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.

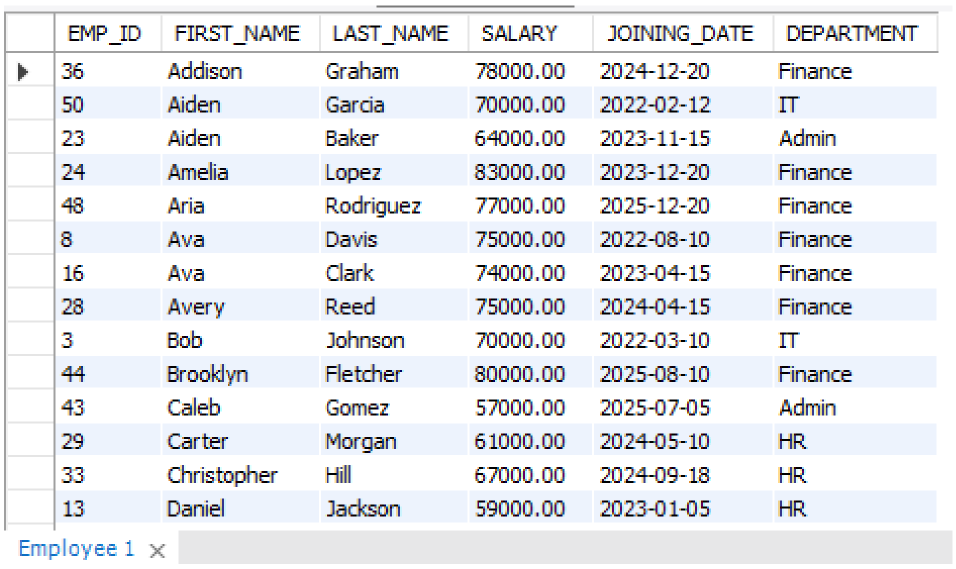
MySQL Queries & Output :

```
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),
```

```sql
  ('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
```

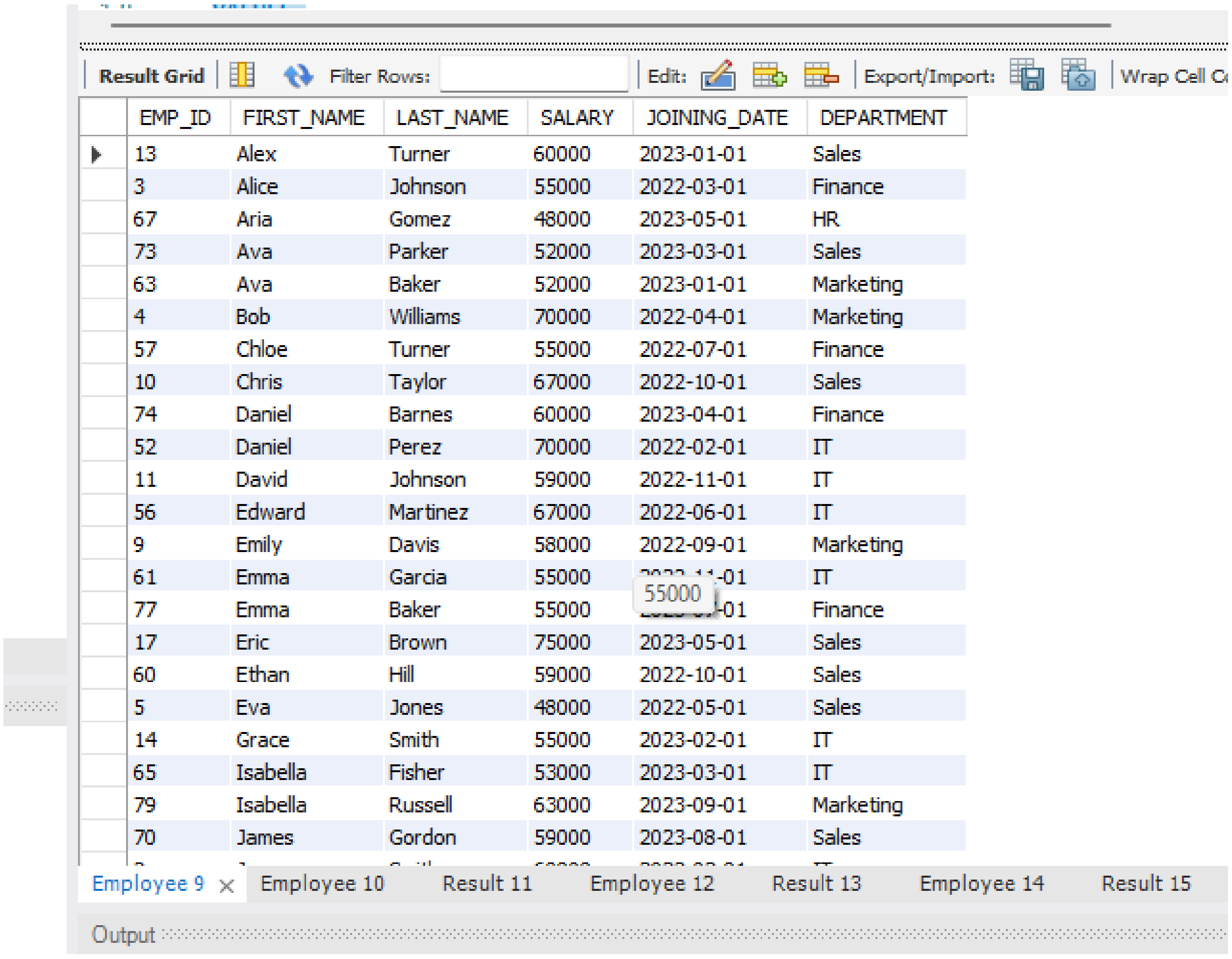
```
FOR EACH ROW
BEGIN
   INSERT INTO employee_backup (EMP_ID, FIRST_NAME, LAST_NAME, SALARY, JOINING_DATE, DEPARTMENT)
   VALUES (OLD.EMP_ID, OLD.FIRST_NAME, OLD.LAST_NAME, OLD.SALARY, OLD.JOINING_DATE,
OLD.DEPARTMENT);
END;
//
DELIMITER ;


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.



EMP_ID	FIRST_NAME	LAST_NAME	SALARY	JOINING_DATE	DEPARTMENT
36	Addison	Graham	78000.00	2024-12-20	Finance
50	Aiden	Garcia	70000.00	2022-02-12	IT
23	Aiden	Baker	64000.00	2023-11-15	Admin
24	Amelia	Lopez	83000.00	2023-12-20	Finance
48	Aria	Rodriguez	77000.00	2025-12-20	Finance
8	Ava	Davis	75000.00	2022-08-10	Finance
16	Ava	Clark	74000.00	2023-04-15	Finance
28	Avery	Reed	75000.00	2024-04-15	Finance
3	Bob	Johnson	70000.00	2022-03-10	IT
44	Brooklyn	Fletcher	80000.00	2025-08-10	Finance
43	Caleb	Gomez	57000.00	2025-07-05	Admin
29	Carter	Morgan	61000.00	2024-05-10	HR
33	Christopher	Hill	67000.00	2024-09-18	HR
13	Daniel	Jackson	59000.00	2023-01-05	HR

Employee 1 ×



Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell C

EMP_ID	FIRST_NAME	LAST_NAME	SALARY	JOINING_DATE	DEPARTMENT
13	Alex	Turner	60000	2023-01-01	Sales
3	Alice	Johnson	55000	2022-03-01	Finance
67	Aria	Gomez	48000	2023-05-01	HR
73	Ava	Parker	52000	2023-03-01	Sales
63	Ava	Baker	52000	2023-01-01	Marketing
4	Bob	Williams	70000	2022-04-01	Marketing
57	Chloe	Turner	55000	2022-07-01	Finance
10	Chris	Taylor	67000	2022-10-01	Sales
74	Daniel	Barnes	60000	2023-04-01	Finance
52	Daniel	Perez	70000	2022-02-01	IT
11	David	Johnson	59000	2022-11-01	IT
56	Edward	Martinez	67000	2022-06-01	IT
9	Emily	Davis	58000	2022-09-01	Marketing
61	Emma	Garcia	55000	2022-11-01	IT
77	Emma	Baker	55000	2023-07-01	Finance
17	Eric	Brown	75000	2023-05-01	Sales
60	Ethan	Hill	59000	2022-10-01	Sales
5	Eva	Jones	48000	2022-05-01	Sales
14	Grace	Smith	55000	2023-02-01	IT
65	Isabella	Fisher	53000	2023-03-01	IT
79	Isabella	Russell	63000	2023-09-01	Marketing
70	James	Gordon	59000	2023-08-01	Sales

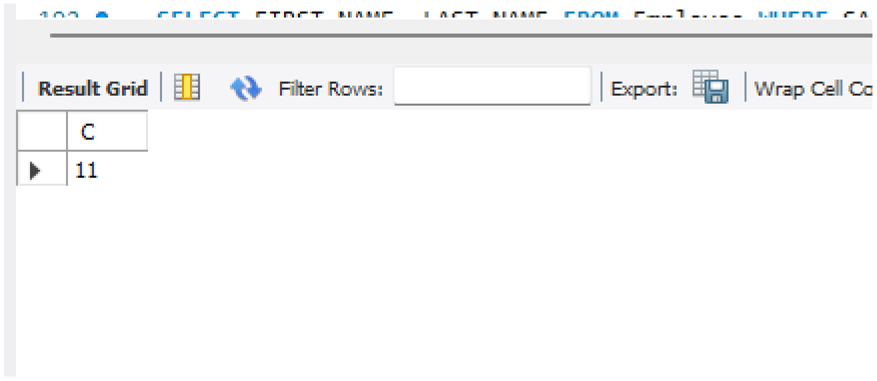
55000

Employee 9 × | Employee 10 | Result 11 | Employee 12 | Result 13 | Employee 14 | Result 15

Output

SELECT * FROM Employee ORDER BY FIRST_NAME ASC, DEPARTMENT DESC;

-- 2 SQL query to fetch the count of employees working in the department 'Admin'.

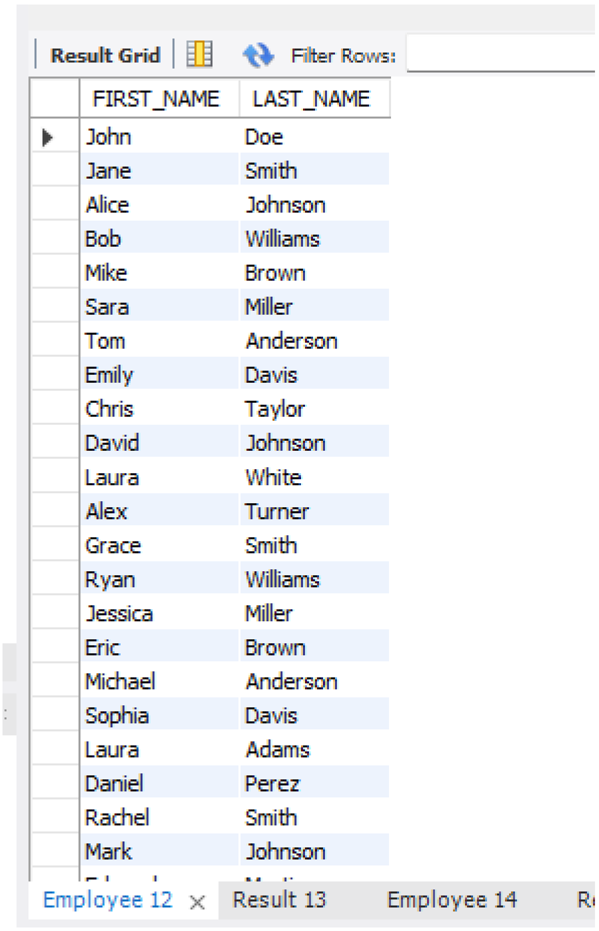


Result Grid | Filter Rows: | Export: | Wrap Cell Co

	C
▶	11

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;

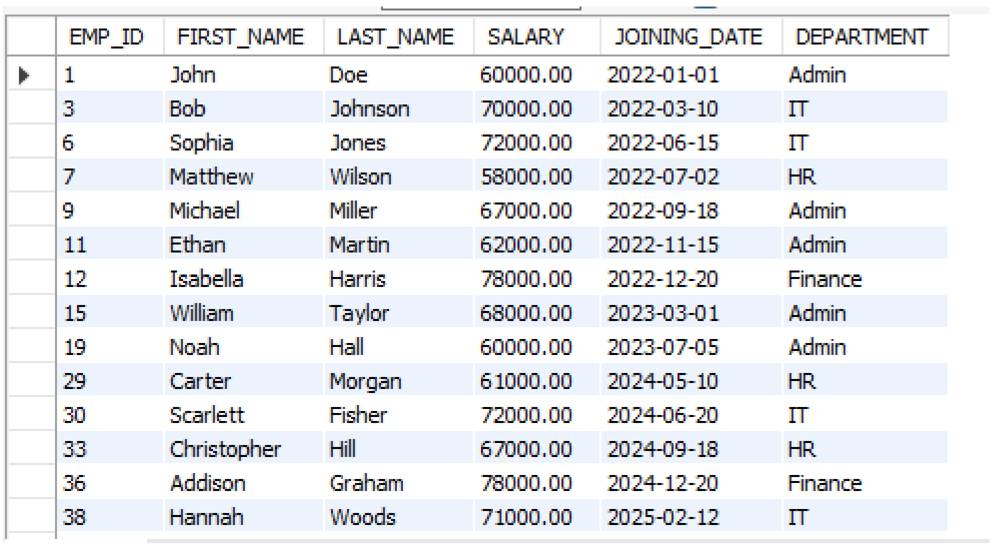


Result Grid | Filter Rows:

	FIRST_NAME	LAST_NAME
▶	John	Doe
	Jane	Smith
	Alice	Johnson
	Bob	Williams
	Mike	Brown
	Sara	Miller
	Tom	Anderson
	Emily	Davis
	Chris	Taylor
	David	Johnson
	Laura	White
	Alex	Turner
	Grace	Smith
	Ryan	Williams
	Jessica	Miller
	Eric	Brown
	Michael	Anderson
	Sophia	Davis
	Laura	Adams
	Daniel	Perez
	Rachel	Smith
	Mark	Johnson

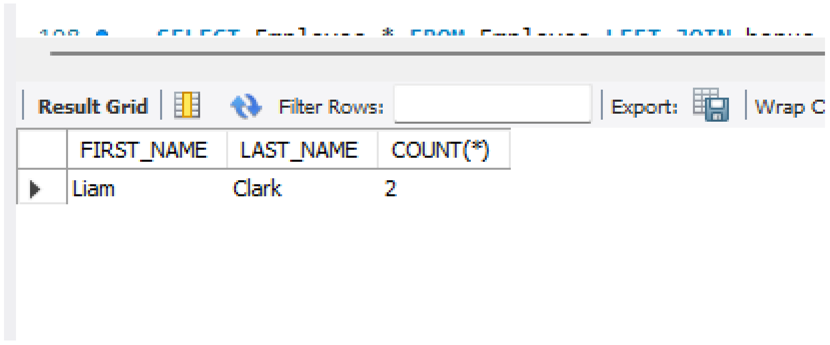
Employee 12 × | Result 13 | Employee 14

-- 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



EMP_ID	FIRST_NAME	LAST_NAME	SALARY	JOINING_DATE	DEPARTMENT
1	John	Doe	60000.00	2022-01-01	Admin
3	Bob	Johnson	70000.00	2022-03-10	IT
6	Sophia	Jones	72000.00	2022-06-15	IT
7	Matthew	Wilson	58000.00	2022-07-02	HR
9	Michael	Miller	67000.00	2022-09-18	Admin
11	Ethan	Martin	62000.00	2022-11-15	Admin
12	Isabella	Harris	78000.00	2022-12-20	Finance
15	William	Taylor	68000.00	2023-03-01	Admin
19	Noah	Hall	60000.00	2023-07-05	Admin
29	Carter	Morgan	61000.00	2024-05-10	HR
30	Scarlett	Fisher	72000.00	2024-06-20	IT
33	Christopher	Hill	67000.00	2024-09-18	HR
36	Addison	Graham	78000.00	2024-12-20	Finance
38	Hannah	Woods	71000.00	2025-02-12	IT

-- 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;

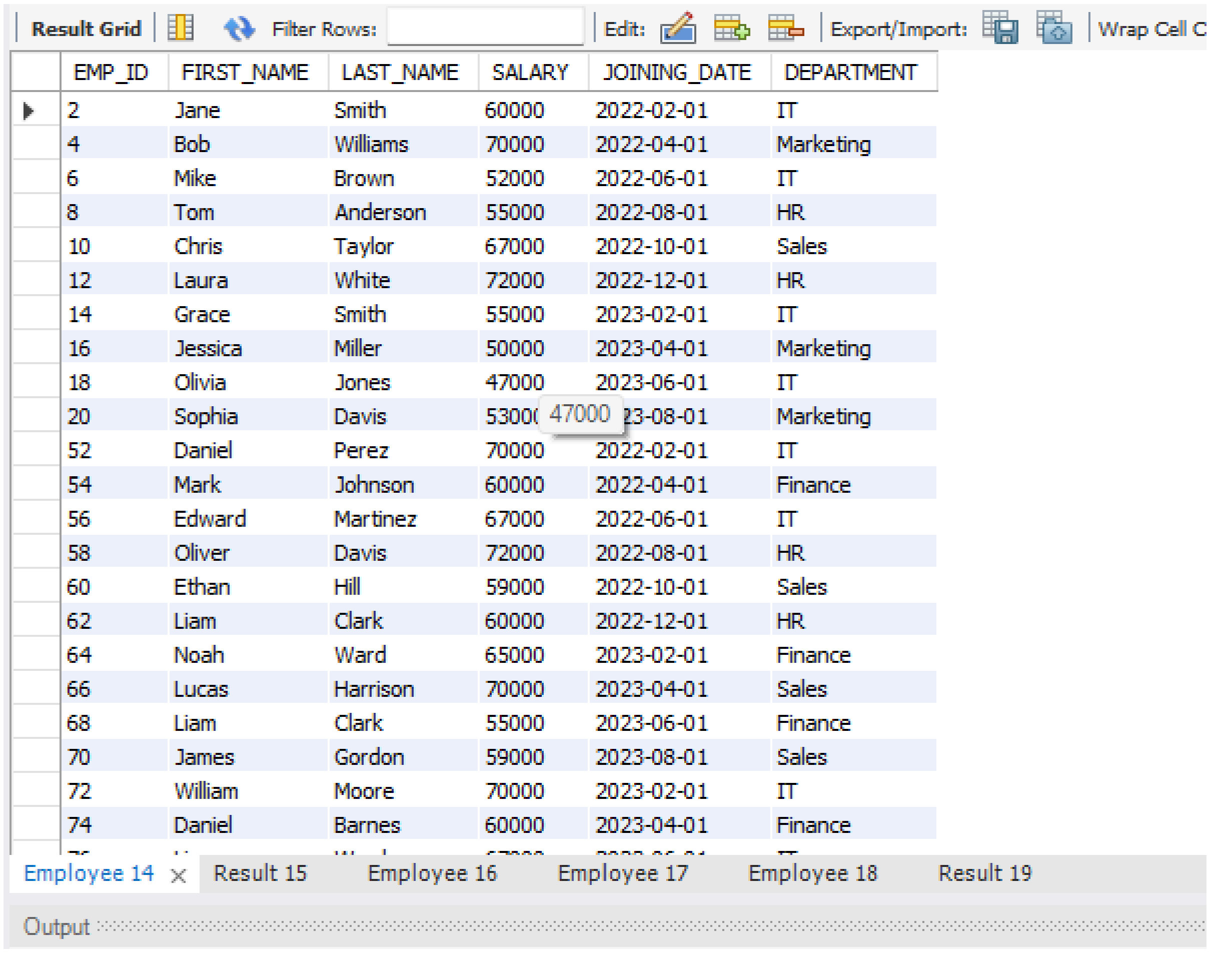


```
108 ●    SELECT Employee.* FROM Employee LEFT JOIN bonus
```

	FIRST_NAME	LAST_NAME	COUNT(*)
▶	Liam	Clark	2

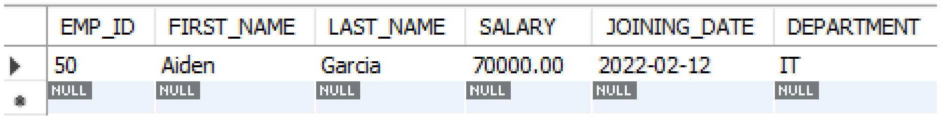
-- 6. SQL query to show only even rows from a table.

SELECT * FROM Employee WHERE MOD(EMP_ID, 2) = 0;



	EMP_ID	FIRST_NAME	LAST_NAME	SALARY	JOINING_DATE	DEPARTMENT
▶	2	Jane	Smith	60000	2022-02-01	IT
	4	Bob	Williams	70000	2022-04-01	Marketing
	6	Mike	Brown	52000	2022-06-01	IT
	8	Tom	Anderson	55000	2022-08-01	HR
	10	Chris	Taylor	67000	2022-10-01	Sales
	12	Laura	White	72000	2022-12-01	HR
	14	Grace	Smith	55000	2023-02-01	IT
	16	Jessica	Miller	50000	2023-04-01	Marketing
	18	Olivia	Jones	47000	2023-06-01	IT
	20	Sophia	Davis	5300(47000	23-08-01	Marketing
	52	Daniel	Perez	70000	2022-02-01	IT
	54	Mark	Johnson	60000	2022-04-01	Finance
	56	Edward	Martinez	67000	2022-06-01	IT
	58	Oliver	Davis	72000	2022-08-01	HR
	60	Ethan	Hill	59000	2022-10-01	Sales
	62	Liam	Clark	60000	2022-12-01	HR
	64	Noah	Ward	65000	2023-02-01	Finance
	66	Lucas	Harrison	70000	2023-04-01	Sales
	68	Liam	Clark	55000	2023-06-01	Finance
	70	James	Gordon	59000	2023-08-01	Sales
	72	William	Moore	70000	2023-02-01	IT
	74	Daniel	Barnes	60000	2023-04-01	Finance

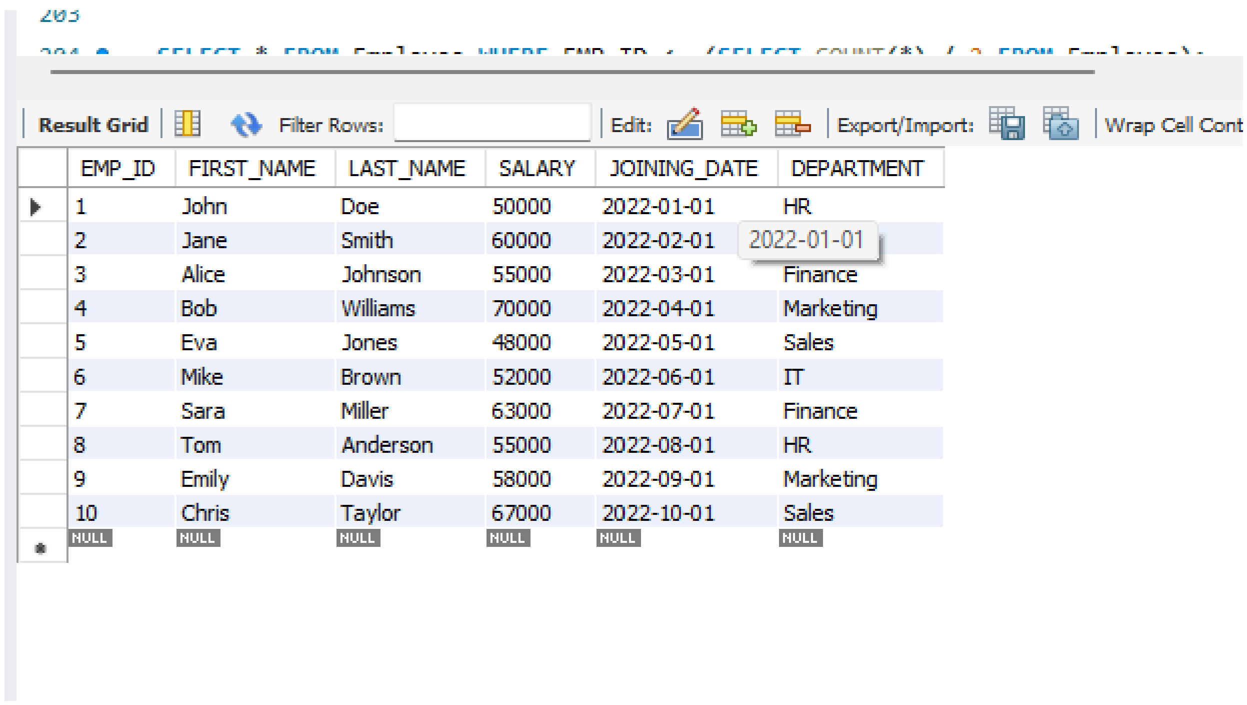
-- 7. SQL query to show records from one table that another table does not have. Find employees in employee table that do not exist in bonus table.
DELETE FROM Bonus WHERE EMP_REF_ID = 50;
SELECT * FROM Employee WHERE EMP_ID NOT IN (SELECT EMP_REF_ID FROM Bonus);



	EMP_ID	FIRST_NAME	LAST_NAME	SALARY	JOINING_DATE	DEPARTMENT
▶	50	Aiden	Garcia	70000.00	2022-02-12	IT
*	NULL	NULL	NULL	NULL	NULL	NULL

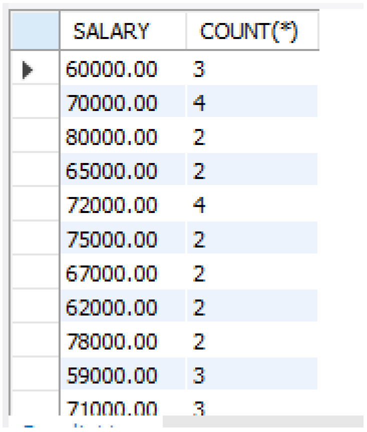
-- 8. SQL query to show the top n (say 10) records of a table.

SELECT * FROM Employee LIMIT 10;



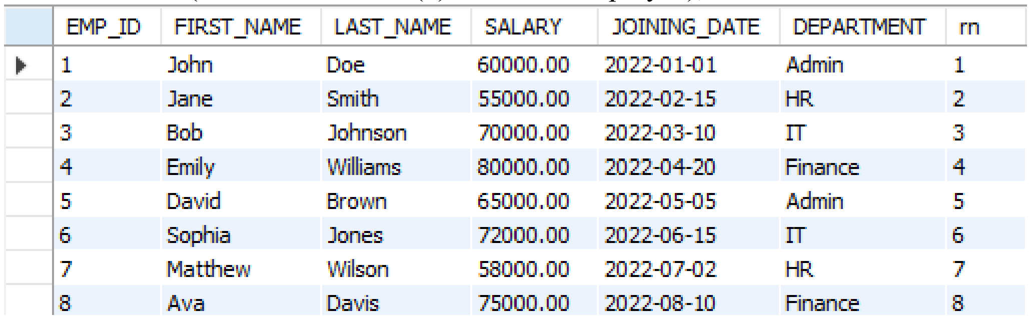
EMP_ID	FIRST_NAME	LAST_NAME	SALARY	JOINING_DATE	DEPARTMENT
1	John	Doe	50000	2022-01-01	HR
2	Jane	Smith	60000	2022-02-01	2022-01-01
3	Alice	Johnson	55000	2022-03-01	Finance
4	Bob	Williams	70000	2022-04-01	Marketing
5	Eva	Jones	48000	2022-05-01	Sales
6	Mike	Brown	52000	2022-06-01	IT
7	Sara	Miller	63000	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
NULL	NULL	NULL	NULL	NULL	NULL

-- 9. Find people who have the same salary.
SELECT SALARY, COUNT(*) FROM Employee GROUP BY SALARY HAVING COUNT(*) > 1;



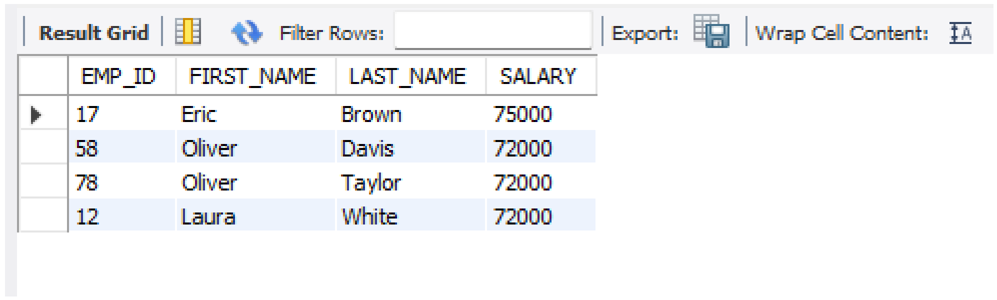
SALARY	COUNT(*)
60000.00	3
70000.00	4
80000.00	2
65000.00	2
72000.00	4
75000.00	2
67000.00	2
62000.00	2
78000.00	2
59000.00	3
71000.00	3

-- 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);
```



EMP_ID	FIRST_NAME	LAST_NAME	SALARY	JOINING_DATE	DEPARTMENT	rn
1	John	Doe	60000.00	2022-01-01	Admin	1
2	Jane	Smith	55000.00	2022-02-15	HR	2
3	Bob	Johnson	70000.00	2022-03-10	IT	3
4	Emily	Williams	80000.00	2022-04-20	Finance	4
5	David	Brown	65000.00	2022-05-05	Admin	5
6	Sophia	Jones	72000.00	2022-06-15	IT	6
7	Matthew	Wilson	58000.00	2022-07-02	HR	7
8	Ava	Davis	75000.00	2022-08-10	Finance	8

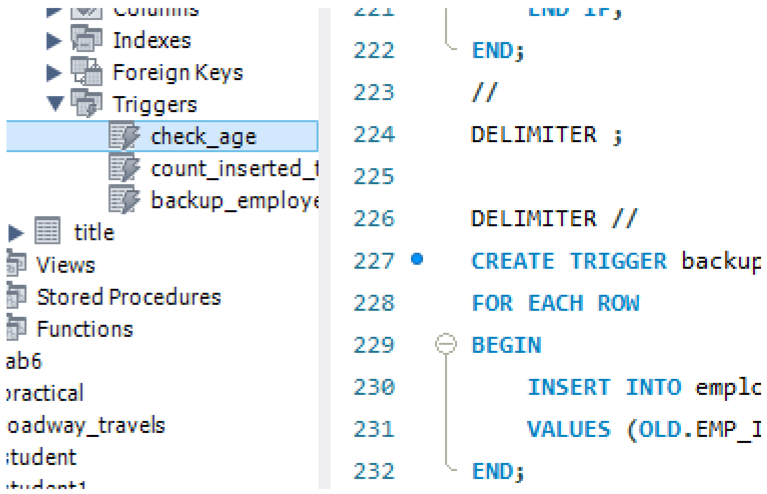
-- 11. Find the highest 2 salaries without LIMIT or TOP.
SELECT DISTINCT SALARY FROM Employee ORDER BY SALARY DESC LIMIT 2;



EMP_ID	FIRST_NAME	LAST_NAME	SALARY
17	Eric	Brown	75000
58	Oliver	Davis	72000
78	Oliver	Taylor	72000
12	Laura	White	72000

-- 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');
```

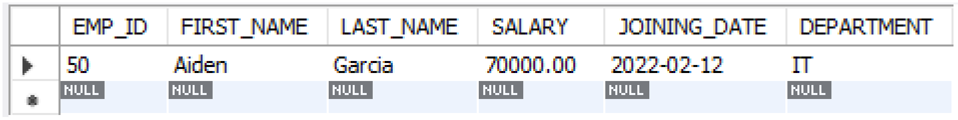


```
    END IF;
END;
//
DELIMITER ;

DELIMITER //
CREATE TRIGGER backup
FOR EACH ROW
BEGIN
    INSERT INTO emplo
    VALUES (OLD.EMP_I
END;
```

-- 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;
```

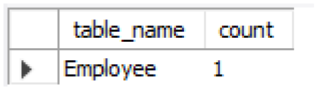
DELETE FROM Bonus WHERE EMP_REF_ID = 50;
DELETE FROM Employee WHERE EMP_ID = 50;
select * FROM employee_backup;



	EMP_ID	FIRST_NAME	LAST_NAME	SALARY	JOINING_DATE	DEPARTMENT
▶	50	Aiden	Garcia	70000.00	2022-02-12	IT
*	NULL	NULL	NULL	NULL	NULL	NULL

-- 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;
```



table_name	count
Employee	1

Conclusion:

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

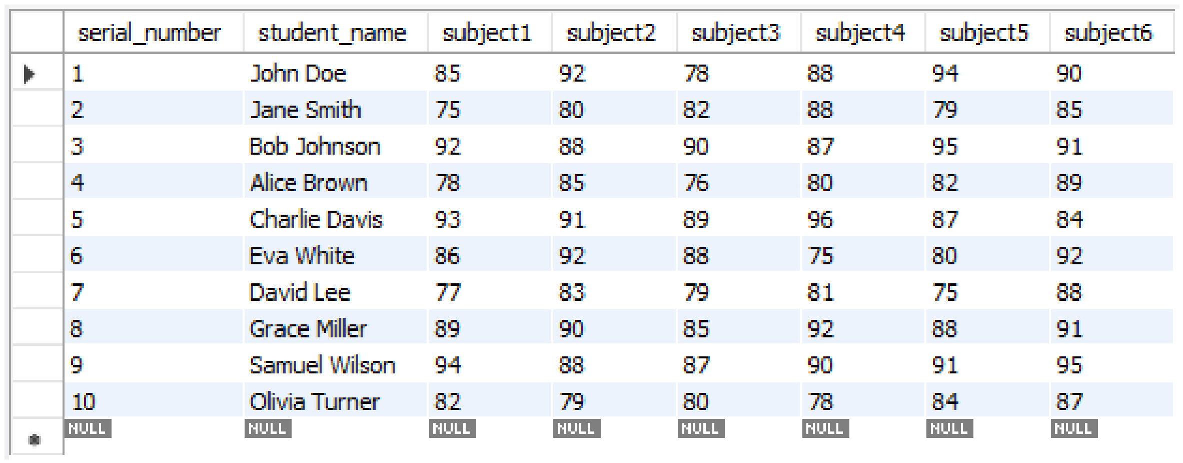
Assignment 2

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);
```

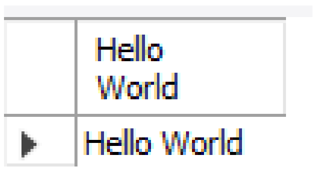
```
-- Select all records from the student table
SELECT * FROM student;
```



serial_number	student_name	subject1	subject2	subject3	subject4	subject5	subject6
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
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

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

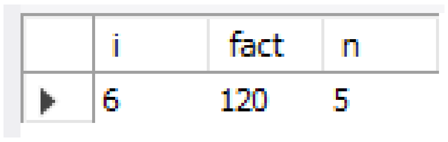
```
DELIMITER //
CREATE PROCEDURE temp()
BEGIN
  SELECT "Hello World";
END;
//
DELIMITER
;
-- Call the temp stored procedure
CALL temp();
```



Hello World
Hello World

-- Create a stored procedure to calculate factorial

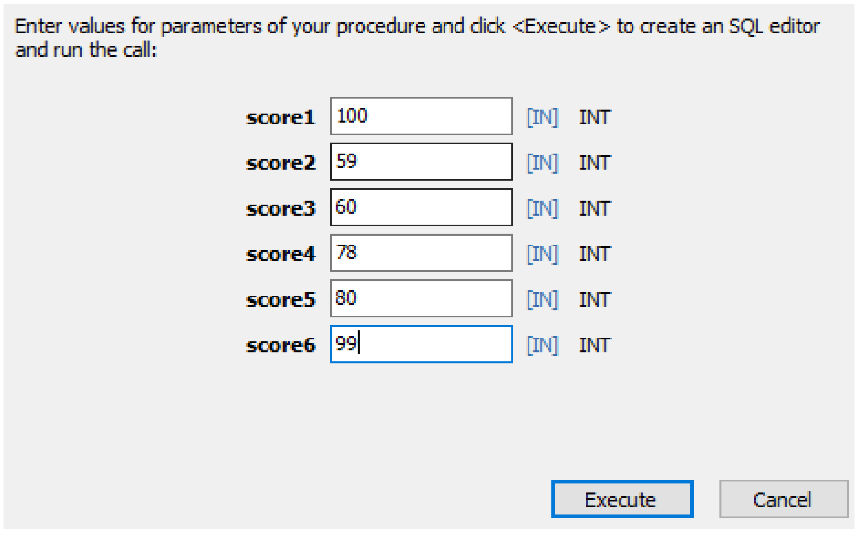
```
DELIMITER //
CREATE PROCEDURE facto(IN n INT)
BEGIN
  DECLARE i INT DEFAULT 1;
  DECLARE fact INT DEFAULT 1;
  factorial: LOOP
    SET fact = fact * i;
    SET i = i + 1;
    IF i <= n THEN
      ITERATE
    factorial;
    END IF;
    LEAVE factorial;
  END LOOP;

  SELECT i, fact,
n;
END;
//
DELIMITER
;

call studentdata.facto(5);
```



```
  i    fact   n
▶ 6    120    5
```

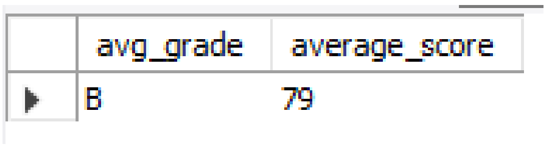
-- Create a stored function to calculate the average grade for 6 subjects

```
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
;
```



Enter values for parameters of your procedure and click <Execute> to create an SQL editor and run the call:

score1	100	[IN]	INT
score2	59	[IN]	INT
score3	60	[IN]	INT
score4	78	[IN]	INT
score5	80	[IN]	INT
score6	99	[IN]	INT

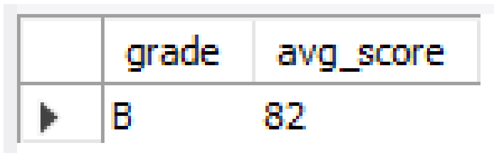


avg_grade	average_score
B	79

-- 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);
```



grade	avg_score
B	82

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.

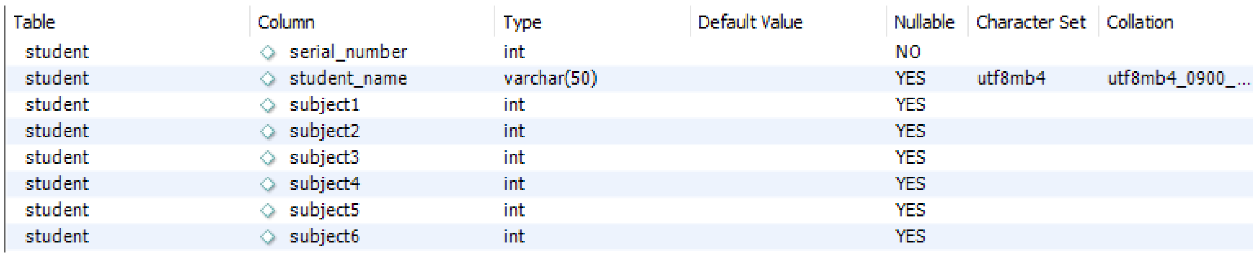
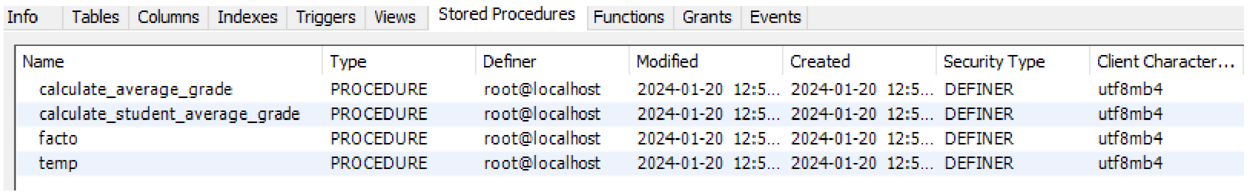


Table	Column	Type	Default Value	Nullable	Character Set	Collation
student	serial_number	int		NO		
student	student_name	varchar(50)		YES	utf8mb4	utf8mb4_0900_...
student	subject1	int		YES		
student	subject2	int		YES		
student	subject3	int		YES		
student	subject4	int		YES		
student	subject5	int		YES		
student	subject6	int		YES		

Figure - Database of student table



Name	Type	Definer	Modified	Created	Security Type	Client Character...
calculate_average_grade	PROCEDURE	root@localhost	2024-01-20 12:5...	2024-01-20 12:5...	DEFINER	utf8mb4
calculate_student_average_grade	PROCEDURE	root@localhost	2024-01-20 12:5...	2024-01-20 12:5...	DEFINER	utf8mb4
facto	PROCEDURE	root@localhost	2024-01-20 12:5...	2024-01-20 12:5...	DEFINER	utf8mb4
temp	PROCEDURE	root@localhost	2024-01-20 12:5...	2024-01-20 12:5...	DEFINER	utf8mb4

Figure - Procedure using PL/SQL

Assignment 3

Aim:

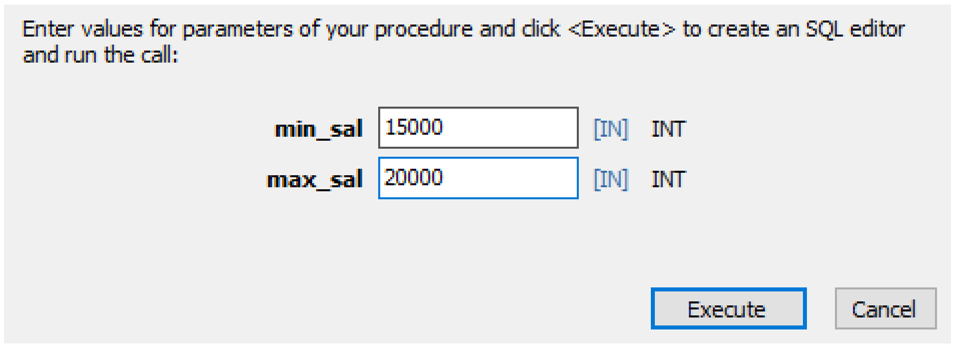
Consider the following table to write PL/SQL code as specified under
Teacher (t_no, f_name, l_name, salary, supervisor, joining_date, birth_date, title)
Class (class_no, t_no, room_no)
Pay_scale (Min_limit, Max_limit, grade)

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:
   a) if salary< 10000 then bonus is 10% of the salary.
   b) if salary is between 10000 and 20000 then bonus is 20% of the salary.
   c) if salary is between 20000 and 25000 then bonus is 25% of the salary.
   d) 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 :

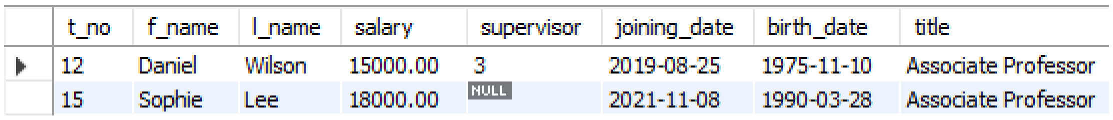
```
create DATABASE Teachers;
use Teachers;

-- 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
;
```



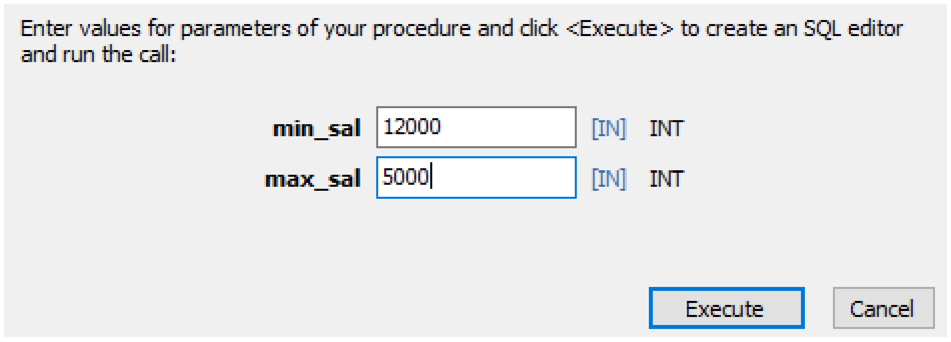
Enter values for parameters of your procedure and click <Execute> to create an SQL editor and run the call:

min_sal	15000	[IN]	INT
max_sal	20000	[IN]	INT



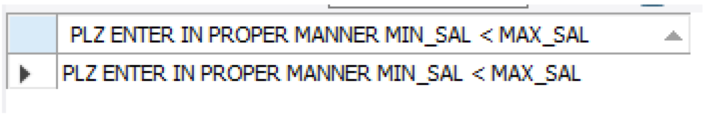
t_no	f_name	l_name	salary	supervisor	joining_date	birth_date	title
12	Daniel	Wilson	15000.00	3	2019-08-25	1975-11-10	Associate Professor
15	Sophie	Lee	18000.00	NULL	2021-11-08	1990-03-28	Associate Professor

Figure shows a Procedure of 1 output and input



Enter values for parameters of your procedure and click <Execute> to create an SQL editor and run the call:

min_sal	12000	[IN]	INT
max_sal	5000	[IN]	INT



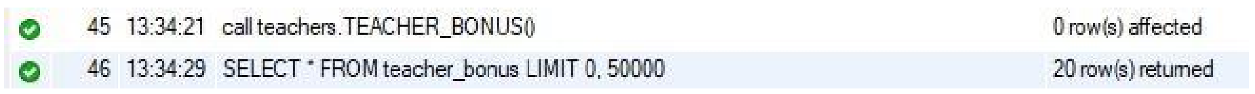
```
PLZ ENTER IN PROPER MANNER MIN_SAL < MAX_SAL
PLZ ENTER IN PROPER MANNER MIN_SAL < MAX_SAL
```

Figure shows a Procedure of 1 output and input wrong input

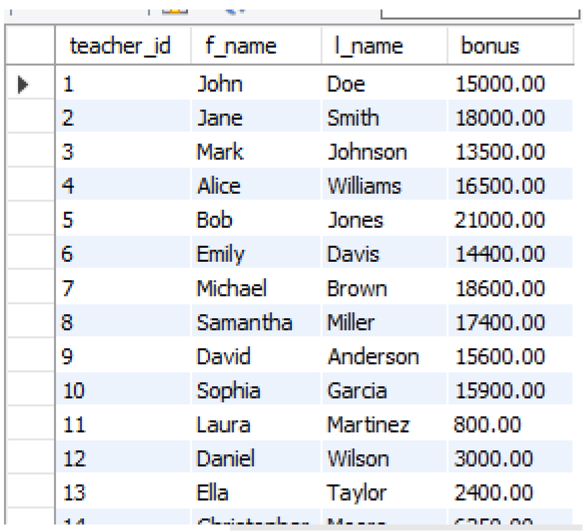
```
-- Task 2: Calculate the bonus amount using a cursor
DELIMITER //
CREATE PROCEDURE TEACHER_BONUS()
BEGIN
    -- Declare variables to store fetched data
    DECLARE v_teacher_id INT;
    DECLARE v_f_name VARCHAR(250);
    DECLARE v_l_name VARCHAR(250);
    DECLARE v_salary DECIMAL(10, 2);
    DECLARE 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;
```



```
45  13:34:21  call teachers.TEACHER_BONUS()                     0 row(s) affected
46  13:34:29  SELECT * FROM teacher_bonus LIMIT 0, 50000        20 row(s) returned
```



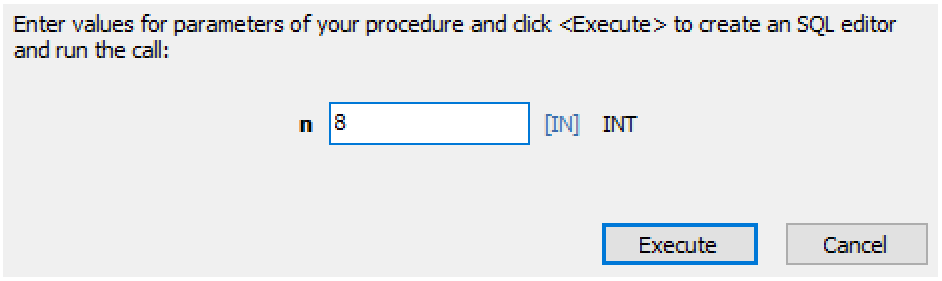
teacher_id	f_name	l_name	bonus
1	John	Doe	15000.00
2	Jane	Smith	18000.00
3	Mark	Johnson	13500.00
4	Alice	Williams	16500.00
5	Bob	Jones	21000.00
6	Emily	Davis	14400.00
7	Michael	Brown	18600.00
8	Samantha	Miller	17400.00
9	David	Anderson	15600.00
10	Sophia	Garcia	15900.00
11	Laura	Martinez	800.00
12	Daniel	Wilson	3000.00
13	Ella	Taylor	2400.00
[illegible]	[illegible]	[illegible]	[illegible]

Figure shows a Procedure of 2 output

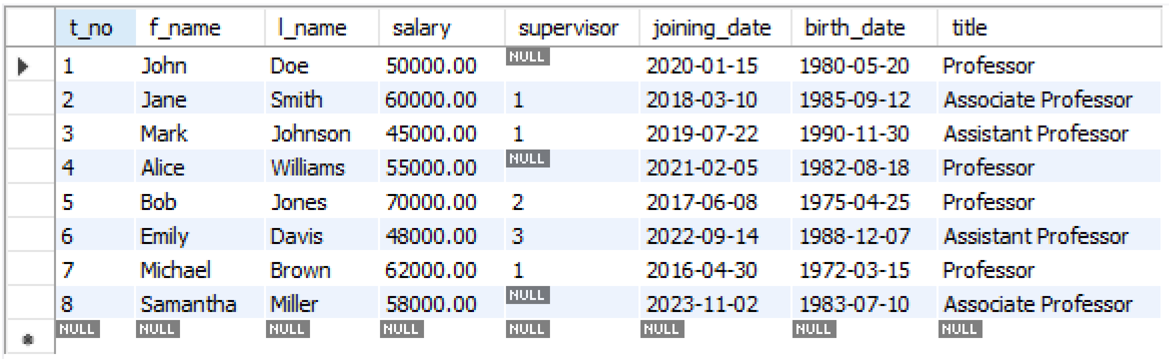
```
-- Task 3: Using a simple LOOP structure, list the first 10 records of the ‘teachers’ table.
DELIMITER //

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;
```


Enter values for parameters of your procedure and click <Execute> to create an SQL editor and run the call:

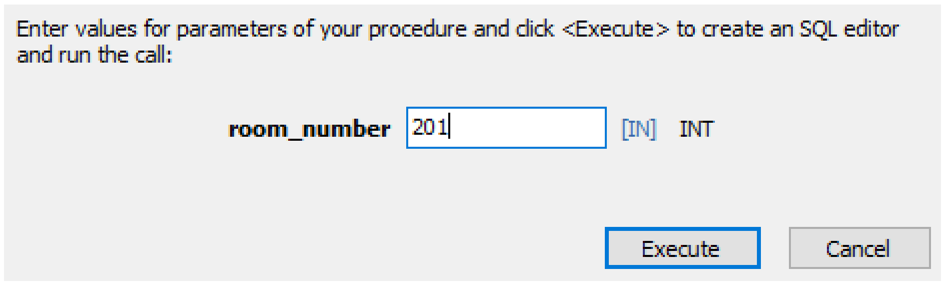
n 8 [IN] INT



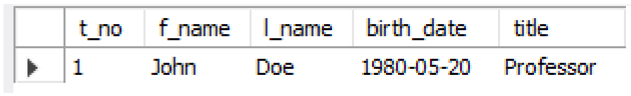
t_no	f_name	l_name	salary	supervisor	joining_date	birth_date	title
1	John	Doe	50000.00	NULL	2020-01-15	1980-05-20	Professor
2	Jane	Smith	60000.00	1	2018-03-10	1985-09-12	Associate Professor
3	Mark	Johnson	45000.00	1	2019-07-22	1990-11-30	Assistant Professor
4	Alice	Williams	55000.00	NULL	2021-02-05	1982-08-18	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
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

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

DELIMITER ;
```

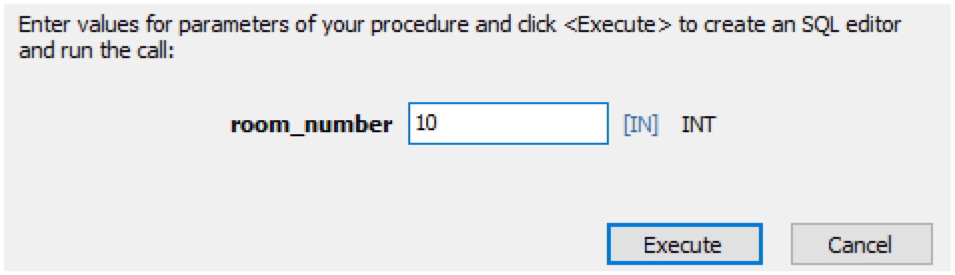

Enter values for parameters of your procedure and click <Execute> to create an SQL editor and run the call:

room_number 201 [IN] INT

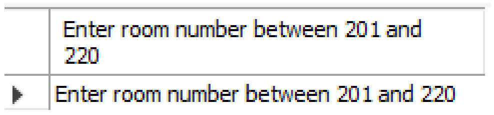


	t_no	f_name	l_name	birth_date	title
▶	1	John	Doe	1980-05-20	Professor

Figure shows a Procedure of 4 input output


Enter values for parameters of your procedure and click <Execute> to create an SQL editor and run the call:

room_number 10 [IN] INT



	Enter room number between 201 and 220
▶	Enter room number between 201 and 220

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.

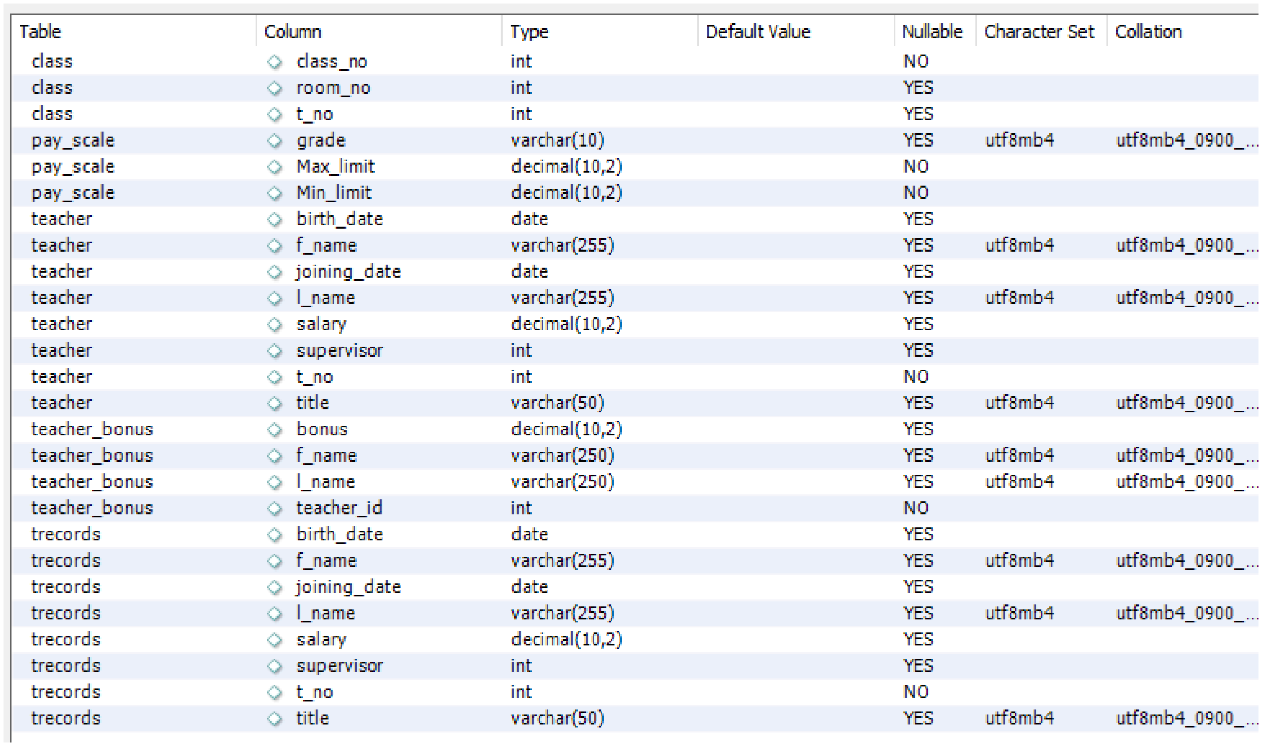
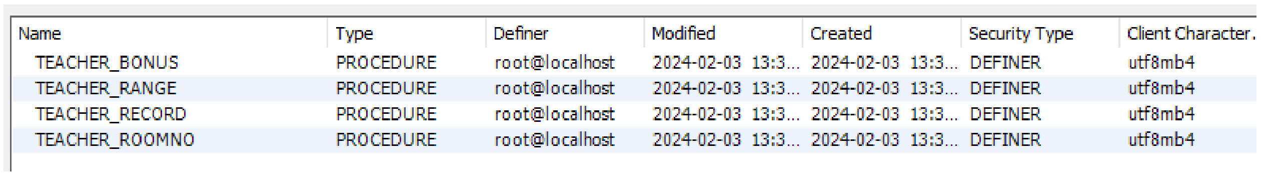


Table	Column	Type	Default Value	Nullable	Character Set	Collation
class	class_no	int		NO		
class	room_no	int		YES		
class	t_no	int		YES		
pay_scale	grade	varchar(10)		YES	utf8mb4	utf8mb4_0900_...
pay_scale	Max_limit	decimal(10,2)		NO		
pay_scale	Min_limit	decimal(10,2)		NO		
teacher	birth_date	date		YES		
teacher	f_name	varchar(255)		YES	utf8mb4	utf8mb4_0900_...
teacher	joining_date	date		YES		
teacher	l_name	varchar(255)		YES	utf8mb4	utf8mb4_0900_...
teacher	salary	decimal(10,2)		YES		
teacher	supervisor	int		YES		
teacher	t_no	int		NO		
teacher	title	varchar(50)		YES	utf8mb4	utf8mb4_0900_...
teacher_bonus	bonus	decimal(10,2)		YES		
teacher_bonus	f_name	varchar(250)		YES	utf8mb4	utf8mb4_0900_...
teacher_bonus	l_name	varchar(250)		YES	utf8mb4	utf8mb4_0900_...
teacher_bonus	teacher_id	int		NO		
trecords	birth_date	date		YES		
trecords	f_name	varchar(255)		YES	utf8mb4	utf8mb4_0900_...
trecords	joining_date	date		YES		
trecords	l_name	varchar(255)		YES	utf8mb4	utf8mb4_0900_...
trecords	salary	decimal(10,2)		YES		
trecords	supervisor	int		YES		
trecords	t_no	int		NO		
trecords	title	varchar(50)		YES	utf8mb4	utf8mb4_0900_...

Figure shows a database table for given task



Name	Type	Definer	Modified	Created	Security Type	Client Character
TEACHER_BONUS	PROCEDURE	root@localhost	2024-02-03 13:3...	2024-02-03 13:3...	DEFINER	utf8mb4
TEACHER_RANGE	PROCEDURE	root@localhost	2024-02-03 13:3...	2024-02-03 13:3...	DEFINER	utf8mb4
TEACHER_RECORD	PROCEDURE	root@localhost	2024-02-03 13:3...	2024-02-03 13:3...	DEFINER	utf8mb4
TEACHER_ROOMNO	PROCEDURE	root@localhost	2024-02-03 13:3...	2024-02-03 13:3...	DEFINER	utf8mb4

Figure shows all 4 Procedure for given task

Assignment 4

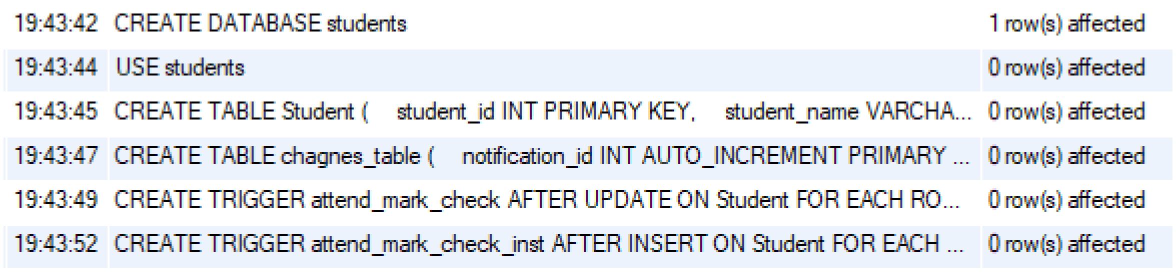
Aim:

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:

a) Whenever attendance is updated, check if the attendance is less than 85%; if so notify the Head of Department concerned.
b) 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 :

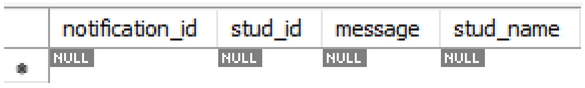
```
CREATE DATABASE
students; USE students;

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
;
```



```
19:43:42  CREATE DATABASE students                                                          1 row(s) affected
19:43:44  USE students                                                                      0 row(s) affected
19:43:45  CREATE TABLE Student (    student_id INT PRIMARY KEY,    student_name VARCHA...   0 row(s) affected
19:43:47  CREATE TABLE chagnes_table (    notification_id INT AUTO_INCREMENT PRIMARY ...    0 row(s) affected
19:43:49  CREATE TRIGGER attend_mark_check AFTER UPDATE ON Student FOR EACH RO...           0 row(s) affected
19:43:52  CREATE TRIGGER attend_mark_check_inst AFTER INSERT ON Student FOR EACH ...        0 row(s) affected
```

Figure 4.1 shows above queries are successfully executed

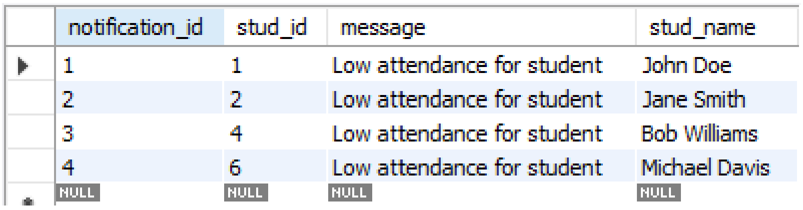
select * from chagnes_table;



	notification_id	stud_id	message	stud_name
*	NULL	NULL	NULL	NULL

Figure 4.2 null table

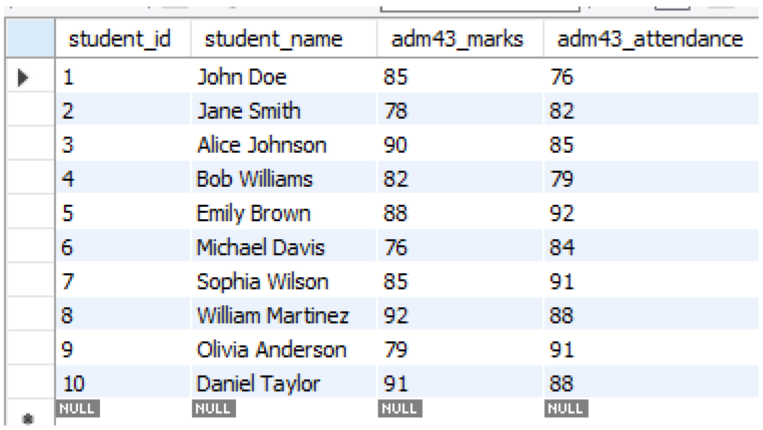
```
INSERT INTO Student (student_id, student_name, adm43_marks, adm43_attendance)
VALUES
(1, 'John Doe', 85, 76),
(2, 'Jane Smith', 78, 82),
(3, 'Alice Johnson', 90, 85),
(4, 'Bob Williams', 82, 79),
(5, 'Emily Brown', 88, 92),
(6, 'Michael Davis', 76, 84),
(7, 'Sophia Wilson', 85, 91),
(8, 'William Martinez', 92, 88),
(9, 'Olivia Anderson', 79, 91),
(10, 'Daniel Taylor', 91, 88);

select * from chagnes_table;
```



notification_id	stud_id	message	stud_name
1	1	Low attendance for student	John Doe
2	2	Low attendance for student	Jane Smith
3	4	Low attendance for student	Bob Williams
4	6	Low attendance for student	Michael Davis
NULL	NULL	NULL	NULL

Figure 4.3 Table has changes by insertion trigger

```
select * from student;
```



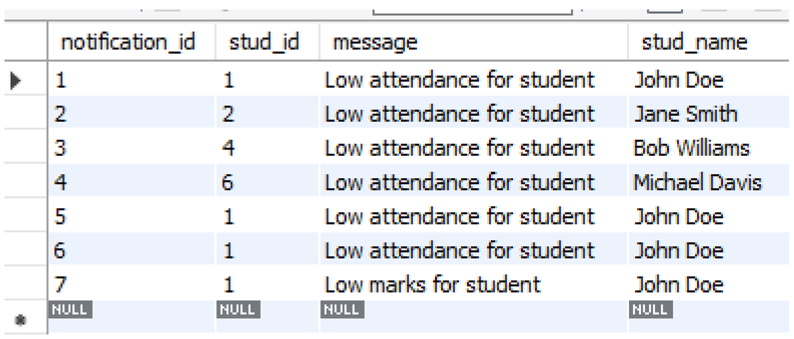
student_id	student_name	adm43_marks	adm43_attendance
1	John Doe	85	76
2	Jane Smith	78	82
3	Alice Johnson	90	85
4	Bob Williams	82	79
5	Emily Brown	88	92
6	Michael Davis	76	84
7	Sophia Wilson	85	91
8	William Martinez	92	88
9	Olivia Anderson	79	91
10	Daniel Taylor	91	88
NULL	NULL	NULL	NULL

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;

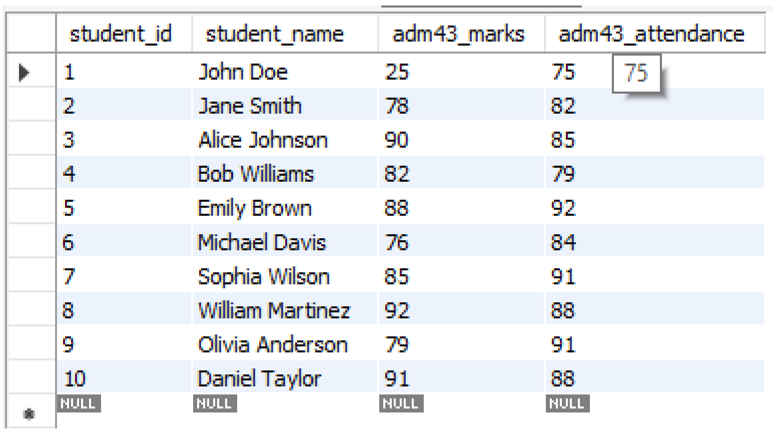
select * from chagnes_table;



notification_id	stud_id	message	stud_name
1	1	Low attendance for student	John Doe
2	2	Low attendance for student	Jane Smith
3	4	Low attendance for student	Bob Williams
4	6	Low attendance for student	Michael Davis
5	1	Low attendance for student	John Doe
6	1	Low attendance for student	John Doe
7	1	Low marks for student	John Doe
NULL	NULL	NULL	NULL

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

select * from student;



student_id	student_name	adm43_marks	adm43_attendance
1	John Doe	25	75
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
NULL	NULL	NULL	NULL

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:

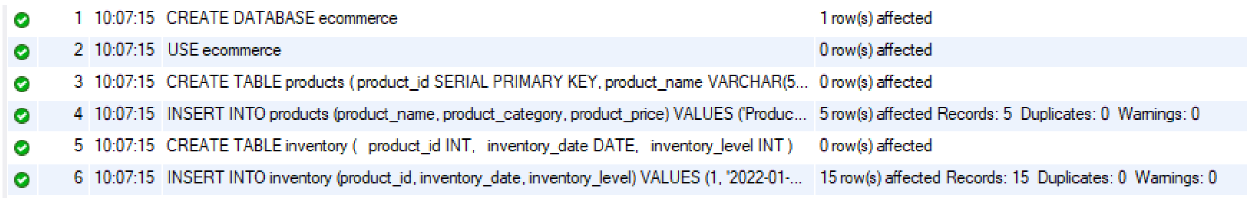
How to analyze ecommerce Inventory

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
);
```

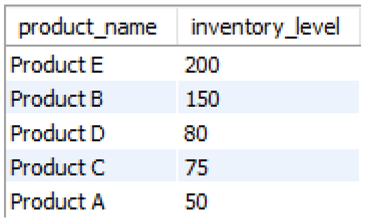
```
INSERT INTO inventory (product_id, inventory_date, inventory_level)
VALUES (1, '2022-01-01', 100),
       (2, '2022-01-01', 200),
       (3, '2022-01-01', 150),
       (4, '2022-01-01', 75),
       (5, '2022-01-01', 250),
       (1, '2022-01-02', 80),
       (2, '2022-01-02', 180),
       (3, '2022-01-02', 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	10:07:15	CREATE DATABASE ecommerce	1 row(s) affected
✓	2	10:07:15	USE ecommerce	0 row(s) affected
✓	3	10:07:15	CREATE TABLE products (product_id SERIAL PRIMARY KEY, product_name VARCHAR(5...	0 row(s) affected
✓	4	10:07:15	INSERT INTO products (product_name, product_category, product_price) VALUES ('Produc...	5 row(s) affected Records: 5 Duplicates: 0 Warnings: 0
✓	5	10:07:15	CREATE TABLE inventory (product_id INT, inventory_date DATE, inventory_level INT)	0 row(s) affected
✓	6	10:07:15	INSERT INTO inventory (product_id, inventory_date, inventory_level) VALUES (1, '2022-01-...	15 row(s) affected Records: 15 Duplicates: 0 Warnings: 0

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

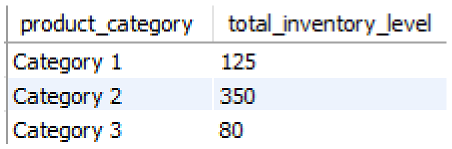
```
SELECT p.product_name, i.inventory_level
FROM products p
JOIN inventory i ON p.product_id = i.product_id
WHERE i.inventory_date = (SELECT MAX(inventory_date) FROM inventory)
ORDER BY i.inventory_level DESC
LIMIT 5;
```



product_name	inventory_level
Product E	200
Product B	150
Product D	80
Product C	75
Product A	50

2) What is the total inventory level for each product category on the most recent inventory date ?

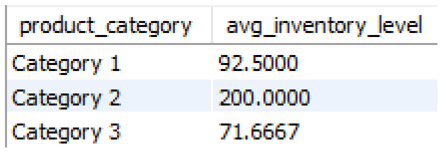
```
SELECT p.product_category, SUM(i.inventory_level) AS total_inventory_level
FROM products p
JOIN inventory i ON p.product_id = i.product_id
WHERE i.inventory_date = (SELECT MAX(inventory_date) FROM inventory)
GROUP BY p.product_category;
```



product_category	total_inventory_level
Category 1	125
Category 2	350
Category 3	80

3) What is the average inventory level for each product category for the month of January 2022 ?

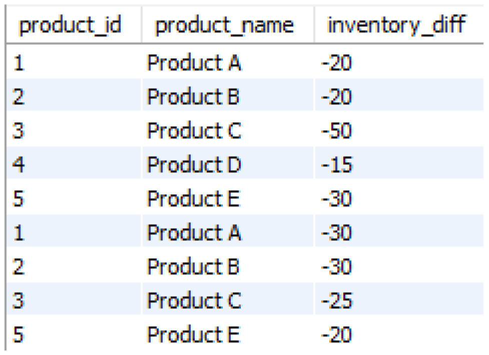
```
SELECT p.product_category, AVG(i.inventory_level) AS
avg_inventory_level FROM products p
JOIN inventory i ON p.product_id = i.product_id
WHERE i.inventory_date >= '2022-01-01' AND i.inventory_date < '2022-02-01'
GROUP BY p.product_category;
```



product_category	avg_inventory_level
Category 1	92.5000
Category 2	200.0000
Category 3	71.6667

4) Which products had a decrease in inventory level from the previous inventory date to the current inventory date ?

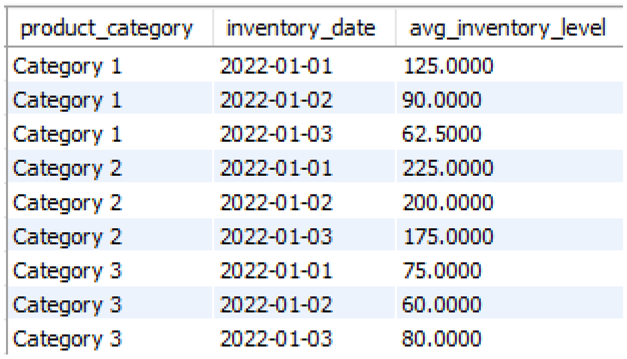
```
SELECT i1.product_id, p.product_name, i1.inventory_level - i2.inventory_level
AS inventory_diff
FROM inventory i1
JOIN inventory i2 ON i1.product_id = i2.product_id
    AND i1.inventory_date = i2.inventory_date + INTERVAL 1 day
JOIN products p ON i1.product_id = p.product_id
WHERE i1.inventory_level < i2.inventory_level;
```



product_id	product_name	inventory_diff
1	Product A	-20
2	Product B	-20
3	Product C	-50
4	Product D	-15
5	Product E	-30
1	Product A	-30
2	Product B	-30
3	Product C	-25
5	Product E	-20

5) 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;
```



product_category	inventory_date	avg_inventory_level
Category 1	2022-01-01	125.0000
Category 1	2022-01-02	90.0000
Category 1	2022-01-03	62.5000
Category 2	2022-01-01	225.0000
Category 2	2022-01-02	200.0000
Category 2	2022-01-03	175.0000
Category 3	2022-01-01	75.0000
Category 3	2022-01-02	60.0000
Category 3	2022-01-03	80.0000

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)

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

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

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

D) 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

12. 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)
)
```