**INDIAN INSTITUTE OF INFORMATION TECHNOLOGY SURAT**



# LAB REPORT

### on

**ADVANCED DATABASE MANAGEMENT (CS 604)**

**Submitted by**

### [ANKIT GONDHA] (UI21CS10)

**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 | 03-15 | 21-01-2024 | 29-01-2024 |  |
| **2** | Implement PL/SQL commands | 16-20 | 21-01-2024 | 29-01-2024 |  |
| **3** | Write PL/SQL code as specified in tasks | 21-29 | 30-01-2024 | 5-02-2024 |  |
| **4** | Create TRIGGERS and implement given task | 30-35 | 06-02-2024 | 06-02-2024 |  |
| **5** | Analyze ecommerce Inventory | 36-38 | 13-02-2024 | 22-02-2024 |  |
| **6** | Object Oriented PL/SQL | 39-44 | 20-03-2024 | 16-04-2024 |  |
| **7** | PostgreSQL Practice | 45-49 | 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.

### Code :

CREATE DATABASE lab1\_final;

USE lab1\_final;

CREATE TABLE Employee (

EMP\_ID INT PRIMARY KEY,

FIRST\_NAME VARCHAR(50),

LAST\_NAME VARCHAR(50),

SALARY DECIMAL(10, 2),

JOINING\_DATE DATE,

DEPARTMENT VARCHAR(50)

);

CREATE TABLE Bonus (

EMP\_REF\_ID INT,

BONUS\_AMOUNT DECIMAL(10, 2),

BONUS\_DATE DATE,

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

);

CREATE TABLE Title (

EMP\_REF\_ID INT,

EMP\_TITLE VARCHAR(50),

AFFECTED\_FROM DATE,

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

);

INSERT INTO Employee VALUES

(1, 'John', 'Doe', 60000, '2022-01-01', 'Admin'),

(2, 'Jane', 'Smith', 75000, '2022-02-15', 'IT'),

(3, 'Michael', 'Johnson', 55000, '2022-03-10', 'Finance'),

(4, 'Emily', 'Davis', 80000, '2022-04-05', 'Marketing'),

(5, 'Alex', 'Brown', 70000, '2022-05-20', 'Admin'),

(6, 'Jessica', 'Lee', 65000, '2022-06-12', 'IT'),

(7, 'Brian', 'Taylor', 72000, '2022-07-08', 'Finance'),

(8, 'Sophia', 'Clark', 68000, '2022-08-15', 'Marketing'),

(9, 'Daniel', 'White', 60000, '2022-09-03', 'Admin'),

(10, 'Olivia', 'Turner', 78000, '2022-10-22', 'IT'),

(11, 'Ethan', 'Miller', 58000, '2022-11-15', 'Finance'),

(12, 'Mia', 'Moore', 85000, '2022-12-01', 'Marketing'),

(13, 'Jacob', 'Hill', 72000, '2023-01-10', 'Admin'),

(14, 'Ava', 'Cooper', 67000, '2023-02-20', 'IT'),

(15, 'William', 'Baker', 74000, '2023-03-15', 'Finance'),

(16, 'Emma', 'Harris', 62000, '2023-04-05', 'Marketing'),

(17, 'Alexander', 'Ward', 78000, '2023-05-12', 'Admin'),

(18, 'Grace', 'Fisher', 69000, '2023-06-18', 'IT'),

(19, 'Benjamin', 'Chapman', 71000, '2023-07-25', 'Finance'),

(20, 'Lily', 'Lopez', 60000, '2023-08-10', 'Marketing'),

(21, 'Carter', 'Wright', 73000, '2023-09-03', 'Admin'),

(22, 'Chloe', 'Cooper', 68000, '2023-10-22', 'IT'),

(23, 'Owen', 'Perry', 77000, '2023-11-15', 'Finance'),

(24, 'Harper', 'Gray', 65000, '2023-12-01', 'Marketing'),

(25, 'Mason', 'Reid', 80000, '2024-01-10', 'Admin'),

(26, 'Scarlett', 'Lane', 70000, '2024-02-20', 'IT'),

(27, 'Logan', 'Ferguson', 72000, '2024-03-15', 'Finance'),

(28, 'Aria', 'Wood', 69000, '2024-04-05', 'Marketing'),

(29, 'Sebastian', 'Hunter', 75000, '2024-05-12', 'Admin'),

(30, 'Avery', 'Grant', 67000, '2024-06-18', 'IT'),

(31, 'Jackson', 'Ross', 80000, '2024-07-25', 'Finance'),

(32, 'Sophie', 'Mitchell', 62000, '2024-08-10', 'Marketing'),

(33, 'Gabriel', 'Barnes', 74000, '2024-09-03', 'Admin'),

(34, 'Madison', 'Wells', 69000, '2024-10-22', 'IT'),

(35, 'Elijah', 'Perry', 77000, '2024-11-15', 'Finance'),

(36, 'Hannah', 'Bryant', 64000, '2024-12-01', 'Marketing'),

(37, 'Caleb', 'Mason', 72000, '2025-01-10', 'Admin'),

(38, 'Aubrey', 'Newton', 67000, '2025-02-20', 'IT'),

(39, 'Lincoln', 'Clark', 80000, '2025-03-15', 'Finance'),

(40, 'Penelope', 'Harrison', 64000, '2025-04-05', 'Marketing'),

(41, 'Grayson', 'Hudson', 73000, '2025-05-12', 'Admin'),

(42, 'Ella', 'Dixon', 68000, '2025-06-18', 'IT'),

(43, 'Nathan', 'Stone', 77000, '2025-07-25', 'Finance'),

(44, 'Sofia', 'Porter', 65000, '2025-08-10', 'Marketing'),

(45, 'Liam', 'Fletcher', 79000, '2025-09-03', 'Admin'),

(46, 'Aaliyah', 'Gibson', 66000, '2025-10-22', 'IT'),

(47, 'Jack', 'Wagner', 75000, '2025-11-15', 'Finance'),

(48, 'Nora', 'Lloyd', 67000, '2025-12-01', 'Marketing'),

(49, 'Eli', 'Bennett', 71000, '2026-01-10', 'Admin'),

(50, 'Amelia', 'Sharp', 74000, '2026-02-20', 'IT');

INSERT INTO Bonus VALUES

(1, 5000, '2022-02-01'),

(2, 3000, '2022-03-10'),

(3, 2000, '2022-04-15'),

(4, 6000, '2022-05-02'),

(5, 4000, '2022-06-18'),

(6, 2500, '2022-07-25'),

(7, 3500, '2022-08-10'),

(8, 4500, '2022-09-20'),

(9, 1000, '2022-10-05'),

(10, 6000, '2022-11-15'),

(11, 1500, '2022-12-01'),

(12, 5000, '2023-01-10'),

(13, 3000, '2023-02-20'),

(14, 2000, '2023-03-15'),

(15, 6000, '2023-04-05'),

(16, 4000, '2023-05-12'),

(17, 2500, '2023-06-18'),

(18, 3500, '2023-07-25'),

(19, 4500, '2023-08-10'),

(20, 1000, '2023-09-03'),

(21, 3000, '2023-10-22'),

(22, 2000, '2023-11-15'),

(23, 6000, '2023-12-01'),

(24, 4000, '2024-01-10'),

(25, 2500, '2024-02-20'),

(26, 3500, '2024-03-15'),

(27, 4500, '2024-04-05'),

(28, 1000, '2024-05-12'),

(29, 6000, '2024-06-18'),

(30, 1500, '2024-07-25'),

(31, 5000, '2024-08-10'),

(32, 3000, '2024-09-03'),

(33, 2000, '2024-10-22'),

(34, 6000, '2024-11-15'),

(35, 4000, '2024-12-01'),

(36, 2500, '2025-01-10'),

(37, 3500, '2025-02-20'),

(38, 4500, '2025-03-15'),

(39, 1000, '2025-04-05'),

(40, 6000, '2025-05-12'),

(41, 1500, '2025-06-18'),

(42, 5000, '2025-07-25'),

(43, 3000, '2025-08-10'),

(44, 2000, '2025-09-03'),

(45, 6000, '2025-10-22'),

(46, 4000, '2025-11-15'),

(47, 2500, '2025-12-01'),

(48, 3500, '2026-01-10'),

(49, 4500, '2026-02-20'),

(50, 1000, '2026-03-15');

-- Inserting 50 records into the Title Table

INSERT INTO Title VALUES

(1, 'Manager', '2022-01-01'),

(2, 'Developer', '2022-02-20'),

(3, 'Analyst', '2022-03-15'),

(4, 'Supervisor', '2022-04-20'),

(5, 'Coordinator', '2022-05-05'),

(6, 'Designer', '2022-06-30'),

(7, 'Specialist', '2022-07-10'),

(8, 'Team Lead', '2022-08-25'),

(9, 'Assistant', '2022-09-15'),

(10, 'Senior Developer', '2022-10-30'),

(11, 'Junior Analyst', '2022-11-15'),

(12, 'Project Manager', '2022-12-01'),

(13, 'UI/UX Designer', '2023-01-10'),

(14, 'Technical Lead', '2023-02-20'),

(15, 'Data Scientist', '2023-03-15'),

(16, 'Marketing Manager', '2023-04-05'),

(17, 'HR Coordinator', '2023-05-12'),

(18, 'Financial Analyst', '2023-06-18'),

(19, 'Operations Specialist', '2023-07-25'),

(20, 'Product Manager', '2023-08-10'),

(21, 'QA Engineer', '2023-09-03'),

(22, 'Systems Analyst', '2023-10-22'),

(23, 'Sales Representative', '2023-11-15'),

(24, 'Customer Support', '2023-12-01'),

(25, 'Network Engineer', '2024-01-10'),

(26, 'Content Writer', '2024-02-20'),

(27, 'UX Researcher', '2024-03-15'),

(28, 'Business Analyst', '2024-04-05'),

(29, 'Legal Counsel', '2024-05-12'),

(30, 'Financial Planner', '2024-06-18'),

(31, 'IT Specialist', '2024-07-25'),

(32, 'Event Coordinator', '2024-08-10'),

(33, 'Logistics Manager', '2024-09-03'),

(34, 'Security Analyst', '2024-10-22'),

(35, 'Public Relations', '2024-11-15'),

(36, 'Graphic Designer', '2024-12-01'),

(37, 'Database Administrator', '2025-01-10'),

(38, 'Health and Safety', '2025-02-20'),

(39, 'Software Engineer', '2025-03-15'),

(40, 'Recruitment Specialist', '2025-04-05'),

(41, 'Technical Support', '2025-05-12'),

(42, 'Facilities Manager', '2025-06-18'),

(43, 'Executive Assistant', '2025-07-25'),

(44, 'Quality Assurance', '2025-08-10'),

(45, 'Market Researcher', '2025-09-03'),

(46, 'Systems Administrator', '2025-10-22'),

(47, 'Creative Director', '2025-11-15'),

(48, 'Data Analyst', '2025-12-01'),

(49, 'E-commerce Specialist', '2026-01-10'),

(50, 'Software Architect', '2026-02-20');

-- Query1

SELECT \* FROM Employee

ORDER BY FIRST\_NAME ASC, DEPARTMENT DESC;

-- Query2

SELECT COUNT(\*) AS EmployeeCount

FROM Employee

WHERE DEPARTMENT = 'Admin';

-- Query3

SELECT FIRST\_NAME, LAST\_NAME

FROM Employee

WHERE SALARY BETWEEN 50000 AND 100000;

-- Query4

SELECT e.\*

FROM Employee e

JOIN Title t ON e.EMP\_ID = t.EMP\_REF\_ID

WHERE t.EMP\_TITLE = 'Manager';

-- Query5

SELECT EMP\_ID, COUNT(\*)

FROM Employee

GROUP BY EMP\_ID

HAVING COUNT(\*) > 1;

-- Query6

SELECT \*

FROM Employee

WHERE EMP\_ID % 2 = 0;

-- Query7

SELECT e.\*

FROM Employee e

LEFT JOIN Bonus b ON e.EMP\_ID = b.EMP\_REF\_ID

WHERE b.EMP\_REF\_ID IS NULL;

-- Query8

SELECT \*

FROM Employee

LIMIT 10;

-- Query9

SELECT SALARY, COUNT(\*)

FROM Employee

GROUP BY SALARY

HAVING COUNT(\*) > 1;

-- Query10

SELECT \*

FROM Employee

ORDER BY EMP\_ID;

-- Query11

SELECT DISTINCT SALARY

FROM Employee

ORDER BY SALARY DESC

LIMIT 2;

-- Query12

DELIMITER //

CREATE TRIGGER before\_insert\_employee

BEFORE INSERT ON Employee

FOR EACH ROW

BEGIN

IF DATEDIFF(CURDATE(), NEW.JOINING\_DATE) < 6570 THEN

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'Employee must be at least 18 years old.';

END IF;

END;

//

DELIMITER ;

-- Query13

DELIMITER //

CREATE TRIGGER before\_delete\_employee

BEFORE DELETE ON Employee

FOR EACH ROW

BEGIN

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

END;

//

DELIMITER ;

-- Query14

DELIMITER //

CREATE TRIGGER after\_insert\_employee

AFTER INSERT ON Employee

FOR EACH ROW

BEGIN

INSERT INTO insert\_count VALUES (NEW.EMP\_ID, NOW());

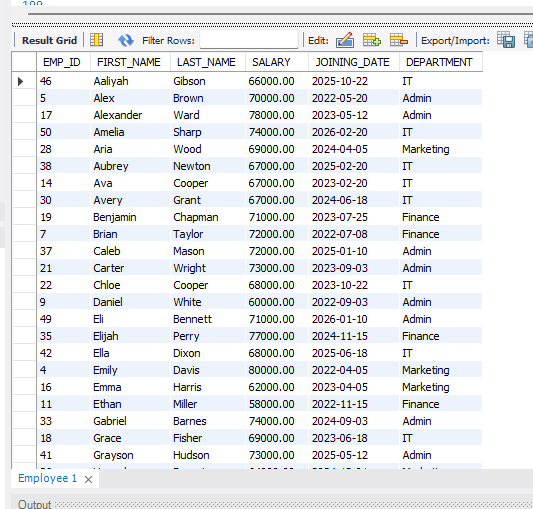
END;

//

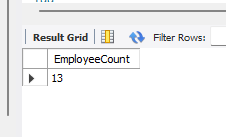
DELIMITER ;

### Output:

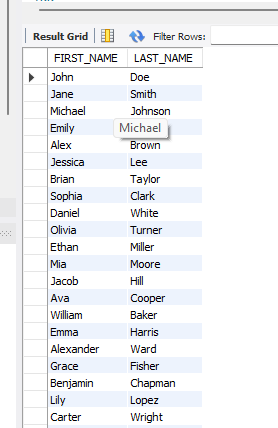
**-- 1. SQL query to print all Employee details from the Employee table order by FIRST\_NAME Ascending and DEPARTMENT Descending.**



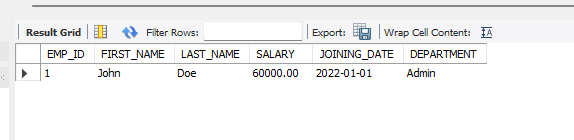
**-- 2 SQL query to fetch the count of employees working in the department ‘Admin’.**



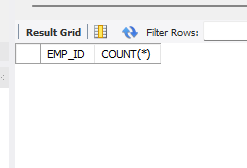
**-- 3. SQL query to fetch Employee names with salaries >= 50000 and <= 100000.**



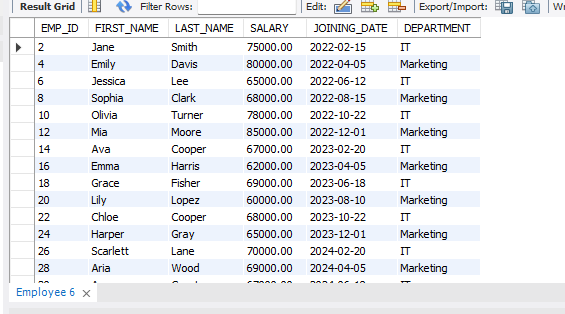
**-- 4. SQL query to print details of the Workers who are also Managers.**



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



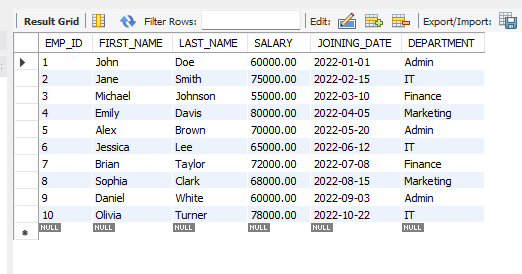
**-- 6. SQL query to show only even rows from a table.**



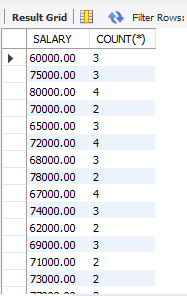
**-- 7. SQL query to show records from one table that another table does not have. Find employees in employee table that do not exist in bonus table.**



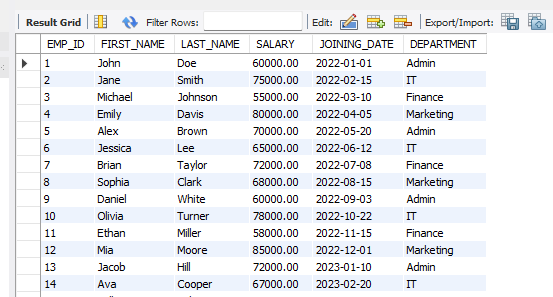
**-- 8. SQL query to show the top n (say 10) records of a table.**



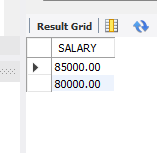
**-- 9. Find people who have the same salary.**



**-- 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 joining date less than current date 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 the number of new tuples inserted using each insert statemet**

### Conclusion:

By organizing data into separate tables and establishing relationships between them using foreign keys, the organization ensures efficient data management and retrieval. The Employee table serves as the central repository for employee information, while the Bonus and Title tables capture additional details related to bonuses and job titles respectively. This structured approach facilitates various queries and analyses, enabling the organization to extract meaningful insights and make informed decisions based on the data stored in the database.

# Assignment 2

### Aim:

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

### Queries & Output :

CREATE TABLE subjects(

id int,

subject varchar(255),

marks int

);

INSERT INTO subjects(id, subject,marks)

values(1, 'ML',92 ),

(2, 'HPC', 65),

(3, 'AR/VR',97 ),

(4, 'WE', 82 ),

(5, 'CC&BD', 75 ),

(5, 'ADMS', 68 );

CREATE DEFINER=`root`@`localhost` PROCEDURE `Task2`()

begin

declare total\_marks int default 0;

declare average\_marks int default 0;

declare grade varchar(1);

Select SUM(marks) into total\_marks from subjects;

Select AVG(marks) into average\_marks from subjects;

if average\_marks>=80 then

set grade = 'A';

elseif average\_marks>=65 then

set grade = 'B';

elseif average\_marks>=50 then

set grade = 'C';

elseif average\_marks>=33 then

set grade = 'D';

else

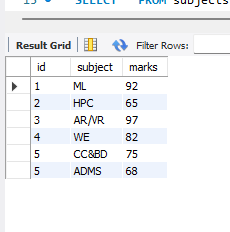
set grade = 'F';

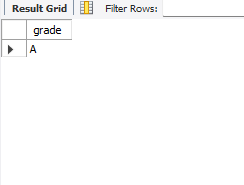
end if;

select grade;

end

### OUTPUT:





#### -- 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';

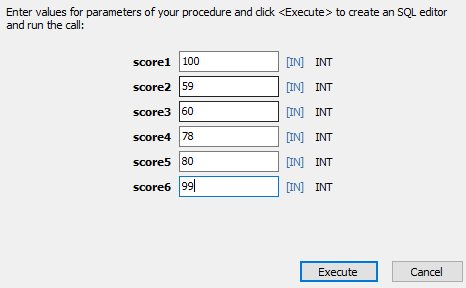
ELSEIF average\_score >= 50 THEN SET avg\_grade = 'D';

##### 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 Using PL/SQL, this code block efficiently calculates the total and average marks, and subsequently assigns a grade based on the calculated average. It's a straightforward approach to automate the grading process for a student's performance in multiple subjects. By encapsulating these calculations within a PL/SQL block, the code offers reusability and maintainability, making it easier to modify or integrate into larger systems as needed.

# 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 lab3;

use lab3;

CREATE TABLE Teacher (

t\_no INT PRIMARY KEY,

f\_name VARCHAR(50),

l\_name VARCHAR(50),

salary DECIMAL(10,2),

supervisor INT,

joining\_date DATE,

birth\_date DATE,

title VARCHAR(50)

);

CREATE TABLE Class (

class\_no INT PRIMARY KEY,

t\_no INT,

room\_no INT,

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

);

CREATE TABLE Pay\_scale (

grade INT PRIMARY KEY,

Min\_limit DECIMAL(10,2),

Max\_limit DECIMAL(10,2)

);

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

VALUES

(1, 'John', 'Doe', 50000, NULL, '2020-01-01', '1985-05-10', 'Math Teacher'),

(2, 'Jane', 'Smith', 55000, 1, '2019-08-15', '1980-12-20', 'Science Teacher'),

(3, 'Alice', 'Johnson', 60000, NULL, '2021-03-10', '1990-03-25', 'English Teacher'),

(4, 'Michael', 'Williams', 52000, 1, '2022-02-20', '1988-07-15', 'History Teacher'),

(5, 'Emily', 'Brown', 58000, NULL, '2020-05-05', '1983-09-30', 'Art Teacher'),

(6, 'David', 'Jones', 53000, 3, '2018-11-11', '1982-11-05', 'Music Teacher'),

(7, 'Sarah', 'Davis', 56000, NULL, '2017-09-01', '1987-06-18', 'Physical Education Teacher'),

(8, 'Christopher', 'Martinez', 59000, 2, '2019-06-30', '1986-04-12', 'Computer Science Teacher'),

(9, 'Jessica', 'Garcia', 54000, NULL, '2021-10-25', '1992-01-28', 'Foreign Language Teacher'),

(10, 'Daniel', 'Rodriguez', 57000, 3, '2016-12-12', '1984-08-22', 'Drama Teacher');

INSERT INTO Class (class\_no, t\_no, room\_no)

VALUES

(101, 1, 101),

(102, 2, 102),

(103, 3, 101),

(104, 4, 104),

(105, 5, 105),

(106, 6, 102),

(107, 7, 103),

(108, 8, 105),

(109, 9, 104),

(110, 10, 106);

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

VALUES

(1, 40000, 60000),

(2, 60001, 80000),

(3, 80001, 100000),

(4, 100001, 120000),

(5, 120001, 140000),

(6, 140001, 160000),

(7, 160001, 180000),

(8, 180001, 200000),

(9, 200001, 220000),

(10, 220001, 240000);

DELIMITER //

CREATE PROCEDURE PrintTeachersInRange(IN min\_salary INT, IN max\_salary INT)

BEGIN

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

END//

DELIMITER ;

CREATE PROCEDURE CalculateBonus()

BEGIN

DECLARE done INT DEFAULT 0;

DECLARE t\_no INT;

DECLARE salary DECIMAL(10, 2);

DECLARE bonus DECIMAL(10, 2);

DECLARE cur CURSOR FOR SELECT t\_no, salary FROM Teacher;

DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;

OPEN cur;

read\_loop: LOOP

FETCH cur INTO t\_no, salary;

IF done THEN

LEAVE read\_loop;

END IF;

IF salary < 10000 THEN

SET bonus = salary \* 0.1;

ELSEIF salary BETWEEN 10000 AND 20000 THEN

SET bonus = salary \* 0.2;

ELSEIF salary BETWEEN 20000 AND 25000 THEN

SET bonus = salary \* 0.25;

ELSE

SET bonus = salary \* 0.3;

END IF;

-- Output the result

SELECT CONCAT('Teacher with ID ', t\_no, ' has a bonus of ', bonus) AS result;

END LOOP;

CLOSE cur;

END

DELIMITER //

CREATE PROCEDURE ListFirst10Teachers()

BEGIN

DECLARE counter INT DEFAULT 0;

DECLARE t\_no INT;

DECLARE f\_name VARCHAR(50);

DECLARE l\_name VARCHAR(50);

DECLARE birth\_date DATE;

DECLARE title VARCHAR(50);

DECLARE cur CURSOR FOR SELECT t\_no, f\_name, l\_name, birth\_date, title FROM Teacher LIMIT 10; -- Limit the result to the first 10 records

DECLARE CONTINUE HANDLER FOR NOT FOUND SET counter = 10;

OPEN cur;

read\_loop: LOOP

FETCH cur INTO t\_no, f\_name, l\_name, birth\_date, title;

IF counter >= 10 THEN

LEAVE read\_loop;

END IF;

-- Output the result directly

SELECT CONCAT('Teacher ', t\_no, ': ', f\_name, ' ', l\_name, ', Birth Date: ', birth\_date, ', Title: ', title) AS result;

SET counter = counter + 1;

END LOOP;

CLOSE cur;

END //

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

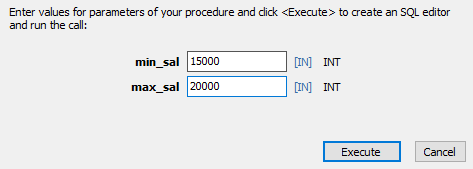


Figure shows a Procedure of 1 output and input

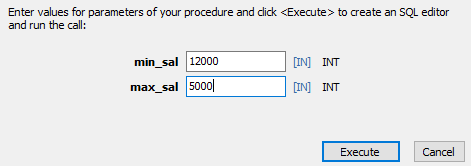


Figure shows a Procedure of 1 output and input wrong input

-- 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;

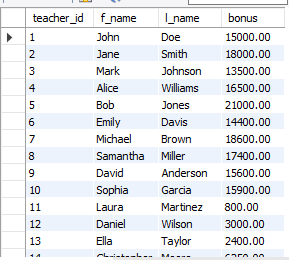


Figure shows a Procedure of 2 output

-- 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;

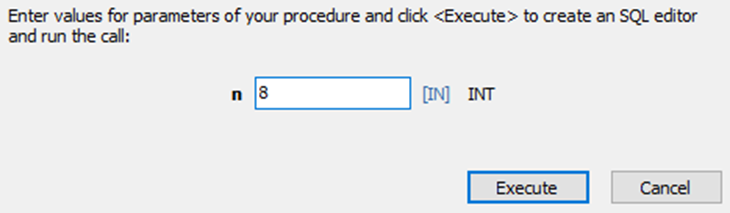
##### END WHILE;

CLOSE Teach\_REC;

END //

DELIMITER ;

SELECT \* FROM TRECORDS;

****

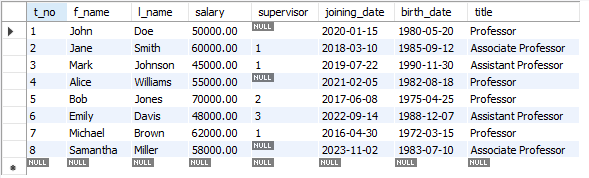


Figure shows a Procedure of 3 input output

-- 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 ;

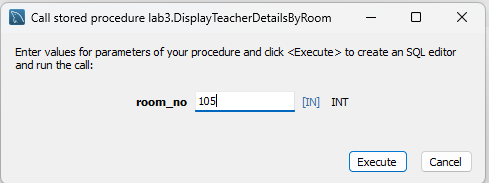
****



Figure shows a Procedure of 4 input output

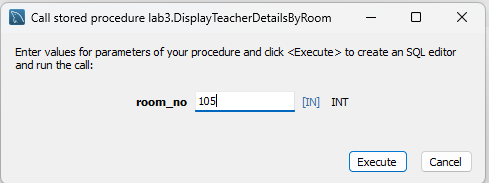
****



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

**Conclusion :**

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.

# 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:

create database lab4;

use lab4;

CREATE TABLE marks (

sname VARCHAR(10) PRIMARY KEY,

m1 INTEGER,

m2 INTEGER,

m3 INTEGER,

m4 INTEGER,

m5 INTEGER,

m6 INTEGER

);

INSERT INTO marks VALUES ('Ankit', 10, 10, 10, 10, 10, 10);

INSERT INTO marks VALUES ('Dev', 11, 11, 11, 11, 11, 11);

INSERT INTO marks VALUES ('Preet', 20, 11, 18, 17, 10, 11);

DELIMITER $$

CREATE TRIGGER marks\_check\_trigger BEFORE INSERT ON marks

FOR EACH ROW

BEGIN

DECLARE minimum\_percentage INT DEFAULT 40;

DECLARE max\_marks INT DEFAULT 100; -- Assuming maximum marks

IF NEW.m1 < max\_marks \* minimum\_percentage / 100 THEN

SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'Marks less than 40% in M1';

END IF;

IF NEW.m2 < max\_marks \* minimum\_percentage / 100 THEN

SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'Marks less than 40% in M2';

END IF;

IF NEW.m3 < max\_marks \* minimum\_percentage / 100 THEN

SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'Marks less than 40% in M3';

END IF;

IF NEW.m4 < max\_marks \* minimum\_percentage / 100 THEN

SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'Marks less than 40% in M4';

END IF;

IF NEW.m5 < max\_marks \* minimum\_percentage / 100 THEN

SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'Marks less than 40% in M5';

END IF;

IF NEW.m6 < max\_marks \* minimum\_percentage / 100 THEN

SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'Marks less than 40% in M6';

END IF;

END$$

DELIMITER ;

insert into marks values('Darshan',20,60,55,77,88,99);

insert into marks values('Chirag',20,60,55,77,88,99);

CREATE TABLE attendance (

sname VARCHAR(10) PRIMARY KEY,

att1 INTEGER,

att2 INTEGER,

att3 INTEGER,

att4 INTEGER,

att5 INTEGER,

att6 INTEGER

);

INSERT INTO attendance VALUES ('Ankit', 10, 10, 10, 10, 10, 10);

INSERT INTO attendance VALUES ('Dev', 11, 11, 11, 11, 11, 11);

INSERT INTO attendance VALUES ('Darshan', 10, 11, 8, 7, 10, 11);

DELIMITER $$

CREATE DEFINER = CURRENT\_USER TRIGGER `lab4`.`attendance\_AFTER\_UPDATE` AFTER UPDATE ON `attendance` FOR EACH ROW

BEGIN

DECLARE total\_classes INTEGER;

SET total\_classes := 50; -- Change the total\_classes value as per your requirement

IF NEW.att1 / total\_classes \* 100 < 85 THEN

SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'Attendance less than 85% in Subject 1';

END IF;

IF NEW.att2 / total\_classes \* 100 < 85 THEN

SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'Attendance less than 85% in Subject 2';

END IF;

IF NEW.att3 / total\_classes \* 100 < 85 THEN

SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'Attendance less than 85% in Subject 3';

END IF;

IF NEW.att4 / total\_classes \* 100 < 85 THEN

SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'Attendance less than 85% in Subject 4';

END IF;

IF NEW.att5 / total\_classes \* 100 < 85 THEN

SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'Attendance less than 85% in Subject 5';

END IF;

IF NEW.att6 / total\_classes \* 100 < 85 THEN

SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'Attendance less than 85% in Subject 6';

END IF;

END$$

DELIMITER ;

UPDATE attendance

SET att1 = 53

WHERE sname = 'Ankit';

UPDATE attendance

SET att1 = 153

WHERE sname = 'Dev';

UPDATE attendance SET att1=46,att2=45,att3=47,att4=48,att5=49,att6=473 WHERE sname='Darshan';

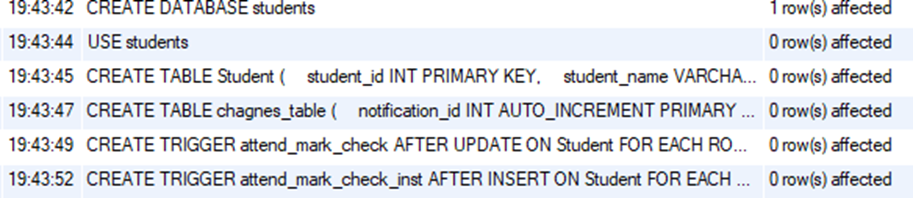


Figure 4.1 shows above queries are successfully executed select \* from chagnes\_table;



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 changes by insertion trigger

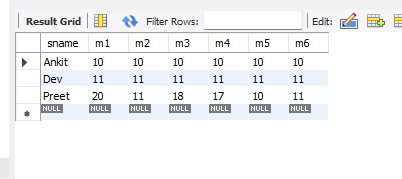
****

Figure 4.4 student table

****

Figure 4.5 changes in table as updation happen as compare to fig 4.3

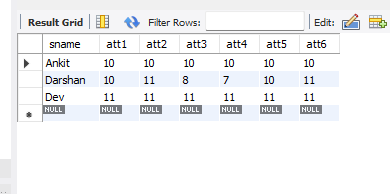
****

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:

Through the use of PL/SQL, these tasks are accomplished efficiently, leveraging the power of cursors, loops, and conditional statements. By organizing and processing data stored in the "Teacher" table, the code provides valuable insights such as teacher salary details, bonus calculations, and specific teacher information based on room numbers. This enables effective management and analysis of teacher-related data within the database system.

# 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 ?

### Code :

CREATE DATABASE lab5;

use lab5;

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', 20),

('Product B', 'Category 2', 40),

('Product C', 'Category 1', 50),

('Product D', 'Category 3', 70),

('Product E', 'Category 1', 90),

('Product F', 'Category 2', 100);

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),

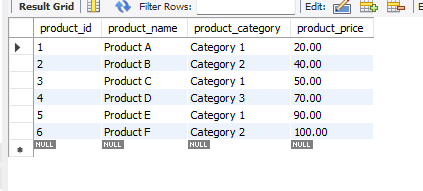
(1, '2022-01-03', 50),

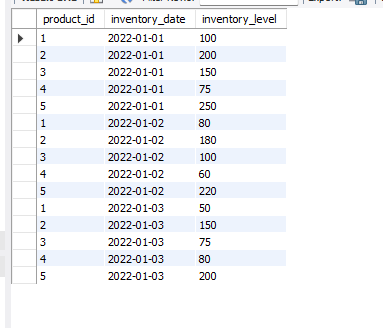
(2, '2022-01-03', 150),

(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 ?

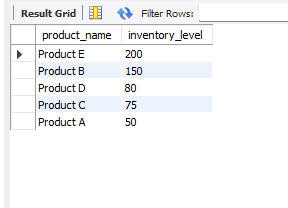
SELECT p.product\_name, i.inventory\_level FROM products p

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;



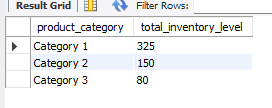
* + 1. What is the total inventory level for each product category on the most recent inventory date ?

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;

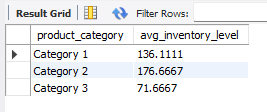


* + 1. What is the average inventory level for each product category for the month of January 2022 ?

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 ?

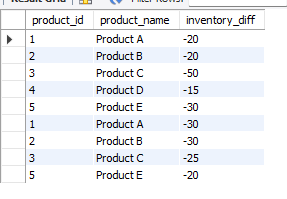
SELECT i1.product\_id, p.product\_name, i1.inventory\_level - i2.inventory\_level AS inventory\_diff

FROM inventory i1

JOIN inventory i2 ON i1.product\_id = i2.product\_id AND i1.inventory\_date = DATE\_ADD(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

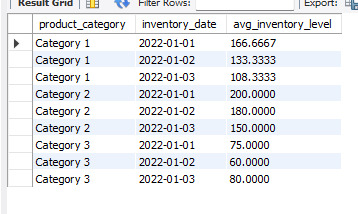
FROM products p

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.

**Code**:

1. **Write a PL/SQL code to create a class for a "Person" with attributes such as name, age, and address.**

CREATE TYPE Person AS OBJECT (

name VARCHAR2(50),

age NUMBER,

address VARCHAR2(100)

);

DECLARE

p1 Person;

BEGIN

p1 := Person(‘Ankit', 30, 'Surat');

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

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

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

END;

1. **Write a PL/SQL code to Implement methods in the "Person" class to display the details and update the age.**

CREATE OR REPLACE TYPE Person AS OBJECT (

id NUMBER,

name VARCHAR2(100),

age NUMBER,

CREATE OR REPLACE TYPE BODY Person AS

MEMBER FUNCTION displayDetails RETURN VARCHAR2 IS BEGIN

RETURN 'Person ID: ' || self.id || ', Name: ' || self.name || ', Age: '

self.age; END;

MEMBER PROCEDURE updateAge(newAge NUMBER) IS BEGIN

self.age := newAge; END;

END;

CREATE OR REPLACE PACKAGE PersonPackage AS

FUNCTION displayDetails(p Person) RETURN VARCHAR2;

PROCEDURE updateAge(p IN OUT Person, newAge NUMBER);

END;

CREATE OR REPLACE PACKAGE BODY PersonPackage AS

FUNCTION displayDetails(p Person) RETURN VARCHAR2 IS

BEGIN

RETURN p.displayDetails();

END;

PROCEDURE updateAge(p IN OUT Person, newAge NUMBER) IS BEGIN

p.updateAge(newAge);

END;

END;

DECLARE

p1 Person := Person(1, 'Ankit', 30);

p2 Person := Person(2, 'Bhavik', 25);

BEGIN

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

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

PersonPackage.updateAge(p1, 31);

DBMS\_OUTPUT.PUT\_LINE('After updating age:');

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

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

END;

1. **Write a PL/SQL code to implement a method to calculate the annual bonus based on the salary in the "Employee" class.**

CREATE OR REPLACE TYPE Employee AS OBJECT ( emp\_id NUMBER,

emp\_name VARCHAR2(100),

salary NUMBER,

MEMBER FUNCTION calculate\_bonus RETURN NUMBER

);

CREATE OR REPLACE TYPE BODY Employee AS

MEMBER FUNCTION calculate\_bonus RETURN NUMBER IS

bonus\_percentage NUMBER;

bonus\_amount NUMBER;

BEGIN

IF self.salary < 50000 THEN

bonus\_percentage := 0.1;

ELSIF self.salary < 100000 THEN

bonus\_percentage := 0.15;

ELSE

bonus\_percentage := 0.2;

END IF;

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

RETURN bonus\_amount;

END;

END;

DECLARE

emp\_obj Employee;

emp\_bonus NUMBER;

BEGIN

emp\_obj := Employee(1, 'Ankit', 65000);

emp\_bonus := emp\_obj.calculate\_bonus;

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

1. **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.**

CREATE OR REPLACE TYPE Employee AS OBJECT ( emp\_id NUMBER,

emp\_name VARCHAR2(100),

salary NUMBER

NOT FINAL;

CREATE OR REPLACE TYPE Manager UNDER Employee (

number\_of\_employees NUMBER

);

CREATE TABLE employees\_data OF Employee

INSERT INTO employees\_data VALUES (1, 'Ankit', 150000)

INSERT INTO employees\_data VALUES (2, 'Bhavik', 80000)

INSERT INTO employees\_data VALUES (3, 'Dev', 120000)

CREATE TABLE managers\_data OF Manager

INSERT INTO managers\_data VALUES (4, 'Ankit', 150000, 10)

DECLARE

emp\_data Employee;

mgr\_data Manager;

BEGIN

FOR emp\_data IN (SELECT \* FROM employees\_data) LOOP

DBMS\_OUTPUT.PUT\_LINE('Employee ID: ' || emp\_data.emp\_id || ', Name: '

emp\_data.emp\_name || ', Salary: ' || emp\_data.salary);

END LOOP;

FOR mgr\_data IN (SELECT \* FROM managers\_data) LOOP

DBMS\_OUTPUT.PUT\_LINE('Manager ID: ' || mgr\_data.emp\_id || ', Name: '

mgr\_data.emp\_name || ', Salary: ' || mgr\_data.salary || ', Number of Employees: ' || mgr\_data.number\_of\_employees);

END LOOP; END;

#### Output:

1)

Type created.

Statement processed. Name: Ankit

Age: 21

Address: IIIT SURAT

2)

Type created.

Type created.

Statement processed.

Person ID: 1, Name: Ankit, Age: 20 Person ID: 2, Name: Bhavik, Age: 28

Updated age:

Person ID: 1, Name: Ankit, Age: 25 Person ID: 2, Name: Bhavik, Age: 28

3)

Type created.

Type created.

Statement processed.

Employee Bonus: 9750

4)

Type created.

Type created.

Type created.

Type created.

Statement processed.

Employee Bonus: 4000

Statement processed.

Manager Bonus: 16275

**Conclusion:**

This PL/SQL code demonstrates the principles of object-oriented programming within the

context of Oracle Database. By defining classes and subclasses, along with methods to operate

on their attributes, we establish a foundation for modeling complex real-world entities and their

behaviors. Such an approach promotes code reusability, modularity, and maintainability,

allowing for the creation of scalable and robust database applications. Through these examples,

we showcase how to structure and extend classes to encapsulate data and behavior, thereby

enabling efficient and organized development within the Oracle PL/SQL environment.

# 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.

### Code :

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)

)

**Conclusion:**

Creating tables in PostgreSQL involves careful consideration of data structure and integrity constraints to ensure efficient data storage and retrieval. These SQL statements showcase various scenarios encountered during table creation, demonstrating PostgreSQL's versatility and robustness in handling diverse database requirements. With PostgreSQL's rich feature set and SQL support, database administrators can design and manage databases effectively, facilitating the development of reliable and scalable applications.