Practical-2: Subquery-join operations on Relational Schema

- **USING (practical 1)**

1. Count the customers with grades above Bangalore's average.

Code:

SELECT COUNT(*) FROM customer WHERE grade / (SELECT AVG(grade)FROM customer WHERE city='Banglore');

```
mysql> SELECT COUNT(*) FROM customer WHERE grade / (SELECT AVG(grade)F
ROM customer WHERE city='Banglore');
+----------+
| COUNT(*) |
+----------+
|        0 |
+----------+
1 row in set (0.00 sec)
```

2. Find the name and numbers of all salesmen who had more than one customer.

Code:

SELECT s.name,s.salesman_id

FROM salseman s

FROM salesman s

JOIN customer c ON s.salesman_id=c.salesman_id

GROUP BY s.salesman_id

HAVING COUNT(c.customer_id)>1;

```
mysql> SELECT s.name,s.salesman_id
    -> FROM salesman s
    -> JOIN customer c ON s.salesman_id=c.salesman_id
    -> GROUP BY s.salesman_id
    -> HAVING COUNT(c.customer_id)>1;
+------------+-------------+
| name       | salesman_id |
+------------+-------------+
| James Hoog |        5001 |
| Nail Knite |        5002 |
+------------+-------------+
2 rows in set (0.00 sec)
```

3. List all salesmen and indicate those who have and don't have customers in their cities

(Use UNION operation.)

Code:

```
SELECT s.name,'No customers in city' AS status
   -> FROM salesman s
   -> JOIN customer c ON s.salesman_id=c.salesman_id
   -> WHERE s.city=c.city
   -> GROUP BY s.name
   -> UNION
   -> SELECT s.name,'No customers in city' AS status
   -> FROM salesman s
   -> LEFT JOIN customer c ON s.salesman_id=c.salesman_id AND s.city=c.city
   -> WHERE c.customer_id IS NULL;
```

```
mysql> SELECT s.name,'No customers in city' AS status
    -> FROM salesman s
    -> JOIN customer c ON s.salesman_id=c.salesman_id
    -> WHERE s.city=c.city
    -> GROUP BY s.name
    -> UNION
    -> SELECT s.name,'No customers in city' AS status
    -> FROM salesman s
    -> LEFT JOIN customer c ON s.salesman_id=c.salesman_id AND s.city=
c.city
    -> WHERE c.customer_id IS NULL;
+------------+----------------------+
| name       | status               |
+------------+----------------------+
| James Hoog | No customers in city |
| Mc Lyon    | No customers in city |
| Nail Knite | No customers in city |
| Lauson Hen | No customers in city |
| Pit Alex   | No customers in city |
| Paul Adam  | No customers in city |
+------------+----------------------+
6 rows in set (0.01 sec)
```

4. Create a view that finds the salesman who has the customer with the highest order of a

day.

Code:

```
CREATE VIEW SalesmanWithHighestOrder AS
SELECT S.salesman_id, S.name, O.order_date, MAX(O.purch_amt) AS max_order_amount
```

FROM Salesman S

JOIN Customer C ON S.salesman_id = C.salesman_id

JOIN `orders` O ON C.customer_id = O.customer_id GROUP BY

S.salesman_id, S.name, O.order_date; select * from

SalesmanWithHighestOrder;

Output:

```
mysql> CREATE VIEW SalesmanWithHighestOrder AS
    -> SELECT S.salesman_id, S.name, O.order_date, MAX(O.purch_amt) AS max_order_amount
    -> FROM Salesman S
    -> JOIN Customer C ON S.salesman_id = C.salesman_id
    -> JOIN `orders` O ON C.customer_id = O.customer_id
    -> GROUP BY S.salesman_id, S.name, O.order_date;
Query OK, 0 rows affected (0.01 sec)

mysql> select * from SalesmanWithHighestOrder;
+-------------+------------+------------+------------------+
| salesman_id | name       | order_date | max_order_amount |
+-------------+------------+------------+------------------+
|        5001 | James Hoog | 2016-07-27 |          2400.60 |
|        5001 | James Hoog | 2016-09-10 |          5760.00 |
|        5001 | James Hoog | 2016-10-05 |            65.26 |
|        5002 | Nail Knite | 2016-06-27 |           250.45 |
|        5002 | Nail Knite | 2016-09-10 |           948.50 |
|        5002 | Nail Knite | 2016-10-05 |           150.50 |
|        5003 | Lauson Hen | 2016-08-17 |           110.50 |
|        5003 | Lauson Hen | 2016-10-10 |          2480.40 |
|        5005 | Pit Alex   | 2016-09-10 |           270.65 |
|        5006 | Mc Lyon    | 2016-10-10 |          1983.43 |
|        5007 | Paul Adam  | 2016-08-17 |            75.29 |
+-------------+------------+------------+------------------+
11 rows in set (0.03 sec)
```

5. Demonstrate the DELETE operation by removing salesman with id 1000. All his orders

must also be deleted

Code:

DELETE FROM orders WHERE salesman_id=1000;

Query OK, 0 rows affected (0.00 sec)

DELETE FROM salesman WHERE salesman_id=1000;

Query OK, 0 rows affected (0.00 sec)

```
mysql> DELETE FROM orders WHERE salesman_id=1000;
Query OK, 0 rows affected (0.00 sec)

mysql> DELETE FROM salesman WHERE salesman_id=1000;
Query OK, 0 rows affected (0.00 sec)
```

2. Design ERD for the following schema and execute the following Queries on it:

Consider the schema for Movie Database:

ACTOR (Act_id, Act_Name, Act_Gender)

DIRECTOR (Dir_id, Dir_Name, Dir_Phone)

MOVIES (Mov_id, Mov_Title, Mov_Year, Mov_Lang, Dir_id)

MOVIE_CAST (Act_id, Mov_id, Role)

RATING (Mov_id, Rev_Stars)

Code:

CREATE TABLE ACTOR (

ACT_ID INT (3),

ACT_NAME VARCHAR (20),

ACT_GENDER CHAR (1), PRIMARY KEY

(ACT_ID));

Output:

```
mysql> CREATE TABLE ACTOR (
    ->     ACT_ID INT(3),
    ->     ACT_NAME VARCHAR(20),
    ->     ACT_GENDER CHAR(1),
    ->     PRIMARY KEY (ACT_ID)
    -> );
Query OK, 0 rows affected (0.02 sec)
```

2. Design ERD for the following schema and execute the following Queries on it:

Consider the schema for Movie Database:

ACTOR (Act_id, Act_Name, Act_Gender)

DIRECTOR (Dir_id, Dir_Name, Dir_Phone)

MOVIES (Mov_id, Mov_Title, Mov_Year, Mov_Lang, Dir_id)

MOVIE_CAST (Act_id, Mov_id, Role)

RATING (Mov_id, Rev_Stars)

Code:

CREATE TABLE ACTOR (

ACT_ID INT (3),

ACT_NAME VARCHAR (20),

ACT_GENDER CHAR (1), PRIMARY KEY

(ACT_ID));

Output:

```
mysql> CREATE TABLE ACTOR (
    ->     ACT_ID INT(3),
    ->     ACT_NAME VARCHAR(20),
    ->     ACT_GENDER CHAR(1),
    ->     PRIMARY KEY (ACT_ID)
    -> );
Query OK, 0 rows affected (0.02 sec)
```

CREATE TABLE DIRECTOR (DIR_ID INT (3),

DIR_NAME VARCHAR (20),

DIR_PHONE INT (10), PRIMARY KEY

(DIR_ID));

Output:

```
mysql> CREATE TABLE DIRECTOR (
    -> DIR_ID INT (3),
    -> DIR_NAME VARCHAR (20),
    -> DIR_PHONE INT (10),
    -> PRIMARY KEY (DIR_ID));
Query OK, 0 rows affected (0.01 sec)
```

Code:

CREATE TABLE MOVIES (

MOV_ID INT (4),

MOV_TITLE VARCHAR (25),

MOV_YEAR INT (4),

MOV_LANG VARCHAR (12),

DIR_ID INT (3),

PRIMARY KEY (MOV_ID),

FOREIGN KEY (DIR_ID) REFERENCES DIRECTOR (DIR_ID));

Output:

```
mysql> CREATE TABLE MOVIES (
    -> MOV_ID INT (4),
    -> MOV_TITLE VARCHAR (25),
    -> MOV_YEAR INT (4),
    -> MOV_LANG VARCHAR (12),
    -> DIR_ID INT (3),
    -> PRIMARY KEY (MOV_ID),
    -> FOREIGN KEY (DIR_ID) REFERENCES DIRECTOR (DIR_ID));
Query OK, 0 rows affected (0.05 sec)
```

CREATE TABLE MOVIE_CAST (

ACT_ID INT (3),

MOV_ID INT (4),

OLE VARCHAR (10),

PRIMARY KEY (ACT_ID, MOV_ID),

FOREIGN KEY (ACT_ID) REFERENCES ACTOR (ACT_ID), FOREIGN KEY (MOV_ID)

REFERENCES MOVIES (MOV_ID));

Output:

```
mysql> CREATE TABLE MOVIE_CAST (
    -> ACT_ID INT (3),
    -> MOV_ID INT (4),
    ->
    -> OLE VARCHAR (10),
    -> PRIMARY KEY (ACT_ID, MOV_ID),
    -> FOREIGN KEY (ACT_ID) REFERENCES ACTOR (ACT_ID),
    -> FOREIGN KEY (MOV_ID) REFERENCES MOVIES (MOV_ID));
Query OK, 0 rows affected (0.01 sec)
```

Code:

CREATE TABLE RATING (

MOV_ID INT (4),

REV_STARS VARCHAR (25),

PRIMARY KEY (MOV_ID),

FOREIGN KEY (MOV_ID) REFERENCES MOVIES (MOV_ID));

Output:

```
mysql> CREATE TABLE RATING (
    -> MOV_ID INT (4),
    -> REV_STARS VARCHAR (25),
    -> PRIMARY KEY (MOV_ID),
    -> FOREIGN KEY (MOV_ID) REFERENCES MOVIES (MOV_ID));
Query OK, 0 rows affected (0.01 sec)
```

INSERT INTO ACTOR VALUES (301,'ANUSHKA','F');

INSERT INTO ACTOR VALUES (302,'PRABHAS','M');

INSERT INTO ACTOR VALUES (303,'PUNITH','M');

INSERT INTO ACTOR VALUES (304,'JERMY','M');

Output:

```
mysql> INSERT INTO ACTOR VALUES (301,'ANUSHKA','F');
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO ACTOR VALUES (302,'PRABHAS','M');
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO ACTOR VALUES (303,'PUNITH','M');
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO ACTOR VALUES (304,'JERMY','M');
Query OK, 1 row affected (0.00 sec)
```

Code:

INSERT INTO DIRECTOR VALUES (60,'RAJAMOULI', 875161100);

INSERT INTO DIRECTOR VALUES (61,'HITCHCOCK', 776613891);

INSERT INTO DIRECTOR VALUES (62,'FARAN', 998677653);

INSERT INTO DIRECTOR VALUES (63,'STEVEN SPIELBERG', 898977653);

Output:

```
mysql> INSERT INTO DIRECTOR VALUES (60,'RAJAMOULI', 875161100);
Query OK, 1 row affected (0.03 sec)

mysql> INSERT INTO DIRECTOR VALUES (61,'HITCHCOCK', 776613891);
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO DIRECTOR VALUES (62,'FARAN', 998677653);
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO DIRECTOR VALUES (63,'STEVEN SPIELBERG', 898977653);
Query OK, 1 row affected (0.00 sec)
```

INSERT INTO MOVIES VALUES (1001,'BAHUBALI-2', 2017, 'TELAGU', 60);

INSERT INTO MOVIES VALUES (1002,'BAHUBALI-1', 2015, 'TELAGU', 60);

INSERT INTO MOVIES VALUES (1003,'AKASH', 2008, 'KANNADA', 61);

INSERT INTO MOVIES VALUES (1004,'WAR HORSE', 2011, 'ENGLISH', 63);

Output:

```
mysql> INSERT INTO MOVIES VALUES (1001,'BAHUBALI-2', 2017, 'TELAGU', 60);
Query OK, 1 row affected (0.01 sec)

mysql> INSERT INTO MOVIES VALUES (1002,'BAHUBALI-1', 2015, 'TELAGU', 60);
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO MOVIES VALUES (1003,'AKASH', 2008, 'KANNADA', 61);
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO MOVIES VALUES (1004,'WAR HORSE', 2011, 'ENGLISH', 63);
Query OK, 1 row affected (0.01 sec)
```

Code:

INSERT INTO MOVIE_CAST VALUES (301, 1002, 'HEROINE');

INSERT INTO MOVIE_CAST VALUES (301, 1001, 'HEROINE');

INSERT INTO MOVIE_CAST VALUES (303, 1003, 'HERO');

INSERT INTO MOVIE_CAST VALUES (303, 1002, 'GUEST');

INSERT INTO MOVIE_CAST VALUES (304, 1004, 'HERO'); **Output:**

```
mysql> INSERT INTO MOVIE_CAST VALUES (301, 1002, 'HEROINE');
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO MOVIE_CAST VALUES (301, 1001, 'HEROINE');
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO MOVIE_CAST VALUES (303, 1003, 'HERO');
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO MOVIE_CAST VALUES (303, 1002, 'GUEST');
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO MOVIE_CAST VALUES (304, 1004, 'HERO');
Query OK, 1 row affected (0.02 sec)
```

INSERT INTO RATING VALUES (1001, 4);

INSERT INTO RATING VALUES (1002, 2);

INSERT INTO RATING VALUES (1003, 5); INSERT INTO RATING

VALUES (1004, 4);

Output:

```
mysql> INSERT INTO RATING VALUES (1001, 4);
Query OK, 1 row affected (0.02 sec)

mysql> INSERT INTO RATING VALUES (1002, 2);
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO RATING VALUES (1003, 5);
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO RATING VALUES (1004, 4);
Query OK, 1 row affected (0.01 sec)
```

Write SQL queries to

1. **List the titles of all movies directed by 'Hitchcock'.**

Code:

SELECT MOV_TITLE

FROM MOVIES m

JOIN DIRECTOR d ON m.DIR_ID = d.DIR_ID

WHERE d.DIR_NAME = 'HITCHCOCK'; **Output:**

```
mysql> SELECT MOV_TITLE
    -> FROM MOVIES m
    -> JOIN DIRECTOR d ON m.DIR_ID = d.DIR_ID
    -> WHERE d.DIR_NAME = 'HITCHCOCK';
+-----------+
| MOV_TITLE |
+-----------+
| AKASH     |
+-----------+
1 row in set (0.00 sec)
```

2. Find the movie names where one or more actors acted in two or more movies.

Code:

SELECT DISTINCT m.MOV_TITLE

FROM MOVIES m

JOIN MOVIE_CAST mc ON m.MOV_ID = mc.MOV_ID

WHERE mc.ACT_ID IN (

  SELECT ACT_ID

```
    FROM MOVIE_CAST
    GROUP BY ACT_ID
    HAVING COUNT(DISTINCT MOV_ID) >= 2
);
```

Output:

```
mysql> SELECT DISTINCT m.MOV_TITLE
    -> FROM MOVIES m
    -> JOIN MOVIE_CAST mc ON m.MOV_ID = mc.MOV_ID
    -> WHERE mc.ACT_ID IN (
    ->     SELECT ACT_ID
    ->     FROM MOVIE_CAST
    ->     GROUP BY ACT_ID
    ->     HAVING COUNT(DISTINCT MOV_ID) >= 2
    -> );
+------------+
| MOV_TITLE  |
+------------+
| BAHUBALI-2 |
| BAHUBALI-1 |
| AKASH      |
+------------+
3 rows in set (0.03 sec)
```

3.List all actors who acted in a movie before 2000 and also in a movie after 2015 (use JOIN operation).

Code:

```
SELECT DISTINCT a.ACT_NAME
FROM ACTOR a
JOIN MOVIE_CAST mc1 ON a.ACT_ID = mc1.ACT_ID
JOIN MOVIES m1 ON mc1.MOV_ID = m1.MOV_ID
JOIN MOVIE_CAST mc2 ON a.ACT_ID = mc2.ACT_ID
JOIN MOVIES m2 ON mc2.MOV_ID = m2.MOV_ID WHERE m1.MOV_YEAR < 2000 AND
m2.MOV_YEAR > 2015;
```

Output:

```
mysql> SELECT DISTINCT a.ACT_NAME
    -> FROM ACTOR a
    -> JOIN MOVIE_CAST mc1 ON a.ACT_ID = mc1.ACT_ID
    -> JOIN MOVIES m1 ON mc1.MOV_ID = m1.MOV_ID
    -> JOIN MOVIE_CAST mc2 ON a.ACT_ID = mc2.ACT_ID
    -> JOIN MOVIES m2 ON mc2.MOV_ID = m2.MOV_ID
    -> WHERE m1.MOV_YEAR < 2000 AND m2.MOV_YEAR > 2015;
Empty set (0.00 sec)
```

4.Find the title of movies and number of stars for each movie that has at least one rating and find the highest number of stars that movie received. Sort the result by movie title.

Code:

```
SELECT m.MOV_TITLE, r.REV_STARS, (
   SELECT MAX(r1.REV_STARS)
   FROM RATING r1
   WHERE r1.MOV_ID = m.MOV_ID
) AS MAX_STARS
FROM MOVIES m
JOIN RATING r ON m.MOV_ID = r.MOV_ID
ORDER BY m.MOV_TITLE;
```

Output:

```
mysql> SELECT m.MOV_TITLE, r.REV_STARS, (
    ->     SELECT MAX(r1.REV_STARS)
    ->     FROM RATING r1
    ->     WHERE r1.MOV_ID = m.MOV_ID
    -> ) AS MAX_STARS
    -> FROM MOVIES m
    -> JOIN RATING r ON m.MOV_ID = r.MOV_ID
    -> ORDER BY m.MOV_TITLE;
+------------+-----------+-----------+
| MOV_TITLE  | REV_STARS | MAX_STARS |
+------------+-----------+-----------+
| AKASH      | 5         | 5         |
| BAHUBALI-1 | 2         | 2         |
| BAHUBALI-2 | 4         | 4         |
| WAR HORSE  | 5         | 5         |
+------------+-----------+-----------+
4 rows in set (0.00 sec)
```

5.Update rating of all movies directed by 'Steven Spielberg' to 5.

Code:

```
UPDATE RATING
SET REV_STARS = '5'
WHERE MOV_ID IN (
   SELECT m.MOV_ID
   FROM MOVIES m
   JOIN DIRECTOR d ON m.DIR_ID = d.DIR_ID
   WHERE d.DIR_NAME = 'STEVEN SPIELBERG'
);
```

Output:

```
mysql> UPDATE RATING
    -> SET REV_STARS = '5'
    -> WHERE MOV_ID IN (
    ->     SELECT m.MOV_ID
    ->     FROM MOVIES m
    ->     JOIN DIRECTOR d ON m.DIR_ID = d.DIR_ID
    ->     WHERE d.DIR_NAME = 'STEVEN SPIELBERG'
    -> );
Query OK, 1 row affected (0.03 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

3. Design ERD for the following schema and execute the following Queries on it:

Code:

```
CREATE TABLE students (
stno INT PRIMARY KEY,
 name VARCHAR(50),
addr VARCHAR(255),
city VARCHAR(50),
 state VARCHAR(2),
 zip VARCHAR(10)
);


CREATE TABLE INSTRUCTORS (
```

```
empno INT PRIMARY KEY,     name
VARCHAR(50),
 rank VARCHAR(20),
roomno VARCHAR(10),
 telno  VARCHAR(15)
);
 CREATE TABLE COURSES (
   cno INT PRIMARY KEY,
   cname VARCHAR(50),
   cr INT,     cap INT
);
CREATE TABLE GRADES (
   stno INT,
   empno INT,     cno INT,
   sem VARCHAR(10),
   year INT,     grade INT,
   PRIMARY KEY (stno),
   FOREIGN KEY (stno) REFERENCES students(stno),
   FOREIGN KEY (empno) REFERENCES INSTRUCTORS(empno),
   FOREIGN KEY (cno) REFERENCES COURSES(cno)
);

CREATE TABLE ADVISING (
   stno INT,     empno INT,
   PRIMARY KEY (stno, empno),
   FOREIGN KEY (stno) REFERENCES students(stno),
   FOREIGN KEY (empno) REFERENCES INSTRUCTORS(empno)
);
```

Output:

```
mysql> CREATE TABLE students (
    ->     stno INT PRIMARY KEY,
    ->     name VARCHAR(50),
    ->     addr VARCHAR(255),
    ->     city VARCHAR(50),
    ->     state VARCHAR(2),
    ->     zip VARCHAR(10)
    -> );
Query OK, 0 rows affected (0.04 sec)

mysql>
mysql> CREATE TABLE INSTRUCTORS (
    ->     empno INT PRIMARY KEY,
    ->     name VARCHAR(50),
    ->     rank VARCHAR(20),
    ->     roomno VARCHAR(10),
    ->     telno VARCHAR(15)
    -> );
Query OK, 0 rows affected (0.01 sec)

mysql>
mysql> CREATE TABLE COURSES (
    ->     cno INT PRIMARY KEY,
    ->     cname VARCHAR(50),
    ->     cr INT,
    ->     cap INT
    -> );
Query OK, 0 rows affected (0.01 sec)
```

```
mysql> CREATE TABLE GRADES (
    ->     stno INT,
    ->     empno INT,
    ->     cno INT,
    ->     sem VARCHAR(10),
    ->     year INT,
    ->     grade INT,
    ->     PRIMARY KEY (stno),
    ->     FOREIGN KEY (stno) REFERENCES students(stno),
    ->     FOREIGN KEY (empno) REFERENCES INSTRUCTORS(empno),
    ->     FOREIGN KEY (cno) REFERENCES COURSES(cno)
    -> );
Query OK, 0 rows affected (0.04 sec)

mysql>
mysql> CREATE TABLE ADVISING (
    ->     stno INT,
    ->     empno INT,
    ->     PRIMARY KEY (stno, empno),
    ->     FOREIGN KEY (stno) REFERENCES students(stno),
    ->     FOREIGN KEY (empno) REFERENCES INSTRUCTORS(empno)
    -> );
Query OK, 0 rows affected (0.04 sec)
```

Code:

INSERT INTO COURSES (cno, cname, cr, cap)

VALUES

(1, 'Math101', 3, 30),

```
(2, 'CS210', 4, 25),
 (3, 'Physics101', 3, 20);

INSERT INTO students (stno, name)
VALUES
(1, 'John Doe'),
 (2, 'Jane Smith'),
(3, 'Alice Johnson');

INSERT INTO instructors (empno, name)VALUES
 (101, 'Instructor A'),
(102, 'Instructor B'),
(103, 'Instructor C');
INSERT INTO GRADES (stno, empno, cno, sem, year, grade)
VALUES
   (1, 101, 1, 'Fall', 2021, 85),
   (2, 102, 2, 'Fall', 2021, 92),
   (3, 103, 3, 'Fall', 2021, 78);

INSERT INTO ADVISING (stno, empno)
VALUES
   (1, 101),
   (2, 102),
   (3, 103);
```

Output:

```
mysql> INSERT INTO COURSES (cno, cname, cr, cap)
    -> VALUES
    ->     (1, 'Math101', 3, 30),
    ->     (2, 'CS210', 4, 25),
    ->     (3, 'Physics101', 3, 20);
Query OK, 3 rows affected (0.04 sec)
Records: 3  Duplicates: 0  Warnings: 0
```

```
mysql> INSERT INTO students (stno, name)
    -> VALUES
    ->     (1, 'John Doe'),
    ->     (2, 'Jane Smith'),
    ->     (3, 'Alice Johnson');
Query OK, 3 rows affected (0.01 sec)
Records: 3  Duplicates: 0  Warnings: 0
```

```
mysql> INSERT INTO instructors (empno, name)
    -> VALUES
    ->     (101, 'Instructor A'),
    ->     (102, 'Instructor B'),
    ->     (103, 'Instructor C');
Query OK, 3 rows affected (0.03 sec)
Records: 3  Duplicates: 0  Warnings: 0

mysql> INSERT INTO GRADES (stno, empno, cno, sem, year, grade)
    -> VALUES
    ->     (1, 101, 1, 'Fall', 2021, 85),
    ->     (2, 102, 2, 'Fall', 2021, 92),
    ->     (3, 103, 3, 'Fall', 2021, 78);
Query OK, 3 rows affected (0.02 sec)
Records: 3  Duplicates: 0  Warnings: 0

mysql> INSERT INTO ADVISING (stno, empno)
    -> VALUES
    ->     (1, 101),
    ->     (2, 102),
    ->     (3, 103);
Query OK, 3 rows affected (0.02 sec)
Records: 3  Duplicates: 0  Warnings: 0
```

For odd rollnumbers(any 10)

1. **Find the names of students who took some four-credit courses.**

Code:

SELECT DISTINCT s.name

FROM students s

JOIN grades g ON s.stno = g.stno

JOIN courses c ON g.cno = c.cno

WHERE c.cr = 4;

Output:

```
mysql> SELECT DISTINCT s.name
    -> FROM students s
    -> JOIN grades g ON s.stno = g.stno
    -> JOIN courses c ON g.cno = c.cno
    -> WHERE c.cr = 4;
+------------+
| name       |
+------------+
| Jane Smith |
+------------+
1 row in set (0.00 sec)
```

2.Find the names of students who took every four-credit course.

Code:

```
SELECT s.name
FROM students s
WHERE NOT EXISTS (
  SELECT 1
  FROM courses c
  WHERE c.cr = 4 AND NOT EXISTS (
    SELECT 1
    FROM grades g
    WHERE g.stno = s.stno AND g.cno = c.cno
  )
);
```

Output:

```
mysql> SELECT s.name
    -> FROM students s
    -> WHERE NOT EXISTS (
    ->     SELECT 1
    ->     FROM courses c
    ->     WHERE c.cr = 4 AND NOT EXISTS (
    ->         SELECT 1
    ->         FROM grades g
    ->         WHERE g.stno = s.stno AND g.cno = c.cno
    ->     )
    -> );
+------------+
| name       |
+------------+
| Jane Smith |
+------------+
1 row in set (0.00 sec)
```

3.Find the names of students who took a course with an instructor who is also their advisor.

Code:

```
SELECT DISTINCT s.name
FROM students s
JOIN grades g ON s.stno = g.stno
JOIN instructors i ON g.empno = i.empno
JOIN advising a ON s.stno = a.stno
WHERE g.empno = a.empno;
```

Output:

```
mysql> SELECT DISTINCT s.name
    -> FROM students s
    -> JOIN grades g ON s.stno = g.stno
    -> JOIN instructors i ON g.empno = i.empno
    -> JOIN advising a ON s.stno = a.stno
    -> WHERE g.empno = a.empno;
+---------------+
| name          |
+---------------+
| John Doe      |
| Jane Smith    |
| Alice Johnson |
+---------------+
3 rows in set (0.00 sec)
```

4.Find the names of students who took cs210 and cs310.

Code:

```
SELECT s.name
FROM students s
WHERE EXISTS (
   SELECT 1
   FROM grades g
   JOIN courses c ON g.cno = c.cno
   WHERE s.stno = g.stno AND c.cname = 'cs210'
)
AND EXISTS (
   SELECT 1
   FROM grades g
```

```
    JOIN courses c ON g.cno = c.cno
    WHERE s.stno = g.stno AND c.cname = 'cs310'
);
```

Output:

```
mysql> SELECT s.name
    -> FROM students s
    -> WHERE EXISTS (
    ->     SELECT 1
    ->     FROM grades g
    ->     JOIN courses c ON g.cno = c.cno
    ->     WHERE s.stno = g.stno AND c.cname = 'cs210'
    -> )
    -> AND EXISTS (
    ->     SELECT 1
    ->     FROM grades g
    ->     JOIN courses c ON g.cno = c.cno
    ->     WHERE s.stno = g.stno AND c.cname = 'cs310'
    -> );
Empty set (0.00 sec)
```

5.Find the names of all students whose advisor is not a full professor.

Code:

```
SELECT DISTINCT s.name
FROM students s
JOIN advising a ON s.stno = a.stno
JOIN instructors i ON a.empno = i.empno
WHERE i.rank <> 'Full Professor';
```

Output:

```
mysql> SELECT DISTINCT s.name
    -> FROM students s
    -> JOIN advising a ON s.stno = a.stno
    -> JOIN instructors i ON a.empno = i.empno
    -> WHERE i.rank <> 'Full Professor';
Empty set (0.00 sec)
```

6..Find instructors who taught students who are advised by another instructor who shares the same room.

Code:

```
SELECT DISTINCT i1.name
FROM instructors i1
JOIN grades g ON i1.empno = g.empno
JOIN advising a ON g.stno = a.stno
```

JOIN instructors i2 ON a.empno = i2.empno

WHERE i1.roomno = i2.roomno AND i1.empno <> i2.empno;

Output:

```
mysql> SELECT DISTINCT i1.name
    -> FROM instructors i1
    -> JOIN grades g ON i1.empno = g.empno
    -> JOIN advising a ON g.stno = a.stno
    -> JOIN instructors i2 ON a.empno = i2.empno
    -> WHERE i1.roomno = i2.roomno AND i1.empno <> i2.empno;
Empty set (0.00 sec)
```

7.Find course numbers for courses that enroll exactly two students

Code:

SELECT g.cno

FROM grades g

GROUP BY g.cno

HAVING COUNT(DISTINCT g.stno) = 2;

Output:

```
mysql> SELECT g.cno
    -> FROM grades g
    -> GROUP BY g.cno
    -> HAVING COUNT(DISTINCT g.stno) = 2;
Empty set (0.00 sec)
```

8.Find the names of all students for whom no other student lives in the same city.

Code:

SELECT s1.name

FROM students s1

WHERE NOT EXISTS (

  SELECT 1

  FROM students s2

  WHERE s1.city = s2.city AND s1.stno <> s2.stno

);

Output:

```
mysql> SELECT s1.name
    -> FROM students s1
    -> WHERE NOT EXISTS (
    ->     SELECT 1
    ->     FROM students s2
    ->     WHERE s1.city = s2.city AND s1.stno <> s2.stno
    -> );
+---------------+
| name          |
+---------------+
| John Doe      |
| Jane Smith    |
| Alice Johnson |
+---------------+
3 rows in set (0.00 sec)
```

9.Find course numbers of courses taken by students who live in Boston and which are taught by an associate professor.

Code:

SELECT DISTINCT g.cno

FROM grades g

JOIN students s ON g.stno = s.stno

JOIN instructors i ON g.empno = i.empno

WHERE s.city = 'Boston' AND i.rank = 'Associate Professor';

Output:

```
mysql> SELECT DISTINCT g.cno
    -> FROM grades g
    -> JOIN students s ON g.stno = s.stno
    -> JOIN instructors i ON g.empno = i.empno
    -> WHERE s.city = 'Boston' AND i.rank = 'Associate Professor';
Empty set (0.00 sec)
```

10.Find the telephone numbers of instructors who teach a course taken by any student who lives in Boston.
Code:

SELECT DISTINCT i.telno

FROM instructors i

JOIN grades g ON i.empno = g.empno

JOIN students s ON g.stno = s.stno

WHERE s.city = 'Boston';

Output:

```
mysql> SELECT DISTINCT i.telno
    -> FROM instructors i
    -> JOIN grades g ON i.empno = g.empno
    -> JOIN students s ON g.stno = s.stno
    -> WHERE s.city = 'Boston';
Empty set (0.00 sec)
```

11.Find names of students who took every course taken by Richard Pierce.

Code:

```
SELECT s.name
FROM students s
WHERE NOT EXISTS (
   SELECT 1
   FROM grades g1
   JOIN students rp ON rp.name = 'Richard Pierce'
   JOIN grades g2 ON rp.stno = g2.stno
   WHERE g1.cno = g2.cno AND g1.stno <> s.stno
);
```

Output:

```
mysql> SELECT s.name
    -> FROM students s
    -> WHERE NOT EXISTS (
    ->     SELECT 1
    ->     FROM grades g1
    ->     JOIN students rp ON rp.name = 'Richard Pierce'
    ->     JOIN grades g2 ON rp.stno = g2.stno
    ->     WHERE g1.cno = g2.cno AND g1.stno <> s.stno
    -> );
+---------------+
| name          |
+---------------+
| John Doe      |
| Jane Smith    |
| Alice Johnson |
+---------------+
3 rows in set (0.00 sec)
```

12.Find the names of students who took only one course.

Code:

```
SELECT s.name
```

FROM students s

JOIN grades g ON s.stno = g.stno

GROUP BY s.stno, s.name

HAVING COUNT(DISTINCT g.cno) = 1;

Output:

```
mysql> SELECT s.name
    -> FROM students s
    -> JOIN grades g ON s.stno = g.stno
    -> GROUP BY s.stno, s.name
    -> HAVING COUNT(DISTINCT g.cno) = 1;
+---------------+
| name          |
+---------------+
| John Doe      |
| Jane Smith    |
| Alice Johnson |
+---------------+
3 rows in set (0.01 sec)
```

13.Find the names of instructors who teach no course.

Code:

SELECT i.name

FROM instructors i

LEFT JOIN grades g ON i.empno = g.empno

WHERE g.cno IS NULL;

Output:

```
mysql> SELECT i.name
    -> FROM instructors i
    -> LEFT JOIN grades g ON i.empno = g.empno
    -> WHERE g.cno IS NULL;
Empty set (0.00 sec)
```

14.Find the names of the instructors who taught only one course during the spring semester of 2001.

Code:

SELECT i.name

FROM instructors i

JOIN grades g ON i.empno = g.empno

WHERE g.sem = 'Spring' AND g.year = 2001

GROUP BY i.empno, i.name HAVING

COUNT(DISTINCT g.cno) = 1;

Output:

```
mysql> SELECT i.name
    -> FROM instructors i
    -> JOIN grades g ON i.empno = g.empno
    -> WHERE g.sem = 'Spring' AND g.year = 2001
    -> GROUP BY i.empno, i.name
    -> HAVING COUNT(DISTINCT g.cno) = 1;
Empty set (0.00 sec)
```

<u>For even rollnumbers(any 10)</u>

1.Find the names of students who took only four-credit courses.

Code:

SELECT s.name
FROM students s
JOIN grades g ON s.stno = g.stno
JOIN courses c ON g.cno = c.cno
GROUP BY s.stno, s.name
HAVING COUNT(DISTINCT CASE WHEN c.cr = 4 THEN g.cno END) =
COUNT(DISTINCT g.cno)
AND COUNT(DISTINCT CASE WHEN c.cr <> 4 THEN g.cno END) =
0;

Output:

```
mysql> SELECT s.name
    -> FROM students s
    -> JOIN grades g ON s.stno = g.stno
    -> JOIN courses c ON g.cno = c.cno
    -> GROUP BY s.stno, s.name
    -> HAVING COUNT(DISTINCT CASE WHEN c.cr = 4 THEN g.cno END) = COUNT(DISTINCT g.cno)
    ->        AND COUNT(DISTINCT CASE WHEN c.cr <> 4 THEN g.cno END) = 0;
+------------+
| name       |
+------------+
| Jane Smith |
+------------+
1 row in set (0.00 sec)
```

2.Find the names of students who took no four-credit courses.

Code:

SELECT s.name

FROM students s

WHERE NOT EXISTS (

SELECT 1

FROM grades g

JOIN courses c ON g.cno = c.cno

WHERE g.stno = s.stno AND c.cr = 4

);

Output:

```
mysql> SELECT s.name
    -> FROM students s
    -> WHERE NOT EXISTS (
    ->     SELECT 1
    ->     FROM grades g
    ->     JOIN courses c ON g.cno = c.cno
    ->     WHERE g.stno = s.stno AND c.cr = 4
    -> );
+---------------+
| name          |
+---------------+
| John Doe      |
| Alice Johnson |
+---------------+
2 rows in set (0.00 sec)
```

3.Find the names of students who took cs210 or cs310.

Code:

SELECT DISTINCT s.name

FROM students s

JOIN grades g ON s.stno = g.stno

JOIN courses c ON g.cno = c.cno WHERE c.cname IN

('cs210', 'cs310');

Output:

```
mysql> SELECT DISTINCT s.name
    -> FROM students s
    -> JOIN grades g ON s.stno = g.stno
    -> JOIN courses c ON g.cno = c.cno
    -> WHERE c.cname IN ('cs210', 'cs310');
+------------+
| name       |
+------------+
| Jane Smith |
+------------+
1 row in set (0.00 sec)
```

4.Find names of all students who have a cs210 grade higher than the highest grade given in cs310 and did not take any course with Prof. Evans.

Code:

SELECT DISTINCT s.name

```
FROM students s
JOIN grades g1 ON s.stno = g1.stno
JOIN courses c1 ON g1.cno = c1.cno
WHERE c1.cname = 'cs210' AND g1.grade > (
   SELECT MAX(g2.grade)
   FROM grades g2
   JOIN courses c2 ON g2.cno = c2.cno
   WHERE c2.cname = 'cs310'
)
AND NOT EXISTS (
   SELECT 1
   FROM grades g3
   JOIN instructors i ON g3.empno = i.empno
   WHERE g3.stno = s.stno AND i.name = 'Prof. Evans'
);
```

Output:

```
mysql> SELECT DISTINCT s.name
    -> FROM students s
    -> JOIN grades g1 ON s.stno = g1.stno
    -> JOIN courses c1 ON g1.cno = c1.cno
    -> WHERE c1.cname = 'cs210' AND g1.grade > (
    ->     SELECT MAX(g2.grade)
    ->     FROM grades g2
    ->     JOIN courses c2 ON g2.cno = c2.cno
    ->     WHERE c2.cname = 'cs310'
    -> )
    -> AND NOT EXISTS (
    ->     SELECT 1
    ->     FROM grades g3
    ->     JOIN instructors i ON g3.empno = i.empno
    ->     WHERE g3.stno = s.stno AND i.name = 'Prof. Evans'
    -> );
Empty set (0.00 sec)
```

5. Find course numbers for courses that enrol at least two students; solve the same query for courses that enroll at least three students.

Code:

```
-- For courses with at least two students
SELECT g.cno
FROM grades g
GROUP BY g.cno
HAVING COUNT(DISTINCT g.stno) >= 2;


-- For courses with at least three students
SELECT g.cno
FROM grades g
GROUP BY g.cno
HAVING COUNT(DISTINCT g.stno) >= 3;
```

Output:

```
mysql> -- For courses with at least two students
mysql> SELECT g.cno
    -> FROM grades g
    -> GROUP BY g.cno
    -> HAVING COUNT(DISTINCT g.stno) >= 2;
Empty set (0.00 sec)

mysql>
mysql> -- For courses with at least three students
mysql> SELECT g.cno
    -> FROM grades g
    -> GROUP BY g.cno
    -> HAVING COUNT(DISTINCT g.stno) >= 3;
Empty set (0.00 sec)
```

6. Find the names of students who obtained the highest grade in cs210.

Code:

```
SELECT s.name
FROM students s
JOIN grades g ON s.stno = g.stno
JOIN courses c ON g.cno = c.cno
WHERE c.cname = 'cs210' AND g.grade = (
   SELECT MAX(grade)
   FROM grades g1
   JOIN courses c1 ON g1.cno = c1.cno
```

```
    WHERE c1.cname = 'cs210'
);
```

Output:

```
mysql> SELECT s.name
    -> FROM students s
    -> JOIN grades g ON s.stno = g.stno
    -> JOIN courses c ON g.cno = c.cno
    -> WHERE c.cname = 'cs210' AND g.grade = (
    ->     SELECT MAX(grade)
    ->     FROM grades g1
    ->     JOIN courses c1 ON g1.cno = c1.cno
    ->     WHERE c1.cname = 'cs210'
    -> );
+------------+
| name       |
+------------+
| Jane Smith |
+------------+
1 row in set (0.00 sec)
```

7.Find the names of instructors who teach courses attended by students who took a course with an instructor who is an assistant professor.

Code:

```
SELECT DISTINCT i1.name
FROM instructors i1
JOIN grades g ON i1.empno = g.empno
JOIN students s ON g.stno = s.stno
JOIN grades g2 ON s.stno = g2.stno
JOIN instructors i2 ON g2.empno = i2.empno
WHERE i2.rank = 'Assistant Professor';
```

Output:

```
mysql> SELECT DISTINCT i1.name
    -> FROM instructors i1
    -> JOIN grades g ON i1.empno = g.empno
    -> JOIN students s ON g.stno = s.stno
    -> JOIN grades g2 ON s.stno = g2.stno
    -> JOIN instructors i2 ON g2.empno = i2.empno
    -> WHERE i2.rank = 'Assistant Professor';
Empty set (0.00 sec)
```

8.Find the lowest grade of a student who took a course during the spring of 2003.

Code:

```
SELECT MIN(g.grade)
FROM grades g
WHERE g.sem = 'Spring' AND g.year = 2003;
```

Output:

```
mysql> SELECT MIN(g.grade)
    -> FROM grades g
    -> WHERE g.sem = 'Spring' AND g.year = 2003;
+--------------+
| MIN(g.grade) |
+--------------+
|         NULL |
+--------------+
1 row in set (0.00 sec)
```

9.Find the names for students such that if prof. Evans teaches a course, then the student takes that course (although not necessarily with prof. Evans).

Code:

```
SELECT s.name
FROM students s
WHERE NOT EXISTS (
  SELECT 1
  FROM courses c
  WHERE EXISTS (
    SELECT 1
    FROM grades g
    WHERE g.stno = s.stno AND g.cno = c.cno
  ) AND EXISTS (
    SELECT 1
    FROM grades g
    JOIN instructors i ON g.empno = i.empno
    WHERE g.cno = c.cno AND i.name = 'Prof. Evans'
  )
);
```

Output:

```
mysql> SELECT s.name
    -> FROM students s
    -> WHERE NOT EXISTS (
    ->     SELECT 1
    ->     FROM courses c
    ->     WHERE EXISTS (
    ->         SELECT 1
    ->         FROM grades g
    ->         WHERE g.stno = s.stno AND g.cno = c.cno
    ->     ) AND EXISTS (
    ->         SELECT 1
    ->         FROM grades g
    ->         JOIN instructors i ON g.empno = i.empno
    ->         WHERE g.cno = c.cno AND i.name = 'Prof. Evans'
    ->     )
    -> );
+---------------+
| name          |
+---------------+
| John Doe      |
| Jane Smith    |
| Alice Johnson |
+---------------+
3 rows in set (0.00 sec)
```

10.**Find the names of students whose advisor did not teach them any course.**

Code:

SELECT s.name

FROM students s

JOIN advising a ON s.stno = a.stno

LEFT JOIN grades g ON s.stno = g.stno AND g.empno = a.empno

WHERE g.empno IS NULL;

Output:

```
mysql> SELECT s.name
    -> FROM students s
    -> JOIN advising a ON s.stno = a.stno
    -> LEFT JOIN grades g ON s.stno = g.stno AND g.empno = a.empno
    -> WHERE g.empno IS NULL;
Empty set (0.00 sec)
```

11.Find the names of students who have failed all their courses (failing is defined as a grade less than 60).

Code:

```
SELECT s.name
FROM students s
JOIN grades g ON s.stno = g.stno
GROUP BY s.stno, s.name
HAVING MIN(g.grade) < 60 AND MAX(g.grade) < 60;
```

Output:

```
mysql> SELECT s.name
    -> FROM students s
    -> JOIN grades g ON s.stno = g.stno
    -> GROUP BY s.stno, s.name
    -> HAVING MIN(g.grade) < 60 AND MAX(g.grade) < 60;
Empty set (0.00 sec)
```

12.Find the highest grade of a student who never took cs110.

Code:

```
SELECT MAX(g.grade)
FROM grades g
WHERE g.stno NOT IN (
   SELECT g2.stno
   FROM grades g2
   JOIN courses c ON g2.cno = c.cno
   WHERE c.cname = 'cs110'
)
GROUP BY g.stno;
```

Output:

```
mysql> SELECT MAX(g.grade)
    -> FROM grades g
    -> WHERE g.stno NOT IN (
    ->     SELECT g2.stno
    ->     FROM grades g2
    ->     JOIN courses c ON g2.cno = c.cno
    ->     WHERE c.cname = 'cs110'
    -> )
    -> GROUP BY g.stno;
+--------------+
| MAX(g.grade) |
+--------------+
|           85 |
|           92 |
|           78 |
+--------------+
3 rows in set (0.00 sec)
```

13. Find the names of students who do not have an advisor.

Code:

SELECT s.name

FROM students s

LEFT JOIN advising a ON s.stno = a.stno

WHERE a.empno IS NULL;

Output:

```
mysql> SELECT s.name
    -> FROM students s
    -> LEFT JOIN advising a ON s.stno = a.stno
    -> WHERE a.empno IS NULL;
Empty set (0.00 sec)
```

14. Find names of courses taken by students who do not live in Massachusetts (MA).

Code:

SELECT DISTINCT c.cname

FROM students s

JOIN grades g ON s.stno = g.stno

JOIN courses c ON g.cno = c.cno

WHERE s.state <> 'MA';

Output:

```
mysql> SELECT DISTINCT c.cname
    -> FROM students s
    -> JOIN grades g ON s.stno = g.stno
    -> JOIN courses c ON g.cno = c.cno
    -> WHERE s.state <> 'MA';
Empty set (0.00 sec)
```