Advance Database Management System

Practical 2: Performing Subquery-Join Operations on Relational Schema

```
Microsoft Windows [Version 10.0.22631.4602]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Rasika>cd C:\Users\Rasika\Downloads
```

```
mysql> CREATE DATABASE practical1;
Query OK, 1 row affected (0.02 sec)

mysql> USE practical1;
Database changed
```

```
C:\Users\Rasika\Downloads>mysql -u root -p practical1 < prac1.sql
Enter password: ****
```

```
C:\Users\Rasika\Downloads>mysql -u root -p
Enter password: ****
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 14
Server version: 8.0.34 MySQL Community Server - GPL

Copyright (c) 2000, 2023, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> USE practical1;
Database changed
```

```
mysql> show TABLES;
+----------------------+
| Tables_in_practical1 |
+----------------------+
| customer             |
| orders               |
| salesman             |
+----------------------+
3 rows in set (0.00 sec)
```

TABLES:

```
mysql> select * from customer;
+-------------+---------------+------------+-------+-------------+
| customer_id | customer_name | city       | grade | salesman_id |
+-------------+---------------+------------+-------+-------------+
|        3001 | Brad Guzan    | London     |  NULL |        NULL |
|        3002 | Nick Rimando  | New York   |   100 |        5001 |
|        3003 | Jozy Altidor  | Moncow     |   200 |        5007 |
|        3004 | Fabian Johns  | Paris      |   300 |        5006 |
|        3005 | Graham Zusi   | California |   200 |        5002 |
|        3007 | Brad Davis    | New York   |   200 |        5001 |
|        3008 | Julian Green  | London     |   300 |        5002 |
|        3009 | Geoff Camero  | Berlin     |   100 |        NULL |
+-------------+---------------+------------+-------+-------------+
8 rows in set (0.00 sec)
```

```
mysql> select * from orders;
+----------+--------------+------------+-------------+-------------+
| order_no | purchase_amt | order_date | customer_id | salesman_id |
+----------+--------------+------------+-------------+-------------+
|    70001 |       150.50 | 2016-10-05 |        3005 |        5002 |
|    70002 |        65.25 | 2016-10-05 |        3002 |        5001 |
|    70003 |      2480.40 | 2016-10-10 |        3009 |        NULL |
|    70004 |       110.50 | 2016-08-17 |        3009 |        NULL |
|    70005 |      2400.60 | 2016-07-27 |        3007 |        5001 |
|    70007 |       948.50 | 2016-09-10 |        3005 |        5002 |
|    70008 |      5760.00 | 2016-09-10 |        3002 |        5001 |
|    70009 |       270.65 | 2016-09-10 |        3001 |        NULL |
|    70010 |      1983.43 | 2016-10-10 |        3004 |        5006 |
|    70011 |        75.29 | 2016-08-17 |        3003 |        5007 |
|    70012 |       250.45 | 2016-06-27 |        3008 |        5002 |
+----------+--------------+------------+-------------+-------------+
11 rows in set (0.00 sec)
```

```
mysql> select * from salesman;
+-------------+------------+----------+------------+
| salesman_id | name       | city     | commission |
+-------------+------------+----------+------------+
|        5001 | James Hoog | New York |       0.15 |
|        5002 | Nail Knite | Paris    |       0.13 |
|        5003 | Lauson Hen |          |       0.12 |
|        5005 | Pit Alex   | London   |       0.11 |
|        5006 | Mc Lyon    | Paris    |       0.14 |
|        5007 | Paul Adam  | Rome     |       0.13 |
+-------------+------------+----------+------------+
6 rows in set (0.00 sec)
```

1. **USING (practical 1):**

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

```
mysql> SELECT grade, COUNT(DISTINCT customer_id)
    -> FROM customer
    -> GROUP BY grade
    -> HAVING grade > (
    ->     SELECT AVG(grade)
    ->     FROM customer
    ->     WHERE city = 'Bangalore'
    -> );
Empty set (0.00 sec)
```

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

```
mysql> SELECT salesman_id, name
    -> FROM salesman A
    -> WHERE 1 < (
    ->     SELECT COUNT(*)
    ->     FROM customer
    ->     WHERE salesman_id = A.salesman_id
    -> );
+-------------+------------+
| salesman_id | name       |
+-------------+------------+
|        5001 | James Hoog |
|        5002 | Nail Knite |
+-------------+------------+
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).

```
mysql> SELECT
    ->     s.salesman_id,
    ->     s.name,
    ->     c.customer_name,
    ->     s.commission
    -> FROM
    ->     salesman s
    -> JOIN
    ->     customer c ON s.city = c.city
    ->
    -> UNION
    ->
    -> SELECT
    ->     s.salesman_id,
    ->     s.name,
    ->     'NO MATCH' AS customer_name,
    ->     s.commission
    -> FROM
    ->     salesman s
    -> WHERE
    ->     s.city NOT IN (SELECT city FROM customer)
    ->
    -> ORDER BY
    ->     name DESC;
```

```
+-------------+------------+---------------+------------+
| salesman_id | name       | customer_name | commission |
+-------------+------------+---------------+------------+
|        5005 | Pit Alex   | Brad Guzan    |       0.11 |
|        5005 | Pit Alex   | Julian Green  |       0.11 |
|        5007 | Paul Adam  | NO MATCH      |       0.13 |
|        5002 | Nail Knite | Fabian Johns  |       0.13 |
|        5006 | Mc Lyon    | Fabian Johns  |       0.14 |
|        5003 | Lauson Hen | NO MATCH      |       0.12 |
|        5001 | James Hoog | Nick Rimando  |       0.15 |
|        5001 | James Hoog | Brad Davis    |       0.15 |
+-------------+------------+---------------+------------+
8 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.

```
mysql> CREATE VIEW ELITESALESMAN AS
    -> SELECT
    ->     o.order_date,
    ->     s.salesman_id,
    ->     s.name
    -> FROM
    ->     salesman s
    -> JOIN
    ->     orders o ON s.salesman_id = o.salesman_id
    -> WHERE
    ->     o.purchase_amt = (
    ->         SELECT MAX(purchase_amt)
    ->         FROM orders o2
    ->         WHERE o2.order_date = o.order_date
    ->     );
Query OK, 0 rows affected (0.03 sec)

mysql> SELECT * FROM ELITESALESMAN;
+------------+-------------+------------+
| order_date | salesman_id | name       |
+------------+-------------+------------+
| 2016-07-27 |        5001 | James Hoog |
| 2016-09-10 |        5001 | James Hoog |
| 2016-10-05 |        5002 | Nail Knite |
| 2016-06-27 |        5002 | Nail Knite |
+------------+-------------+------------+
4 rows in set (0.01 sec)
```

5) Demonstrate the DELETE operation by removing salesman with id 1000. All his orders must also be deleted.

```
mysql> DELETE FROM orders WHERE salesman_id = 1000;
Query OK, 0 rows affected (0.01 sec)

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

mysql> SELECT * FROM orders WHERE salesman_id = 1000;
Empty set (0.00 sec)

mysql> SELECT * FROM salesman WHERE salesman_id = 1000;
Empty set (0.00 sec)
```

2. Design ERD for the following schema and execute the following Queries on it:
   - Consider the schema for Movie Database:

i) ACTOR (Act_id, Act_Name, Act_Gender)
ii) DIRECTOR (Dir_id, Dir_Name, Dir_Phone)
iii) MOVIES (Mov_id, Mov_Title, Mov_Year, Mov_Lang, Dir_id)
iv) MOVIE_CAST (Act_id, Mov_id, Role)
v) RATING (Mov_id, Rev_Stars)

```
mysql> CREATE DATABASE MOVIES;
Query OK, 1 row affected (0.01 sec)
```

```
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, 1 warning (0.03 sec)
```

```
mysql> INSERT INTO ACTOR VALUES (301, 'ANUSHKA', 'F');
Query OK, 1 row affected (0.01 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 (0.01 sec)

mysql> INSERT INTO ACTOR VALUES (304, 'JERMY', 'M');
Query OK, 1 row affected (0.01 sec)
```

```
mysql> select * from ACTOR;
+--------+----------+------------+
| ACT_ID | ACT_NAME | ACT_GENDER |
+--------+----------+------------+
|    301 | ANUSHKA  | F          |
|    302 | PRABHAS  | M          |
|    303 | PUNITH   | M          |
|    304 | JERMY    | M          |
+--------+----------+------------+
4 rows in set (0.00 sec)
```

```
mysql> CREATE TABLE DIRECTOR (
    ->     DIR_ID INT(3),
    ->     DIR_NAME VARCHAR(20),
    ->     DIR_PHONE BIGINT(10),
    ->     PRIMARY KEY (DIR_ID)
    -> );
Query OK, 0 rows affected, 2 warnings (0.03 sec)
```

```
mysql> select * from DIRECTOR;
+--------+-----------------+------------+
| DIR_ID | DIR_NAME        | DIR_PHONE  |
+--------+-----------------+------------+
|     60 | RAJAMOULI       | 8751611001 |
|     61 | HITCHCOCK       | 7766138911 |
|     62 | FARAN           | 9986776531 |
|     63 | STEVEN SPIELBERG | 8989776530 |
+--------+-----------------+------------+
4 rows in set (0.00 sec)
```

```
mysql> CREATE TABLE MOVIES (
    ->     MOV_ID INT(4),
    ->     MOV_TITLE VARCHAR(25),
    ->     MOV_YEAR YEAR,
    ->     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, 2 warnings (0.06 sec)
```

```
mysql> select * from MOVIES;
+--------+------------+----------+----------+--------+
| MOV_ID | MOV_TITLE  | MOV_YEAR | MOV_LANG | DIR_ID |
+--------+------------+----------+----------+--------+
|   1001 | BAHUBALI-2 |     2017 | TELUGU   |     60 |
|   1002 | BAHUBALI-1 |     2015 | TELUGU   |     60 |
|   1003 | AKASH      |     2008 | KANNADA  |     61 |
|   1004 | WAR HORSE  |     2011 | ENGLISH  |     63 |
+--------+------------+----------+----------+--------+
4 rows in set (0.00 sec)
```

```
mysql> CREATE TABLE MOVIE_CAST (
    ->     ACT_ID INT(3),
    ->     MOV_ID INT(4),
    ->     ROLE 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, 2 warnings (0.06 sec)
```

```
mysql> select * from MOVIE_CAST;
+--------+--------+---------+
| ACT_ID | MOV_ID | ROLE    |
+--------+--------+---------+
|    301 |   1001 | HEROINE |
|    301 |   1002 | HEROINE |
|    303 |   1002 | GUEST   |
|    303 |   1003 | HERO    |
|    304 |   1004 | HERO    |
+--------+--------+---------+
5 rows in set (0.00 sec)
```

```
mysql> CREATE TABLE RATING (
    ->     MOV_ID INT(4),
    ->     REV_STARS TINYINT(1),
    ->     PRIMARY KEY (MOV_ID),
    ->     FOREIGN KEY (MOV_ID) REFERENCES MOVIES(MOV_ID)
    -> );
Query OK, 0 rows affected, 2 warnings (0.03 sec)
```

```
mysql> select * from RATING;
+--------+-----------+
| MOV_ID | REV_STARS |
+--------+-----------+
|   1001 |         4 |
|   1002 |         2 |
|   1003 |         5 |
|   1004 |         4 |
+--------+-----------+
4 rows in set (0.00 sec)
```

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

```
mysql> SELECT MOV_TITLE
    -> FROM MOVIES
    -> WHERE DIR_ID = (SELECT DIR_ID FROM DIRECTOR WHERE 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:**

```
mysql> SELECT MOV_TITLE
    -> FROM MOVIES
    -> WHERE MOV_ID IN (
    ->     SELECT MOV_ID
    ->     FROM MOVIE_CAST
    ->     WHERE 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 (0.01 sec)
```

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

```
mysql> SELECT DISTINCT A.ACT_NAME
    -> FROM ACTOR A
    -> JOIN MOVIE_CAST MC ON A.ACT_ID = MC.ACT_ID
    -> JOIN MOVIES M ON MC.MOV_ID = M.MOV_ID
    -> WHERE M.MOV_YEAR < 2000
    -> AND A.ACT_ID IN (
    ->     SELECT MC2.ACT_ID
    ->     FROM MOVIE_CAST MC2
    ->     JOIN MOVIES M2 ON MC2.MOV_ID = M2.MOV_ID
    ->     WHERE 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:**

```
mysql> SELECT MOV_TITLE, MAX(REV_STARS) AS HIGHEST_STARS
    -> FROM MOVIES M
    -> JOIN RATING R ON M.MOV_ID = R.MOV_ID
    -> GROUP BY MOV_TITLE
    -> ORDER BY MOV_TITLE;
+------------+---------------+
| MOV_TITLE  | HIGHEST_STARS |
+------------+---------------+
| AKASH      |             5 |
| BAHUBALI-1 |             2 |
| BAHUBALI-2 |             4 |
| WAR HORSE  |             4 |
+------------+---------------+
4 rows in set (0.01 sec)
```

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

```
mysql> UPDATE RATING
    -> SET REV_STARS = 5
    -> WHERE MOV_ID IN (
    ->     SELECT MOV_ID
    ->     FROM MOVIES
    ->     WHERE DIR_ID = (SELECT DIR_ID FROM DIRECTOR WHERE DIR_NAME = 'Steven Spielberg')
    -> );
Query OK, 1 row affected (0.01 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> select * from RATING;
+--------+-----------+
| MOV_ID | REV_STARS |
+--------+-----------+
|   1001 |         4 |
|   1002 |         2 |
|   1003 |         5 |
|   1004 |         5 |
+--------+-----------+
4 rows in set (0.00 sec)
```

3. **<u>Design ERD for the following schema and execute the following Queries on it:</u>**

```
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> INSERT INTO students (stno, name, addr, city, state, zip) VALUES
    -> (1011, 'Edwards P. David', '10 Red Rd.', 'Newton', 'MA', '02159'),
    -> (2415, 'Grogan A. Mary', '8 Walnut St.', 'Malden', 'MA', '02148'),
    -> (2661, 'Mixon Leatha', '100 School St.', 'Brookline', 'MA', '02146'),
    -> (2890, 'McLane Sandy', '30 Cass Rd.', 'Boston', 'MA', '02122'),
    -> (3442, 'Novak Roland', '42 Beacon St.', 'Nashua', 'NH', '03060'),
    -> (3566, 'Pierce Richard', '70 Park St.', 'Brookline', 'MA', '02146'),
    -> (4022, 'Prior Lorraine', '8 Beacon St.', 'Boston', 'MA', '02125'),
    -> (5544, 'Rawlings Jerry', '15 Pleasant Dr.', 'Boston', 'MA', '02115'),
    -> (5571, 'Lewis Jerry', '1 Main Rd.', 'Providence', 'RI', '02904');
Query OK, 9 rows affected (0.01 sec)
Records: 9  Duplicates: 0  Warnings: 0
```

```
mysql> select * from students;
+------+-----------------+----------------+------------+-------+-------+
| stno | name            | addr           | city       | state | zip   |
+------+-----------------+----------------+------------+-------+-------+
| 1011 | Edwards P. David | 10 Red Rd.     | Newton     | MA    | 02159 |
| 2415 | Grogan A. Mary   | 8 Walnut St.   | Malden     | MA    | 02148 |
| 2661 | Mixon Leatha     | 100 School St. | Brookline  | MA    | 02146 |
| 2890 | McLane Sandy     | 30 Cass Rd.    | Boston     | MA    | 02122 |
| 3442 | Novak Roland     | 42 Beacon St.  | Nashua     | NH    | 03060 |
| 3566 | Pierce Richard   | 70 Park St.    | Brookline  | MA    | 02146 |
| 4022 | Prior Lorraine   | 8 Beacon St.   | Boston     | MA    | 02125 |
| 5544 | Rawlings Jerry   | 15 Pleasant Dr. | Boston     | MA    | 02115 |
| 5571 | Lewis Jerry      | 1 Main Rd.     | Providence | RI    | 02904 |
+------+-----------------+----------------+------------+-------+-------+
9 rows in set (0.00 sec)
```

```
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 (0.02 sec)

mysql>  INSERT INTO INSTRUCTORS (empno, name, `rank`, roomno, telno) VALUES
    -> (019, 'Evans Robert', 'Professor', '82', '7122'),
    -> (023, 'Exxon George', 'Professor', '90', '9101'),
    -> (056, 'Sawyer Kathy', 'Assoc. Prof.', '91', '5110'),
    -> (126, 'Davis William', 'Assoc. Prof.', '72', '5411'),
    -> (234, 'Will Samuel', 'Assist. Prof.', '90', '7024');
Query OK, 5 rows affected (0.01 sec)
Records: 5  Duplicates: 0  Warnings: 0
```

```
mysql> select * from instructors;
+-------+--------------+---------------+--------+-------+
| empno | name         | rank          | roomno | telno |
+-------+--------------+---------------+--------+-------+
|    19 | Evans Robert | Professor     | 82     | 7122  |
|    23 | Exxon George | Professor     | 90     | 9101  |
|    56 | Sawyer Kathy | Assoc. Prof.  | 91     | 5110  |
|   126 | Davis William | Assoc. Prof. | 72     | 5411  |
|   234 | Will Samuel  | Assist. Prof. | 90     | 7024  |
+-------+--------------+---------------+--------+-------+
5 rows in set (0.00 sec)
```

```
mysql> CREATE TABLE COURSES (
    ->     cno VARCHAR(10) PRIMARY KEY,
    ->     cname VARCHAR(50),
    ->     cr INT,
    ->     cap INT
    -> );
Query OK, 0 rows affected (0.02 sec)

mysql> INSERT INTO COURSES (cno, cname, cr, cap) VALUES
    -> ('cs110', 'Introduction to Computing', 4, 120),
    -> ('cs210', 'Computer Programming', 4, 100),
    -> ('cs240', 'Computer Architecture', 3, 100),
    -> ('cs310', 'Data Structures', 3, 60),
    -> ('cs350', 'Higher Level Languages', 3, 50),
    -> ('cs410', 'Software Engineering', 3, 40),
    -> ('cs460', 'Graphics', 3, 30);
Query OK, 7 rows affected (0.01 sec)
Records: 7  Duplicates: 0  Warnings: 0
```

```
mysql> select * from courses;
+-------+--------------------------+------+------+
| cno   | cname                    | cr   | cap  |
+-------+--------------------------+------+------+
| cs110 | Introduction to Computing |    4 |  120 |
| cs210 | Computer Programming      |    4 |  100 |
| cs240 | Computer Architecture     |    3 |  100 |
| cs310 | Data Structures           |    3 |   60 |
| cs350 | Higher Level Languages    |    3 |   50 |
| cs410 | Software Engineering      |    3 |   40 |
| cs460 | Graphics                  |    3 |   30 |
+-------+--------------------------+------+------+
7 rows in set (0.00 sec)
```

```
mysql> CREATE TABLE GRADES (
    ->     stno INT,
    ->     empno INT,
    ->     cno VARCHAR(10),
    ->     sem VARCHAR(10),
    ->     year INT,
    ->     grade INT,
    ->     PRIMARY KEY (stno, cno, sem, year),
    ->     FOREIGN KEY (stno) REFERENCES STUDENTS(stno),
    ->     FOREIGN KEY (empno) REFERENCES INSTRUCTORS(empno),
    ->     FOREIGN KEY (cno) REFERENCES COURSES(cno)
    -> );
Query OK, 0 rows affected (0.05 sec)
```

```
mysql> INSERT INTO GRADES (stno, empno, cno, sem, year, grade) VALUES
    -> (1011, 019, 'cs110', 'Fall', 2001, 40),
    -> (2661, 019, 'cs110', 'Fall', 2001, 80),
    -> (3566, 019, 'cs110', 'Fall', 2001, 95),
    -> (5544, 019, 'cs110', 'Fall', 2001, 100),
    -> (1011, 023, 'cs110', 'Spring', 2002, 75),
    -> (4022, 023, 'cs110', 'Spring', 2002, 60),
    -> (3566, 019, 'cs240', 'Spring', 2002, 100),
    -> (5571, 019, 'cs240', 'Spring', 2002, 50),
    -> (2415, 019, 'cs240', 'Spring', 2002, 100),
    -> (3442, 234, 'cs410', 'Spring', 2002, 60),
    -> (5571, 234, 'cs410', 'Spring', 2002, 80),
    -> (1011, 019, 'cs210', 'Fall', 2002, 90),
    -> (2661, 019, 'cs210', 'Fall', 2002, 70),
    -> (3566, 019, 'cs210', 'Fall', 2002, 90),
    -> (5571, 019, 'cs210', 'Spring', 2003, 85),
    -> (4022, 019, 'cs210', 'Spring', 2003, 70),
    -> (5544, 056, 'cs240', 'Spring', 2003, 70),
    -> (1011, 056, 'cs240', 'Spring', 2003, 90),
    -> (4022, 056, 'cs240', 'Spring', 2003, 80),
    -> (2661, 234, 'cs310', 'Spring', 2003, 100),
    -> (4022, 234, 'cs310', 'Spring', 2003, 75);
Query OK, 21 rows affected (0.01 sec)
Records: 21  Duplicates: 0  Warnings: 0
```

```
mysql> select * from grades;
+------+-------+-------+--------+------+-------+
| stno | empno | cno   | sem    | year | grade |
+------+-------+-------+--------+------+-------+
| 1011 |    19 | cs110 | Fall   | 2001 |    40 |
| 1011 |    23 | cs110 | Spring | 2002 |    75 |
| 1011 |    19 | cs210 | Fall   | 2002 |    90 |
| 1011 |    56 | cs240 | Spring | 2003 |    90 |
| 2415 |    19 | cs240 | Spring | 2002 |   100 |
| 2661 |    19 | cs110 | Fall   | 2001 |    80 |
| 2661 |    19 | cs210 | Fall   | 2002 |    70 |
| 2661 |   234 | cs310 | Spring | 2003 |   100 |
| 3442 |   234 | cs410 | Spring | 2002 |    60 |
| 3566 |    19 | cs110 | Fall   | 2001 |    95 |
| 3566 |    19 | cs210 | Fall   | 2002 |    90 |
| 3566 |    19 | cs240 | Spring | 2002 |   100 |
| 4022 |    23 | cs110 | Spring | 2002 |    60 |
| 4022 |    19 | cs210 | Spring | 2003 |    70 |
| 4022 |    56 | cs240 | Spring | 2003 |    80 |
| 4022 |   234 | cs310 | Spring | 2003 |    75 |
| 5544 |    19 | cs110 | Fall   | 2001 |   100 |
| 5544 |    56 | cs240 | Spring | 2003 |    70 |
| 5571 |    19 | cs210 | Spring | 2003 |    85 |
| 5571 |    19 | cs240 | Spring | 2002 |    50 |
| 5571 |   234 | cs410 | Spring | 2002 |    80 |
+------+-------+-------+--------+------+-------+
21 rows in set (0.00 sec)
```

```
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)

mysql> INSERT INTO ADVISING (stno, empno) VALUES
    -> (1011, 019),
    -> (2415, 019),
    -> (2661, 023),
    -> (2890, 023),
    -> (3442, 056),
    -> (3566, 126),
    -> (4022, 234),
    -> (5544, 023),
    -> (5571, 234);
Query OK, 9 rows affected (0.01 sec)
Records: 9  Duplicates: 0  Warnings: 0
```

```
mysql> select * from advising;
+------+-------+
| stno | empno |
+------+-------+
| 1011 |    19 |
| 2415 |    19 |
| 2661 |    23 |
| 2890 |    23 |
| 5544 |    23 |
| 3442 |    56 |
| 3566 |   126 |
| 4022 |   234 |
| 5571 |   234 |
+------+-------+
9 rows in set (0.00 sec)
```

For odd roll numbers(any 10)

1. Names of students who took some four-credit courses:

```
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             |
+------------------+
| Edwards P. David |
| Mixon Leatha     |
| Pierce Richard   |
| Prior Lorraine   |
| Rawlings Jerry   |
| Lewis Jerry      |
+------------------+
6 rows in set (0.01 sec)
```

2. Names of students who took every four-credit course:

```
mysql> SELECT s.name
    -> FROM STUDENTS s
    -> WHERE NOT EXISTS (
    ->     SELECT c.cno
    ->     FROM COURSES c
    ->     WHERE c.cr = 4
    ->     AND NOT EXISTS (
    ->         SELECT g.cno
    ->         FROM GRADES g
    ->         WHERE g.cno = c.cno AND g.stno = s.stno
    ->     )
    -> );
+------------------+
| name             |
+------------------+
| Edwards P. David |
| Mixon Leatha     |
| Pierce Richard   |
| Prior Lorraine   |
+------------------+
4 rows in set (0.00 sec)
```

3. Names of students who took a course with an instructor who is also their advisor:

```
mysql> SELECT DISTINCT s.name
    -> FROM STUDENTS s
    -> JOIN ADVISING a ON s.stno = a.stno
    -> JOIN GRADES g ON s.stno = g.stno AND g.empno = a.empno;
+------------------+
| name             |
+------------------+
| Edwards P. David |
| Grogan A. Mary   |
| Prior Lorraine   |
| Lewis Jerry      |
+------------------+
4 rows in set (0.00 sec)
```

4. Names of students who took both cs210 and cs310:

```
mysql> SELECT s.name
    -> FROM STUDENTS s
    -> WHERE EXISTS (
    ->     SELECT g.cno FROM GRADES g WHERE g.stno = s.stno AND g.cno = 'cs210'
    -> )
    -> AND EXISTS (
    ->     SELECT g.cno FROM GRADES g WHERE g.stno = s.stno AND g.cno = 'cs310'
    -> );
+---------------+
| name          |
+---------------+
| Mixon Leatha  |
| Prior Lorraine |
+---------------+
2 rows in set (0.00 sec)
```

5. Names of students whose advisor is not a full professor:

```
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 != 'Professor';
+----------------+
| name           |
+----------------+
| Novak Roland   |
| Pierce Richard |
| Prior Lorraine |
| Lewis Jerry    |
+----------------+
4 rows in set (0.00 sec)
```

6. Instructors who taught students advised by another instructor in the same room:

```
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 ADVISING a ON s.stno = a.stno
    -> JOIN INSTRUCTORS i2 ON a.empno = i2.empno
    -> WHERE i1.roomno = i2.roomno AND i1.empno != i2.empno;
+--------------+
| name         |
+--------------+
| Will Samuel  |
| Exxon George |
+--------------+
2 rows in set (0.00 sec)
```

7. Course numbers for courses that enroll exactly two students:

```
mysql> SELECT g.cno
    -> FROM GRADES g
    -> GROUP BY g.cno
    -> HAVING COUNT(DISTINCT g.stno) = 2;
+-------+
| cno   |
+-------+
| cs310 |
| cs410 |
+-------+
2 rows in set (0.00 sec)
```

8. Names of students for whom no other student lives in the same city:

```
mysql> SELECT s1.name
    -> FROM STUDENTS s1
    -> WHERE NOT EXISTS (
    ->     SELECT s2.stno
    ->     FROM STUDENTS s2
    ->     WHERE s2.city = s1.city AND s2.stno != s1.stno
    -> );
+------------------+
| name             |
+------------------+
| Edwards P. David |
| Grogan A. Mary   |
| Novak Roland     |
| Lewis Jerry      |
+------------------+
4 rows in set (0.00 sec)
```

9. Course numbers taken by students living in Boston taught by an associate professor:

```
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 = 'Assoc. Prof.';
+-------+
| cno   |
+-------+
| cs240 |
+-------+
1 row in set (0.00 sec)
```

10. Telephone numbers of instructors teaching courses taken by Boston students:

```
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';
+-------+
| telno |
+-------+
| 9101  |
| 7122  |
| 5110  |
| 7024  |
+-------+
4 rows in set (0.00 sec)
```

11. Names of students who took every course taken by Richard Pierce:

```
mysql> SELECT s1.name
    -> FROM STUDENTS s1
    -> WHERE NOT EXISTS (
    ->     SELECT g2.cno
    ->     FROM GRADES g2
    ->     JOIN STUDENTS s2 ON g2.stno = s2.stno
    ->     WHERE s2.name = 'Richard Pierce'
    ->     AND NOT EXISTS (
    ->         SELECT g1.cno
    ->         FROM GRADES g1
    ->         WHERE g1.cno = g2.cno AND g1.stno = s1.stno
    ->     )
    -> );
+-----------------+
| name            |
+-----------------+
| Edwards P. David |
| Grogan A. Mary   |
| Mixon Leatha     |
| McLane Sandy     |
| Novak Roland     |
| Pierce Richard   |
| Prior Lorraine   |
| Rawlings Jerry   |
| Lewis Jerry      |
+-----------------+
9 rows in set (0.00 sec)
```

12. Names of students who took only one course:

```
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           |
+----------------+
| Grogan A. Mary |
| Novak Roland   |
+----------------+
2 rows in set (0.00 sec)
```

13. Names of instructors who teach no course:

```
mysql> SELECT i.name
    -> FROM INSTRUCTORS i
    -> LEFT JOIN GRADES g ON i.empno = g.empno
    -> WHERE g.cno IS NULL;
+---------------+
| name          |
+---------------+
| Davis William |
+---------------+
1 row in set (0.00 sec)
```

14. Names of instructors who taught only one course in Spring 2001:

```
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)
```

For even roll numbers(any 10)

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

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

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

```
mysql> SELECT DISTINCT s.name
    -> FROM students s
    -> WHERE NOT EXISTS (
    ->     SELECT 1
    ->     FROM grades g
    ->     JOIN courses c ON g.cno = c.cno
    ->     WHERE s.stno = g.stno AND c.cr = 4
    -> );
+----------------+
| name           |
+----------------+
| Grogan A. Mary |
| McLane Sandy   |
| Novak Roland   |
+----------------+
3 rows in set (0.00 sec)
```

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

```
mysql> SELECT DISTINCT s.name
    -> FROM students s
    -> JOIN grades g ON s.stno = g.stno
    -> WHERE g.cno IN ('cs210', 'cs310');
+------------------+
| name             |
+------------------+
| Edwards P. David |
| Mixon Leatha     |
| Pierce Richard   |
| Prior Lorraine   |
| Lewis Jerry      |
+------------------+
5 rows 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.

```
mysql> SELECT DISTINCT s.name
    -> FROM students s
    -> JOIN grades g1 ON s.stno = g1.stno
    -> WHERE g1.cno = 'cs210'
    ->   AND g1.grade > (
    ->       SELECT MAX(g2.grade)
    ->       FROM grades g2
    ->       WHERE g2.cno = 'cs310'
    ->   )
    ->   AND s.stno NOT IN (
    ->       SELECT g.stno
    ->       FROM grades g
    ->       WHERE g.empno = (
    ->           SELECT empno FROM instructors WHERE name = 'Evans Robert'
    ->       )
    ->   );
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.

```
mysql> SELECT cno
    -> FROM grades
    -> GROUP BY cno
    -> HAVING COUNT(DISTINCT stno) >= 2;
+-------+
| cno   |
+-------+
| cs110 |
| cs210 |
| cs240 |
| cs310 |
| cs410 |
+-------+
5 rows in set (0.00 sec)
```

```
mysql> SELECT cno
    -> FROM grades
    -> GROUP BY cno
    -> HAVING COUNT(DISTINCT stno) >= 3;
+-------+
| cno   |
+-------+
| cs110 |
| cs210 |
| cs240 |
+-------+
3 rows in set (0.00 sec)
```

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

```
mysql> SELECT s.name
    -> FROM students s
    -> JOIN grades g ON s.stno = g.stno
    -> WHERE g.cno = 'cs210'
    ->   AND g.grade = (
    ->       SELECT MAX(grade)
    ->       FROM grades
    ->       WHERE cno = 'cs210'
    ->   );
+-----------------+
| name            |
+-----------------+
| Edwards P. David |
| Pierce Richard   |
+-----------------+
2 rows 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.

```
mysql> SELECT DISTINCT i.name
    -> FROM instructors i
    -> WHERE i.empno IN (
    ->     SELECT g1.empno
    ->     FROM grades g1
    ->     WHERE g1.stno IN (
    ->         SELECT g2.stno
    ->         FROM grades g2
    ->         JOIN instructors i2 ON g2.empno = i2.empno
    ->         WHERE i2.rank = 'Assist. Prof.'
    ->     )
    -> );
+-------------+
| name        |
+-------------+
| Evans Robert |
| Exxon George |
| Sawyer Kathy |
| Will Samuel  |
+-------------+
4 rows in set (0.00 sec)
```

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

```
mysql> SELECT MIN(grade) AS lowest_grade
    -> FROM grades
    -> WHERE sem = 'Spring' AND year = 2003;
+--------------+
| lowest_grade |
+--------------+
|           70 |
+--------------+
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).

```
mysql> SELECT s.name
    -> FROM students s
    -> WHERE NOT EXISTS (
    ->     SELECT 1
    ->     FROM grades g
    ->     WHERE g.empno = (SELECT empno FROM instructors WHERE name = 'Evans Robert')
    ->       AND NOT EXISTS (
    ->           SELECT 1
    ->           FROM grades g2
    ->           WHERE g2.stno = s.stno AND g2.cno = g.cno
    ->       )
    -> );
+-----------------+
| name            |
+-----------------+
| Edwards P. David |
| Pierce Richard   |
| Prior Lorraine   |
+-----------------+
3 rows in set (0.00 sec)
```

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

```
mysql> SELECT DISTINCT s.name
    -> FROM students s
    -> JOIN advising a ON s.stno = a.stno
    -> WHERE NOT EXISTS (
    ->     SELECT 1
    ->     FROM grades g
    ->     WHERE g.stno = s.stno AND g.empno = a.empno
    -> );
+----------------+
| name           |
+----------------+
| Mixon Leatha   |
| McLane Sandy   |
| Novak Roland   |
| Pierce Richard |
| Rawlings Jerry |
+----------------+
5 rows in 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).

```
mysql> SELECT DISTINCT s.name
    -> FROM students s
    -> WHERE NOT EXISTS (
    ->     SELECT 1
    ->     FROM grades g
    ->     WHERE g.stno = s.stno AND g.grade >= 60
    -> );
+---------------+
| name          |
+---------------+
| McLane Sandy |
+---------------+
1 row in set (0.00 sec)
```

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

```
mysql> SELECT MAX(g.grade) AS highest_grade
    -> FROM grades g
    -> WHERE g.stno NOT IN (
    ->     SELECT stno
    ->     FROM grades
    ->     WHERE cno = 'cs110'
    -> );
+---------------+
| highest_grade |
+---------------+
|           100 |
+---------------+
1 row in set (0.00 sec)
```

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

```
mysql> SELECT s.name
    -> FROM students s
    -> WHERE s.stno NOT IN (SELECT stno FROM advising);
Empty set (0.00 sec)
```

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

```
mysql> SELECT DISTINCT c.cname
    -> FROM courses c
    -> JOIN grades g ON c.cno = g.cno
    -> JOIN students s ON g.stno = s.stno
    -> WHERE s.state != 'MA';
+----------------------+
| cname                |
+----------------------+
| Software Engineering |
| Computer Programming |
| Computer Architecture |
+----------------------+
3 rows in set (0.00 sec)
```