**PRACTICAL 2**

**AIM: Subquery-join operations on Relational Schema.**

1. **Using Practical 1**
2. **Count the customers with grades above Bangalore’s average.**

select COUNT(\*)

from customer

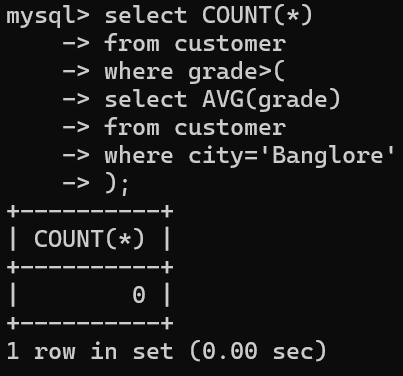
where grade>(

select AVG(grade)

from customer

where city='Banglore'

);



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

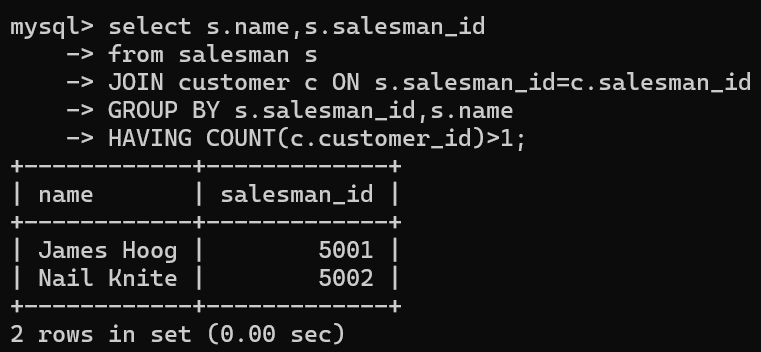
select s.name,s.salesman\_id

from salesman s

JOIN customer c ON s.salesman\_id=c.salesman\_id

GROUP BY s.salesman\_id,s.name

HAVING COUNT(c.customer\_id)>1;



1. **List all salesmen and indicate those who have and don’t have customers in their cities (Use UNION operation.)**

select s.salesman\_id,s.name,'Has Customers' As customer\_status

from salesman s

JOIN customer c ON s.salesman\_id=c.salesman\_id

where s.city=c.city

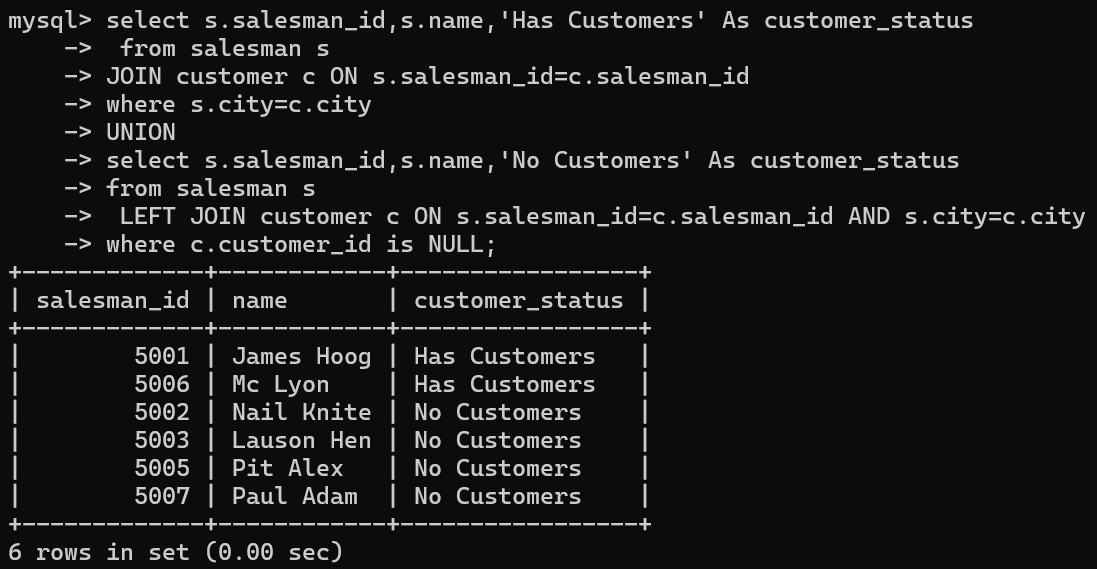
UNION

select s.salesman\_id,s.name,'No Customers' As customer\_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;



1. **Create a view that finds the salesman who has the customer with the highest order of a day.**

create VIEW SalesmanWithHighestOrder As

select s.salesman\_id,s.name,o.order\_date,Max(o.purch\_amt) As max\_order\_amount

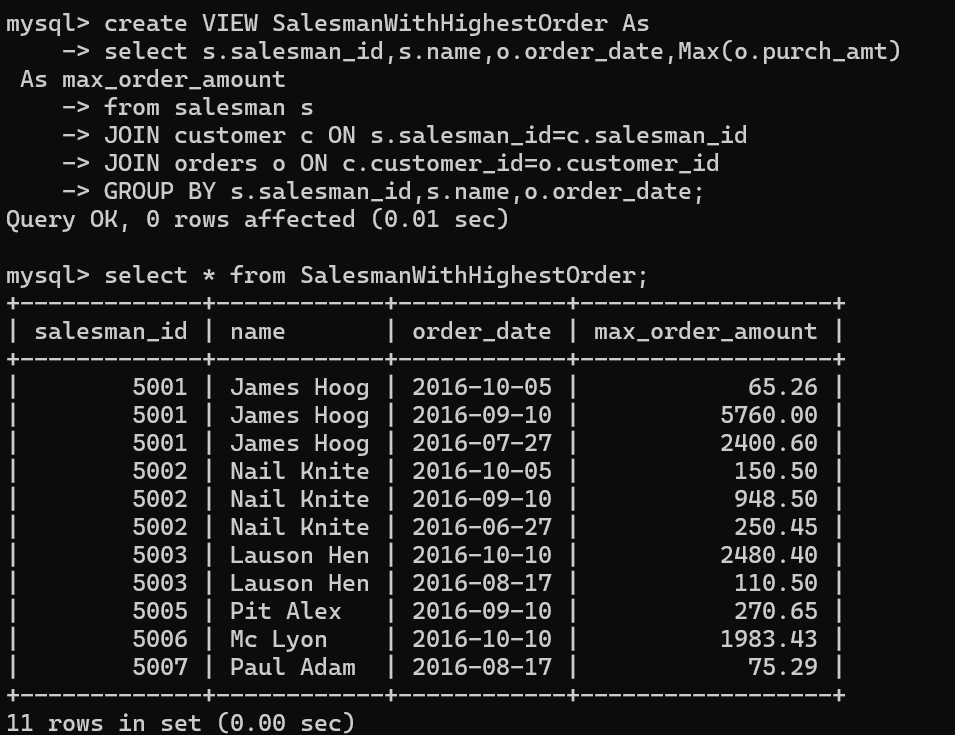
from salesman s

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;

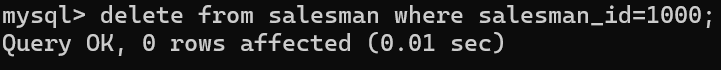


1. **Demonstrate the DELETE operation by removing salesman with id 1000. All his orders must also be deleted.**

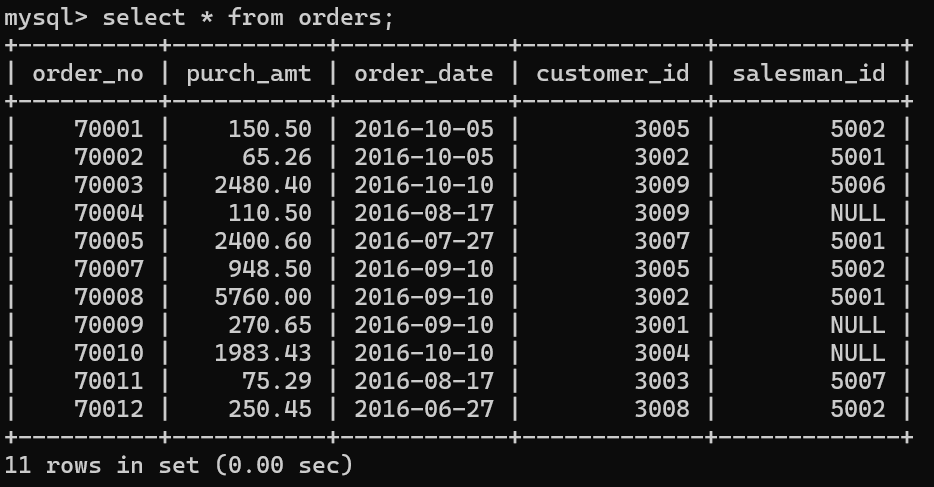
delete from salesman where salesman\_id=1000;

select \* from salesman;

select \* from orders;







1. **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)**

**Create tables for actor, director, movies, movie\_cast, rating.**

create table actor(

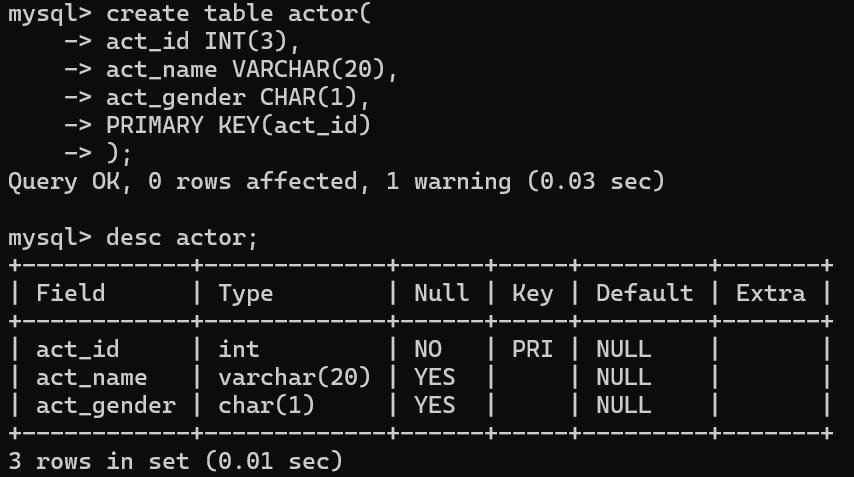
act\_id INT(3),

act\_name VARCHAR(20),

act\_gender CHAR(1),

PRIMARY KEY(act\_id)

);



create table director(

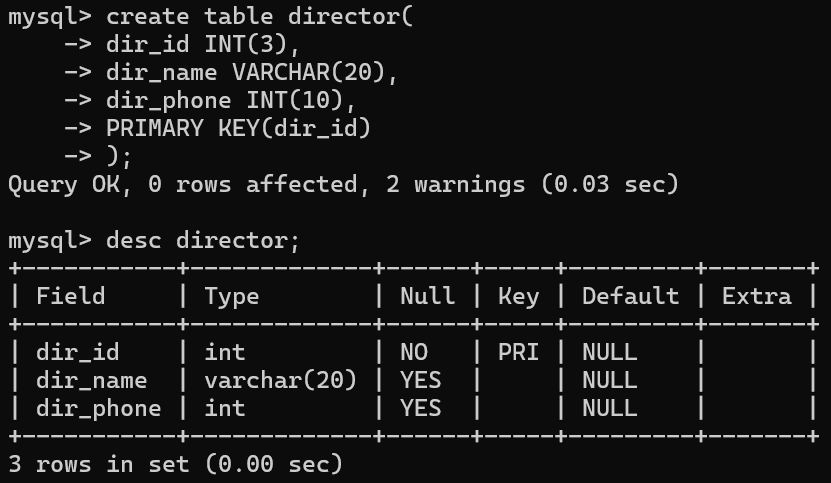
dir\_id INT(3),

dir\_name VARCHAR(20),

dir\_phone INT(10),

PRIMARY KEY(dir\_id)

);



create table movies(

mov\_id INT(4),

mov\_title VARCHAR(25),

mov\_year INT(4),

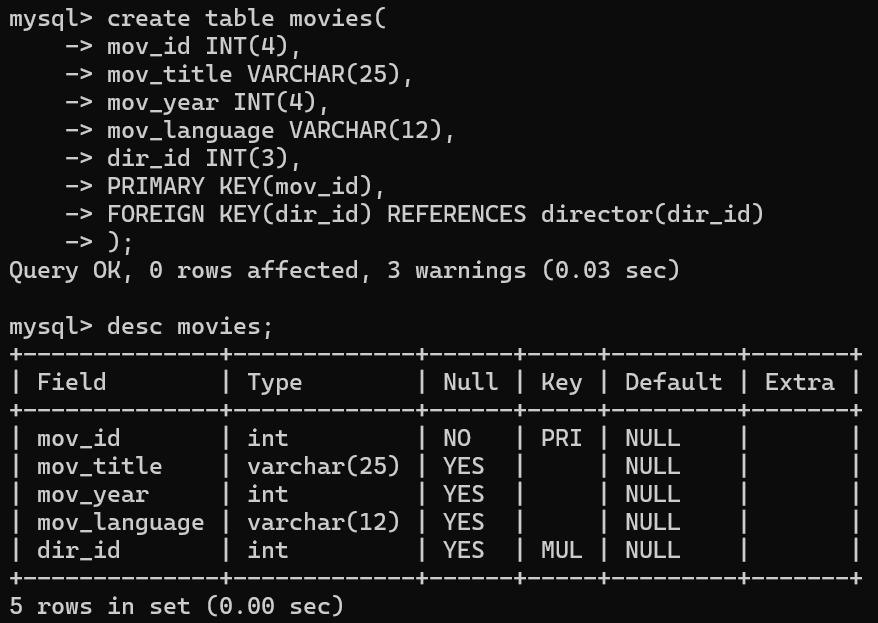
mov\_language VARCHAR(12),

dir\_id INT(3),

PRIMARY KEY(mov\_id),

FOREIGN KEY(dir\_id) REFERENCES director(dir\_id)

);



create table movie\_cast(

act\_id INT(3),

mov\_id INT(4),

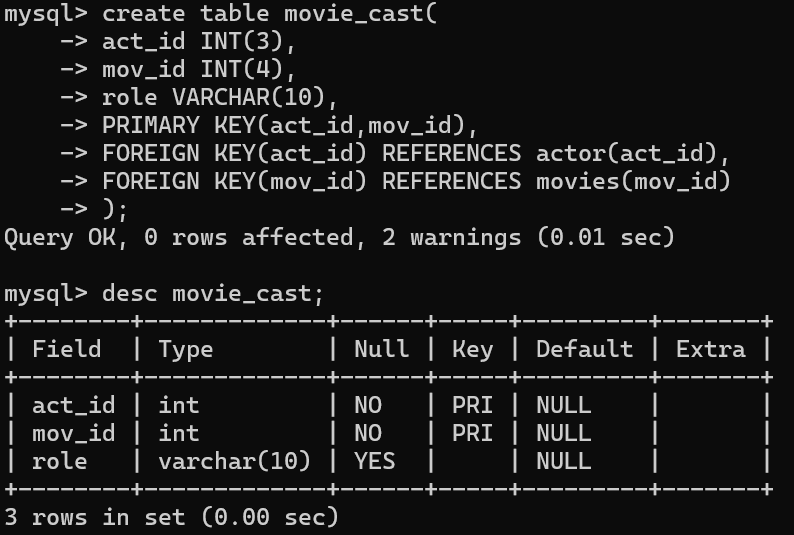
role VARCHAR(10),

PRIMARY KEY(act\_id,mov\_id),

FOREIGN KEY(act\_id) REFERENCES actor(act\_id),

FOREIGN KEY(mov\_id) REFERENCES movies(mov\_id)

);



create table rating(

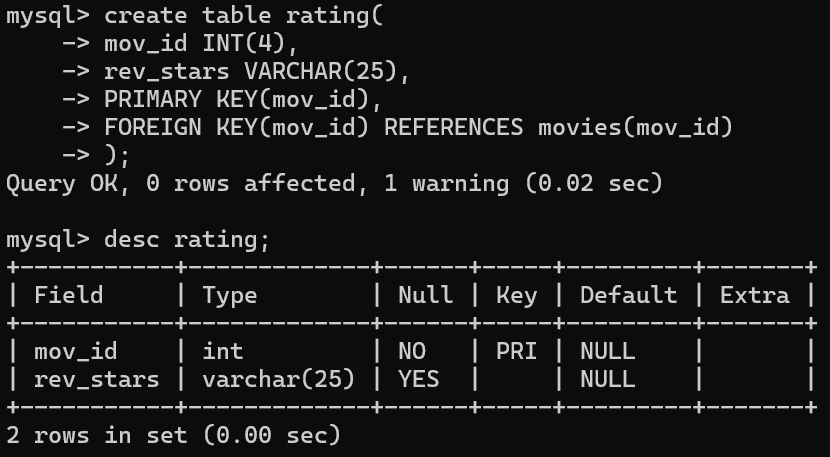
mov\_id INT(4),

rev\_stars VARCHAR(25),

PRIMARY KEY(mov\_id),

FOREIGN KEY(mov\_id) REFERENCES movies(mov\_id)

);



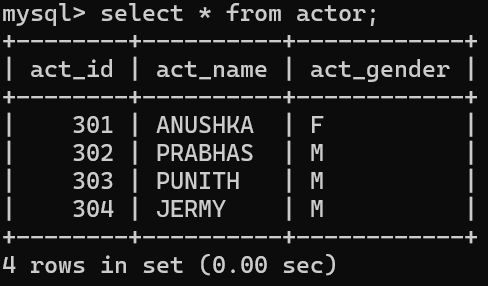
**Insert values into tables.**

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');

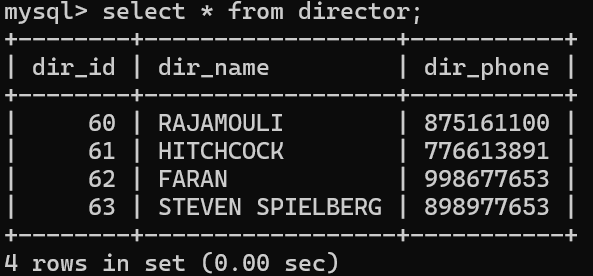


insert into director values(60,'RAJAMOULI', 875161100);

insert into director values(61,'HITCHCOCK', 776613891);

insert into director values(62,'FARAN', 998677653);

insert into director values(63,'STEVEN SPIELBERG', 898977653);

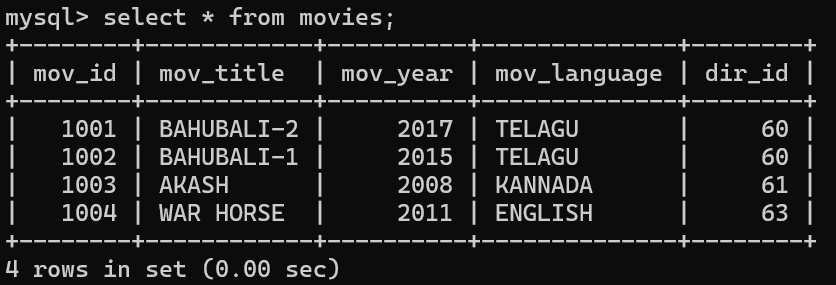


insert into movies values(1001,'BAHUBALI-2', 2017, 'TELAGU', 60);

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);



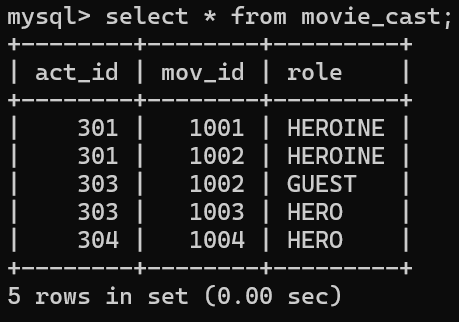
insert into movie\_cast values(301, 1002, 'HEROINE');

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');

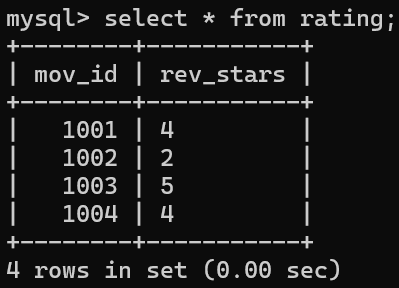


insert into rating values(1001,4);

insert into rating values(1002,2);

insert into rating values(1003,5);

insert into rating values(1004,4);



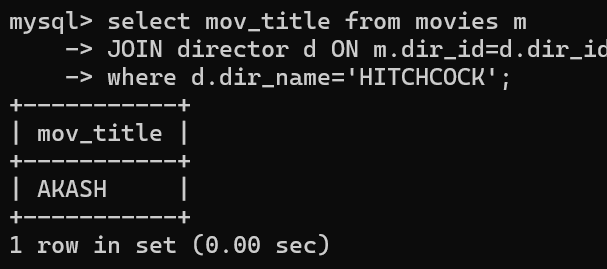
**Write SQL queries to**

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

select mov\_title from movies m

JOIN director d ON m.dir\_id=d.dir\_id

where d.dir\_name='HITCHCOCK';



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

select DISTINCT m.mov\_title

from movies m

JOIN movie\_cast mc ON m.mov\_id=mc.mov\_id

where mc.act\_id IN(

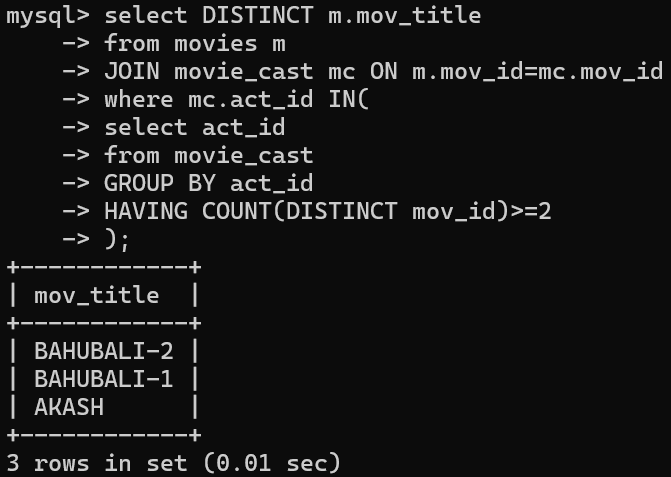
select act\_id

from movie\_cast

GROUP BY act\_id

HAVING COUNT(DISTINCT mov\_id)>=2

);



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

select DISTINCT a.act\_name

from actor a

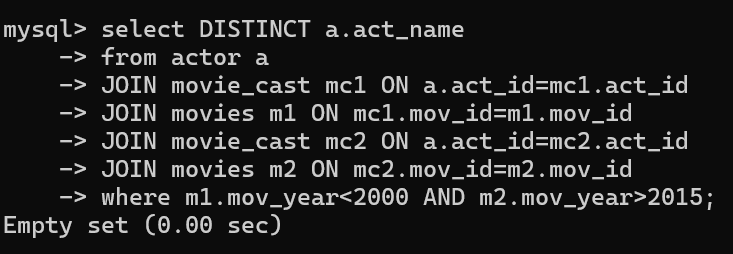
JOIN movie\_cast mc1 ON a.act\_id=mc1.act\_id

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;



1. **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.**

select m.mov\_title,r.rev\_stars,(

select max(r1.rev\_stars)

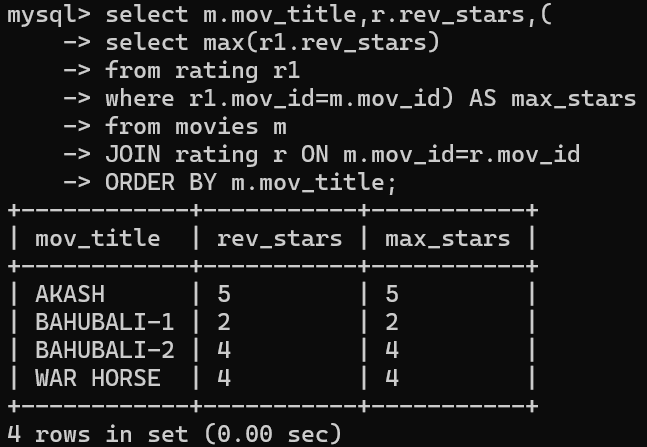
from rating r1

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;



1. **Update rating of all movies directed by ‘Steven Spielberg’ to 5.**

update rating

set rev\_stars='5'

where mov\_id in(

select m.mov\_id

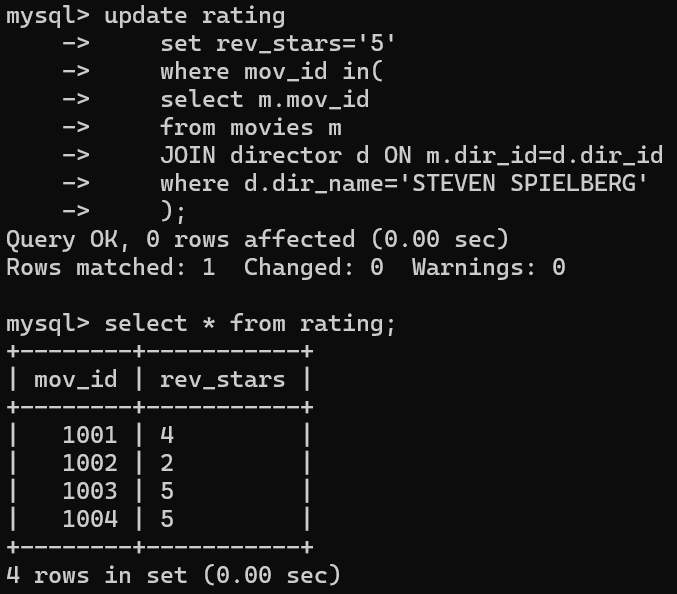
from movies m

JOIN director d ON m.dir\_id=d.dir\_id

where d.dir\_name='STEVEN SPIELBERG'

);

select \* from rating;



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

**Create tables.**

CREATE TABLE students (

stno INT PRIMARY KEY,

name VARCHAR(50),

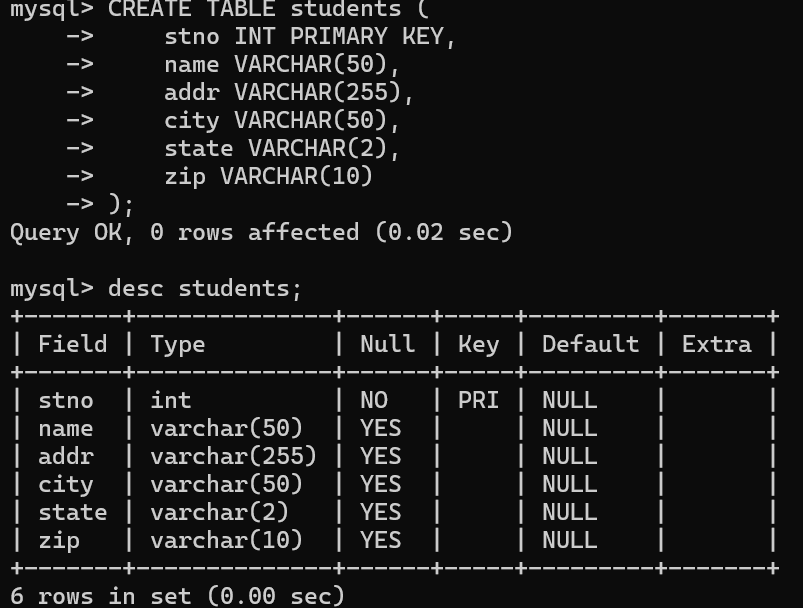
addr VARCHAR(255),

city VARCHAR(50),

state VARCHAR(2),

zip VARCHAR(10)

);



CREATE TABLE INSTRUCTORS (

empno INT PRIMARY KEY,

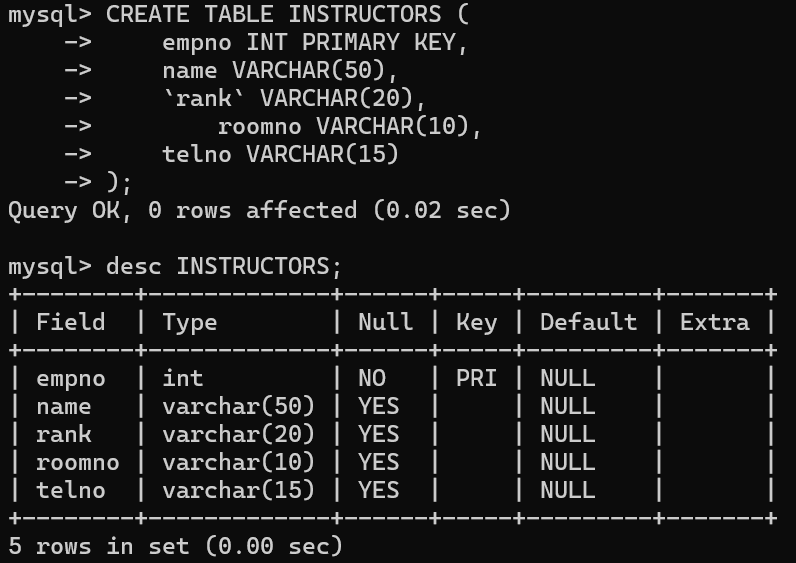
name VARCHAR(50),

`rank` VARCHAR(20),

roomno VARCHAR(10),

telno VARCHAR(15)

);



CREATE TABLE COURSES (

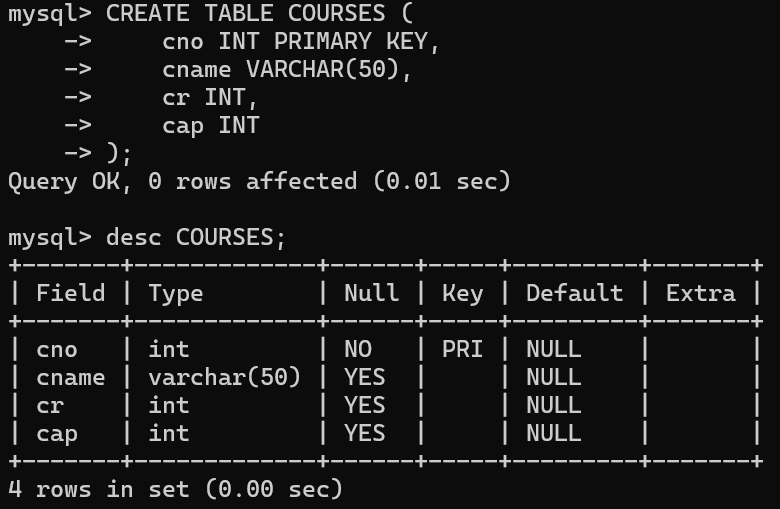
cno INT PRIMARY KEY,

cname VARCHAR(50),

cr INT,

cap INT

);



CREATE TABLE GRADES (

stno INT,

empno INT,

cno INT,

sem VARCHAR(10),

year INT,

grade INT,

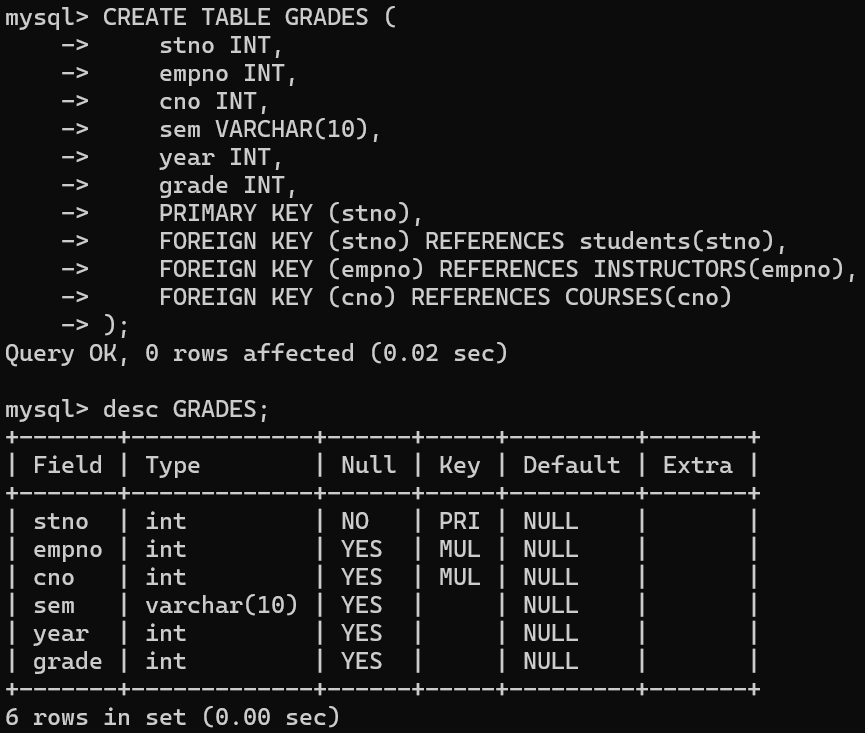
PRIMARY KEY (stno),

FOREIGN KEY (stno) REFERENCES students(stno),

FOREIGN KEY (empno) REFERENCES INSTRUCTORS(empno),

FOREIGN KEY (cno) REFERENCES COURSES(cno)

);



CREATE TABLE ADVISING (

stno INT,

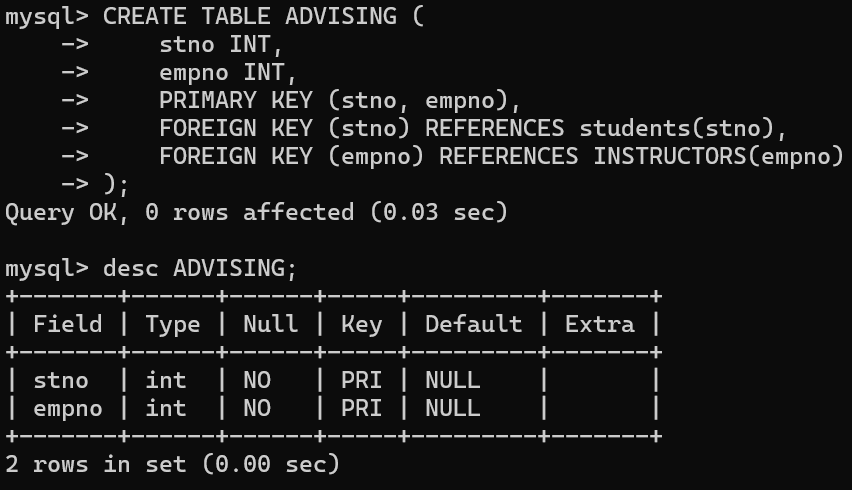
empno INT,

PRIMARY KEY (stno, empno),

FOREIGN KEY (stno) REFERENCES students(stno),

FOREIGN KEY (empno) REFERENCES INSTRUCTORS(empno)

);



**Insert values into tables.**

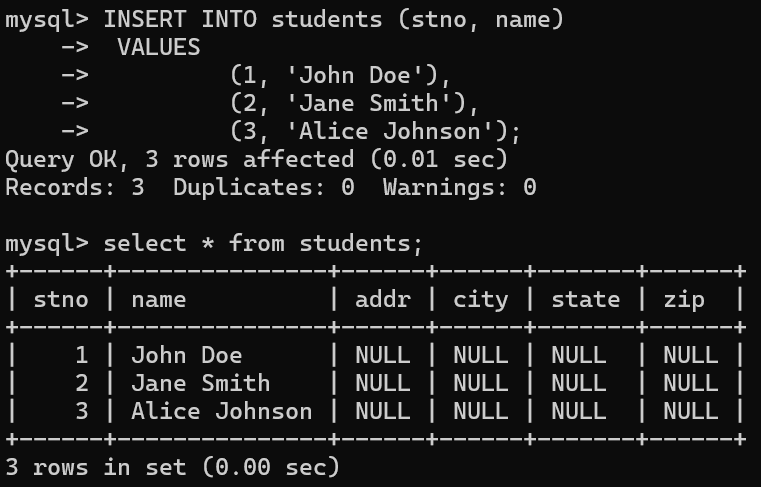
INSERT INTO students (stno, name)

VALUES

(1, 'John Doe'),

(2, 'Jane Smith'),

(3, 'Alice Johnson');



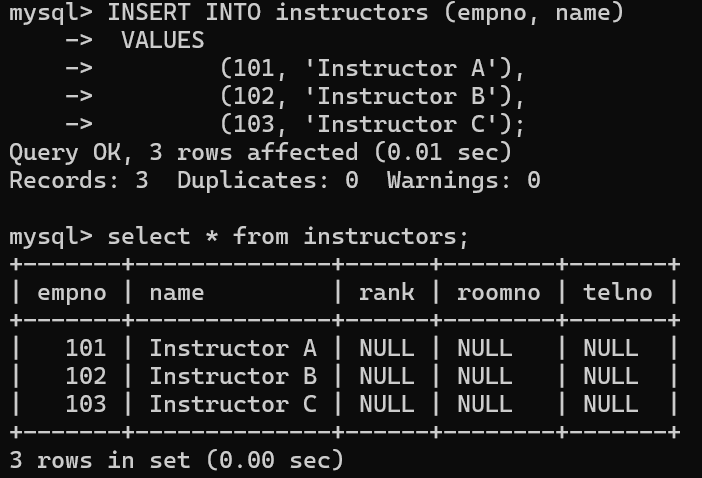
INSERT INTO instructors (empno, name)

VALUES

(101, 'Instructor A'),

(102, 'Instructor B'),

(103, 'Instructor C');



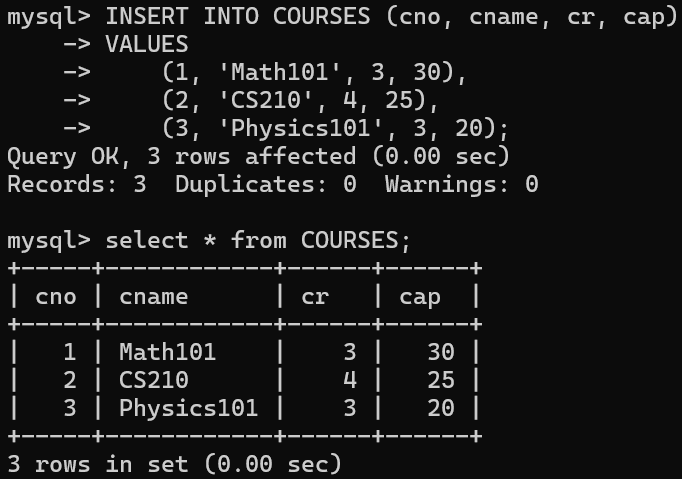
INSERT INTO COURSES (cno, cname, cr, cap)

VALUES

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

(2, 'CS210', 4, 25),

(3, 'Physics101', 3, 20);



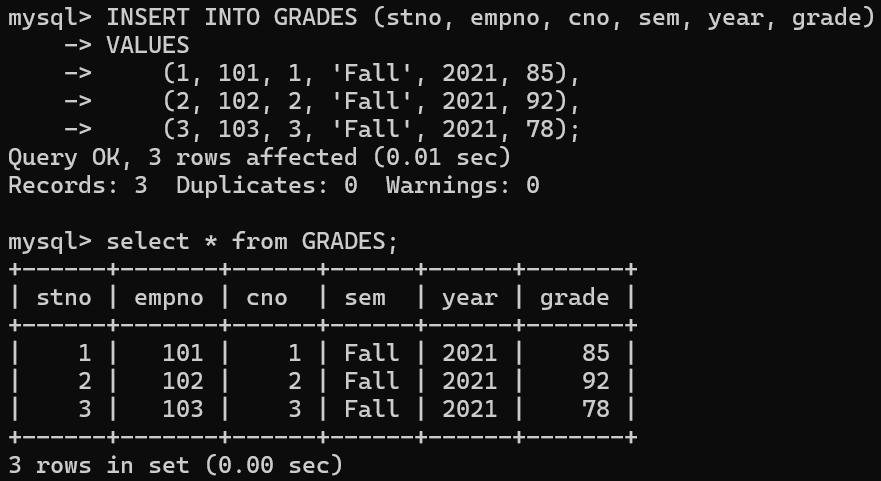
INSERT INTO GRADES (stno, empno, cno, sem, year, grade)

VALUES

(1, 101, 1, 'Fall', 2021, 85),

(2, 102, 2, 'Fall', 2021, 92),

(3, 103, 3, 'Fall', 2021, 78);



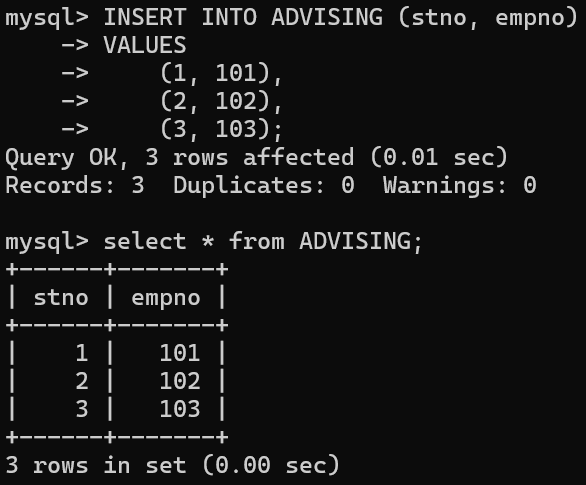
INSERT INTO ADVISING (stno, empno)

VALUES

(1, 101),

(2, 102),

(3, 103);



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

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

select s.name

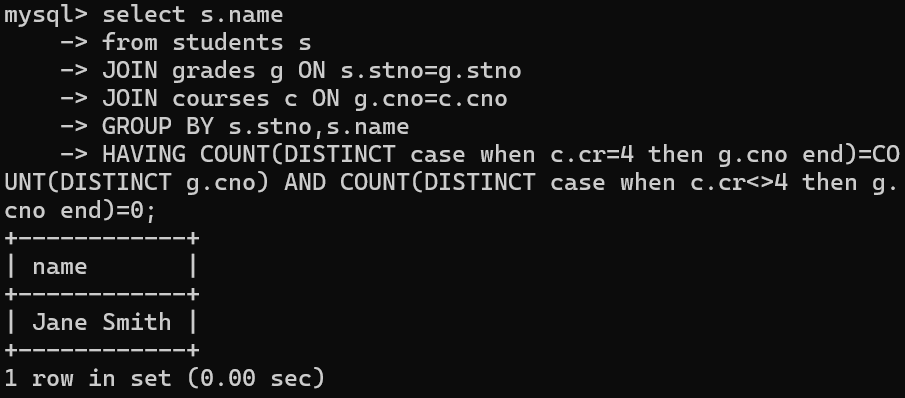
from students s

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;



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

select s.name

from students s

where NOT EXISTS(

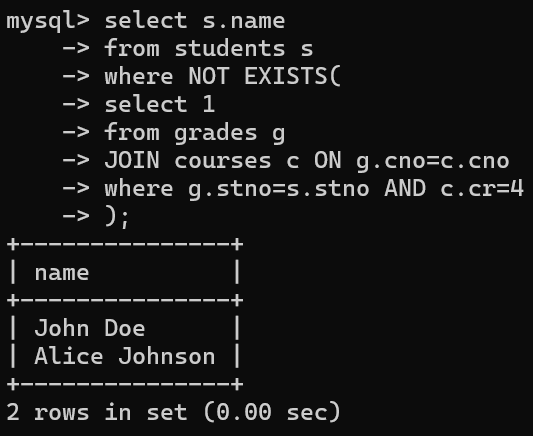
select 1

from grades g

JOIN courses c ON g.cno=c.cno

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

);



1. **Find the names of students who took cs210 or cs310.**

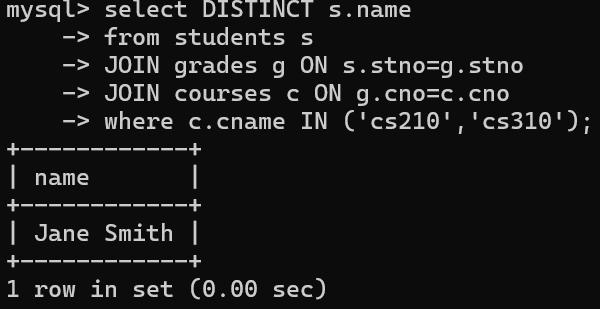
select DISTINCT s.name

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');



1. **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.**

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'

)

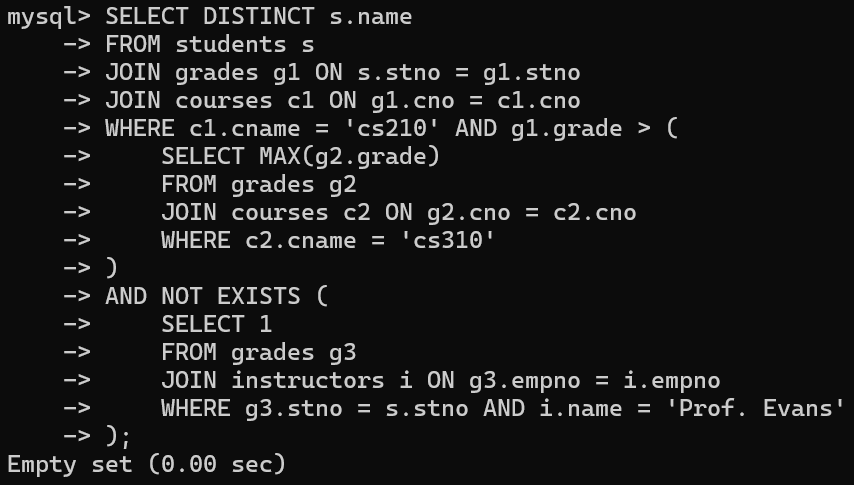
AND NOT EXISTS (

SELECT 1

FROM grades g3

JOIN instructors i ON g3.empno = i.empno

WHERE g3.stno = s.stno AND i.name = 'Prof. Evans'



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

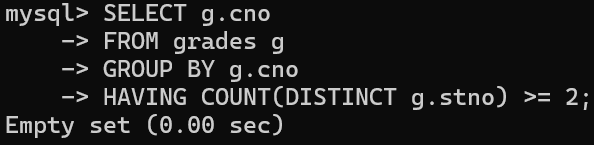
**For courses with at least 2 students.**

SELECT g.cno

FROM grades g

GROUP BY g.cno

HAVING COUNT(DISTINCT g.stno) >= 2;



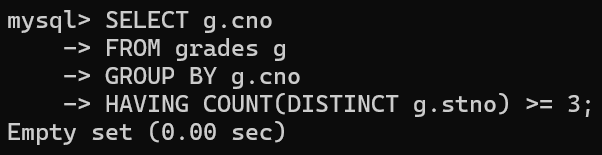
**For courses with at least 3 students.**

SELECT g.cno

-> FROM grades g

-> GROUP BY g.cno

-> HAVING COUNT(DISTINCT g.stno) >= 3;



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

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=(

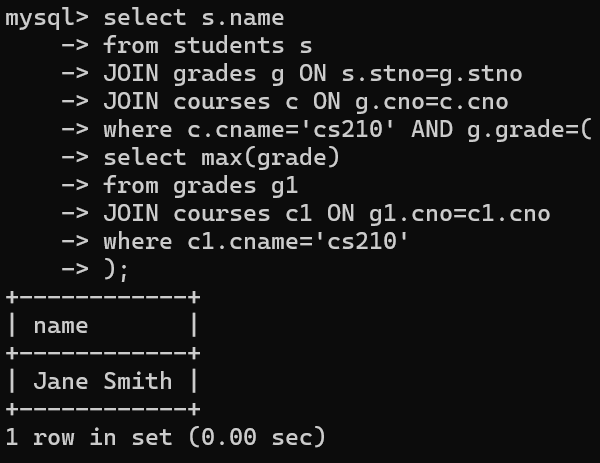
select max(grade)

from grades g1

JOIN courses c1 ON g1.cno=c1.cno

where c1.cname='cs210'

);



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

select DISTINCT i1.name

from instructors i1

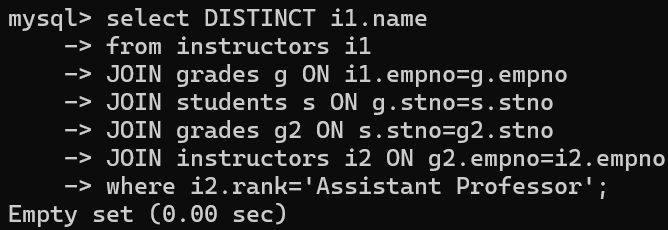
JOIN grades g ON i1.empno=g.empno

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';

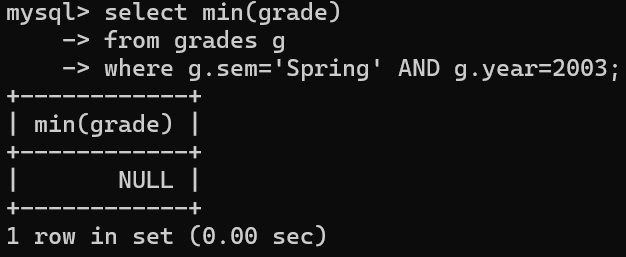


1. **Find the lowest grade of a student who took a course during the spring of 2003.**

select min(grade)

from grades g

where g.sem='Spring' AND g.year=2003;



1. **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).**

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

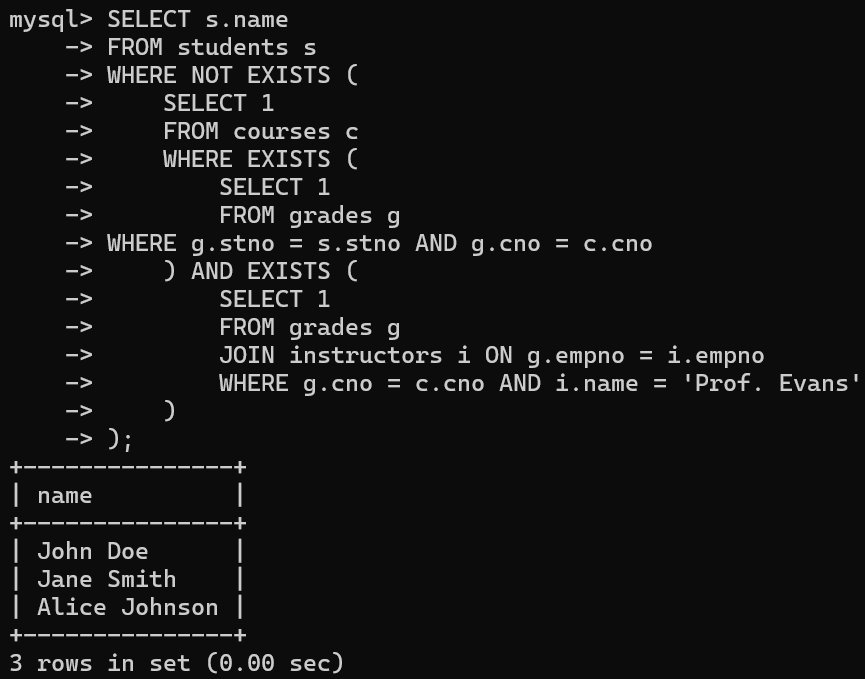
FROM grades g

JOIN instructors i ON g.empno = i.empno

WHERE g.cno = c.cno AND i.name = 'Prof. Evans'

)

);



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

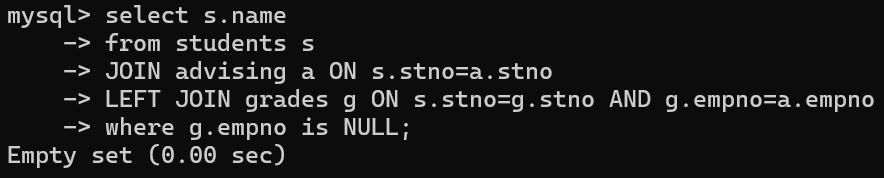
select s.name

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;



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

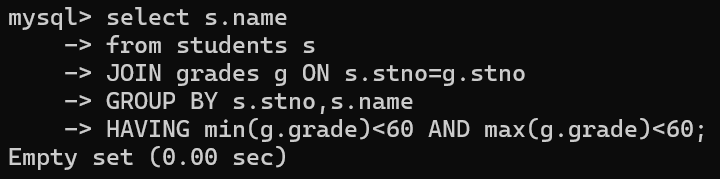
select s.name

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;



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

select max(g.grade)

from grades g

where g.stno NOT in(

select g2.stno

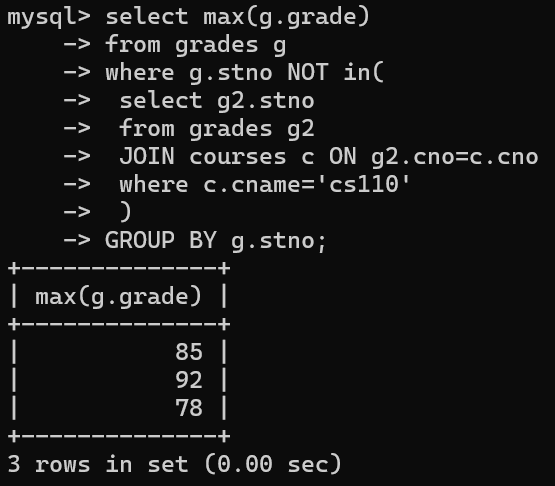
from grades g2

JOIN courses c ON g2.cno=c.cno

where c.cname='cs110'

)

GROUP BY g.stno;



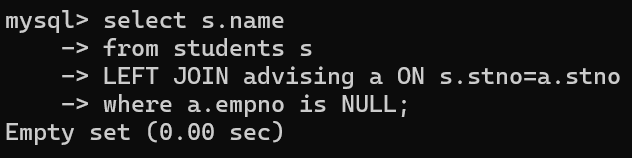
1. **Find the names of students who do not have an advisor.**

select s.name

from students s

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

where a.empno is NULL;



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

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';

