-- Creating Class table CREATE TABLE Class (

class\_no INT PRIMARY KEY, t\_no INT,

room\_no INT,

FOREIGN KEY (t\_no) REFERENCES Teacher(t\_no)

);

-- Creating Pay\_scale table CREATE TABLE Pay\_scale (

Min\_limit DECIMAL(10, 2),

Max\_limit DECIMAL(10, 2), grade VARCHAR(10),

PRIMARY KEY (Min\_limit, Max\_limit)

);

-- Inserting data into Teacher table

INSERT INTO Teacher (t\_no, f\_name, l\_name, salary, supervisor, joining\_date, birth\_date, title) VALUES

(1, 'John', 'Doe', 50000.00, NULL, '2020-01-15', '1980-05-20', 'Professor'),

(2, 'Jane', 'Smith', 60000.00, 1, '2018-03-10', '1985-09-12', 'Associate Professor'),

(3, 'Mark', 'Johnson', 45000.00, 1, '2019-07-22', '1990-11-30', 'Assistant Professor'),

(4, 'Alice', 'Williams', 55000.00, NULL, '2021-02-05', '1982-08-18', 'Professor'),

(5, 'Bob', 'Jones', 70000.00, 2, '2017-06-08', '1975-04-25', 'Professor'),

(6, 'Emily', 'Davis', 48000.00, 3, '2022-09-14', '1988-12-07', 'Assistant Professor'),

(7, 'Michael', 'Brown', 62000.00, 1, '2016-04-30', '1972-03-15', 'Professor'),

(8, 'Samantha', 'Miller', 58000.00, NULL, '2023-11-02', '1983-07-10', 'Associate Professor'),

(9, 'David', 'Anderson', 52000.00, 5, '2020-08-18', '1978-09-28', 'Assistant Professor'),

(10, 'Sophia', 'Garcia', 53000.00, 2, '2019-01-07', '1987-06-03', 'Associate Professor'),

(11, 'Laura', 'Martinez', 8000.00, 1, '2020-04-12', '1982-09-22', 'Assistant Professor'),

(12, 'Daniel', 'Wilson', 15000.00, 3, '2019-08-25', '1975-11-10', 'Associate Professor'),

(13, 'Ella', 'Taylor', 12000.00, 1, '2022-02-18', '1988-05-05', 'Assistant Professor'),

(14, 'Christopher', 'Moore', 25000.00, 2, '2018-06-30', '1980-12-15', 'Professor'),

(15, 'Sophie', 'Lee', 18000.00, NULL, '2021-11-08', '1990-03-28', 'Associate Professor'),

(16, 'Connor', 'Hill', 10000.00, 5, '2017-03-02', '1985-07-18', 'Assistant Professor'),

(17, 'Olivia', 'Allen', 30000.00, 7, '2016-09-14', '1972-10-30', 'Professor'),

(18, 'Jackson', 'Ward', 7000.00, 8, '2023-04-30', '1983-03-25', 'Assistant Professor'),

(19, 'Aria', 'Clark', 11000.00, NULL, '2020-08-18', '1978-05-20', 'Associate Professor'),

(20, 'Logan', 'Evans', 6000.00, 5, '2019-01-07', '1987-11-03', 'Assistant Professor');

-- Inserting data into Class table with different room numbers INSERT INTO Class (class\_no, t\_no, room\_no)

##### VALUES

(101, 1, 201),

(102, 2, 202),

(103, 3, 203),

(104, 4, 204),

(105, 5, 205),

(106, 6, 206),

(107, 7, 207),

(108, 8, 208),

(109, 9, 209),

(110, 10, 210),

(111, 11, 211),

(112, 12, 212),

(113, 13, 213),

(114, 14, 214),

(115, 15, 215),

(116, 16, 216),

(117, 17, 217),

(118, 18, 218),

(119, 19, 219),

(120, 20, 220);

-- Inserting data into Pay\_scale table

INSERT INTO Pay\_scale (Min\_limit, Max\_limit, grade) VALUES

(0.00, 9999.99, 'Grade A'),

(10000.00, 19999.99, 'Grade B'),

(20000.00, 39999.99, 'Grade C'),

(40000.00, 49999.99, 'Grade D'),

(50000.00, 69999.99, 'Grade E'),

(70000.00, 99999.99, 'Grade F');

-- Task 1: Accept a range of salary and print details of teachers from the teacher table. DELIMITER //

CREATE PROCEDURE TEACHER\_RANGE(IN min\_sal INT, IN max\_sal INT) BEGIN

IF min\_sal <= max\_sal THEN

SELECT \* FROM Teacher WHERE salary BETWEEN min\_sal AND max\_sal;

ELSE

SELECT "PLZ ENTER IN PROPER MANNER MIN\_SAL < MAX\_SAL";

END IF;

END;

// DELIMITER ;

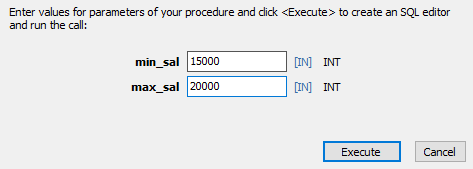


Figure shows a Procedure of 1 output and input

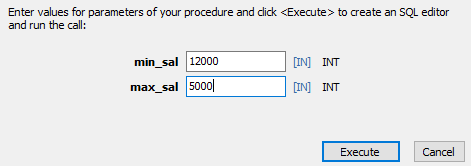


Figure shows a Procedure of 1 output and input wrong input

-- Task 2: Calculate the bonus amount using a cursor DELIMITER //

##### CREATE PROCEDURE TEACHER\_BONUS() BEGIN

-- Declare variables to store fetched data DECLARE v\_teacher\_id INT; DECLARE v\_f\_name VARCHAR(250); DECLARE v\_l\_name VARCHAR(250); DECLARE v\_salary DECIMAL(10, 2); DECLARE v\_bonus DECIMAL(10, 2);

-- Declare cursor

DECLARE Teach\_Bonus CURSOR FOR SELECT t\_no, f\_name, l\_name, salary FROM teacher;

-- Declare handler for NOT FOUND condition DECLARE CONTINUE HANDLER FOR NOT FOUND

SET v\_teacher\_id = NULL;

-- Create a new table to store bonus values CREATE TABLE IF NOT EXISTS teacher\_bonus (

teacher\_id INT PRIMARY KEY, f\_name VARCHAR(250), l\_name VARCHAR(250),

bonus DECIMAL(10, 2)

);

-- Open the cursor OPEN Teach\_Bonus;

-- Fetch and process data from the cursor

FETCH Teach\_Bonus INTO v\_teacher\_id, v\_f\_name, v\_l\_name, v\_salary;

-- Loop through the cursor results WHILE v\_teacher\_id IS NOT NULL DO

-- Calculate bonus based on salary conditions IF v\_salary < 10000 THEN

SET v\_bonus = 0.10 \* v\_salary;

ELSEIF v\_salary BETWEEN 10000 AND 20000 THEN

SET v\_bonus = 0.20 \* v\_salary;

ELSEIF v\_salary BETWEEN 20000 AND 25000 THEN

SET v\_bonus = 0.25 \* v\_salary; ELSE

SET v\_bonus = 0.30 \* v\_salary; END IF;

-- Insert the calculated bonus into the new table

INSERT INTO teacher\_bonus (teacher\_id, f\_name, l\_name, bonus)

VALUES (v\_teacher\_id, v\_f\_name, v\_l\_name, v\_bonus);

-- Fetch the next row

FETCH Teach\_Bonus INTO v\_teacher\_id, v\_f\_name, v\_l\_name, v\_salary; END WHILE;

-- Close the cursor CLOSE Teach\_Bonus;

##### END // DELIMITER ;

CALL TEACHER\_BONUS();

SELECT \* FROM teacher\_bonus;

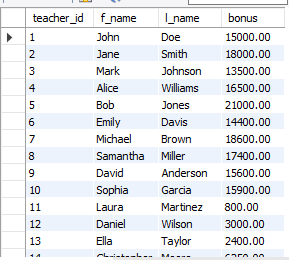


Figure shows a Procedure of 2 output

-- Task 3: Using a simple LOOP structure, list the first 10 records of the ‘teachers’ table. DELIMITER //

CREATE PROCEDURE TEACHER\_RECORD(IN n INT) BEGIN

DECLARE v\_t\_no INT;

DECLARE v\_f\_name VARCHAR(250); DECLARE v\_l\_name VARCHAR(250); DECLARE v\_salary DECIMAL(10, 2);

DECLARE v\_supervisor BOOL;

DECLARE v\_joining\_date DATE; DECLARE v\_birth\_date DATE; DECLARE v\_title VARCHAR(50); DECLARE c INTEGER;

DECLARE Teach\_REC CURSOR FOR

SELECT t\_no, f\_name, l\_name, salary, supervisor, joining\_date, birth\_date, title FROM teacher;

##### DECLARE CONTINUE HANDLER FOR NOT FOUND

SET v\_t\_no = NULL; SET c = 1;

##### CREATE TABLE IF NOT EXISTS TRECORDS (

t\_no int primary key,

f\_name varchar(255), l\_name varchar(255), salary DECIMAL(10,2), supervisor INT, joining\_date date, birth\_date date,

title varchar(50)

);

OPEN Teach\_REC;

FETCH Teach\_REC INTO

v\_t\_no,v\_f\_name,v\_l\_name,v\_salary,v\_supervisor,v\_joining\_date,v\_birth\_date,v\_title;

WHILE c <= n DO

INSERT TRECORDS (t\_no, f\_name, l\_name, salary, supervisor, joining\_date, birth\_date, title)

##### VALUES

(v\_t\_no,v\_f\_name,v\_l\_name,v\_salary,v\_supervisor,v\_joining\_date,v\_birth\_date,v\_title);

FETCH Teach\_REC INTO

v\_t\_no,v\_f\_name,v\_l\_name,v\_salary,v\_supervisor,v\_joining\_date,v\_birth\_date,v\_title; SET c = c + 1;

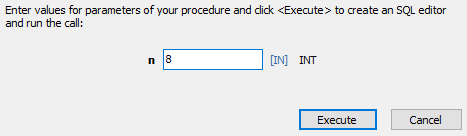
##### END WHILE;

CLOSE Teach\_REC;

END //

DELIMITER ;

SELECT \* FROM TRECORDS;



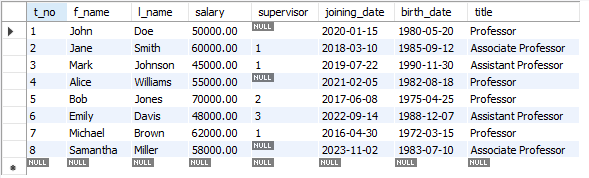


Figure shows a Procedure of 3 input output

-- Task 4: Accept the room number and display teacher details. DELIMITER //

CREATE PROCEDURE TEACHER\_ROOMNO(IN room\_number INT ) BEGIN

IF room\_number >= 201 AND room\_number <= 220 then

select Teacher.t\_no, Teacher.f\_name, Teacher.l\_name, Teacher.birth\_date, Teacher.title from Teacher join Class on Teacher.t\_no = Class.t\_no where room\_number = Class.room\_no ;

##### ELSE

END;

//

select "Enter room number between 201 and 220"; END IF;

##### DELIMITER ;

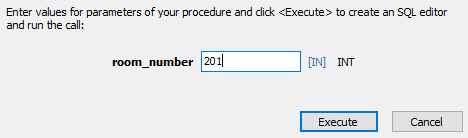




Figure shows a Procedure of 4 input output

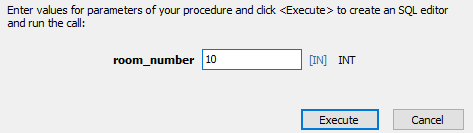




Figure shows a Procedure of 4 output when user enter wrong input

### Conclusion :

In this experiment, I acquired the skills to create a producer using PL/SQL and gained knowledge in utilizing cursors within a program. Specifically, I developed a teacher database as part of the assignment, successfully completing the task. This experience allowed me to familiarize myself with various PL/SQL commands.

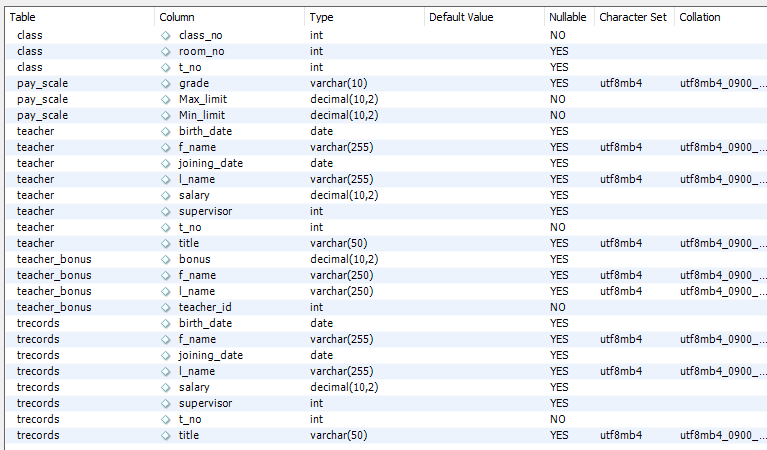


Figure shows a database table for given task

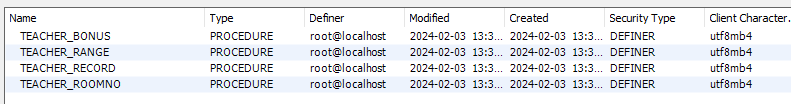


Figure shows all 4 Procedure for given task

# Assignment 4

### Aim:

Design and develop a suitable Student Database application. One of the attributes to me maintained is the attendance of a student in each subject for which he/she has enrolled.

Using TRIGGERS, we write active rules to do the following:

1. Whenever attendance is updated, check if the attendance is less than 85%; if so notify the Head of Department concerned.
2. Whenever the marks in the Internal Assessment Test are entered, check if the marks are less than 40%; if so, notify the Head of the Department concerned.

### Queries & Output :

CREATE DATABASE students; USE students;

CREATE TABLE Student ( student\_id INT PRIMARY KEY, student\_name VARCHAR(50),

adm43\_marks INT CHECK (adm43\_marks >= 0 AND adm43\_marks <= 100), adm43\_attendance INT CHECK (adm43\_attendance >= 0 AND adm43\_attendance <= 100)

);

CREATE TABLE chagnes\_table (

notification\_id INT AUTO\_INCREMENT PRIMARY KEY, stud\_id INT,

message VARCHAR(255), stud\_name VARCHAR(50)

);

##### DELIMITER //

CREATE TRIGGER attend\_mark\_check AFTER UPDATE ON Student

##### FOR EACH ROW BEGIN

IF NEW.adm43\_attendance < 85 THEN

INSERT INTO chagnes\_table (stud\_id, message, stud\_name)

VALUES (NEW.student\_id , 'Low attendance for student ' , NEW.student\_name);

##### END IF;

IF NEW.adm43\_marks < 40 THEN

INSERT INTO chagnes\_table (stud\_id, message, stud\_name)

VALUES (NEW.student\_id ,'Low marks for student ' , NEW.student\_name ); END IF;

END;

// DELIMITER ;

DELIMITER //

CREATE TRIGGER attend\_mark\_check\_inst AFTER INSERT ON Student

##### FOR EACH ROW BEGIN

IF NEW.adm43\_attendance < 85 THEN

INSERT INTO chagnes\_table (stud\_id, message, stud\_name)

VALUES (NEW.student\_id, 'Low attendance for student ' , NEW.student\_name); END IF;

IF NEW.adm43\_marks < 40 THEN

INSERT INTO chagnes\_table (stud\_id, message, stud\_name)

VALUES (NEW.student\_id, 'Low marks for student ' , NEW.student\_name ); END IF;

##### END;

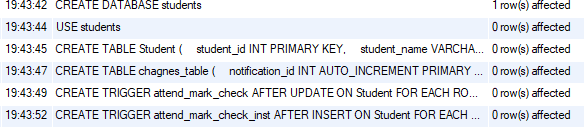
// DELIMITER ;

Figure 4.1 shows above queries are successfully executed select \* from chagnes\_table;



Figure 4.2 null table

INSERT INTO Student (student\_id, student\_name, adm43\_marks, adm43\_attendance) VALUES

(1, 'John Doe', 85, 76),

(2, 'Jane Smith', 78, 82),

(3, 'Alice Johnson', 90, 85),

(4, 'Bob Williams', 82, 79),

(5, 'Emily Brown', 88, 92),

(6, 'Michael Davis', 76, 84),

(7, 'Sophia Wilson', 85, 91),

(8, 'William Martinez', 92, 88),

(9, 'Olivia Anderson', 79, 91),

(10, 'Daniel Taylor', 91, 88); select \* from chagnes\_table;

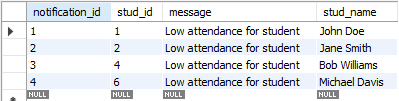


Figure 4.3 Table has changes by insertion trigger

select \* from student;

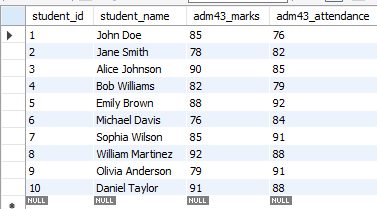


Figure 4.4 student table

UPDATE students.Student SET adm43\_attendance = 75 WHERE student\_id = 1;

UPDATE students.Student SET adm43\_marks = 25 WHERE student\_id = 1;

select \* from chagnes\_table;

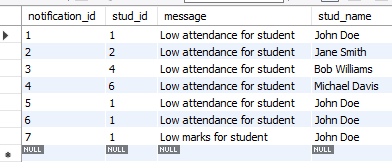


Figure 4.5 changes in table as updation happen as compare to fig 4.3 select \* from student;

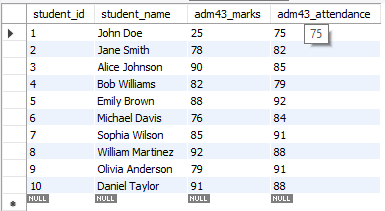


Figure 4.6 changes in student table as updation happen as compare to fig 4.4 and also value updated in table as well as in changes table.

### Conclusion:

In this experiment I learned how to create triggers and also update values in the database with triggers.

# Assignment 5

### Aim:

How to analyze ecommerce Inventory

* 1. What are the top 5 products with the highest inventory levels on the most recent inventory date ?
  2. What is the total inventory level for each product category on the most recent inventory date ?
  3. What is the average inventory level for each product category for the month of January 2022 ?
  4. Which products had a decrease in inventory level from the previous inventory date to the current inventory date ?
  5. What is the overall trend in inventory levels for each product category over the month of January 2022 ?

### Queries & Output :

CREATE DATABASE ecommerce; USE ecommerce;

CREATE TABLE products ( product\_id SERIAL PRIMARY KEY, product\_name VARCHAR(50), product\_category VARCHAR(20), product\_price NUMERIC(10,2)

);

INSERT INTO products (product\_name, product\_category, product\_price) VALUES ('Product A', 'Category 1', 19.99),

('Product B', 'Category 2', 29.99),

('Product C', 'Category 1', 39.99),

('Product D', 'Category 3', 49.99),

('Product E', 'Category 2', 59.99);

CREATE TABLE inventory ( product\_id INT, inventory\_date DATE, inventory\_level INT

);

INSERT INTO inventory (product\_id, inventory\_date, inventory\_level) VALUES (1, '2022-01-01', 100),

(2, '2022-01-01', 200),

(3, '2022-01-01', 150),

(4, '2022-01-01', 75),

(5, '2022-01-01', 250),

(1, '2022-01-02', 80),

(2, '2022-01-02', 180),

(3, '2022-01-02', 100),

(4, '2022-01-02', 60),

(5, '2022-01-02', 220),

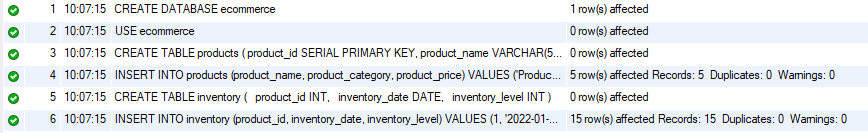
(1, '2022-01-03', 50),

(2, '2022-01-03', 150),

(3, '2022-01-03', 75),

(4, '2022-01-03', 80),

(5, '2022-01-03', 200);



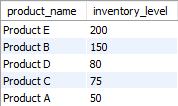
* + 1. What are the top 5 products with the highest inventory levels on the most recent inventory date ?

SELECT p.product\_name, i.inventory\_level FROM products p

JOIN inventory i ON p.product\_id = i.product\_id

WHERE i.inventory\_date = (SELECT MAX(inventory\_date) FROM inventory) ORDER BY i.inventory\_level DESC

##### LIMIT 5;

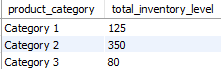


* + 1. What is the total inventory level for each product category on the most recent inventory date ?

SELECT p.product\_category, SUM(i.inventory\_level) AS total\_inventory\_level FROM products p

JOIN inventory i ON p.product\_id = i.product\_id

WHERE i.inventory\_date = (SELECT MAX(inventory\_date) FROM inventory) GROUP BY p.product\_category;

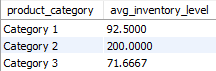


* + 1. What is the average inventory level for each product category for the month of January 2022 ?

SELECT p.product\_category, AVG(i.inventory\_level) AS avg\_inventory\_level FROM products p

JOIN inventory i ON p.product\_id = i.product\_id

WHERE i.inventory\_date >= '2022-01-01' AND i.inventory\_date < '2022-02-01' GROUP BY p.product\_category;



* + 1. Which products had a decrease in inventory level from the previous inventory date to the current inventory date ?

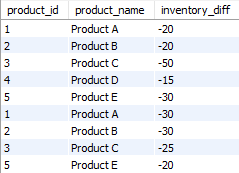
SELECT i1.product\_id, p.product\_name, i1.inventory\_level - i2.inventory\_level AS inventory\_diff

FROM inventory i1

JOIN inventory i2 ON i1.product\_id = i2.product\_id

AND i1.inventory\_date = i2.inventory\_date + INTERVAL 1 day JOIN products p ON i1.product\_id = p.product\_id

WHERE i1.inventory\_level < i2.inventory\_level;



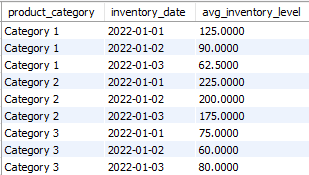
* + 1. What is the overall trend in inventory levels for each product category over the month of January 2022 ?

SELECT p.product\_category, i.inventory\_date, AVG(i.inventory\_level) AS avg\_inventory\_level FROM products p

JOIN inventory i ON p.product\_id = i.product\_id

WHERE i.inventory\_date >= '2022-01-01' AND i.inventory\_date < '2022-02-01' GROUP BY p.product\_category, i.inventory\_date

ORDER BY p.product\_category, i.inventory\_date;



**Conclusion:** In analyzing ecommerce inventory using MySQL queries, insights were gained on top products by inventory, total inventory per category, and average levels for January 2022. Identification of products with decreased inventory highlighted management areas. Trend analysis for January 2022 revealed patterns, aiding in proactive adjustments. Leveraging MySQL for ecommerce inventory analysis enabled actionable insights for optimizing stock levels and improving business performance.

# Assignment 6

### Aim:

(Object Oriented)

1. Write a PL/SQL code to create a class for a "Person" with attributes such as name, age, and address.
2. Write a PL/SQL code to Implement methods in the "Person" class to display the details and update the age.
3. Write a PL/SQL code to implement a method to calculate the annual bonus based on the salary in the "Employee" class.
4. Write a PL/SQL code to create a "Manager" subclass inheriting from the "Employee" class, and add an attribute to store the number of employees managed.

**Queries & Output :**

## A)

### s1:

CREATE TYPE Person AS OBJECT ( name VARCHAR2(50),

age NUMBER,

address VARCHAR2(100)

);

#### Output:

Type created.

### s2:

##### DECLARE

p1 Person;

##### BEGIN

p1 := Person('Kalpan Bariya', 21, 'IIIT SURAT');

DBMS\_OUTPUT.PUT\_LINE('Name: ' || p1.name); DBMS\_OUTPUT.PUT\_LINE('Age: ' || p1.age);

DBMS\_OUTPUT.PUT\_LINE('Address: ' || p1.address); END;

#### Output:

Statement processed. Name: Kalpan Bariya

Age: 30

Address: IIIT SURAT

## B)

### s1:

CREATE OR REPLACE TYPE Person AS OBJECT ( id NUMBER,

name VARCHAR2(100), age NUMBER,

-- displayDetails member function

MEMBER FUNCTION displayDetails RETURN VARCHAR2, MEMBER PROCEDURE updateAge(newAge NUMBER)

);

#### Output:

Type created.

### s2:

CREATE OR REPLACE TYPE BODY Person AS

MEMBER FUNCTION displayDetails RETURN VARCHAR2 IS BEGIN

RETURN 'Person ID: ' || id || ', Name: ' || name || ', Age: ' || age; END;

MEMBER PROCEDURE updateAge(newAge NUMBER) IS BEGIN

age := newAge;

END;

END;

#### Output:

Type created.

### s3:

##### DECLARE

p1 Person := Person(1, 'Kalpan', 20); p2 Person := Person(2, 'KP', 28);

##### BEGIN

DBMS\_OUTPUT.PUT\_LINE(p1.displayDetails()); DBMS\_OUTPUT.PUT\_LINE(p2.displayDetails());

p1.updateAge(25);

DBMS\_OUTPUT.PUT\_LINE('Updated age:'); DBMS\_OUTPUT.PUT\_LINE(p1.displayDetails()); DBMS\_OUTPUT.PUT\_LINE(p2.displayDetails()); END;

#### Output:

Statement processed.

Person ID: 1, Name: Kalpan, Age: 20 Person ID: 2, Name: KP, Age: 28

Updated age:

Person ID: 1, Name: Kalpan, Age: 25 Person ID: 2, Name: KP, Age: 28

## C)

### s1:

CREATE TYPE Employee AS OBJECT (

emp\_id NUMBER, emp\_name VARCHAR2(100), salary NUMBER,

MEMBER FUNCTION calculate\_bonus RETURN NUMBER

);

#### Output:

Type created.

### s2:

CREATE TYPE BODY Employee AS

MEMBER FUNCTION calculate\_bonus RETURN NUMBER IS bonus\_percentage NUMBER;

bonus\_amount NUMBER; BEGIN

IF self.salary< 20000 THEN bonus\_percentage := 0.15;

ELSIF self.salary< 100000 THEN bonus\_percentage := 0.20;

##### ELSE

bonus\_percentage := 0.25; END IF;

bonus\_amount := self.salary \* bonus\_percentage;

RETURN bonus\_amount; END;

END;

#### Output:

Type created.

### s3:

##### DECLARE

emp\_obj Employee; -- an instance of the Employee class emp\_bonus NUMBER; -- a variable to store the bonus amount BEGIN

emp\_obj := Employee(1, 'Kalpan', 5000); emp\_bonus := emp\_obj.calculate\_bonus;

DBMS\_OUTPUT.PUT\_LINE('Employee Bonus: ' || emp\_bonus); END;

#### Output:

Statement processed.

Employee Bonus: 750

## D)

### s1:

CREATE OR REPLACE TYPE Employee AS OBJECT (

emp\_id NUMBER, emp\_name VARCHAR2(90), salary NUMBER,

MEMBER FUNCTION calculate\_bonus RETURN NUMBER

);

#### Output:

Type created.

### s2:

CREATE OR REPLACE TYPE BODY Employee AS

MEMBER FUNCTION calculate\_bonus RETURN NUMBER IS bonus\_percentage NUMBER;

bonus\_amount NUMBER; BEGIN

IF self.salary< 20000THEN bonus\_percentage := 0.15;

ELSIF self.salary< 100000 THEN bonus\_percentage := 0.20;

##### ELSE

bonus\_percentage := 0.25; END IF;

bonus\_amount := self.salary \* bonus\_percentage; RETURN bonus\_amount;

END;

END;

#### Output:

Type created.

### s3:

CREATE OR REPLACE TYPE Manager AS OBJECT (

emp\_id NUMBER, emp\_name VARCHAR2(90), salary NUMBER,

employees\_managed NUMBER, -- Additional

CONSTRUCTOR FUNCTION Manager(

emp\_id NUMBER, emp\_name VARCHAR2, salary NUMBER,

employees\_managed NUMBER

##### ) RETURN SELF AS RESULT,

MEMBER FUNCTION calculate\_bonus RETURN NUMBER

);

#### Output:

Type created.

### s4:

CREATE OR REPLACE TYPE BODY Manager AS

CONSTRUCTOR FUNCTION Manager(

emp\_id NUMBER, emp\_name VARCHAR2, salary NUMBER,

employees\_managed NUMBER

##### ) RETURN SELF AS RESULT IS BEGIN

SELF.emp\_id := emp\_id; SELF.emp\_name := emp\_name; SELF.salary := salary;

SELF.employees\_managed := employees\_managed; RETURN;

##### END;

MEMBER FUNCTION calculate\_bonus RETURN NUMBER IS bonus\_percentage NUMBER;

bonus\_amount NUMBER; BEGIN

IF self.salary< 50000 THEN bonus\_percentage := 0.05;

ELSIF self.salary< 100000 THEN bonus\_percentage := 0.155;

##### ELSE

bonus\_percentage := 0.25; END IF;

bonus\_amount := (self.salary + self.employees\_managed \* 1000) \* bonus\_percentage; RETURN bonus\_amount;

END;

END;

#### Output:

Type created.

### s5:

##### DECLARE

emp\_obj Employee; emp\_bonus NUMBER; BEGIN

emp\_obj := Employee(1, 'Kalpan', 20000); emp\_bonus := emp\_obj.calculate\_bonus;

DBMS\_OUTPUT.PUT\_LINE('Employee Bonus: ' || emp\_bonus); END;

#### Output:

Statement processed.

Employee Bonus: 4000

### s6:

##### DECLARE

manager\_obj Manager; manager\_bonus NUMBER; BEGIN

manager\_obj := Manager(2, 'Kalpan', 90000, 15); manager\_bonus := manager\_obj.calculate\_bonus;

DBMS\_OUTPUT.PUT\_LINE('Manager Bonus: ' || manager\_bonus); END;

#### Output:

Statement processed.

Manager Bonus: 16275

**Conclusion:** In this experiment, I learned how to create and execute an object in Oracle, along with exploring inheritance implementation within this assignment. Additionally, I delved into the implementation of if-else statements, member functions, and member procedures in PL/SQL.

# Assignment 7

### Aim:

1. Write a SQL statement to create a simple table countries including columns country\_id,country\_name and region\_id.
2. Write a SQL statement to create a simple table countries including columns country\_id,country\_name and region\_id which already exist.
3. Write a SQL statement to create the structure of a table dup\_countries similar to countries.
4. Write a SQL statement to create a duplicate copy of countries table including structure and data by name dup\_countries.
5. Write a SQL statement to create a table countries set a constraint NULL.
6. Write a SQL statement to create a table named jobs including columns job\_id, job\_title, min\_salary, max\_salary and check whether the max\_salary amount exceeding the upper limit 25000.
7. Write a SQL statement to create a table named countries including columns country\_id, country\_name and region\_id and make sure that no countries except Italy, India and China will be entered in the table.
8. Write a SQL statement to create a table named countries including columns country\_id,country\_name and region\_id and make sure that no duplicate data against column country\_id will be allowed at the time of insertion.
9. Write a SQL statement to create a table named jobs including columns job\_id, job\_title, min\_salary and max\_salary, and make sure that, the default value for job\_title is blank and min\_salary is 8000 and max\_salary is NULL will be entered automatically at the time of insertion if no value assigned for the specified columns.
10. Write a SQL statement to create a table named countries including columns country\_id, country\_name and region\_id and make sure that the country\_id column will be a key field which will not contain any duplicate data at the time of insertion.
11. Write a SQL statement to create a table countries including columns country\_id, country\_name and region\_id and make sure that the column country\_id will be unique and store an auto-incremented value.

Click me to see the solution

1. Write a SQL statement to create a table countries including columns country\_id, country\_name and region\_id and make sure that the combination of columns country\_id and region\_id will be unique.

### Queries & Output :

CREATE DATABASE AS7; USE AS7;

-- 1. Create a simple table countries CREATE TABLE AS7.countries (

country\_id INT,

country\_name VARCHAR(50), region\_id INT

);

-- 2. Create a table countries if not exists

CREATE TABLE IF NOT EXISTS AS7.countries (

country\_id INT,

country\_name VARCHAR(50), region\_id INT

);

-- 3. Create the structure of table dup\_countries similar to countries CREATE TABLE AS7.dup\_countries LIKE AS7.countries;

-- 4. Create a duplicate copy of countries table including structure and data CREATE TABLE AS7.dup\_countries AS SELECT \* FROM AS7.countries;

-- 5. Create a table countries with a constraint allowing NULL values CREATE TABLE AS7.countries (

country\_id INT,

country\_name VARCHAR(50), region\_id INT,

CONSTRAINT country\_name\_null CHECK (country\_name IS NULL)

);

-- 6. Create a table jobs with max\_salary check constraint CREATE TABLE AS7.jobs (

job\_id INT,

job\_title VARCHAR(50), min\_salary DECIMAL(10,2), max\_salary DECIMAL(10,2),

CONSTRAINT max\_salary\_check CHECK (max\_salary <= 25000)

);

-- 7. Create a table countries with specific allowed country entries CREATE TABLE AS7.countries (

country\_id INT,

country\_name VARCHAR(50), region\_id INT,

CONSTRAINT country\_name\_check CHECK (country\_name IN ('Italy', 'India', 'China'))

);

-- 8. Create a table countries with no duplicate country\_id allowed CREATE TABLE AS7.countries (

country\_id INT PRIMARY KEY, country\_name VARCHAR(50), region\_id INT

);

-- 9. Create a table jobs with default values for specified columns CREATE TABLE AS7.jobs (

job\_id INT,

job\_title VARCHAR(50) DEFAULT '', min\_salary DECIMAL(10,2) DEFAULT 8000,

max\_salary DECIMAL(10,2)

);

-- 10. Create a table countries with country\_id as a key field CREATE TABLE AS7.countries (

country\_id INT UNIQUE, country\_name VARCHAR(50), region\_id INT

);

-- 11. Create a table countries with auto-incremented country\_id and unique constraint CREATE TABLE AS7.countries (

country\_id INT AUTO\_INCREMENT PRIMARY KEY,

country\_name VARCHAR(50), region\_id INT, UNIQUE(country\_id)

);

-- 12. Create a table countries with unique combination of country\_id and region\_id CREATE TABLE AS7.countries (

country\_id INT,

country\_name VARCHAR(50), region\_id INT, UNIQUE(country\_id, region\_id)

)