1. For each department, find the maximum salary of instructors in that department.

```
SELECT dept_name, MAX(salary) AS maximum_salary
FROM instructor
GROUP BY dept_name;
```

dept_name	maximum_salary
Biology	72000.00
Comp. Sci.	92000.00
Elec. Eng.	80000.00
Finance	90000.00
History	62000.00
Music	40000.00
Physics	95000.00

2. Find the IDs of all students who were taught by an instructor named Katz; make sure there are no duplicates in the result.

```
SELECT DISTINCT student.ID
FROM student
JOIN takes ON student.ID = takes.ID
JOIN teaches ON teaches.course_id = takes.course_id
JOIN instructor ON instructor.ID = teaches.ID
WHERE instructor.name = 'Katz';
```

ID
00128
12345
45678
54321
76543
98765

3. Find the ID and title of each course in Comp. Sci. that has had at least one section with afternoon hours (i.e., ends at or after 12:00). (You should eliminate duplicates if any.)

```
SELECT DISTINCT c.course_id, c.title
FROM course AS c
JOIN section AS s
ON c.course_id = s.course_id
JOIN time_slot AS t
ON t.time_slot_id = s.time_slot_id
WHERE c.dept_name = 'Comp. Sci.'
 AND t.start_hr >= 12;
```

course_id	title
CS-101	Intro. to Computer Science
CS-315	Robotics

4. Find the IDs and titles of all the courses that are (direct) prerequisite to the Robotics course.

```
SELECT c1.course_id AS prerequisite_id, c1.title AS prerequisite_title
FROM course AS c
JOIN prereq AS p
ON p.course_id = c.course_id
JOIN course AS c1
ON p.prereq_id = c1.course_id
WHERE c.title = 'Robotics';
```

prerequisite_id	prerequisisite_title
CS-101	Intro. to Computer Science

5. Find the IDs and names of all instructors earning the highest salary (there may be more than one with the same salary).

```
SELECT ID, name
FROM instructor
WHERE salary = (SELECT MAX(salary) FROM instructor);
```

ID	name
22222	Einstein
NULL	NULL

6. Find the enrollment (number of students) in each section that was offered in Spring 2017. The result columns should be course id, section id, students num. You do not need to output sections with 0 students.

```
SELECT sec.course_id, sec.sec_id, COUNT(*) AS student_num
FROM takes AS t
INNER JOIN section AS sec
ON t.course_id = sec.course_id AND t.sec_id = sec.sec_id
WHERE sec.year = 2017 AND sec.semester = 'Spring'
GROUP BY sec.course_id, sec.sec_id;
```

course_id	sec_id	student_num
CS-190	2	2
EE-181	1	1

7. Rewrite the preceding query, but also output sections with 0 students.

```
SELECT sec.course_id, sec.sec_id, COUNT(t.ID) AS student_num
FROM section AS sec
LEFT JOIN takes AS t
ON sec.course_id = t.course_id AND sec.sec_id = t.sec_id
WHERE sec.year = 2017 AND sec.semester = 'Spring'
GROUP BY sec.course_id, sec.sec_id;
```

course_id	sec_id	student_num
CS-190	1	0
CS-190	2	2
EE-181	1	1

8. Find the IDs and names of all instructors who have taught at least 3 different courses.

```
SELECT i.ID, i.name, COUNT(DISTINCT t.course_id) AS courses_taught
FROM instructor AS i
JOIN teaches AS t
ON i.ID = t.ID
GROUP BY i.ID, i.name
HAVING COUNT(DISTINCT t.course_id) >= 3;
```

ID	name	courses_taug...
10101	Srinivasan	3

9. Find the ID and name of the student with the highest number of 'A' grades (there may be more than one such student)

```
SELECT s.ID, s.name, COUNT(t.grade) AS a_grade_count

FROM student AS s

JOIN takes AS t ON s.ID = t.ID

WHERE t.grade = 'A'

GROUP BY s.ID, s.name

HAVING COUNT(t.grade) = (

  SELECT MAX(a_grade_count)
```

```
    FROM (
        SELECT COUNT(grade) AS a_grade_count
        FROM takes
        WHERE grade = 'A'
        GROUP BY ID
    ) AS subquery
);
```

ID	name	a_grade_count
12345	Shankar	3

10. Find the ID and name of each History student who has not taken any Music courses.

```
SELECT s.ID, s.name
FROM student AS s
WHERE s.dept_name = 'History'
AND NOT EXISTS (
    SELECT 1
    FROM takes AS t
    JOIN course AS c ON t.course_id = c.course_id
    WHERE s.ID = t.ID
    AND c.dept_name = 'Music'
);
```

ID	name
19991	Brandt

11. Find the ID and name of each instructor who has never given an ’A’ grade in any course she or he has taught. (Instructors who have never taught a course trivially satisfy this condition.)

```
SELECT DISTINCT i.ID, i.name

FROM instructor AS i

LEFT JOIN teaches AS t ON i.ID = t.ID

LEFT JOIN takes AS tk ON t.course_id = tk.course_id AND t.sec_id = tk.sec_id

WHERE i.ID NOT IN (

  SELECT t.ID

  FROM teaches AS t

  JOIN takes AS tk ON t.course_id = tk.course_id AND t.sec_id = tk.sec_id

  WHERE tk.grade = 'A' OR tk.grade = 'A-'

);
```

ID	name
12121	Wu
22222	Einstein
32343	El Said
33456	Gold
58583	Califieri
76543	Singh
98345	Kim

12. Rewrite the preceding query, but also ensure that you include only instructors who have given at least one other non-null grade in some course.

```
SELECT DISTINCT i.ID, i.name

FROM instructor AS i

LEFT JOIN teaches AS t ON i.ID = t.ID
```

```
LEFT JOIN takes AS tk ON t.course_id = tk.course_id AND t.sec_id = tk.sec_id
WHERE i.ID NOT IN (
   SELECT t.ID
   FROM teaches AS t
   JOIN takes AS tk ON t.course_id = tk.course_id AND t.sec_id = tk.sec_id
   WHERE tk.grade = 'A' OR tk.grade = 'A-'
)
AND i.ID IN (
   SELECT t.ID
   FROM teaches AS t
   JOIN takes AS tk ON t.course_id = tk.course_id AND t.sec_id = tk.sec_id
   WHERE tk.grade IS NOT NULL
);
```

ID	name
32343	El Said
12121	Wu
22222	Einstein
98345	Kim

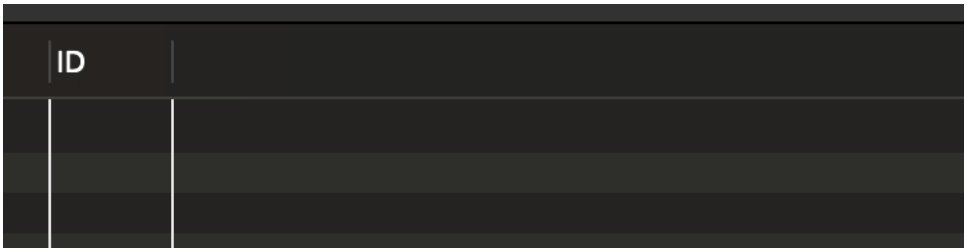
13. For each student who have retaken a course at least once (i.e., the student has taken the course at least twice), show the student's ID, name and the course ID.

```
SELECT s.ID, s.name, t.course_id
FROM student AS s
JOIN takes AS t ON s.ID = t.ID
GROUP BY s.ID, s.name, t.course_id
HAVING COUNT(t.course_id) >= 2;
```

ID	name	course_id
45678	Levy	CS-101

14. Find the IDs of those students who have retaken at least three distinct courses at least once (i.e., the student has taken the course at least twice)

```
SELECT t.ID

FROM takes AS t

GROUP BY t.ID

HAVING COUNT(DISTINCT t.course_id) >= 3 AND SUM(CASE WHEN t.course_id IS NOT NULL THEN 1 ELSE 0 END) >= 6;
```



15. Find the IDs and names of those instructors who have taught every course in their department.

```
SELECT i.ID, i.name

FROM instructor AS i

JOIN teaches AS t ON i.ID = t.ID

JOIN course AS c ON t.course_id = c.course_id

WHERE i.dept_name = c.dept_name

GROUP BY i.ID, i.name

HAVING COUNT(DISTINCT c.course_id) = (

  SELECT COUNT(*)

  FROM course AS c2
```

```
    WHERE c2.dept_name = i.dept_name

);
```

ID	name
12121	Wu
15151	Mozart
22222	Einstein
32343	El Said
98345	Kim

1. Create a new course ”CS-001” in the Comp. Sci. department, titled ”Weekly Seminar”, with 2 credits.

```
insert into course values ('CS-001', 'Weekly Seminar', 'Comp. Sci.', '2');
```

```
479  11:56:44  insert into course values ('CS-001', 'Weekly Semi...  1 row(s) affected
```

2. Create a section of this course in Spring 2022, with sec id of 1, and with the location of this section not yet specified.

```
insert into section values ('CS-001', '1', 'Spring', '2022', NULL, NULL, NULL);
```

```
480  11:57:20  insert into section values ('CS-001', '1', 'Spring', '...  1 row(s) affected
```

3. Enroll every student in the Comp. Sci. department in the above section (Hint: use INSERT INTO SELECT statement).

```
INSERT INTO takes (ID, course_id, sec_id, semester, year)
SELECT s.ID, 'CS-001', '1', 'Spring', '2022'
FROM student AS s
WHERE s.dept_name = 'Comp. Sci.'
```

```
481  12:02:38  INSERT INTO takes (ID, course_id, sec_id, seme...  4 row(s) affected Records: 4  Duplicates: 0  Warnings: 0
```

4. Delete enrollments in the above section where the student’s ID is 12345.

```
DELETE from takes
WHERE ID = '12345'
    AND course_id = 'CS-001'
    AND sec_id = '1'
    And semester = 'Spring'
    And year = '2022';
```

```
482  12:03:18  DELETE from takes WHERE ID = '12345'  AND co...  1 row(s) affected
```

5.Delete the course CS-001. What happened to the section and enrollments of this course?

```
DELETE FROM course
WHERE course_id = 'CS-001';
```

```
483  12:03:49  DELETE FROM course WHERE course_id = 'CS-0...  1 row(s) affected
```

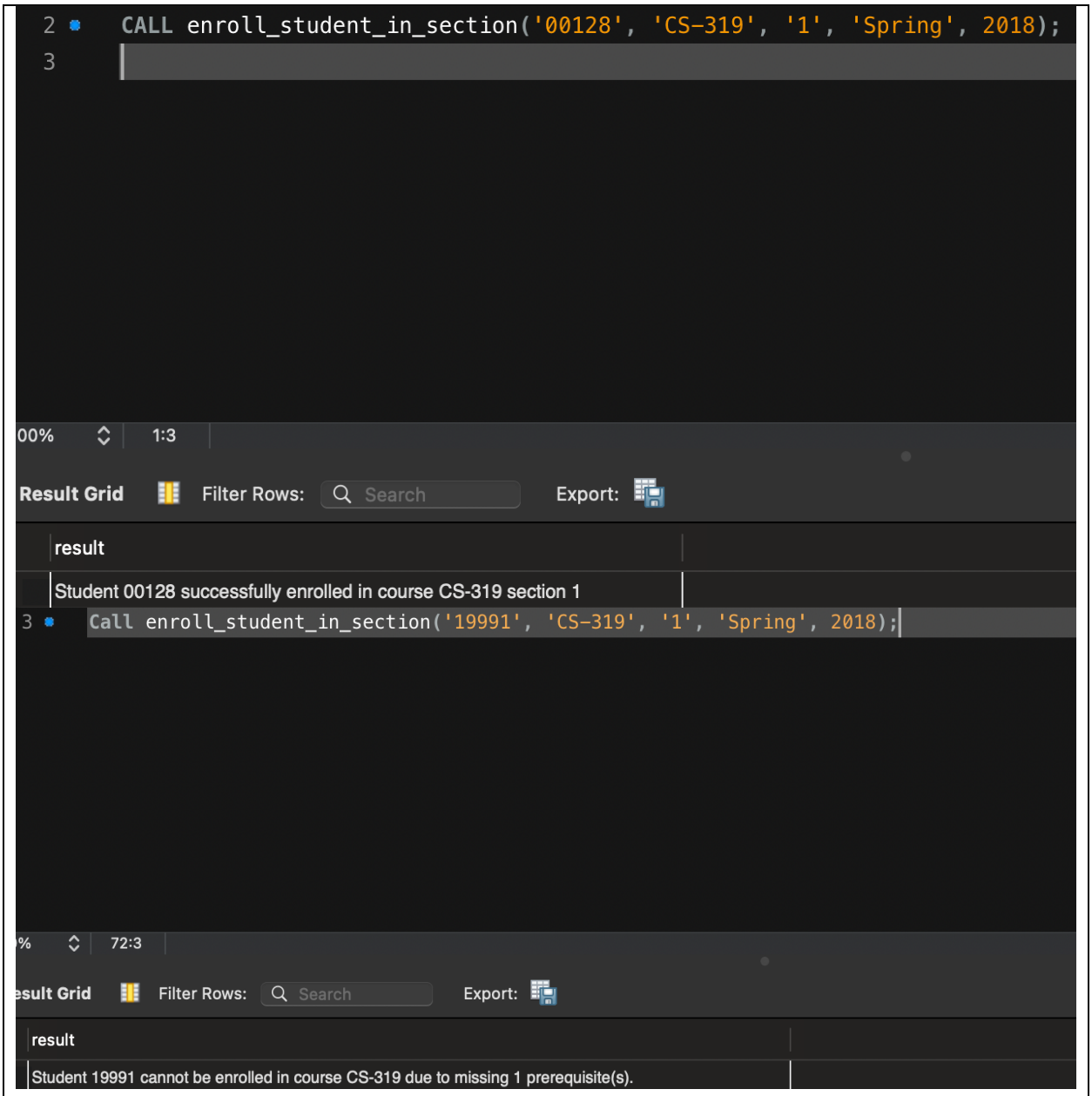
Since constraint on the course table have ON DELETE CASCADE constraint, the enrollment is deleted automatically with the deletion of the CS-001.

3. Create a stored procedure that enrolls a student into a section of a course, but only if the student has completed the prerequisites for that course. If the prerequisites are not met, the procedure should return a message indicating that the student cannot be enrolled.
Call the procedure for a student who meets the prerequisites and a student who does not meet the prerequisites, and show the output in each case.

```
DELIMITER //

CREATE PROCEDURE enroll_student_in_section (
   IN input_student_id VARCHAR(5),
   IN input_course_id VARCHAR(8),
   IN input_sec_id VARCHAR(8),
   IN input_semester VARCHAR(6),
   IN input_year DECIMAL(4,0)
)
BEGIN
   DECLARE prereq_count INT;
   DECLARE completed_count INT;
   DECLARE prereq_missing INT;
   DECLARE already_enrolled INT;


   SELECT COUNT(*) INTO already_enrolled
   FROM takes
   WHERE ID = input_student_id
     AND course_id = input_course_id
     AND sec_id = input_sec_id
     AND semester = input_semester
     AND year = input_year;


   IF already_enrolled > 0 THEN
      SELECT CONCAT('Student ', input_student_id, ' is already enrolled in course ',
input_course_id, ' section ', input_sec_id) AS result;
   ELSE

      SELECT COUNT(*) INTO prereq_count
      FROM prereq
      WHERE course_id = input_course_id;
```

```
    SELECT COUNT(*) INTO completed_count
    FROM prereq AS p
    JOIN takes AS t ON p.prereq_id = t.course_id
    WHERE t.ID = input_student_id
      AND t.grade IS NOT NULL
      AND t.grade <> 'F'
      AND p.course_id = input_course_id;


    IF completed_count = prereq_count THEN

      INSERT INTO takes (ID, course_id, sec_id, semester, year, grade)
      VALUES (input_student_id, input_course_id, input_sec_id, input_semester,
input_year, NULL);


      SELECT CONCAT('Student ', input_student_id, ' successfully enrolled in course ',
input_course_id, ' section ', input_sec_id) AS result;
    ELSE

      SET prereq_missing = prereq_count - completed_count;
      SELECT CONCAT('Student ', input_student_id, ' cannot be enrolled in course ',
input_course_id, ' due to missing ', prereq_missing, ' prerequisite(s).') AS result;
    END IF;
  END IF;

END //

DELIMITER ;
```

```
485  12:04:56    CREATE PROCEDURE enroll_student_in_section...  0 row(s) affected
```


2 CALL enroll_student_in_section('00128', 'CS-319', '1', 'Spring', 2018);
3
00%
1:3
Result Grid
Filter Rows:
Search
Export:
result
Student 00128 successfully enrolled in course CS-319 section 1
3 Call enroll_student_in_section('19991', 'CS-319', '1', 'Spring', 2018);
%
72:3
esult Grid
Filter Rows:
Search
Export:
result
Student 19991 cannot be enrolled in course CS-319 due to missing 1 prerequisite(s).

4.Write SQL DDL commands to create a database for an insurance company according to the following schemas:
person(driver id, name, address)
car(license plate, model, year)
owns(driver id, license plate)
accident(report number, date, location)
participated(report number, license plate, driver id, damage amount)
Make any reasonable assumptions about data types, and be sure to declare primary and foreign keys.

```
CREATE TABLE person (
   driver_id INT PRIMARY KEY,
   name VARCHAR(50) NOT NULL,
   address VARCHAR(150)
);

CREATE TABLE car (
   license_plate VARCHAR(15) PRIMARY KEY,
   model VARCHAR(50) NOT NULL,
   year INT CHECK (year >= 1886)
);

CREATE TABLE owns (
   driver_id INT,
   license_plate VARCHAR(15),
   PRIMARY KEY (driver_id, license_plate),
   FOREIGN KEY (driver_id) REFERENCES person(driver_id) ON DELETE CASCADE,
   FOREIGN KEY (license_plate) REFERENCES car(license_plate) ON DELETE CASCADE
);


CREATE TABLE accident (
   report_number INT PRIMARY KEY,
   date DATE NOT NULL,
   location VARCHAR(150) NOT NULL
);


CREATE TABLE participated (
   report_number INT,
   license_plate VARCHAR(15),
```

```
  driver_id INT,
  damage_amount DECIMAL(10, 2),
  PRIMARY KEY (report_number, license_plate, driver_id),
  FOREIGN KEY (report_number) REFERENCES accident(report_number) ON DELETE
CASCADE,
  FOREIGN KEY (license_plate) REFERENCES car(license_plate) ON DELETE
CASCADE,
  FOREIGN KEY (driver_id) REFERENCES person(driver_id) ON DELETE CASCADE
);
```

```sql
CREATE TABLE person (
    driver_id INT PRIMARY KEY,
    name VARCHAR(50) NOT NULL,
    address VARCHAR(150)
);

CREATE TABLE car (
    license_plate VARCHAR(15) PRIMARY KEY,
    model VARCHAR(50) NOT NULL,
    year INT CHECK (year >= 1886)
);

CREATE TABLE owns (
    driver_id INT,
    license_plate VARCHAR(15),
    PRIMARY KEY (driver_id, license_plate),
    FOREIGN KEY (driver_id) REFERENCES person(driver_id) ON DELETE CASCADE,
    FOREIGN KEY (license_plate) REFERENCES car(license_plate) ON DELETE CASCADE
);
CREATE TABLE accident (
    report_number INT PRIMARY KEY,
    date DATE NOT NULL,
    location VARCHAR(150) NOT NULL
);


CREATE TABLE participated (
    report_number INT,
    license_plate VARCHAR(15),
    driver_id INT,
    damage_amount DECIMAL(10, 2),
    PRIMARY KEY (report_number, license_plate, driver_id),
    FOREIGN KEY (report_number) REFERENCES accident(report_number) ON DELETE CASCADE,
    FOREIGN KEY (license_plate) REFERENCES car(license_plate) ON DELETE CASCADE,
    FOREIGN KEY (driver_id) REFERENCES person(driver_id) ON DELETE CASCADE
);
```

```
490  12:05:52  CREATE TABLE participated (  report_number I...  0 row(s) affected
```