# Practical - 2

## 20-12-24

## Group-by and Sub-Queries

**Q1) USING (practical - 1)**

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

```
mysql> SELECT COUNT(*) AS customers
-> FROM customer
-> WHERE grade > (
-> SELECT A VG(grade)
-> FROM customer
-> WHERE city = 'Bangalore'
-> );
+--------------------------+
| customers_avg_grade |
+--------------------------+
              | 0 |
+--------------------------+
1 row in set (0.03 sec)
```

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

```
mysql> SELECT s.salesman
-> FROM salesman s
-> JOIN customer c ON s.salesman=c.salesman_id
-> GROUP BY s.salesman_id
-> HAVING COUNT(c.customer_id>1);
```

```
+-------------+------------+
| salesman_id | name       |
+-------------+------------+
|        5001 | James Hooq |
|        5002 | Nail Knite |
+-------------+------------+
2 rows in set (0.01 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
-> FROM salesman s
-> JOIN customer c ON s.salesman
-> WHERE s.city = c.city
->
```

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

```
+-------------+------------+----------+---------------+
| salesman_id | name       | city     | status        |
+-------------+------------+----------+---------------+
|        5001 | James Hooq | New York | Has Customers |
|        5006 | Mc Lyon    | Paris    | Has Customers |
|        5005 | Pit Alex   | London   | No Customers  |
+-------------+------------+----------+---------------+
3 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 highest_order_per_day AS
   -> SELECT o.order_date,
   -> o.salesman_id,
   -> o.customer_id,
   -> o.purch_amt
   -> FROM `order` o
   -> JOIN customer c ON o.customer_id = c.customer_id
   -> JOIN salesman s ON o.salesman_id = s.salesman_id
   -> WHERE (o.order_date, o.purch_amt) IN (
   -> SELECT order_date, MAX(purch_amt)
   -> FROM `order`
   -> GROUP BY order_date
   -> );
Query OK, 0 rows affected (0.02 sec)
```

**5.Demonstrate the DELETE operation by removing salesman with id 1000.**

**All his orders must also be deleted**

```
mysql> DELETE FROM `order`
   -> WHERE salesman_id = 5001; Query OK, 3 rows affected (0.01 sec)
mysql> select * from salesman;
   + +          +          +          +
| salesman_id | name   | city      | commission |
```

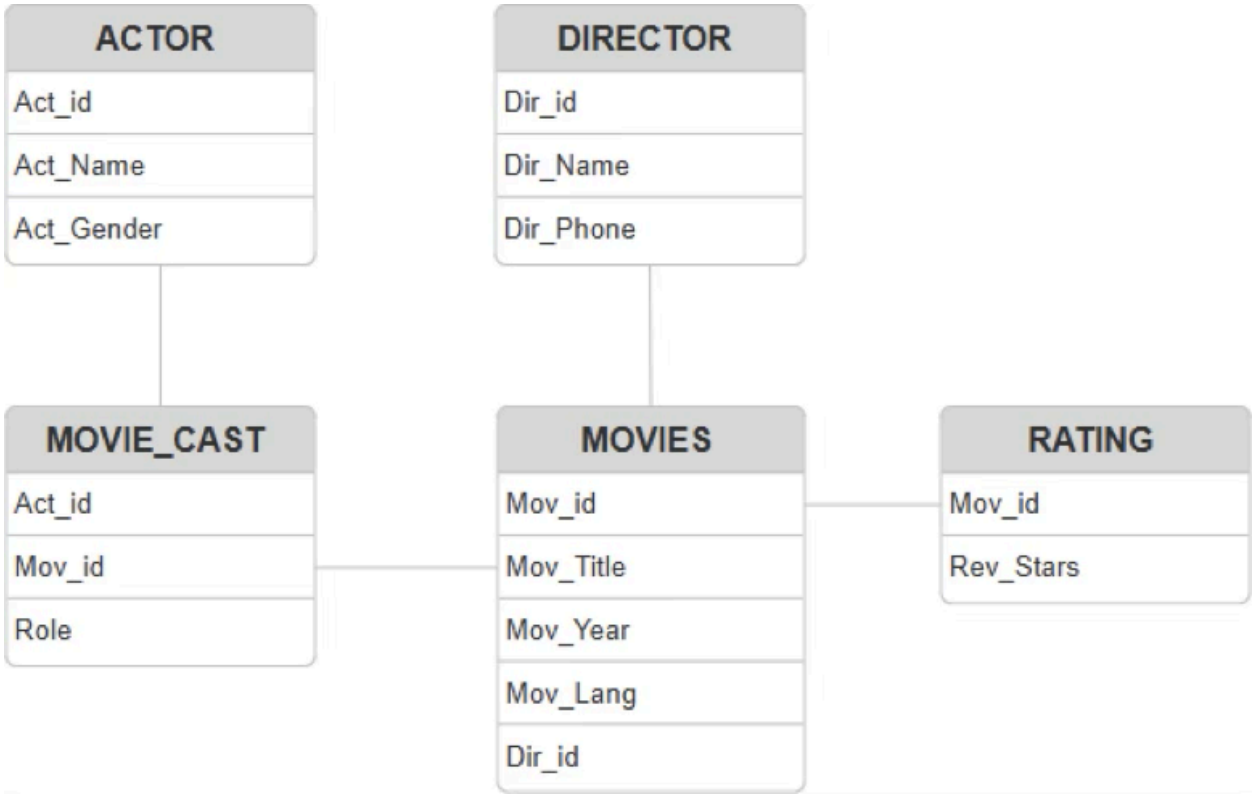
```
+ +         +         +         +
|      5001 | James Hooq | New York |   0.15 |
|      5002 | Nail Knite | Paris    |   0.13 |
|      5003 | Lauson Hen | NULL     |   0.12 |
|      5005 | Pit Alex   | London   |   0.11 |
|      5006 | Mc Lyon    | Paris    |   0.14 |
|      5007 | Paul Adam  | Rome     |   0.13 |
+ +         +         +         +
6 rows in set (0.00 sec)
```

```
mysql> select * from `order`;
+ +         +         +         +         +
| order_no | purch_amt | order_date | customer_id | salesman_id |
+ +         +         +         +         +
|      70001 |  150.5 | 2016-10-05 |       3005 |    5002 |
|      70003 |  2480.4 | 2016-10-10 |       3009 |    NULL |
|      70004 |  110.5 | 2016-08-17 |       3009 |    NULL |
|      70007 |  948.5 | 2016-09-10 |       3005 |    5002 |
|      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 |
+ +         +         +         +         +
8 rows in set (0.00 sec)
```

**Q2) 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)**




```
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 (0.02 sec)
mysql> INSERT INTO ACTOR VALUES (301,'ANUSHKA','F');
Query OK, 1 row affected (0.02 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)
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 (0.01 sec)
mysql> INSERT INTO DIRECTOR VALUES (60, 'RAJAMOULI', 8751611001);
Query OK, 1 row affected (0.01 sec)
mysql> INSERT INTO DIRECTOR VALUES (61, 'HITCHCOCK', 7766138911);
Query OK, 1 row affected (0.00 sec)
mysql> INSERT INTO DIRECTOR VALUES (62, 'FARAN', 9986776531);
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO DIRECTOR VALUES (63, 'STEVEN SPIELBERG', 8989776530);
Query OK, 1 row affected (0.00 sec)
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, 3 warnings (0.02 sec)
mysql> INSERT INTO MOVIES VALUES (1001, 'BAHUBALI-2', 2017, 'TELAGU', 60);
Query OK, 1 row affected (0.00 sec)
mysql> INSERT INTO MOVIES VALUES (1002, 'BAHUBALI-1', 2015, 'TELAGU', 60);
Query OK, 1 row affected (0.01 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 (0.00 sec)
mysql> CREATE TABLE MOVIE_CAST (
   -> ACT_ID INT(3),

   -> MOV_ID INT(4),
   -> ROLE_NAME VARCHAR(30),
   -> 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 (0.03 sec)
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 (0.01 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 (0.00 sec)
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, 1 warning (0.02 sec)
mysql> INSERT INTO RATING VALUES (1001, 4);
Query OK, 1 row affected (0.00 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)
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> select * from movie_cast;
  +  +         +         +
| ACT_ID | MOV_ID | ROLE_NAME |
  +  +         +         +
|      301 |     1001 | HEROINE           |
```

```
|      301 |      1002 | HEROINE           |
|      303 |      1002 | GUEST    |
|      303 |      1003 | HERO     |
|      304 |      1004 | HERO     |
  +  +          +          +
5 rows in set (0.00 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> 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 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 (0.01 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
   -> OR m.MOV_YEAR > 2015;
+    +
| ACT_NAME |
+    +
| ANUSHKA |
+    +
1 row in 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.**

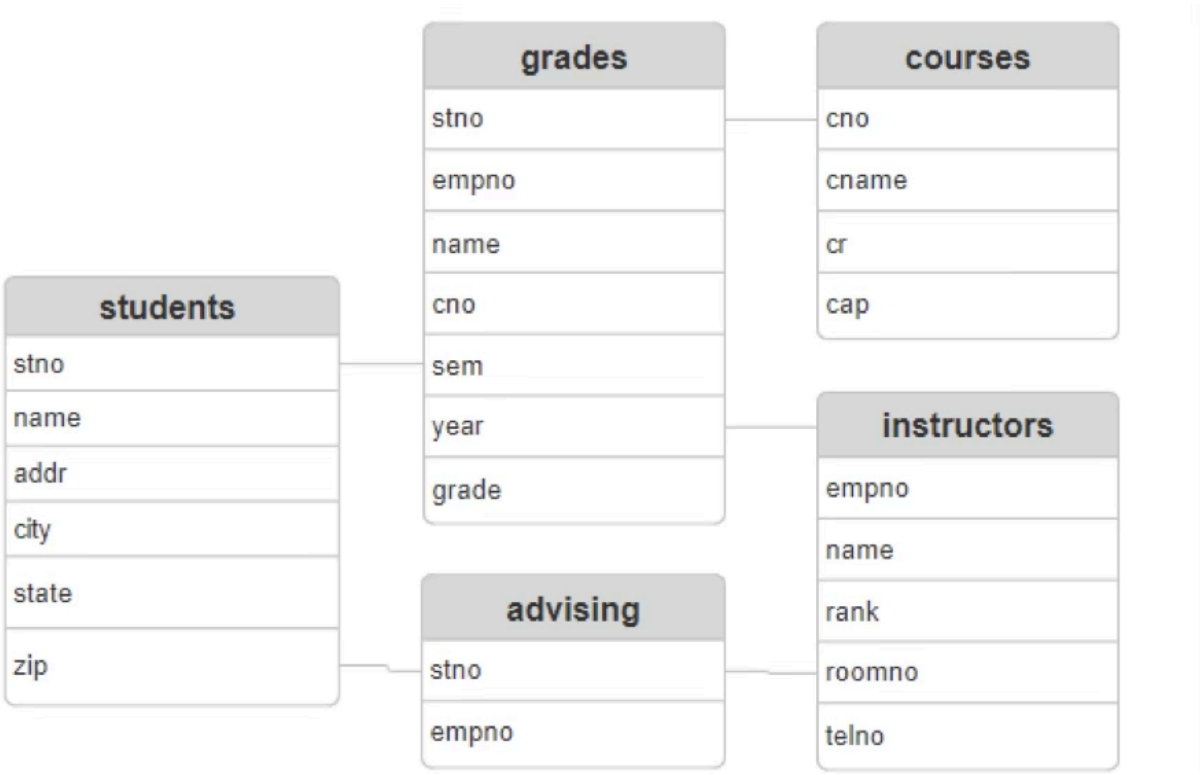
```
mysql> SELECT m.MOV_TITLE, r.REV_STARS
   -> FROM movies m
   -> JOIN rating r ON m.MOV_ID = r.MOV_ID
   -> WHERE r.REV_STARS IS NOT NULL
   -> ORDER BY m.MOV_TITLE;
+    +          +
| MOV_TITLE | REV_STARS |
+    +          +
```

| AKASH | 5 |
| BAHUBALI-1 | 2 |
| BAHUBALI-2 | 4 |
| WAR HORSE | 4 |
+ + +
4 rows in set (0.00 sec)

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

```
mysql> UPDATE rating r
   -> JOIN movies m ON r.MOV_ID = m.MOV_ID
   -> JOIN director d ON m.DIR_ID = d.DIR_ID
   -> SET r.REV_STARS = 5
   -> WHERE d.DIR_NAME = 'STEVEN SPIELBERG';
Query OK, 1 row affected (0.02 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)
```

**Q3) Design ERD for the following schema and execute the following Queries on it:**




```
mysql> CREATE TABLE STUDENTS (
    -> stno INT,
    -> name VARCHAR(255),
    -> addr VARCHAR(255),
    -> city VARCHAR(255),
    -> state CHAR(2),
    -> zip CHAR(5)
    -> );
Query OK, 0 rows affected (0.07 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> CREATE TABLE INSTRUCTORS (
```

```
->     empno INT,
   -> name VARCHAR(255),
   -> rankk VARCHAR(255), -- Changed rank to rankk
   -> roomno INT,
   -> telno INT
   -> );
Query OK, 0 rows affected (0.02 sec)

mysql> INSERT INTO INSTRUCTORS (empno, name, rankk, roomno, telno) VALUES -- Changed rank
to rankk
   -> (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> CREATE TABLE COURSES (
   -> cno VARCHAR(255),
   -> cname VARCHAR(255),
   -> cr INT,
   -> cap INT
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> 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> select * from instructors;

| empno | name | rankk | 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 |

| stno | empno | cno | sem | year | grade |
|---|---|---|---|---|---|
| 1011 | 19 | cs110 | Fall | 2001 | 40 |
| 2661 | 19 | cs110 | Fall | 2001 | 40 |
| 3566 | 19 | cs110 | Fall | 2001 | 95 |
| 5544 | 19 | cs110 | Fall | 2001 | 100 |
| 1011 | 23 | cs110 | Spring | 2002 | 75 |
| 4022 | 23 | cs110 | Spring | 2002 | 60 |

| | | | | | |
|---|---|---|---|---|---|
| 3566 | 19 | cs240 | Spring | 2002 | 100 |
| 5571 | 19 | cs240 | Spring | 2002 | 50 |
| 2415 | 19 | cs240 | Spring | 2002 | 100 |
| 3442 | 234 | cs410 | Spring | 2002 | 60 |
| 5571 | 234 | cs410 | Spring | 2002 | 80 |
| 1011 | 19 | cs210 | Fall | 2002 | 90 |
| 2661 | 19 | cs210 | Fall | 2002 | 70 |
| 3566 | 19 | cs210 | Fall | 2002 | 90 |
| 5571 | 19 | cs210 | Spring | 2003 | 85 |
| 4022 | 19 | cs210 | Spring | 2003 | 70 |

mysql> select * from grades;

| | | | | | |
|---|---|---|---|---|---|
| 5544 | 56 | cs240 | Spring | 2003 | 70 |
| 1011 | 56 | cs240 | Spring | 2003 | 90 |
| 4022 | 56 | cs240 | Spring | 2003 | 80 |
| 2661 | 234 | cs310 | Spring | 2003 | 100 |
| 4022 | 234 | cs310 | Spring | 2003 | 75 |

21 rows in set (0.00 sec)

mysql> select * from ADVISING ;

| stno | empno |
|---|---|
| 1011 | 19 |
| 2415 | 19 |
| 2661 | 23 |
| 2890 | 23 |
| 3442 | 56 |
| 3566 | 126 |
| 4022 | 234 |
| 5544 | 23 |

| 5571 | 234 |
+ + +
9 rows in set (0.00 sec)

**For even roll numbers(any 10)**

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

```
mysql> SELECT DISTINCT s.name
   -> FROM STUDENTS s
   -> WHERE NOT EXISTS (
   -> SELECT *
   -> FROM GRADES g, COURSES c
   -> WHERE s.stno = g.stno AND g.cno = c.cno 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, GRADES g
   -> WHERE s.stno = g.stno AND (g.cno = 'cs210' OR g.cno = '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 Physics101 and did not take any course with Prof. Evans.**

```
mysql> SELECT DISTINCT s.name
   -> FROM STUDENTS s, GRADES g1, GRADES g2
-> WHERE s.stno = g1.stno AND g1.cno = 'cs210' AND s.stno = g2.stno AND g2.cno = 'cs310' AND
g1.grade > (SELECT MAX(grade) FROM GRADES WHERE cno = 'cs310')
   -> AND NOT EXISTS (
   -> SELECT *
   -> FROM GRADES g3, INSTRUCTORS i
   -> WHERE s.stno = g3.stno AND g3.empno = i.empno AND i.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> -- At least two students mysql> SELECT g.cno
   -> FROM GRADES g
   -> GROUP BY g.cno
   -> HAVING COUNT(DISTINCT g.stno) >= 2;
 +    +
| cno |
 +    +
| cs110 |
| cs210 |
| cs240 |
| cs310 |
| cs410 |
 +    +
5 rows in set (0.00 sec)
mysql>
mysql> -- At least three students mysql> SELECT g.cno
-> FROM GRADES g
   -> GROUP BY g.cno
   -> HAVING COUNT(DISTINCT g.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, GRADES g
    -> WHERE s.stno = g.stno AND 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 i1.name
    -> FROM INSTRUCTORS i1, GRADES g1, STUDENTS s1, GRADES g2, INSTRUCTORS i2
-> WHERE i1.empno = g1.empno AND g1.stno = s1.stno AND s1.stno = g2.stno AND g2.empno =
i2.empno AND i2.rankk = '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(g.grade)
    -> FROM GRADES g
    -> WHERE g.sem = 'Spring' AND g.year = 2003;
+    +
| MIN(g.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 DISTINCT s.name
    -> FROM STUDENTS s
    -> WHERE NOT EXISTS (
    -> SELECT *
    -> FROM COURSES c, GRADES g1, INSTRUCTORS i
->     WHERE c.cno = g1.cno AND g1.empno = i.empno AND i.name = 'Evans Robert' AND NOT
EXISTS (
    -> SELECT *
    -> FROM GRADES g2
    -> WHERE s.stno = g2.stno AND g2.cno = c.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, ADVISING a
    -> WHERE s.stno = a.stno AND NOT EXISTS (
    -> SELECT *
    -> FROM GRADES g
    -> WHERE s.stno = g.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 *
   -> FROM GRADES g
   -> WHERE s.stno = g.stno AND g.grade >= 60
   -> );
+    +
| name           |
+    +
| McLane Sandy |
+    +
1 row in set (0.00 sec)
```