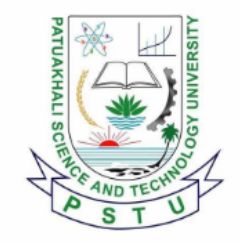


Patuakhali Science and Technology University
Faculty of Computer Science and Engineering

CCE 224 :: Database System Sessional

Sessional Project Report

Title : Full lab report
Submission Date : Sat 15, June 2025

Submitted to,

Prof. Dr. Md Samsuzzaman
Professor,
Department of Computer and Communication Engineering,
Patuakhali Science and Technology University.

Submitted by,

Md. Sharafat Karim
ID : 2102024,
Reg: 10151

Contents

Lab Problems

1. Problem set 1

1. Find the names of those departments whose budget is higher than that of Astronomy. List them in alphabetic order.

```
SELECT
    dept_name
FROM
    department
WHERE
    budget > (SELECT
            budget
        FROM
            department
        WHERE
            dept_name = 'Astronomy');
```

```
# dept_name
'Athletics'
'Biology'
'Cybernetics'
'Finance'
'History'
'Math'
'Physics'
'Psychology'
```

2. Display a list of all instructors, showing each instructor's ID and the number of sections taught. Make sure to show the number of sections as 0 for instructors who have not taught any section.

```
SELECT
    I.ID, COUNT(T.ID) as number_of_sections
FROM
    instructor AS I
        NATURAL LEFT JOIN
    teaches AS T
GROUP BY I.ID
ORDER BY number_of_sections;
```

```
# ID, number_of_sections
'35579', '0'
'52647', '0'
'50885', '0'
'57180', '0'
'58558', '0'
'59795', '0'
'63395', '0'
'64871', '0'
'72553', '0'
'4034', '0'
'37687', '0'
'74426', '0'
'78699', '0'
'79653', '0'
'31955', '0'
```

```
'95030', '0'
'96895', '0'
'16807', '0'
'97302', '0'
'15347', '1'
'73623', '1'
'65931', '1'
'80759', '1'
'90376', '1'
'90643', '1'
'48570', '1'
'25946', '1'
'50330', '1'
'4233', '1'
'42782', '1'
'48507', '1'
'14365', '2'
'63287', '2'
'3335', '2'
'28400', '2'
'81991', '2'
'28097', '2'
'41930', '3'
'19368', '3'
'34175', '3'
'43779', '4'
'95709', '4'
'3199', '4'
'36897', '5'
'77346', '6'
'79081', '6'
'74420', '6'
'99052', '9'
'6569', '10'
'22591', '13'
```

3. For each student who has retaken a course at least twice (i.e., the student has taken the course at least three times), show the course ID and the student's ID. Please display your results in order of course ID and do not display duplicate rows.

```
select distinct course_id, ID
  from takes
    group by ID, course_id having count(*) > 2
    order by course_id;
```

```
# course_id, ID
'362', '16480'
'362', '16969'
'362', '27236'
'362', '39925'
'362', '39978'
'362', '44881'
'362', '49611'
'362', '5414'
'362', '69581'
'362', '9993'
```

4. Find the names of Biology students who have taken at least 3 Accounting courses.

```
select name
from student natural join (
  select ID from takes
  where course_id in ( select course_id from course
    where dept_name = "Accounting")
    group by ID having count(*) > 2
    ) as T where dept_name = "Biology";
```

```
SELECT s.name
FROM student s
JOIN takes t ON s.ID = t.ID
JOIN course c ON t.course_id = c.course_id
WHERE s.dept_name = 'Biology'
AND c.dept_name = 'Accounting'
GROUP BY s.ID
HAVING COUNT(*) > 2;
```

```
# name
'Michael'
'Dalton'
'Shoji'
'Wehen'
'Uchiyama'
'Schill'
'Kaminsky'
'Giannoulis'
```

5. Find the sections that had maximum enrollment in Fall 2010.

```
SELECT course_id, sec_id
FROM takes
WHERE semester = 'Fall' AND year = 2010
GROUP BY course_id, sec_id
HAVING COUNT(ID) = (
    SELECT MAX(enrollment_count)
    FROM (
        SELECT COUNT(ID) AS enrollment_count
        FROM takes
        WHERE semester = 'Fall' AND year = 2010
        GROUP BY course_id, sec_id
    ) AS subquery
);
```

```
select course_id, sec_id
from takes
WHERE semester = 'Fall' AND year = 2010
group by course_id, sec_id
order by count(*) desc
limit 1;
```

```
# course_id, sec_id
'867', '2'
```

6. Find student names and the number of law courses taken for students who have taken at least half of the available law courses. (These courses are named things like 'Tort Law' or 'Environmental Law').

```sql
select name, count(*)
from student as st
natural join takes as tt
where tt.course_id in (
  select course_id
    from course
    where title like "%Law%"
)
group by st.ID
having count(*) > (
  select count(*)/2
    from course
    where title like "%Law%"
);
```

```
# name, count(*)
'Nakajima', '4'
'Nikut', '4'
'Hahn-', '4'
'Nanda', '4'
'Schinag', '4'
```

7. Find the rank and name of the 10 students who earned the most A grades (A-, A, A+). Use alphabetical order by name to break ties. Note: the browser SQLite does not support window functions.

```sql
select row_number() over () as rnk, P.name from (
  select name, count(*) as cnt
  from student S
  join takes T on S.ID = T.ID
  where T.grade in ("A+", "A", "A-")
  group by T.ID
    ) as P
order by P.cnt desc, P.name
limit 10;

select row_number() over(order by count(*) desc, name) as rnk, name
from student st
natural join takes tt
where tt.grade in ("A+", "A", "A-")
group by ID
order by count(*) desc, name
limit 10;
```

```
# rnk, name
'1', 'Lepp'
'2', 'Eller'
'3', 'Masri'
'4', 'Vries'
'5', 'Åström'
'6', 'Gandhi'
'7', 'Greene'
'8', 'Haigh'
'9', 'McCarter'
'10', 'Sanchez'
```

2. Problem set 2

1. Find out the ID and salary of the instructors.

```
select ID, salary from instructor;
```

```
-- small database
# ID, salary
'10101', '65000.00'
'12121', '90000.00'
'15151', '40000.00'
'22222', '95000.00'
'32343', '60000.00'
'33456', '87000.00'
'45565', '75000.00'
'58583', '62000.00'
'76543', '80000.00'
'76766', '72000.00'
'83821', '92000.00'
'98345', '80000.00'

-- big database
# ID, salary
'14365', '32241.56'
'15347', '72140.88'
'16807', '98333.65'
'19368', '124651.41'
'22591', '59706.49'
'25946', '90891.69'
'28097', '35023.18'
'28400', '84982.92'
'31955', '71351.42'
'3199', '82534.37'
'3335', '80797.83'
'34175', '115469.11'
'35579', '62579.61'
'36897', '43770.36'
'37687', '104563.38'
'4034', '61387.56'
'41930', '50482.03'
'4233', '88791.45'
'42782', '34272.67'
'43779', '79070.08'
'48507', '107978.47'
'48570', '87549.80'
'50330', '108011.81'
'50885', '32570.50'
'52647', '87958.01'
'57180', '43966.29'
'58558', '66143.25'
'59795', '48803.38'
'63287', '103146.87'
'63395', '94333.99'
'64871', '45310.53'
'6569', '105311.38'
'65931', '79866.95'
'72553', '46397.59'
```

```
'73623', '90038.09'
'74420', '121141.99'
'74426', '106554.73'
'77346', '99382.59'
'78699', '59303.62'
'79081', '47307.10'
'79653', '89805.83'
'80759', '45538.32'
'81991', '77036.18'
'90376', '117836.50'
'90643', '57807.09'
'95030', '54805.11'
'95709', '118143.98'
'96895', '119921.41'
'97302', '51647.57'
'99052', '93348.83'
```

2. Find out the ID and salary of the instructor who gets more than $85,000.

```
select ID, salary from instructor
where salary > 85000;

-- small db
# ID, salary
'12121', '90000.00'
'22222', '95000.00'
'33456', '87000.00'
'83821', '92000.00'

-- big db
# ID, salary
'16807', '98333.65'
'19368', '124651.41'
'25946', '90891.69'
'34175', '115469.11'
'37687', '104563.38'
'4233', '88791.45'
'48507', '107978.47'
'48570', '87549.80'
'50330', '108011.81'
'52647', '87958.01'
'63287', '103146.87'
'63395', '94333.99'
'6569', '105311.38'
'73623', '90038.09'
'74420', '121141.99'
'74426', '106554.73'
'77346', '99382.59'
'79653', '89805.83'
'90376', '117836.50'
'95709', '118143.98'
'96895', '119921.41'
'99052', '93348.83'
```

3. Find out the department names and their budget at the university.

```
select dept_name, budget from department;
```

```
-- small db
# dept_name, budget
'Biology', '90000.00'
'Comp. Sci.', '100000.00'
'Elec. Eng.', '85000.00'
'Finance', '120000.00'
'History', '50000.00'
'Music', '80000.00'
'Physics', '70000.00'

-- big db
# dept_name, budget
'Accounting', '441840.92'
'Astronomy', '617253.94'
'Athletics', '734550.70'
'Biology', '647610.55'
'Civil Eng.', '255041.46'
'Comp. Sci.', '106378.69'
'Cybernetics', '794541.46'
'Elec. Eng.', '276527.61'
'English', '611042.66'
'Finance', '866831.75'
'Geology', '406557.93'
'History', '699140.86'
'Languages', '601283.60'
'Marketing', '210627.58'
'Math', '777605.11'
'Mech. Eng.', '520350.65'
'Physics', '942162.76'
'Pol. Sci.', '573745.09'
'Psychology', '848175.04'
'Statistics', '395051.74'
```

4. List out the names of the instructors from Computer Science who have more than $70,000.

```
desc instructor;
select name from instructor
where salary > 70000;

-- small db
# name
'Wu'
'Einstein'
'Gold'
'Katz'
'Singh'
'Crick'
'Brandt'
'Kim'

-- big db
# name
'Bawa'
'Yazdi'
'Wieland'
'Liley'
'Atanassov'
```

```
'Moreira'
'Gustafsson'
'Bourrier'
'Bondi'
'Arias'
'Luo'
'Romero'
'Lent'
'Sarkar'
'Shuming'
'Bancilhon'
'Jaekel'
'McKinnon'
'Mingoz'
'Pimenta'
'Sullivan'
'Voronina'
'Kenje'
'Mahmoud'
'Levine'
'Valtchev'
'Bietzk'
'Sakurai'
'Mird'
'Dale'
```

5. For all instructors in the university who have taught some course, find their names and the course ID of

all courses they taught.

```
select I.name, T.course_id
from instructor I
natural join teaches T
order by I.name;

-- small db
# name, course_id
'Brandt', 'CS-190'
'Brandt', 'CS-190'
'Brandt', 'CS-319'
'Crick', 'BIO-101'
'Crick', 'BIO-301'
'Einstein', 'PHY-101'
'El Said', 'HIS-351'
'Katz', 'CS-101'
'Katz', 'CS-319'
'Kim', 'EE-181'
'Mozart', 'MU-199'
'Srinivasan', 'CS-101'
'Srinivasan', 'CS-315'
'Srinivasan', 'CS-347'
'Wu', 'FIN-201'

-- large db
# name, course_id
'Atanassov', '603'
```

```
'Atanassov', '604'
'Bawa', '457'
'Bietzk', '158'
'Bondi', '571'
'Bondi', '274'
'Bondi', '747'
'Bourrier', '960'
'Bourrier', '949'
'Choll', '461'
'DAgostino', '663'
'DAgostino', '338'
'DAgostino', '338'
'DAgostino', '352'
'DAgostino', '400'
'DAgostino', '400'
'DAgostino', '991'
'DAgostino', '642'
'DAgostino', '599'
'DAgostino', '482'
'DAgostino', '962'
'DAgostino', '972'
'DAgostino', '867'
'Dale', '893'
'Dale', '237'
'Dale', '629'
'Dale', '237'
'Dale', '927'
'Dale', '748'
'Dale', '802'
'Dale', '158'
'Dale', '496'
'Gustafsson', '169'
'Gustafsson', '169'
'Gustafsson', '631'
'Gustafsson', '561'
'Jaekel', '852'
'Jaekel', '334'
'Kean', '366'
'Kean', '808'
'Lembr', '200'
'Lembr', '843'
'Lent', '626'
'Liley', '192'
'Luo', '679'
'Mahmoud', '704'
'Mahmoud', '735'
'Mahmoud', '735'
'Mahmoud', '493'
'Mahmoud', '864'
'Mahmoud', '486'
'Mingoz', '362'
'Mingoz', '527'
'Mingoz', '137'
'Mingoz', '362'
'Mingoz', '426'
'Mingoz', '304'
```

```
'Mingoz', '319'
'Mingoz', '445'
'Mingoz', '349'
'Mingoz', '362'
'Morris', '696'
'Morris', '791'
'Morris', '795'
'Morris', '313'
'Morris', '242'
'Pimenta', '875'
'Queiroz', '559'
'Romero', '105'
'Romero', '489'
'Romero', '105'
'Romero', '476'
'Sakurai', '468'
'Sakurai', '960'
'Sakurai', '258'
'Sakurai', '270'
'Sarkar', '867'
'Shuming', '468'
'Sullivan', '694'
'Tung', '692'
'Tung', '401'
'Tung', '421'
'Ullman ', '408'
'Ullman ', '974'
'Ullman ', '345'
'Ullman ', '200'
'Ullman ', '760'
'Ullman ', '408'
'Valtchev', '702'
'Valtchev', '415'
'Vicentino', '793'
'Voronina', '376'
'Voronina', '239'
'Voronina', '959'
'Voronina', '443'
'Voronina', '612'
'Voronina', '443'
'Wieland', '581'
'Wieland', '591'
'Wieland', '545'
```

6. Find the names of all instructors whose salary is greater than at least one instructor in the Biology department.

```
select name
from instructor
where dept_name = "Biology"
and salary > ( select min(S.salary) from (
  select salary
    from instructor
    where dept_name = "Biology"
) as S );

-- big db
```

```
# name
'Valtchev'
```

7. Find the advisor of the student with ID 12345

```
select * from advisor
where s_ID = 12345;

-- small db
# s_ID, i_ID
'12345', '10101'
```

8. Find the average salary of all instructors.

```
select avg(I.salary) average_salary from
(select salary from instructor) as I;

-- small db
# average_salary
'74833.333333'

-- big db
# average_salary
'77600.188200'
```

9. Find the names of all departments whose building name includes the substring 'Watson'.

```
select dept_name from department
where building like "%Watson%";

-- small db
# dept_name
'Biology'
'Physics'
```

10. Find the names of instructors with salary amounts between $90,000 and $100,000.

```
select name from instructor
where salary between 90000 and 100000;

-- small db
# name
'Wu'
'Einstein'
'Brandt'

-- large db
# name
'Yazdi'
'Liley'
'McKinnon'
'Sullivan'
'Mahmoud'
'Dale'
```

11. Find the instructor names and the courses they taught for all instructors in the Biology department who

have taught some course.

```
select name, course_id
from instructor
natural left join teaches
where dept_name = "Biology";

-- small db
# name, course_id
'Crick', 'BIO-101'
'Crick', 'BIO-301'

-- large db
# name, course_id
'Queiroz', '559'
'Valtchev', '415'
'Valtchev', '702'
```

12. Find the courses taught in Fall-2009 semester.

```
select course_id from teaches
where semester = "Fall" and year = 2009;

-- big db
# course_id
'105'
'237'
'242'
'304'
'334'
'486'
'960'
```

13. Find the set of all courses taught either in Fall-2009 or in Spring-2010.

```
select course_id from teaches
where (semester = "Fall" and year = 2009) or (semester = "Spring" and year = 2010);

( select course_id from teaches
where semester = "Fall" and year = 2009 )
union ( select course_id from teaches
where semester = "Spring" and year = 2010 );

-- big db
# course_id
'105'
'237'
'242'
'270'
'304'
'334'
'443'
'486'
'493'
'679'
'692'
'735'
'960'
```

14. Find the set of all courses taught in the Fall-2009 as well as in Spring-2010.

```
select course_id from teaches
where (semester = "Fall" and year = 2009) and (semester = "Spring" and year = 2010);

( select course_id from teaches
where semester = "Fall" and year = 2009 )
intersect ( select course_id from teaches
where semester = "Spring" and year = 2010 );
```

15. Find all courses taught in the Fall-2009 semester but not in the Spring-2010 semester.

```
select course_id from teaches
where (semester = "Fall" and year = 2009) and not (semester = "Spring" and year =
2010);

( select course_id from teaches
where semester = "Fall" and year = 2009 )
except ( select course_id from teaches
where semester = "Spring" and year = 2010 );

-- big db
# course_id
'105'
'237'
'242'
'304'
'334'
'486'
'960'
```

16. Find all instructors who appear in the instructor relation with null values for salary.

```
select * from instructor
where salary = NULL;
```

17. Find the average salary of instructors in the Finance department.

```
select avg(T.salary) from
( select salary from instructor
where dept_name = "Finance" ) as T;

-- small db
# avg(T.salary)
'85000.000000'

-- big db
# avg(T.salary)
'105311.380000'
```

18. Find the total number of instructors who teach a course in the Spring-2010 semester.

```
select  count(T.id) from
( select id from instructor
  natural join teaches
    where semester = "Spring"
    and year = 2010 ) as T ;

-- big db
# count(T.id)
'6'
```

19. Find the average salary in each department.

```
select dept_name, avg(salary)
from department
natural join instructor
group by dept_name;

-- small db
# dept_name, avg(salary)
'Biology', '72000.000000'
'Comp. Sci.', '77333.333333'
'Elec. Eng.', '80000.000000'
'Finance', '85000.000000'
'History', '61000.000000'
'Music', '40000.000000'
'Physics', '91000.000000'

-- big db
# dept_name, avg(salary)
'Accounting', '48716.592500'
'Athletics', '77098.198000'
'Pol. Sci.', '100053.073333'
'Psychology', '61143.050000'
'Languages', '57421.856667'
'English', '72089.050000'
'Statistics', '67795.441667'
'Elec. Eng.', '74162.740000'
'Comp. Sci.', '98133.470000'
'Marketing', '84097.437500'
'Astronomy', '79070.080000'
'Mech. Eng.', '79813.020000'
'Physics', '114576.900000'
'Cybernetics', '96346.567500'
'Finance', '105311.380000'
'Geology', '99382.590000'
'Biology', '61287.250000'
```

20. Find the number of instructors in each department who teach a course in the Spring-2010 semester.

```
select dept_name, count(ID)
from department
natural join instructor
where ID in (
  select ID from teaches
    where semester = "Spring"
    and year = 2010
)
group by dept_name;

--big db
# dept_name, count(ID)
'Athletics', '1'
'English', '2'
'Physics', '1'
'Geology', '1'
```

21. List out the departments where the average salary of the instructors is more than $42,000.

```sql
select dept_name
from department D
where ( select avg(salary) from (
  select salary
    from instructor I
    where D.dept_name = I.dept_name
) as T ) > 42000;

select distinct dept_name
from instructor
group by dept_name
having avg(salary) > 42000;

-- small db
# dept_name
'Biology'
'Comp. Sci.'
'Elec. Eng.'
'Finance'
'History'
'Physics'

-- big db
# dept_name
'Accounting'
'Astronomy'
'Athletics'
'Biology'
'Comp. Sci.'
'Cybernetics'
'Elec. Eng.'
'English'
'Finance'
'Geology'
'Languages'
'Marketing'
'Mech. Eng.'
'Physics'
'Pol. Sci.'
'Psychology'
'Statistics'
```

22. For each course section offered in 2009, find the average total credits (tot cred) of all students enrolled

in the section, if the section had at least 2 students.

```sql
select course_id, sec_id, avg(tot_cred)
from takes
natural join student
where year = 2009
group by sec_id, course_id
having count(*) > 1;

-- big db
# course_id, sec_id, avg(tot_cred)
'105', '1', '68.3578'
```

```
'237', '2', '65.6656'
'242', '1', '64.4576'
'304', '1', '64.9023'
'334', '1', '62.8806'
'486', '1', '64.8980'
'604', '1', '65.7233'
'960', '1', '66.0847'
'972', '1', '65.2607'
```

23. Find all the courses taught in both the Fall-2009 and Spring-2010 semesters.

```
select course_id from teaches
where (semester = "Fall" and year = 2009) and (semester = "Spring" and year = 2010);

( select course_id from teaches
where semester = "Fall" and year = 2009 )
intersect ( select course_id from teaches
where semester = "Spring" and year = 2010 );
```

24. Find all the courses taught in the Fall-2009 semester but not in the Spring-2010 semester.

```
select course_id from teaches
where (semester = "Fall" and year = 2009) and not (semester = "Spring" and year =
2010);

( select course_id from teaches
where semester = "Fall" and year = 2009 )
except ( select course_id from teaches
where semester = "Spring" and year = 2010 );

-- big db
# course_id
'105'
'237'
'242'
'304'
'334'
'486'
'960'
```

25. Select the names of instructors whose names are neither ‘Mozart’ nor ‘Einstein’.

```
select name from instructor
where name not in ("Mozart", "Einstein");

-- small db
# name
'Srinivasan'
'Wu'
'El Said'
'Gold'
'Katz'
'Califieri'
'Singh'
'Crick'
'Brandt'
'Kim'
```

26. Find the total number of (distinct) students who have taken course sections taught by the instructor

with ID 110011.

```
select count(T.ID)
from takes T
natural join section S
where "110011" in (
  select ID
    from teaches ST
    where T.course_id = ST.course_id
    and T.sec_id = ST.sec_id
    and T.semester = ST.semester
    and T.year = ST.year
);

SELECT COUNT(DISTINCT t.ID) AS total_students
FROM takes t
JOIN teaches te ON t.course_id = te.course_id
               AND t.sec_id = te.sec_id
               AND t.semester = te.semester
               AND t.year = te.year
WHERE te.ID = '110011';
```

27. Find the ID and names of all instructors whose salary is greater than at least one instructor in the History

department.

```
select ID, name
from instructor
where salary > ( select min(T.salary) from (
  select salary
    from instructor
    where dept_name = "History"
) as T );

-- small db
# ID, name
'10101', 'Srinivasan'
'12121', 'Wu'
'22222', 'Einstein'
'33456', 'Gold'
'45565', 'Katz'
'58583', 'Califieri'
'76543', 'Singh'
'76766', 'Crick'
'83821', 'Brandt'
'98345', 'Kim'
```

28. Find the names of all instructors that have a salary value greater than that of each instructor in the

Biology department.

```
select name
from instructor
where salary > ( select max(T.salary) from (
  select salary
    from instructor
    where dept_name = "Biology"
) as T );
```

```
-- small db
# name
'Wu'
'Einstein'
'Gold'
'Katz'
'Singh'
'Brandt'
'Kim'

-- big db
# name
'Yazdi'
'Wieland'
'Liley'
'Atanassov'
'Gustafsson'
'Bourrier'
'Bondi'
'Arias'
'Luo'
'Romero'
'Lent'
'Sarkar'
'Shuming'
'Bancilhon'
'Jaekel'
'McKinnon'
'Mingoz'
'Pimenta'
'Sullivan'
'Voronina'
'Kenje'
'Mahmoud'
'Levine'
'Bietzk'
'Sakurai'
'Mird'
'Dale'
```

29. Find the departments that have the highest average salary.

```
select dept_name
from instructor
group by dept_name
order by avg(salary) desc
limit 1;

-- small db
# dept_name
'Physics'

-- small db
# dept_name
'Physics'
```

30. Find all courses taught in both the Fall 2009 semester and in the Spring-2010 semester.

```
select course_id from teaches
where (semester = "Fall" and year = 2009) and (semester = "Spring" and year = 2010);

( select course_id from teaches
where semester = "Fall" and year = 2009 )
intersect ( select course_id from teaches
where semester = "Spring" and year = 2010 );
```

31. Find all students who have taken all the courses offered in the Biology department.

```
with cnt as (
  select count(course_id) as ct
    from course
    where dept_name = "Biology"
)
select ID
from takes
where course_id in (
  select course_id
    from course
    where dept_name = "Biology"
)
group by ID
having count(*) = (
  select ct from cnt
);
```

32. Find all courses that were offered at most once in 2009.

```
select course_id
from takes
where year = 2009
group by course_id
having count(*) = 1;
```

33. Find all courses that were offered at least twice in 2009.

```
select course_id
from takes
where year = 2009
group by course_id
having count(*) > 1;
```

34. Find the average instructors' salaries of those departments where the average salary is greater than $42,000.

```
select dept_name
from department D
where ( select avg(salary) from (
  select salary
    from instructor I
    where D.dept_name = I.dept_name
) as T ) > 42000;

select distinct dept_name
from instructor
group by dept_name
having avg(salary) > 42000;
```

```
-- small db
# dept_name
'Biology'
'Comp. Sci.'
'Elec. Eng.'
'Finance'
'History'
'Physics'

-- big db
# dept_name
'Accounting'
'Astronomy'
'Athletics'
'Biology'
'Comp. Sci.'
'Cybernetics'
'Elec. Eng.'
'English'
'Finance'
'Geology'
'Languages'
'Marketing'
'Mech. Eng.'
'Physics'
'Pol. Sci.'
'Psychology'
'Statistics'
```

35. Find the maximum across all departments of the total salary at each department.

```
select sum(salary)
from department
natural join instructor
group by dept_name
order by sum(salary) desc
limit 1;

SELECT MAX(total_salary) AS max_total_salary
FROM (
    SELECT dept_name, SUM(salary) AS total_salary
    FROM instructor
    GROUP BY dept_name
) AS dept_salaries;

-- small db
# max_total_salary
'232000.00'

-- big db
# max_total_salary
'406772.65'
```

36. List all departments along with the number of instructors in each department.

```
select dept_name, count(ID)
from department
natural left join instructor
```

```
group by dept_name;

-- small data
# dept_name, count(ID)
'Biology', '1'
'Comp. Sci.', '3'
'Elec. Eng.', '1'
'Finance', '2'
'History', '2'
'Music', '1'
'Physics', '2'

-- big data
# dept_name, count(ID)
'Accounting', '4'
'Astronomy', '1'
'Athletics', '5'
'Biology', '2'
'Civil Eng.', '0'
'Comp. Sci.', '2'
'Cybernetics', '4'
'Elec. Eng.', '4'
'English', '4'
'Finance', '1'
'Geology', '1'
'History', '0'
'Languages', '3'
'Marketing', '4'
'Math', '0'
'Mech. Eng.', '2'
'Physics', '2'
'Pol. Sci.', '3'
'Psychology', '2'
'Statistics', '6'
```

3. Problem set 3

1. Find the titles of courses in the Comp. Sci. department that have 3 credits.

```
select title
from course
where dept_name = "Comp. Sci."
and credits = 3;

-- small db
# title
'Robotics'
'Image Processing'
'Database System Concepts'

-- large db
# title
'International Finance'
'Computability Theory'
'Japanese'
```

2. Find the IDs of all students who were taught by an instructor named Einstein; make

sure there are no duplicates in the result.

```
select takes_table.ID
from takes takes_table
join teaches teaches_table
  on takes_table.course_id = teaches_table.course_id
    and takes_table.sec_id = teaches_table.sec_id
    and takes_table.semester = teaches_table.semester
    and takes_table.year = teaches_table.year
where teaches_table.ID = ( select ID from instructor
where name = "Einstein" ) ;

-- small db
# ID
'44553'
```

3. Find the ID and name of each student who has taken at least one Comp. Sci. course;

make sure there are no duplicate names in the result.

```
select distinct ID, name
from student
natural join takes
where takes.course_id in (
  select course_id
    from course
    where dept_name = "Comp. Sci."
);

-- small db
# ID, name
'00128', 'Zhang'
'12345', 'Shankar'
'45678', 'Levy'
'54321', 'Williams'
'76543', 'Brown'
'98765', 'Bourikas'

-- big db
-- big output
```

4. Find the course id, section id, and building for each section of a Biology course.

```
select course_id, sec_id, building
from section
natural join course
where dept_name = "Biology";

-- small db
# course_id, sec_id, building
'BIO-101', '1', 'Painter'
'BIO-301', '1', 'Painter'

-- big db
# course_id, sec_id, building
'415', '1', 'Lamberton'
'559', '1', 'Lamberton'
'702', '1', 'Saucon'
```

5. Output instructor names sorted by the ratio of their salary to their department's budget (in ascending order).

```
select name
from instructor
natural left join department
order by (salary/ budget) asc;
```

```
-- small db
# name
'Mozart'
'Srinivasan'
'Singh'
'Wu'
'Katz'
'Crick'
'Brandt'
'Kim'
'El Said'
'Califieri'
'Gold'
'Einstein'
```

```
-- big data
# name
'Konstantinides'
'Kean'
'Tung'
'Queiroz'
'DAgostino'
'Lembr'
'Soisalon-Soininen'
'Yin'
'Desyl'
'Murata'
'Bawa'
'Bertolino'
'Hau'
'Pimenta'
'Ullman '
'Shuming'
'Gutierrez'
'Dale'
'McKinnon'
'Valtchev'
'Mingoz'
'Vicentino'
'Romero'
'Voronina'
'Yazdi'
'Arinb'
'Jaekel'
'Luo'
'Choll'
'Bietzk'
'Pingr'
```

```
'Liley'
'Sarkar'
'Bancilhon'
'Moreira'
'Sakurai'
'Lent'
'Morris'
'Atanassov'
'Wieland'
'Mahmoud'
'Arias'
'Gustafsson'
'Dusserre'
'Levine'
'Sullivan'
'Kenje'
'Mird'
'Bourrier'
'Bondi'
```

6. Output instructor names and buildings for each building an instructor has taught in.

Include instructor names who have not taught any classes (the building name should be NULL in this case).

```
select name, building
from instructor
natural left join teaches
natural left join section;

-- small db
# name, building
'Srinivasan', 'Packard'
'Srinivasan', 'Watson'
'Srinivasan', 'Taylor'
'Wu', 'Packard'
'Mozart', 'Packard'
'Einstein', 'Watson'
'El Said', 'Painter'
'Gold', NULL
'Katz', 'Packard'
'Katz', 'Watson'
'Califieri', NULL
'Singh', NULL
'Crick', 'Painter'
'Crick', 'Painter'
'Brandt', 'Taylor'
'Brandt', 'Taylor'
'Brandt', 'Taylor'
'Kim', 'Taylor'

-- big db
# name, building
'Lembr', 'Saucon'
'Lembr', 'Fairchild'
'Bawa', 'Saucon'
'Yazdi', NULL
```

```
'Wieland', 'Saucon'
'Wieland', 'Alumni'
'Wieland', 'Saucon'
'DAgostino', 'Fairchild'
'DAgostino', 'Stabler'
'DAgostino', 'Lambeau'
'DAgostino', 'Lambeau'
'DAgostino', 'Main'
'DAgostino', 'Whitman'
'DAgostino', 'Chandler'
'DAgostino', 'Saucon'
'DAgostino', 'Fairchild'
'DAgostino', 'Taylor'
'DAgostino', 'Nassau'
'DAgostino', 'Taylor'
'DAgostino', 'Lamberton'
'Liley', 'Polya'
'Kean', 'Saucon'
'Kean', 'Polya'
'Atanassov', 'Taylor'
'Atanassov', 'Bronfman'
'Moreira', NULL
'Gustafsson', 'Gates'
'Gustafsson', 'Drown'
'Gustafsson', 'Main'
'Gustafsson', 'Taylor'
'Bourrier', 'Saucon'
'Bourrier', 'Lamberton'
'Bondi', 'Main'
'Bondi', 'Power'
'Bondi', 'Gates'
'Soisalon-Soininen', NULL
'Morris', 'Fairchild'
'Morris', 'Chandler'
'Morris', 'Saucon'
'Morris', 'Polya'
'Morris', 'Lamberton'
'Arias', NULL
'Murata', NULL
'Tung', 'Saucon'
'Tung', 'Gates'
'Tung', 'Taylor'
'Luo', 'Saucon'
'Vicentino', 'Nassau'
'Romero', 'Chandler'
'Romero', 'Taylor'
'Romero', 'Drown'
'Romero', 'Lamberton'
'Lent', 'Lamberton'
'Sarkar', 'Lamberton'
'Shuming', 'Power'
'Konstantinides', NULL
'Bancilhon', NULL
'Hau', NULL
'Dusserre', NULL
'Desyl', NULL
```

```
'Jaekel', 'Taylor'
'Jaekel', 'Gates'
'McKinnon', NULL
'Gutierrez', NULL
'Mingoz', 'Fairchild'
'Mingoz', 'Lamberton'
'Mingoz', 'Rathbone'
'Mingoz', 'Saucon'
'Mingoz', 'Lamberton'
'Mingoz', 'Alumni'
'Mingoz', 'Bronfman'
'Mingoz', 'Lamberton'
'Mingoz', 'Alumni'
'Mingoz', 'Saucon'
'Pimenta', 'Power'
'Yin', NULL
'Sullivan', 'Alumni'
'Voronina', 'Taylor'
'Voronina', 'Power'
'Voronina', 'Whitman'
'Voronina', 'Gates'
'Voronina', 'Lamberton'
'Voronina', 'Saucon'
'Kenje', NULL
'Mahmoud', 'Whitman'
'Mahmoud', 'Lamberton'
'Mahmoud', 'Taylor'
'Mahmoud', 'Drown'
'Mahmoud', 'Taylor'
'Mahmoud', 'Power'
'Pingr', NULL
'Ullman ', 'Chandler'
'Ullman ', 'Taylor'
'Ullman ', 'Taylor'
'Ullman ', 'Taylor'
'Ullman ', 'Garfield'
'Ullman ', 'Polya'
'Levine', NULL
'Queiroz', 'Lamberton'
'Valtchev', 'Lamberton'
'Valtchev', 'Saucon'
'Bietzk', 'Whitman'
'Cholll', 'Main'
'Arinb', NULL
'Sakurai', 'Main'
'Sakurai', 'Power'
'Sakurai', 'Lambeau'
'Sakurai', 'Power'
'Mird', NULL
'Bertolino', NULL
'Dale', 'Taylor'
'Dale', 'Power'
'Dale', 'Fairchild'
'Dale', 'Taylor'
'Dale', 'Stabler'
'Dale', 'Saucon'
```

```
'Dale', 'Saucon'
'Dale', 'Fairchild'
'Dale', 'Saucon'
```

4. 70 Queries (SQL Server)

```
--1. Find out the ID and salary of the instructors.
select ID, salary from instructor;

--2. Find out the ID and salary of the instructor who gets more than $85,000.
select ID, salary from instructor where salary > 85000;

--3. Find out the department names and their budget at the university.
select dept_name, budget from department;

--4. List out the names of the instructors from Computer Science who have more than
$70,000.
select name from instructor where dept_name='Comp. Sci.' and salary>70000;

--5. For all instructors in the university who have taught some course, find their
names and the course
--ID of all courses they taught.
select name, course_id
from instructor
left join teaches
on instructor.ID=teaches.ID;

--6. Find the names of all instructors whose salary is greater than at least one
instructor in the Biology
--department.
select name from instructor
where salary > (select min(salary)
  from instructor where dept_name='Biology');

--7. Find the advisor of the student with ID 12345
select i_id from advisor where s_ID=12345;

--8. Find the average salary of all instructors.
select avg(salary) from instructor;

--9. Find the names of all departments whose building name includes the substring
Watson.
select dept_name from department
where building like '%Watson%';

--10. Find the names of instructors with salary amounts between $90,000 and $100,000.
select name from instructor
where salary between 90000 and 100000;

--11. Find the instructor names and the courses they taught for all instructors in
the Biology
--department who have taught some course.
select name, course.title from instructor
join teaches on instructor.ID=teaches.ID
join course on teaches.course_id=course.course_id
where instructor.dept_name='Biology';
```

```sql
--12. Find the courses taught in Fall-2009 semester.
select course_id from teaches
where semester='Fall' and year=2009;

--13. Find the set of all courses taught either in Fall-2009 or in Spring-2010.
select course_id from teaches
where (semester='Fall' and year=2009)
 or (semester='Spring' and year=2010);

--14. Find the set of all courses taught in the Fall-2009 as well as in Spring-2010.
select course_id from teaches
where (semester='Fall' and year=2009)
 and (semester='Spring' and year=2010);

--15. Find all courses taught in the Fall-2009 semester but not in the Spring-2010
semester.
select course_id from teaches
where (semester='Fall' and year=2009)
 and not (semester='Spring' and year=2010);

--16. Find all instructors who appear in the instructor relation with null values for
salary.
select * from instructor where salary is NULL;

--17. Find the average salary of instructors in the Finance department.
select avg(salary) from instructor where dept_name='Finance';

--18. Find the total number of instructors who teach a course in the Spring-2010
semester.
select count(distinct ID) from teaches
where (semester='Spring' and year=2010);

--19. Find the average salary in each department.
select dept_name, AVG(salary)
from instructor
group by dept_name;

--20. Find the number of instructors in each department who teach a course in the
Spring-2010
--semester.
select count(distinct ID) from teaches
where (semester='Spring' and year=2010);

--21. List out the departments where the average salary of the instructors is more
than $42,000.
select dept_name
from instructor
group by dept_name
having AVG(salary)>42000;

--22. For each course section offered in 2009, find the average total credits (tot
cred) of all students
--enrolled in the section, if the section had at least 2 students.
select course_id, sec_id, AVG(tot_cred) as average_tot
from takes join student
on takes.ID=student.ID
```

```
group by course_id, sec_id
having count(student.ID)>1;

--23. Find all the courses taught in both the Fall-2009 and Spring-2010 semesters.
select course_id from teaches
where (semester='Fall' and year=2009)
 and (semester='Spring' and year=2010);

--24. Find all the courses taught in the Fall-2009 semester but not in the
Spring-2010 semester.
select course_id from teaches
where (semester='Fall' and year=2009)
 and not (semester='Spring' and year=2010);

--25. Select the names of instructors whose names are neither &lt;Mozart= nor
&lt;Einstein=.
select name from instructor
where name != 'Mozart' and name !='Einstein';

--26. Find the total number of (distinct) students who have taken course sections
taught by the
--instructor with ID 110011.
select count(distinct takes.ID)
from instructor
join teaches on teaches.ID=instructor.ID
join takes on
 takes.course_id=teaches.course_id and
 takes.sec_id=teaches.sec_id and
 takes.semester=teaches.semester and
 takes.year=teaches.year
where instructor.ID=10101
group by instructor.ID;

--27. Find the ID and names of all instructors whose salary is greater than at least
one instructor in the
--History department.
select ID, name from instructor
where salary > (select min(salary)
 from instructor where dept_name='History');

--28. Find the names of all instructors that have a salary value greater than that of
each instructor in
--the Biology department.
select name
from instructor
where salary > (
 select min(salary)
 from instructor
 where dept_name='Biology'
);

--29. Find the departments that have the highest average salary.
select top 1
dept_name
from instructor
group by dept_name
```

```sql
order by avg(salary) desc;

--30. Find all courses taught in both the Fall 2009 semester and in the Spring-2010
semester.
select course_id
from teaches
where semester='Fall' and year=2009
 and semester='Spring' and year=2010;

--31. Find all students who have taken all the courses offered in the Biology
department.
select student.ID, name, count(student.ID)
from student
join takes
 on takes.ID=student.ID
join course
 on takes.course_id=course.course_id
where course.dept_name='Biology'
group by student.ID, name
having count(student.ID) = (
 select count(*)
 from course
 where dept_name='Biology'
);

--32. Find all courses that were offered at most once in 2009.
select teaches.course_id, title
from teaches
join course on course.course_id=teaches.course_id
where year=2009
group by teaches.course_id, title
having count(*)=1;

--33. Find all courses that were offered at least twice in 2009.
select teaches.course_id, title
from teaches
join course on course.course_id=teaches.course_id
where year=2009
group by teaches.course_id, title
having count(*)>=2;

--34. Find the average instructors9 salaries of those departments where the average
salary is greater
--than $42,000.
select dept_name
from instructor
group by dept_name
having AVG(salary)>42000;

--35. Find the maximum across all departments of the total salary at each department.
select top 1
dept_name, sum(salary)
from instructor
group by dept_name
order by sum(salary) desc;
```

```sql
--36. List all departments along with the number of instructors in each department.
select dept_name, count(*)
from instructor
group by dept_name;

--37. Find the titles of courses in the Comp. Sci. department that has 3 credits.
select title
from course
where dept_name='Comp. SCi.'
 and credits=3;

--38. Find the IDs of all students who were taught by an instructor named Einstein;
make sure there
--are no duplicates in the result.
select distinct takes.ID
from instructor
join teaches
 on teaches.ID=instructor.ID
join takes
 on takes.course_id=teaches.course_id
 and takes.sec_id=teaches.sec_id
 and takes.semester=teaches.semester
 and takes.year=teaches.year
where name='Einstein';

--39. Find the highest salary of any instructor.
select max(salary)
from instructor;

--40. Find all instructors earning the highest salary (there may be more than one
with the same
--salary).
select ID, name
from instructor
group by ID, name
order by sum(salary) desc;

--41. Find the enrollment of each section that was offered in Autumn-2009.
select sec_id, course_id
from takes
where semester='Autumn' and year=2009;

--42. Find the maximum enrollment, across all sections, in Autumn-2009.
select top 1
sec_id, course_id
from takes
where semester='Autumn' and year=2009
group by sec_id, course_id
order by count(*) desc;

--43. Find the salaries after the following operation: Increase the salary of each
instructor in the Comp.
--Sci. department by 10%.
update instructor
set salary = salary * 1.10;
where dept_name = 'Comp. Sci.';
```

```sql
--44. Find all students who have not taken a course.
select student.ID, name
from student
left join takes
 on student.ID=takes.ID
where course_id is NULL;

--45. List all course sections offered by the Physics department in the Fall-2009
semester, with the
--building and room number of each section.
select  teaches.sec_id, teaches.course_id, section.building, room_number
from teaches
join course
 on teaches.course_id=course.course_id
join section
 on teaches.course_id=section.course_id
 and teaches.sec_id=section.sec_id
 and teaches.semester=section.semester
 and teaches.year=section.year
join department
 on course.dept_name=department.dept_name
where course.dept_name='Physics'
 and teaches.semester='Fall'
 and teaches.year=2009
group by teaches.sec_id, teaches.course_id, section.building, room_number;

--46. Find the student names who take courses in Spring-2010 semester at Watson
Building.
select name
from student
join takes
 on student.ID=takes.ID
join section
 on takes.course_id=section.course_id
 and takes.sec_id=section.sec_id
 and takes.semester=section.semester
 and takes.year=section.year
where building='Watson'
 and takes.semester='Spring'
 and takes.year=2010;

--47. List the students who take courses teaches by Brandt.
select distinct takes.ID
from instructor
join teaches
 on teaches.ID=instructor.ID
join takes
 on takes.course_id=teaches.course_id
 and takes.sec_id=teaches.sec_id
 and takes.semester=teaches.semester
 and takes.year=teaches.year
where name='Brandt';

--48. Find out the average salary of the instructor in each department.
select dept_name, avg(salary) from instructor group by dept_name;
```

```sql
--49. Find the number of students who take the course titled 8Intro. To Computer
Science.
select count(*)
from student
join takes
 on takes.ID=student.ID
join course
 on takes.course_id=course.course_id
where title='Intro. To Computer Science';

--50. Find out the total salary of the instructors of the Computer Science department
who take a
--course(s) in Watson building.
select sum(salary)
from instructor
join teaches
 on instructor.ID=teaches.ID
join section
 on teaches.course_id=section.course_id
 and teaches.sec_id=section.sec_id
 and teaches.semester=section.semester
 and teaches.year=section.year
where dept_name='Comp. Sci.'
 and building = 'Watson';

--51. Find out the course titles which starts between 10:00 to 12:00.
select title
from course
join section
 on course.course_id=section.course_id
join time_slot
 on time_slot.time_slot_id=section.time_slot_id
where start_hr = 10
 or (start_hr > 10 and start_hr < 12)
 or (start_hr = 12 and start_min = 0);

--52. List the course names where CS-1019 is the pre-requisite course.
select title
from course
join prereq
 on course.course_id=prereq.course_id
where prereq_id = 'CS-1019';

--53. List the student names who get more than B+ grades in their respective courses.
select name
from student
join takes
 on student.ID=takes.ID
where grade like '%A%';

--54. Find the student who takes the maximum credit from each department.
select top 1
student.name, student.dept_name, sum(credits)
from student
join takes
```

```
 on student.ID = takes.ID
join course
 on course.course_id = takes.course_id
group by student.dept_name, student.name, student.ID
order by sum(credits) desc;

--55. Find out the student ID and grades who take a course(s) in Spring-2009
semester.
select student.ID, grade
from student
join takes
 on student.ID=takes.ID
where semester='Spring'
 and year=2009;

--56. Find the building(s) where the student takes the course titled Image
Processing.
select building
from section
join course
 on section.course_id = course.course_id
where title = 'Image Processing';

--57. Find the room no. and the building where the student from Fall-2009 semester
can take a
--course(s)
select building, room_number
from section
where semester = 'Fall'
 and year = 2009;

--58. Find the names of those departments whose budget is higher than that of
Astronomy. List
--them in alphabetic order
select dept_name
from department
where budget > (
 select budget
 from department
 where dept_name = 'Astronomy'
);

--59. Display a list of all instructors, showing each instructor&amp;#39;s ID and the
number of sections
--taught. Make sure to show the number of sections as 0 for instructors who have not
taught
--any section
select instructor.ID, count(distinct sec_id)
from instructor
left join teaches
 on instructor.ID = teaches.ID
group by instructor.ID;

--60. For each student who has retaken a course at least twice (i.e., the student has
taken the
--course at least three times), show the course ID and the student&amp;#39;s ID. Please
```

```
display your
--results in order of course ID and do not display duplicate rows
select ID, course_id
from takes
group by ID, course_id
having count(*) > 2;

--61. Find the names of Biology students who have taken at least 3 Accounting courses
select name
from student
join takes
 on student.ID = takes.ID
where dept_name = 'Biology'
 and course_id in (
  select course_id
  from course
  where dept_name = 'Accounting'
 )
group by student.ID, name
having count(*) > 2;

--62. Find the sections that had maximum enrollment in Fall 2010
select top 1
sec_id, course_id
from takes
where semester='Fall' and year=2010
group by sec_id, course_id
order by count(ID) desc;

SELECT course_id, sec_id
FROM takes
WHERE semester = 'Fall' AND year = 2010
GROUP BY course_id, sec_id
HAVING COUNT(ID) = (
    SELECT MAX(enrollment_count)
    FROM (
        SELECT COUNT(ID) AS enrollment_count
        FROM takes
        WHERE semester = 'Fall' AND year = 2010
        GROUP BY course_id, sec_id
    ) AS subquery
);

--63. Find student names and the number of law courses taken for students who have
taken at
--least half of the available law courses. (These courses are named things like
&#39;Tort Law&#39; or
--&#39;Environmental Law&#39;
select name, count(*)
from student
join takes on student.ID=takes.ID
join course on takes.course_id=course.course_id
where title like '%Law%'
group by student.ID, name
having count(*) > (
 select count(*)/2
```

```
 from course
 where title like '%Law%'
);

--64. Find the rank and name of the 10 students who earned the most A grades (A-, A,
A+).
--Use alphabetical order by name to break ties.
select top 10
row_number() over(order by count(*) desc, name) as rank, name
from student
join takes
 on student.id = takes.id
where grade in ('A-', 'A', 'A+')
group by name, student.ID
order by count(*) desc, name;

--65. Find the titles of courses in the Comp. Sci. department that have 3 credits.
select title
from course
where credits = 3 and dept_name = 'Comp. Sci.';

--66. Find the IDs of all students who were taught by an instructor named Einstein;
make sure there
--are no duplicates in the result.
select distinct takes.ID
from instructor
join teaches
 on teaches.ID=instructor.ID
join takes
 on takes.course_id=teaches.course_id
 and takes.sec_id=teaches.sec_id
 and takes.semester=teaches.semester
 and takes.year=teaches.year
where name='Einstein';

--67. Find the ID and name of each student who has taken at least one Comp. Sci.
course; make sure
--there are no duplicate names in the result.
select distinct student.ID, name
from student
join takes
 on student.id = takes.id
where course_id in (
 select course_id
 from  course
 where dept_name = 'Comp. Sci.'
);


--68. Find the course id, section id, and building for each section of a Biology
course.
select section.course_id, sec_id, building
from section
join course
 on section.course_id = course.course_id
where dept_name = 'Biology';
```

```
--69. Output instructor names sorted by the ratio of their salary to their
department&amp;#39;s budget (in
--ascending order).
select name
from instructor
join department
 on instructor.dept_name = department.dept_name
order by (salary / budget);

--70. Output instructor names and buildings for each building an instructor has
taught in. Include
--instructor names who have not taught any classes (the building name should be NULL
in this
--case).
select distinct name, building
from instructor
left join teaches
 on instructor.ID = teaches.ID
left join section
 on teaches.course_id = section.course_id
 and teaches.sec_id = section.sec_id
 and teaches.year = section.year
 and teaches.semester = section.semester;
```

5. W3 Resources

MySQL Basic Select Statement:

1.

```
-- Selecting the first_name column and aliasing it as "First Name"
SELECT first_name "First Name",
-- Selecting the last_name column and aliasing it as "Last Name"
last_name "Last Name"
-- Selecting data from the employees table
FROM employees;
```

2.

```
-- Selecting distinct values from the department_id column
SELECT DISTINCT department_id
-- Selecting data from the employees table
FROM employees;
```

3.

```
-- Selecting all columns from the employees table
SELECT *
-- Selecting data from the employees table
FROM employees
-- Ordering the result set by the first_name column in descending order
ORDER BY first_name DESC;
```

4.

```
-- Selecting the first_name, last_name, and salary columns
SELECT first_name, last_name, salary,
-- Calculating 15% of the salary and aliasing it as "PF" (Provident Fund)
salary * 0.15 AS PF
-- Selecting data from the employees table
FROM employees;
```

5.
```sql
-- Selecting specific columns: employee_id, first_name, last_name, and salary
SELECT employee_id, first_name, last_name, salary
-- Selecting data from the employees table
FROM employees
-- Ordering the result set by the salary column in ascending order
ORDER BY salary;
```

6.
```sql
-- Calculating the sum of salaries for all employees
SELECT SUM(salary)
-- Selecting data from the employees table
FROM employees;
```

7.
```sql
-- Selecting the maximum and minimum salary values from the employees table
SELECT MAX(salary), MIN(salary)
-- Selecting data from the employees table
FROM employees;
```

8.
```sql
-- Calculating the average salary and counting the total number of employees
SELECT AVG(salary), COUNT(*)
-- Selecting data from the employees table
FROM employees;
```

9.
```sql
-- Counting the total number of records (rows) in the employees table
SELECT COUNT(*)
-- Selecting data from the employees table
FROM employees;
```

10.
```sql
-- Counting the total number of distinct job IDs in the employees table
SELECT COUNT(DISTINCT job_id)
-- Selecting data from the employees table
FROM employees;
```

11.
```sql
-- Converting the first_name column values to uppercase
SELECT UPPER(first_name)
-- Selecting data from the employees table
FROM employees;
```

12.
```sql
-- Extracting the substring of the first three characters from the first_name column
SELECT SUBSTRING(first_name, 1, 3)
-- Selecting data from the employees table
FROM employees;
```

13.
```sql
-- Performing arithmetic operations: multiplication and addition
SELECT 171 * 214 + 625 Result;
```

14.

```sql
-- Concatenating the first_name and last_name columns with a space in between
SELECT CONCAT(first_name, ' ', last_name) 'Employee Name'
-- Selecting data from the employees table
FROM employees;
```

15.

```sql
-- Removing leading and trailing whitespace characters from the first_name column
SELECT TRIM(first_name)
-- Selecting data from the employees table
FROM employees;
```

16.

```sql
-- Selecting the first_name and last_name columns
SELECT first_name, last_name,
-- Calculating the sum of the lengths of first_name and last_name columns and
aliasing it as 'Length of Names'
LENGTH(first_name) + LENGTH(last_name) 'Length of Names'
-- Selecting data from the employees table
FROM employees;
```

17.

```sql
-- Selecting all columns from the employees table
SELECT *
-- Selecting data from the employees table
FROM employees
-- Filtering the result set to include only rows where the first_name column contains
a digit
WHERE first_name REGEXP '[0-9]';
```

18.

```sql
-- Selecting the employee_id and first_name columns
SELECT employee_id, first_name
-- Selecting data from the employees table
FROM employees
-- Limiting the result set to only include the first 10 rows
LIMIT 10;
```

19.

```sql
-- Selecting the first_name, last_name, and calculating the monthly salary
SELECT first_name, last_name,
-- Dividing the salary by 12 to calculate the monthly salary and rounding it to 2
decimal places
ROUND(salary / 12, 2) AS 'Monthly Salary'
-- Selecting data from the employees table
FROM employees;
MySQL Restricting and Sorting data:
```

1.

```sql
-- Selecting the first_name, last_name, and salary columns
SELECT first_name, last_name, salary
-- Selecting data from the employees table
FROM employees
-- Filtering the result set to include only rows where the salary is not between
10000 and 15000
WHERE salary NOT BETWEEN 10000 AND 15000;
```

```
2.
-- Selecting the first_name, last_name, and department_id columns
SELECT first_name, last_name, department_id
-- Selecting data from the employees table
FROM employees
-- Filtering the result set to include only rows where the department_id is either 30
or 100
WHERE department_id IN (30, 100)
-- Ordering the result set by the department_id column in ascending order
ORDER BY department_id ASC;

3.
-- Selecting the first_name, last_name, salary, and department_id columns
SELECT first_name, last_name, salary, department_id
-- Selecting data from the employees table
FROM employees
-- Filtering the result set to include only rows where the salary is not between
10000 and 15000
-- and the department_id is either 30 or 100
WHERE salary NOT BETWEEN 10000 AND 15000
AND department_id IN (30, 100);

4.
-- Selecting the first_name, last_name, and hire_date columns
SELECT first_name, last_name, hire_date
-- Selecting data from the employees table
FROM employees
-- Filtering the result set to include only rows where the year part of the hire_date
is like '1987%'
WHERE YEAR(hire_date) LIKE '1987%';

5.
-- Selecting the first_name column
SELECT first_name
-- Selecting data from the employees table
FROM employees
-- Filtering the result set to include only rows where the first_name column contains
both 'b' and 'c'
WHERE first_name LIKE '%b%'
AND first_name LIKE '%c%';

6.
-- Selecting the last_name, job_id, and salary columns
SELECT last_name, job_id, salary
-- Selecting data from the employees table
FROM employees
-- Filtering the result set to include only rows where the job_id is either 'IT_PROG'
or 'SH_CLERK'
-- and the salary is not 4500, 10000, or 15000
WHERE job_id IN ('IT_PROG', 'SH_CLERK')
AND salary NOT IN (4500, 10000, 15000);

7.
-- Selecting the last_name column
SELECT last_name
-- Selecting data from the employees table
```

```sql
FROM employees
-- Filtering the result set to include only rows where the last_name column consists
of exactly six characters
WHERE last_name LIKE '______';

8.
-- Selecting the last_name column
SELECT last_name
-- Selecting data from the employees table
FROM employees
-- Filtering the result set to include only rows where the last_name column starts
with two characters followed by 'e' and any other characters
WHERE last_name LIKE '__e%';

9.
-- Selecting distinct values from the job_id column
SELECT DISTINCT job_id
-- Selecting data from the employees table
FROM employees;

10.
-- Selecting the first_name, last_name, salary columns, and calculating 15% of the
salary as PF (Provident Fund)
SELECT first_name, last_name, salary, salary * 0.15 AS PF
-- Selecting data from the employees table
FROM employees;

11.
-- Selecting all columns from the employees table
SELECT *
-- Selecting data from the employees table
FROM employees
-- Filtering the result set to include only rows where the last_name column matches
one of the specified values
WHERE last_name IN ('JONES', 'BLAKE', 'SCOTT', 'KING', 'FORD');

Aggregate Functions and Group by :

1.
-- Counting the number of distinct job IDs in the employees table
SELECT COUNT(DISTINCT job_id)
-- Selecting data from the employees table
FROM employees;

2.
-- Calculating the total sum of salaries for all employees
SELECT SUM(salary)
-- Selecting data from the employees table
FROM employees;

3.
-- Retrieving the minimum salary from the employees table
SELECT MIN(salary)
-- Selecting data from the employees table
FROM employees;
```

```
4.
-- Retrieving the maximum salary among employees with the job_id 'IT_PROG'
SELECT MAX(salary)
-- Selecting data from the employees table
FROM employees
-- Filtering the result set to include only employees with the job_id 'IT_PROG'
WHERE job_id = 'IT_PROG';

5.
-- Calculating the average salary and counting the number of employees in department
90
SELECT AVG(salary), COUNT(*)
-- Selecting data from the employees table
FROM employees
-- Filtering the result set to include only employees in department 90
WHERE department_id = 90;

6.
-- Calculating various statistics (maximum, minimum, sum, average) for the salary
column
SELECT
    -- Rounding the maximum salary to 0 decimal places and aliasing it as 'Maximum'
    ROUND(MAX(salary), 0) 'Maximum',
    -- Rounding the minimum salary to 0 decimal places and aliasing it as 'Minimum'
    ROUND(MIN(salary), 0) 'Minimum',
    -- Rounding the sum of salaries to 0 decimal places and aliasing it as 'Sum'
    ROUND(SUM(salary), 0) 'Sum',
    -- Rounding the average salary to 0 decimal places and aliasing it as 'Average'
    ROUND(AVG(salary), 0) 'Average'
-- Selecting data from the employees table
FROM employees;

7.
-- Counting the number of employees for each job ID
SELECT job_id, COUNT(*)
-- Selecting data from the employees table
FROM employees
-- Grouping the result set by the job_id column
GROUP BY job_id;

8.
-- Calculating the difference between the maximum and minimum salaries
SELECT MAX(salary) - MIN(salary) DIFFERENCE
-- Selecting data from the employees table
FROM employees;

9.
-- Selecting the manager_id and the minimum salary for each manager
SELECT manager_id, MIN(salary)
-- Selecting data from the employees table
FROM employees
-- Filtering the result set to include only rows where manager_id is not NULL
WHERE manager_id IS NOT NULL
-- Grouping the result set by manager_id
GROUP BY manager_id
-- Sorting the result set by the minimum salary in descending order
```

```
ORDER BY MIN(salary) DESC;
```

```
10.
-- Calculating the total salary for each department
SELECT department_id, SUM(salary)
-- Selecting data from the employees table
FROM employees
-- Grouping the result set by department_id
GROUP BY department_id;
```

```
11.
-- Calculating the average salary for each job, excluding 'IT_PROG'
SELECT job_id, AVG(salary)
-- Selecting data from the employees table
FROM employees
-- Filtering the result set to include only rows where the job_id is not 'IT_PROG'
WHERE job_id <> 'IT_PROG'
-- Grouping the result set by job_id
GROUP BY job_id;
```

```
12.
-- Calculating various statistics for salaries of employees in department 90, grouped
by job_id
SELECT job_id, SUM(salary), AVG(salary), MAX(salary), MIN(salary)
-- Selecting data from the employees table
FROM employees
-- Filtering the result set to include only rows where the department_id is '90'
WHERE department_id = '90'
-- Grouping the result set by job_id
GROUP BY job_id;
```

```
13.
-- Retrieving the maximum salary for each job title, filtering out job titles where
the maximum salary is less than 4000
SELECT job_id, MAX(salary)
-- Selecting data from the employees table
FROM employees
-- Grouping the result set by job_id
GROUP BY job_id
-- Filtering the result set to include only groups where the maximum salary is
greater than or equal to 4000
HAVING MAX(salary) >= 4000;
```

```
14.
-- Calculating the average salary and counting the number of employees for each
department, filtering out departments with fewer than 10 employees
SELECT department_id, AVG(salary), COUNT(*)
-- Selecting data from the employees table
FROM employees
-- Grouping the result set by department_id
GROUP BY department_id
-- Filtering the result set to include only groups where the count of employees is
greater than 10
HAVING COUNT(*) > 10;
```

MySQL Subquery :

```
1.
-- Selecting the first name, last name, and salary of employees whose salary is
higher than that of the employee with the last name 'Bull'
SELECT FIRST_NAME, LAST_NAME, SALARY
-- Selecting data from the employees table
FROM employees
-- Filtering the result set to include only employees whose salary is higher than the
salary of the employee with the last name 'Bull'
WHERE SALARY >
    -- Subquery to fetch the salary of the employee with the last name 'Bull'
    (SELECT salary FROM employees WHERE last_name = 'Bull');

2.
-- Selecting the first name and last name of employees
SELECT first_name, last_name
-- Selecting data from the employees table
FROM employees
-- Filtering the result set to include only employees whose department_id is in the
set of department_ids where the department_name is 'IT'
WHERE department_id
IN (SELECT department_id FROM departments WHERE department_name='IT');

3.
-- Selecting the first name and last name of employees
SELECT first_name, last_name
-- Selecting data from the employees table
FROM employees
-- Filtering the result set to include only employees whose manager_id is in the set
of employee_ids
-- where the department_id is in the set of department_ids associated with locations
in the US
WHERE manager_id in
    (SELECT employee_id
    -- Subquery to select employee_ids from the employees table
    FROM employees
    -- Filtering the employee_ids to include only those associated with departments
    -- where the location_id is in the set of location_ids associated with countries
having country_id 'US'
    WHERE department_id
    IN
        (SELECT department_id
        -- Subquery to select department_ids from the departments table
        FROM departments
        -- Filtering the department_ids to include only those associated with
locations
        -- where the country_id is 'US'
        WHERE location_id
        IN
            (SELECT location_id
            -- Subquery to select location_ids from the locations table
            FROM locations
            -- Filtering the location_ids to include only those associated with
countries
            -- having country_id 'US'
            WHERE country_id='US')
```

```
        )
    );
```

4.

```
-- Selecting the first name and last name of employees
SELECT first_name, last_name
-- Selecting data from the employees table
FROM employees
-- Filtering the result set to include only employees whose employee_id is in the set
of manager_ids
WHERE (employee_id IN
    -- Subquery to select manager_ids from the employees table
    (SELECT manager_id FROM employees)
);
```

5.

```
-- Selecting the first name, last name, and salary of employees
SELECT first_name, last_name, salary
-- Selecting data from the employees table
FROM employees
-- Filtering the result set to include only employees whose salary is greater than
the average salary of all employees
WHERE salary >
    -- Subquery to calculate the average salary from the employees table
    (SELECT AVG(salary) FROM employees);
```

6.

```
-- Selecting the first name, last name, and salary of employees
SELECT first_name, last_name, salary
-- Selecting data from the employees table
FROM employees
-- Filtering the result set to include only employees whose salary matches the
minimum salary defined for their job position
WHERE employees.salary =
    -- Subquery to select the minimum salary for each job position from the jobs
table
    (SELECT min_salary
    FROM jobs
    -- Matching the job_id of each employee with the job_id in the jobs table
    WHERE employees.job_id = jobs.job_id);
```

7.

```
Selecting the first name, last name, and salary of employees
SELECT first_name, last_name, salary
-- Selecting data from the employees table
FROM employees
-- Filtering the result set to include only employees who belong to departments with
names starting with 'IT'
-- and have a salary higher than the average salary of all employees
WHERE department_id IN
    -- Subquery to select department_ids from the departments table where the
department_name starts with 'IT'
    (SELECT department_id FROM departments WHERE department_name LIKE 'IT%')
-- Additional condition: Salary of employees should be higher than the average salary
of all employees
AND salary >
```

```
    -- Subquery to calculate the average salary from the employees table
    (SELECT avg(salary) FROM employees);

8.
S-- Selecting the first name, last name, and salary of employees
SELECT first_name, last_name, salary
-- Selecting data from the employees table
FROM employees
-- Filtering the result set to include only employees whose salary is higher than
that of the employee with the last name 'Bell'
WHERE salary >
    -- Subquery to fetch the salary of the employee with the last name 'Bell'
    (SELECT salary FROM employees WHERE last_name = 'Bell')
-- Sorting the result set in ascending order based on the first name of employees
ORDER BY first_name;

9.
-- Selecting all columns from the employees table
SELECT *
-- Selecting data from the employees table
FROM employees
-- Filtering the result set to include only employees whose salary is equal to the
minimum salary among all employees
WHERE salary =
    -- Subquery to find the minimum salary from the employees table
    (SELECT MIN(salary) FROM employees);

10.
-- Selecting all columns from the employees table
SELECT *
-- Selecting data from the employees table
FROM employees
-- Filtering the result set to include only employees whose salary is greater than
all average salaries within their respective departments
WHERE salary >-- Selecting all columns from the employees table
SELECT *
-- Selecting data from the employees table
FROM employees
-- Filtering the result set to include only employees whose salary is greater than
all average salaries within their respective departments
WHERE salary >
    -- Subquery to calculate the average salary for each department and compare with
each employee's salary
    ALL(SELECT avg(salary) FROM employees GROUP BY department_id);

11.
-- Selecting the first name, last name, job ID, and salary of employees
SELECT first_name, last_name, job_id, salary
-- Selecting data from the employees table
FROM employees
-- Filtering the result set to include only employees whose salary is greater than
all salaries of employees with job_id 'SH_CLERK'
WHERE salary >
    -- Subquery to select all salaries of employees with job_id 'SH_CLERK' and
compare with each employee's salary
    ALL (SELECT salary FROM employees WHERE job_id = 'SH_CLERK')
```

```sql
-- Sorting the result set in ascending order based on salary
ORDER BY salary;
```

12.

```sql
-- Selecting the first name and last name of employees who are not managers
SELECT b.first_name, b.last_name
-- Selecting data from the employees table, aliasing it as 'b'
FROM employees b
-- Filtering the result set to include only employees who are not managers
WHERE NOT EXISTS
    -- Subquery to check if there is no employee with manager_id equal to the
employee_id of each employee in the outer query
    (SELECT 'X' FROM employees a WHERE a.manager_id = b.employee_id);
```

13.

```sql
-- Selecting the employee_id, first name, last name, and department name of
employees
SELECT employee_id, first_name, last_name,
-- Subquery to fetch the department name for each employee's department_id
(SELECT department_name FROM departments d
 -- Joining the employees table with the departments table based on the department_id
 WHERE e.department_id = d.department_id) department
-- Selecting data from the employees table, aliasing it as 'e'
 FROM employees e
-- Sorting the result set based on the department name
 ORDER BY department;
```

14.

```sql
-- Selecting the employee_id and first name of employees
SELECT employee_id, first_name
-- Selecting data from the employees table, aliasing it as 'A'
FROM employees AS A
-- Filtering the result set to include only employees whose salary is greater than
the average salary of their department
WHERE salary >
    -- Subquery to calculate the average salary for each department and compare with
each employee's salary
    (SELECT AVG(salary) FROM employees WHERE department_id = A.department_id);
```

15.

```sql
-- Setting user-defined variable @i to 0
SET @i = 0;
-- Selecting the sequential number and employee_id of every second row in the
employees table
SELECT i, employee_id
-- Subquery to generate a sequential number for each row in the employees table,
starting from 1
FROM (SELECT @i := @i + 1 AS i, employee_id FROM employees) a
-- Filtering the result set to include only rows with even sequential numbers
WHERE MOD(a.i, 2) = 0;
```

16.

```sql
-- Selecting distinct salaries from the employees table for which there are exactly 5
distinct salaries greater than or equal to it
SELECT DISTINCT salary
-- Selecting data from the employees table, aliasing it as 'e1'
```

```sql
FROM employees e1
-- Filtering the result set to include only those salaries for which there are
exactly 5 distinct salaries greater than or equal to it
WHERE 5 =
    -- Subquery to count the number of distinct salaries greater than or equal to
each salary
    (SELECT COUNT(DISTINCT salary)
    -- Selecting data from the employees table, aliasing it as 'e2'
    FROM employees e2
    -- Filtering the result set to include only distinct salaries greater than or
equal to the salary in the outer query (e1.salary)
    WHERE e2.salary >= e1.salary);

17.
-- Selecting distinct salaries from the employees table for which there are exactly 4
distinct salaries less than or equal to it
SELECT DISTINCT salary
-- Selecting data from the employees table, aliasing it as 'e1'
FROM employees e1
-- Filtering the result set to include only those salaries for which there are
exactly 4 distinct salaries less than or equal to it
WHERE 4 =
    -- Subquery to count the number of distinct salaries less than or equal to each
salary
    (SELECT COUNT(DISTINCT salary)
    -- Selecting data from the employees table, aliasing it as 'e2'
    FROM employees e2
    -- Filtering the result set to include only distinct salaries less than or equal
to the salary in the outer query (e1.salary)
    WHERE e2.salary <= e1.salary);

18.
-- Selecting all columns from a subset of the employees table, ordered by employee_id
in descending order, and limiting the result to the first 10 rows
SELECT * FROM (
    SELECT * FROM employees ORDER BY employee_id DESC LIMIT 10
) sub
-- Sorting the result set from the subset in ascending order based on employee_id
ORDER BY employee_id ASC;

19.
-- Selecting all columns from the departments table
SELECT *
-- Selecting data from the departments table
FROM departments
-- Filtering the result set to include only departments whose department_id is not
present in the set of department_ids associated with employees
WHERE department_id
NOT IN
    -- Subquery to select department_ids from the employees table
    (select department_id FROM employees);

20.
-- Selecting distinct salary values from the employees table
SELECT DISTINCT salary
-- Selecting data from the employees table, aliasing it as 'a'
```

```
FROM employees a
-- Filtering the result set to include only salary values where there are at most 3
distinct salary values greater than or equal to it
WHERE 3 >=
    -- Subquery to count the number of distinct salaries greater than or equal to
each salary
    (SELECT COUNT(DISTINCT salary)
    -- Selecting data from the employees table, aliasing it as 'b'
    FROM employees b
    -- Filtering the result set to include only distinct salaries greater than or
equal to the salary in the outer query (a.salary)
    WHERE b.salary >= a.salary)
-- Sorting the result set in descending order based on salary
ORDER BY a.salary DESC;

21.
-- Selecting distinct salary values from the employees table
SELECT DISTINCT salary
-- Selecting data from the employees table, aliasing it as 'a'
FROM employees a
-- Filtering the result set to include only salary values where there are at most 3
distinct salary values less than or equal to it
WHERE 3 >=
    -- Subquery to count the number of distinct salaries less than or equal to each
salary
    (SELECT COUNT(DISTINCT salary)
    -- Selecting data from the employees table, aliasing it as 'b'
    FROM employees b
    -- Filtering the result set to include only distinct salaries less than or equal
to the salary in the outer query (a.salary)
    WHERE b.salary <= a.salary)
-- Sorting the result set in descending order based on salary
ORDER BY a.salary DESC;


22.
-- Selecting all columns from the employees table, aliasing it as 'emp1'
SELECT *
-- Selecting data from the employees table, aliasing it as 'emp1'
FROM employees emp1
-- Filtering the result set to include only employees where the count of distinct
salaries greater than the salary of the current employee is 1
WHERE (1) = (
    -- Subquery to count the number of distinct salaries greater than the salary of
each employee
    SELECT COUNT(DISTINCT(emp2.salary))
    -- Selecting data from the employees table, aliasing it as 'emp2'
    FROM employees emp2
    -- Filtering the result set to include only distinct salaries greater than the
salary of the employee in the outer query (emp1.salary)
    WHERE emp2.salary > emp1.salary
);

MySQL Joins :

1.
```

```sql
-- This SQL query selects specific columns from the 'locations' table after
performing a natural join with the 'countries' table.

SELECT
    location_id, -- Selecting the 'location_id' column from the result set.
    street_address, -- Selecting the 'street_address' column from the result set.
    city, -- Selecting the 'city' column from the result set.
    state_province, -- Selecting the 'state_province' column from the result set.
    country_name -- Selecting the 'country_name' column from the result set.
FROM
    locations -- Specifying the 'locations' table.
NATURAL JOIN
    countries; -- Performing a natural join with the 'countries' table based on any
common columns.

2.
-- This SQL query selects specific columns from the 'employees' table after
performing an inner join with the 'departments' table using the 'department_id'
column.

SELECT
    first_name, -- Selecting the 'first_name' column from the result set.
    last_name, -- Selecting the 'last_name' column from the result set.
    department_id, -- Selecting the 'department_id' column from the result set.
    department_name -- Selecting the 'department_name' column from the result set.
FROM
    employees -- Specifying the 'employees' table.
JOIN
    departments -- Specifying the 'departments' table.
USING
    (department_id); -- Performing an inner join using the 'department_id' column,
which is common in both tables.



3.
-- This SQL query selects specific columns from the 'employees' and 'departments'
tables, as well as the 'locations' table, to retrieve information about employees in
the London city.

SELECT
    e.first_name, -- Selecting the 'first_name' column from the 'employees' table and
aliasing it as 'e'.
    e.last_name, -- Selecting the 'last_name' column from the 'employees' table and
aliasing it as 'e'.
    e.job_id, -- Selecting the 'job_id' column from the 'employees' table and
aliasing it as 'e'.
    e.department_id, -- Selecting the 'department_id' column from the 'employees'
table and aliasing it as 'e'.
    d.department_name -- Selecting the 'department_name' column from the
'departments' table and aliasing it as 'd'.
FROM
    employees e -- Specifying the 'employees' table and aliasing it as 'e'.
JOIN
    departments d -- Specifying the 'departments' table and aliasing it as 'd'.
ON
```

```
    (e.department_id = d.department_id) -- Performing a join between 'employees' and
'departments' based on the 'department_id' column.
JOIN
    locations l ON -- Joining the 'locations' table and aliasing it as 'l'.
    (d.location_id = l.location_id) -- Performing a join between 'departments' and
'locations' based on the 'location_id' column.
WHERE
    LOWER(l.city) = 'London'; -- Filtering the result to only include rows where the
city in lowercase is 'London'.
4.
-- This SQL query selects specific columns from the 'employees' table, twice aliased
as 'e' and 'm', to retrieve information about employees and their managers.

SELECT
    e.employee_id 'Emp_Id', -- Selecting the 'employee_id' column from the
'employees' table, aliased as 'e', and renaming it as 'Emp_Id'.
    e.last_name 'Employee', -- Selecting the 'last_name' column from the 'employees'
table, aliased as 'e', and renaming it as 'Employee'.
    m.employee_id 'Mgr_Id', -- Selecting the 'employee_id' column from the
'employees' table, aliased as 'm', and renaming it as 'Mgr_Id'.
    m.last_name 'Manager' -- Selecting the 'last_name' column from the 'employees'
table, aliased as 'm', and renaming it as 'Manager'.
FROM
    employees e -- Specifying the 'employees' table and aliasing it as 'e'.
JOIN
    employees m -- Joining the 'employees' table again and aliasing it as 'm'.
ON
    (e.manager_id = m.employee_id); -- Performing a join between 'employees' and
itself based on the 'manager_id' column to associate employees with their managers.

5.
-- This SQL query selects specific columns from the 'employees' table to retrieve
information about employees hired after an employee with the last name 'Jones'.

SELECT
    e.first_name, -- Selecting the 'first_name' column from the 'employees' table.
    e.last_name, -- Selecting the 'last_name' column from the 'employees' table.
    e.hire_date -- Selecting the 'hire_date' column from the 'employees' table.
FROM
    employees e -- Specifying the 'employees' table and aliasing it as 'e'.
JOIN
    employees davies -- Joining the 'employees' table again and aliasing it as
'davies'.
ON
    (davies.last_name = 'Jones') -- Performing a join based on the condition where
the last name in 'davies' is 'Jones'.
WHERE
    davies.hire_date < e.hire_date; -- Filtering the result to include only employees
hired after the employee with the last name 'Jones'.

6.
-- This SQL query retrieves the count of employees in each department, along with the
department names, from the 'departments' and 'employees' tables.

SELECT
    department_name AS 'Department Name', -- Selecting the 'department_name' column
```

```
from the 'departments' table and aliasing it as 'Department Name'.
    COUNT(*) AS 'No of Employees' -- Counting the number of records (employees) in
each department and aliasing it as 'No of Employees'.
FROM
    departments -- Specifying the 'departments' table.
INNER JOIN
    employees -- Performing an inner join with the 'employees' table.
ON
    employees.department_id = departments.department_id -- Joining the 'employees'
and 'departments' tables based on the 'department_id' column.
GROUP BY
    departments.department_id, department_name -- Grouping the result set by
department ID and department name.
ORDER BY
    department_name; -- Ordering the result set by department name in ascending
order.

7.
-- This SQL query retrieves specific columns from the 'job_history' table and
calculates the duration of each job in days for employees in the specified
department.

SELECT
    employee_id, -- Selecting the 'employee_id' column from the result set.
    job_title, -- Selecting the 'job_title' column from the result set.
    end_date - start_date AS Days -- Calculating the difference between 'end_date'
and 'start_date' columns and aliasing it as 'Days'.
FROM
    job_history -- Specifying the 'job_history' table.
NATURAL JOIN
    jobs -- Performing a natural join with the 'jobs' table.
WHERE
    department_id = 90; -- Filtering the result to include only records where the
department ID is 90.

8.
-- This SQL query retrieves specific columns from the 'departments' and 'employees'
tables to get information about department managers.

SELECT
    d.department_id, -- Selecting the 'department_id' column from the 'departments'
table.
    d.department_name, -- Selecting the 'department_name' column from the
'departments' table.
    d.manager_id, -- Selecting the 'manager_id' column from the 'departments' table.
    e.first_name -- Selecting the 'first_name' column from the 'employees' table and
aliasing it as 'e'.
FROM
    departments d -- Specifying the 'departments' table and aliasing it as 'd'.
INNER JOIN
    employees e -- Performing an inner join with the 'employees' table and aliasing
it as 'e'.
ON
    (d.manager_id = e.employee_id); -- Joining the 'departments' and 'employees'
tables based on the 'manager_id' column to associate departments with their managers.
```

```
9.
-- This SQL query retrieves specific columns from the 'departments', 'employees', and
'locations' tables to get information about department managers and their
corresponding locations.

SELECT
    d.department_name, -- Selecting the 'department_name' column from the
'departments' table.
    e.first_name, -- Selecting the 'first_name' column from the 'employees' table and
aliasing it as 'e'.
    l.city -- Selecting the 'city' column from the 'locations' table and aliasing it
as 'l'.
FROM
    departments d -- Specifying the 'departments' table and aliasing it as 'd'.
JOIN
    employees e -- Performing a join with the 'employees' table and aliasing it as
'e'.
ON
    (d.manager_id = e.employee_id) -- Joining the 'departments' and 'employees'
tables based on the 'manager_id' column to associate departments with their managers.
JOIN
    locations l USING (location_id); -- Performing a join with the 'locations' table
based on the 'location_id' column.

10.
-- This SQL query calculates the average salary for each job title by joining the
'employees' and 'jobs' tables.

SELECT
    job_title, -- Selecting the 'job_title' column from the result set.
    AVG(salary) -- Calculating the average salary and selecting it from the result
set.
FROM
    employees -- Specifying the 'employees' table.
NATURAL JOIN
    jobs -- Performing a natural join with the 'jobs' table.
GROUP BY
    job_title; -- Grouping the result set by job title to calculate the average
salary for each job.

11.
-- This SQL query selects specific columns from the 'employees' and 'jobs' tables and
calculates the difference between each employee's salary and the minimum salary for
their job title.

SELECT
    job_title, -- Selecting the 'job_title' column from the result set.
    first_name, -- Selecting the 'first_name' column from the result set.
    salary - min_salary AS 'Salary - Min_Salary' -- Calculating the difference
between the salary and the minimum salary for each job title and aliasing it as
'Salary - Min_Salary'.
FROM
    employees -- Specifying the 'employees' table.
NATURAL JOIN
    jobs; -- Performing a natural join with the 'jobs' table.
```

12.

```
-- This SQL query retrieves all columns from the 'job_history' table for employees
whose salary is greater than 10000.

SELECT
    jh.* -- Selecting all columns from the 'job_history' table.
FROM
    job_history jh -- Specifying the 'job_history' table and aliasing it as 'jh'.
JOIN
    employees e -- Performing a join with the 'employees' table and aliasing it as
'e'.
ON
    (jh.employee_id = e.employee_id) -- Joining the 'job_history' and 'employees'
tables based on the 'employee_id' column to associate job history with employees.
WHERE
    salary > 10000; -- Filtering the result to include only records where the salary
is greater than 10000.
```

13.

```
-- This SQL query selects specific columns from the 'departments' and 'employees'
tables to retrieve information about department managers with more than 15 years of
experience.

SELECT
    first_name, -- Selecting the 'first_name' column from the result set.
    last_name, -- Selecting the 'last_name' column from the result set.
    hire_date, -- Selecting the 'hire_date' column from the result set.
    salary, -- Selecting the 'salary' column from the result set.
    (DATEDIFF(now(), hire_date))/365 Experience -- Calculating the experience in
years and aliasing it as 'Experience'.
FROM
    departments d -- Specifying the 'departments' table and aliasing it as 'd'.
JOIN
    employees e -- Joining the 'employees' table and aliasing it as 'e'.
ON
    (d.manager_id = e.employee_id) -- Joining the 'departments' and 'employees'
tables based on the 'manager_id' column to associate department managers with their
departments.
WHERE
    (DATEDIFF(now(), hire_date))/365 > 15; -- Filtering the result to include only
records where the experience in years is greater than 15.
```

Date and Time functions :

1.

```
-- This SQL query calculates a date that is three months prior to the current date.

SELECT
date(((PERIOD_ADD -- Calculates a period by adding a specified number of months to a
given period.
```

```
    (EXTRACT(YEAR_MONTH -- Extracts the year and month from the current date
(CURDATE()).
    FROM CURDATE()) -- Specifies the current date.
    ,-3)*100)+1)); -- Subtracts three months from the current date, then multiplies
by 100 to convert it to a period, and finally adds 1 to convert it back to a date.

2.
-- This SQL query calculates the last day of the current month.

SELECT
    (SUBDATE(ADDDATE -- Calculates the date obtained by adding a specified interval
(1 month) to the current date.
    (CURDATE(),INTERVAL 1 MONTH), -- Adds 1 month to the current date (CURDATE()).
    INTERVAL DAYOFMONTH(CURDATE()) DAY)) -- Subtracts the number of days from the
current date to get the last day of the current month.
    AS LastDayOfTheMonth; -- Alias for the calculated last day of the month.

3.
-- This SQL query retrieves distinct dates representing the first day of the week for
each hire date in the 'employees' table.

SELECT DISTINCT( -- Selects distinct values of the result set.
    STR_TO_DATE( -- Converts a string into a date value.
CONCAT( -- Concatenates multiple strings into one string.
YEARWEEK(hire_date), -- Gets the year and week number for each hire date.
            '1' -- Appends '1' to the end of the concatenated string.
        ),
        '%x%v%w' -- Specifies the format of the input string.
    )
)
FROM
employees; -- Specifies the 'employees' table.

4.
-- This SQL query creates a date using the year extracted from the current date and
the day '1'.

SELECT
MAKEDATE( -- Creates a date using the specified year and day.
EXTRACT(YEAR FROM CURDATE()), -- Extracts the year from the current date (CURDATE()).
        1 -- Specifies the day as '1'.
    );

5.
-- This SQL query creates a date using the year extracted from the current date, the
month '12', and the day '31'.

SELECT
    STR_TO_DATE( -- Converts a string into a date value.
CONCAT( -- Concatenates multiple strings into one string.
            12, -- Specifies the month as '12'.
            31, -- Specifies the day as '31'.
EXTRACT(YEAR FROM CURDATE()) -- Extracts the year from the current date (CURDATE()).
        ),
        '%m%d%Y' -- Specifies the format of the input string.
    );
```

```
6.
-- This SQL query calculates the age based on the difference between the current year
and the year of birth.

SELECT
YEAR(CURRENT_TIMESTAMP) - -- Calculates the current year.
YEAR("1967-06-08") - -- Calculates the year from the given birthdate.
    (RIGHT(CURRENT_TIMESTAMP, 5) < -- Checks if the month and day of the current date
are before the month and day of the birthdate.
RIGHT("1967-06-08", 5)) -- Extracts the month and day from the birthdate.
as age; -- Alias for the calculated age.

7.
-- This SQL query formats the current date in a specified format and aliases it as
'Current_date'.

SELECT
    DATE_FORMAT( -- Formats a date value according to the specified format.
CURDATE(), -- Retrieves the current date using the CURDATE() function.
        '%M %e, %Y' -- Specifies the desired format for the date: '%M' for full month
name, '%e' for day of the month without leading zeros, and '%Y' for four-digit year.
    )
    AS 'Current_date'; -- Alias for the formatted current date.

8.
-- This SQL query formats the current date and time in a specified format.

SELECT
    DATE_FORMAT( -- Formats a date and time value according to the specified format.
NOW(), -- Retrieves the current date and time using the NOW() function.
        ''%W %M %Y' -- Specifies the desired format for the date and time: '%W' for
the full weekday name, '%M' for the full month name, and '%Y' for the four-digit
year.
    );

9.
-- This SQL query extracts the year component from the current date and time.

SELECT
EXTRACT(YEAR FROM NOW());

10.
-- This SQL query converts a number representing the number of days since year 0 to a
date value.

SELECT
    FROM_DAYS(730677);

11.
-- This SQL query retrieves the first name and hire date of employees hired within a
specific date range.

SELECT
    FIRST_NAME, -- Selecting the 'FIRST_NAME' column from the 'employees' table.
    HIRE_DATE -- Selecting the 'HIRE_DATE' column from the 'employees' table.
```

```
FROM
employees -- Specifying the 'employees' table.
WHERE
    HIRE_DATE  -- Filtering the rows based on the hire date being within a specific
date range.
    BETWEEN '1987-06-01 00:00:00' -- Specifies the start of the date range.
    AND '1987-07-30  23:59:59'; -- Specifies the end of the date range.

12.
-- This SQL query formats the current date and time in a specific format.

SELECT
date_format( -- Formats a date and time value according to the specified format.
CURDATE(), -- Retrieves the current date using the CURDATE() function.
        '%W %D %M %Y %T' -- Specifies the desired format for the date and time.
    );


13.
-- This SQL query formats the current date in a specific date format.

SELECT
date_format( -- Formats a date value according to the specified format.
CURDATE(), -- Retrieves the current date using the CURDATE() function.
        '%d/%m/%Y' -- Specifies the desired format for the date.
    );

14.
-- This SQL query formats the current date and time in a specific format.

SELECT
date_format( -- Formats a date and time value according to the specified format.
CURDATE(), -- Retrieves the current date using the CURDATE() function.
        '%l:%i %p %b %e, %Y' -- Specifies the desired format for the date and time.
    );

15.
-- This SQL query retrieves the first name and last name of employees hired in June.

SELECT
first_name, -- Selecting the 'first_name' column from the 'employees' table.
last_name -- Selecting the 'last_name' column from the 'employees' table.
FROM
employees -- Specifying the 'employees' table.
WHERE
MONTH(HIRE_DATE) =  6; -- Filtering the rows to include only those where the hire
date month is June.

16.
-- This SQL query retrieves the year part of the hire date for employees and groups
them by year, filtering only those years where the count of employees hired is
greater than 10.

SELECT
    DATE_FORMAT(HIRE_DATE,'%Y') -- Formats the hire date to extract the year part and
returns it as 'YYYY'.
```

```
FROM
employees -- Specifies the 'employees' table.
GROUP BY
    DATE_FORMAT(HIRE_DATE,'%Y') -- Groups the result set by the year part of the hire
date.
HAVING
COUNT(EMPLOYEE_ID) > 10; -- Filters the grouped results to include only those years
where the count of employees is greater than 10.

17.
-- This SQL query retrieves the first name and hire date of employees hired in the
year 1987.

SELECT
    FIRST_NAME, -- Selecting the 'FIRST_NAME' column from the 'employees' table.
    HIRE_DATE -- Selecting the 'HIRE_DATE' column from the 'employees' table.
FROM
employees -- Specifying the 'employees' table.
WHERE
YEAR(HIRE_DATE)=1987; -- Filtering the rows to include only those where the hire date
year is 1987.

18.
-- This SQL query retrieves the department name, first name, and salary of employees
who have been working for more than 5 years and are also managers of their respective
departments.

SELECT
    DEPARTMENT_NAME, -- Selecting the 'DEPARTMENT_NAME' column from the 'departments'
table.
    FIRST_NAME, -- Selecting the 'FIRST_NAME' column from the 'employees' table.
    SALARY -- Selecting the 'SALARY' column from the 'employees' table.
FROM
departments D -- Specifying the 'departments' table and aliasing it as 'D'.
JOIN
employees E -- Joining the 'employees' table and aliasing it as 'E'.
ON
    (D.MANAGER_ID=E.MANAGER_ID) -- Joining the 'departments' and 'employees' tables
based on the manager ID to associate managers with their departments.
WHERE
    (SYSDATE()-HIRE_DATE) / 365 > 5; -- Filtering the result to include only records
where the duration of employment is more than 5 years.

19.
-- This SQL query retrieves the employee ID, last name, hire date, and the last day
of the month for each hire date.

SELECT
employee_id, -- Selecting the 'employee_id' column from the 'employees' table.
last_name, -- Selecting the 'last_name' column from the 'employees' table.
hire_date, -- Selecting the 'hire_date' column from the 'employees' table.
    LAST_DAY(hire_date) -- Using the LAST_DAY() function to calculate the last day of
the month for each hire date.
FROM
employees; -- Specifying the 'employees' table.
```

```
20.
-- This SQL query retrieves the first name of employees, the current date, their hire
date, and calculates their years of employment.

SELECT
    FIRST_NAME, -- Selecting the 'FIRST_NAME' column from the 'employees' table.
SYSDATE(), -- Retrieving the current date using the SYSDATE() function.
    HIRE_DATE, -- Selecting the 'HIRE_DATE' column from the 'employees' table.
DATEDIFF(SYSDATE(), hire_date)/365 -- Calculating the difference in days between the
current date and the hire date, then dividing by 365 to get years of employment.
FROM
employees; -- Specifying the 'employees' table.

21.
-- This SQL query calculates the count of employees hired in each department for each
year, ordered by department ID.

SELECT
    DEPARTMENT_ID, -- Selecting the 'DEPARTMENT_ID' column from the 'employees'
table.
    DATE_FORMAT(HIRE_DATE,'%Y'), -- Formatting the hire date to extract the year and
selecting it as 'YEAR'.
COUNT(EMPLOYEE_ID) -- Counting the number of employees in each department for each
year.
FROM
employees -- Specifying the 'employees' table.
GROUP BY
    DEPARTMENT_ID, DATE_FORMAT(HIRE_DATE, '%Y') -- Grouping the result set by
department ID and year of hire date.
ORDER BY
    DEPARTMENT_ID; -- Ordering the result set by department ID.

MySQL String :

1.
-- This SQL query selects the job_id and concatenates the employee_id values
separated by a space for each group of job_id.

SELECT
job_id,
    GROUP_CONCAT(employee_id, ' ') AS 'Employees ID' -- Concatenates employee_ids
with a space separator and renames the resulting column as 'Employees ID'
FROM
employees
GROUP BY
job_id; -- Groups the result by job_id, so each row represents a unique job_id with
concatenated employee_ids.
2.
-- This SQL statement updates the 'phone_number' column in the 'employees' table.
-- It replaces any occurrence of the substring '124' in the 'phone_number' column
with '999'.
-- The WHERE clause ensures that only rows with phone numbers containing '124' are
updated.

UPDATE employees
SET phone_number = REPLACE(phone_number, '124', '999')
```

```
WHERE phone_number LIKE '%124%';

3.
-- This SQL query selects all columns from the 'employees' table where the length of
the first name is greater than or equal to 8.

SELECT
    * -- Selecting all columns from the 'employees' table.
FROM
employees -- Specifying the 'employees' table.
WHERE
LENGTH(first_name) >= 8; -- Filtering the rows to include only those where the length
of the first name is greater than or equal to 8.

4.
-- This SQL query selects the job_id, maximum salary, and minimum salary from the
jobs table.
-- The LPAD function is used to left-pad the salary values with zeros to ensure they
have a total width of 7 characters.

SELECT
job_id,
LPAD(max_salary, 7, '0') AS ' Max Salary', -- Left-pads the max_salary column with
zeros to ensure a width of 7 characters, and renames the resulting column as 'Max
Salary'.
LPAD(min_salary, 7, '0') AS ' Min Salary'  -- Left-pads the min_salary column with
zeros to ensure a width of 7 characters, and renames the resulting column as 'Min
Salary'.
FROM
jobs;

5.
-- Updating the email addresses of employees
UPDATE employees
-- Setting the email column to a concatenation of the existing email value and
'@example.com'
SET email = CONCAT(email, '@example.com');

6.
-- Selecting the employee_id, first_name, and hire month from the employees table
SELECT employee_id, first_name,
-- Extracting the month portion of the hire_date using the MID function, starting
from the 6th character and taking 2 characters
MID(hire_date, 6, 2) as hire_month
-- Selecting data from the employees table
FROM employees;

7.
-- This SQL query selects the employee ID and extracts a portion of the reversed
email address to create a new column called Email_ID.

SELECT
employee_id, -- Selecting the employee ID from the employees table.

    -- Reverses the email address, extracts a substring starting from the fourth
character, and then reverses it back.
```

```
REVERSE(SUBSTR(REVERSE(email), 4)) as Email_ID

FROM
employees; -- Specifies the table from which data is being retrieved, in this case,
it's the 'employees' table.

8.
-- This SQL query selects all columns from the employees table where the first name
is in uppercase.

SELECT
    * -- Selecting all columns from the employees table.

FROM
employees -- Specifies the table from which data is being retrieved, in this case,
it's the 'employees' table.

WHERE
first_name = BINARY UPPER(first_name);

9.
-- This SQL query selects the last four digits of the phone numbers stored in the
'phone_number' column and aliases the result as 'Ph.No.'.

SELECT
RIGHT(phone_number, 4) as 'Ph.No.' -- Selecting the last four digits of the phone
numbers and aliasing the result as 'Ph.No.'.

FROM
employees; -- Specifies the table from which data is being retrieved, in this case,
it's the 'employees' table.

10.
-- This SQL query selects the location ID, street address, and extracts the last word
from the street address.

SELECT
location_id, -- Selecting the location ID from the locations table.
street_address, -- Selecting the street address from the locations table.

    -- Extracting the last word from the street address by replacing punctuation
marks with spaces,
    -- splitting the address into words, and then selecting the last word.
    SUBSTRING_INDEX(
REPLACE(
REPLACE(
REPLACE(street_address',',' '),
            ')',' '),
        '(',' '),
' ',-1) AS 'Last--word-of-street_address'

FROM
locations; -- Specifies the table from which data is being retrieved, in this case,
it's the 'locations' table.

11.
```

```sql
-- This SQL query selects all columns from the locations table where the length of
the street address is less than or equal to the minimum length of all street
addresses in the locations table.

SELECT
    * -- Selecting all columns from the locations table.

FROM
locations -- Specifies the table from which data is being retrieved, in this case,
it's the 'locations' table.

WHERE
LENGTH(street_address) <= ( -- Filters the rows where the length of the street
address is less than or equal to...

        SELECT
MIN(LENGTH(street_address)) -- ... the minimum length of all street addresses in the
locations table.

        FROM
locations
    );

12.
-- This SQL query selects a portion of the job title from the jobs table.

SELECT
job_title, -- Selecting the job title from the jobs table.

    -- Extracting the substring from the job title starting from the first character
up to the position of the first space.
SUBSTR(job_title, 1, INSTR(job_title, ' ') - 1)

FROM
jobs; -- Specifies the table from which data is being retrieved, in this case, it's
the 'jobs' table.

13.
-- This SQL query selects the first name and last name of employees whose last name
contains the letter 'C' after the second position.

SELECT
first_name, last_name -- Selecting the first name and last name from the employees
table.

FROM
employees -- Specifies the table from which data is being retrieved, in this case,
it's the 'employees' table.

WHERE
INSTR(last_name,'C') > 2; -- Filters the rows where the letter 'C' is found in the
last name after the second position.

14.
-- This SQL query selects the first name and its length from the employees table for
names starting with 'J', 'M', or 'A', and sorts the result by first name.
```

```sql
SELECT
first_name "Name", -- Selecting the first name and aliasing it as "Name".
LENGTH(first_name) "Length" -- Calculating the length of the first name and aliasing
it as "Length".

FROM
employees -- Specifies the table from which data is being retrieved, in this case,
it's the 'employees' table.

WHERE
first_name LIKE 'J%' -- Filters the rows where the first name starts with 'J'.
    OR first_name LIKE 'M%' -- Filters the rows where the first name starts with 'M'.
    OR first_name LIKE 'A%' -- Filters the rows where the first name starts with 'A'.

ORDER BY
first_name; -- Orders the result set by the first name in ascending order.
```

15.

```sql
-- This SQL query selects the first name of employees and left-pads their salary
values with '$' characters up to a total width of 10 characters.

SELECT
first_name, -- Selecting the first name from the employees table.

    -- Left-pads the salary values with '$' characters up to a total width of 10
characters.
LPAD(salary, 10, '$') SALARY

FROM
employees; -- Specifies the table from which data is being retrieved, in this case,
it's the 'employees' table.
```

16.

```sql
-- This SQL query selects the first 8 characters of the first name, repeats '$'
characters based on the salary divided by 1000, and retrieves the original salary
from the employees table, ordering the result by salary in descending order.

SELECT
LEFT(first_name, 8), -- Selecting the first 8 characters of the first name.

    -- Repeats '$' characters a number of times based on the salary divided by 1000.
REPEAT('$', FLOOR(salary/1000)) 'SALARY($)',

salary -- Selecting the original salary from the employees table.

FROM
employees -- Specifies the table from which data is being retrieved, in this case,
it's the 'employees' table.

ORDER BY
salary DESC; -- Orders the result set by salary in descending order.
```

17.

```sql
-- This SQL query selects the employee ID, first name, last name, and hire date from
the employees table
-- where the hire date contains the month and day "07" in the specified format, '%d
%m %Y'.

SELECT
employee_id, -- Selecting the employee ID from the employees table.
first_name, -- Selecting the first name from the employees table.
last_name, -- Selecting the last name from the employees table.
hire_date -- Selecting the hire date from the employees table.

FROM
employees -- Specifies the table from which data is being retrieved, in this case,
it's the 'employees' table.

WHERE
POSITION("07" IN DATE_FORMAT(hire_date, '%d %m %Y')) > 0;
    -- Checks if the position of "07" exists in the formatted hire date string,
    -- indicating that the month and day of hiring are "07".
```

6. Lab Day 3

```sql
-- lab 03

create database universipedia;
use universipedia;


-- classroom test
select * from classroom;

-- # building, room_number, capacity
-- 'Alumni', '143', '47'
-- 'Alumni', '547', '26'
-- 'black', '431', '1000'
-- 'Bronfman', '700', '12'
-- 'Chandler', '375', '10'
-- 'Chandler', '804', '11'
-- 'dark', '431', '9999'
-- 'Drown', '757', '18'
-- 'Fairchild', '145', '27'
-- 'Garfield', '119', '59'
-- 'Gates', '314', '10'
-- 'Gates', '707', '65'
-- 'Grace', '40', '34'
-- 'Lambeau', '348', '51'
-- 'Lamberton', '134', '10'
-- 'Lamberton', '143', '10'
-- 'Main', '425', '22'
-- 'Main', '45', '30'
-- 'Nassau', '45', '92'
-- 'Painter', '86', '97'
-- 'pink', '431', '9999'
-- 'Polya', '808', '28'
-- 'Power', '717', '12'
-- 'Power', '972', '10'
-- 'Rathbone', '261', '60'
```

```
  -- 'Saucon', '113', '109'
  -- 'Saucon', '180', '15'
  -- 'Saucon', '844', '24'
  -- 'Stabler', '105', '113'
  -- 'Taylor', '183', '71'
  -- 'Taylor', '812', '115'
  -- 'White', '432', '100'
  -- 'Whitman', '134', '120'
  -- 'Whitman', '434', '32'

  describe classroom;

  insert into classroom ( building, room_number, capacity )
  value ("dark", 431, 9999);

  -- 10:29:15 insert into classroom ( building, room_number, capacity ) value
("dark", 431, 9999) 1 row(s) affected 0.0036 sec

  CREATE TABLE department (
    dept_name VARCHAR(20),
    building VARCHAR(15),
    budget NUMERIC(12 , 2 ),
    PRIMARY KEY (dept_name)
  );

  CREATE TABLE department (
    dept_name VARCHAR(20),
    building VARCHAR(15),
    budget NUMERIC(12 , 2 ),
    PRIMARY KEY (dept_name)
  );

  create table course (
    course_id varchar(7),
    title varchar(50),
    dept_name varchar (20),
    credits numeric (2, 0),
    primary key (course_id),
    foreign key (dept_name) references department(dept_name)
  );

  create table instructor (
    ID varchar(5),
    number varchar (20) not null,
    dept_name varchar (20),
    salary numeric(8,2),
    primary key (ID),
    foreign key (dept_name) references department
  );

  CREATE TABLE section (
    course_id VARCHAR(8),
    sec_id VARCHAR(8),
    semester VARCHAR(6),
    year NUMERIC(4 , 0 ),
    building VARCHAR(15),
```

```
    room_number VARCHAR(7),
    time_slot_id VARCHAR(4),
    PRIMARY KEY (course_id , sec_id , semester , year),
    FOREIGN KEY (course_id)
      REFERENCES course
  );

  create table teaches (
    ID varchar (5),
    course_id varchar(8),
    sec_id varchar(8),
    semester varchar(6),
    year numeric (4, 0),
    primary key (ID, course_id, sec_id, semester, year),
    foreign key (course_id, sec_id, semester, year) references section,
    foreign key (ID) references instructor
  );

  DROP TABLE IF EXISTS parent;

  CREATE TABLE parent (
    Name VARCHAR(1),
    Age NUMERIC(2, 0),
    PRIMARY KEY (Name, Age)
  );

  DESCRIBE parent;

  CREATE TABLE child (
    Name VARCHAR(1),
    Age NUMERIC(2, 0),
    PRIMARY KEY (Name, Age),
    FOREIGN KEY (Name, Age)
      REFERENCES parent (Name, Age)
  );

  drop table child;
  desc child;

  -- 11:01:04 create table course (  course_id varchar(7),     title varchar(50),
dept_name varchar (20),     credits numeric (2, 0),     primary key (course_id),
foreign key (dept_name) references department(dept_name) ) 0 row(s) affected 0.020
sec
  -- 11:03:46 create table instructor (  ID varchar(5),     number varchar (20) not
null,     dept_name varchar (20),     salary numeric(8,2),     primary key (ID),
foreign key (dept_name) references department )  0 row(s) affected 0.022 sec
  -- 11:06:53 create table section (  course_id varchar(8),     sec_id varchar(8),
semester varchar (6),     year numeric(4, 0),     building varchar (15),
room_number varchar(7),     time_slot_id varchar(4),     primary key (course_id,
sec_id, semester, year),     foreign key (course_id) references course ) 0 row(s)
affected 0.025 sec
  -- 11:14:51 create table teaches (  ID varchar (5),     course_id varchar(8),
sec_id varchar(8),     semester varchar(6),     year numeric (4, 0),     primary key
(ID, course_id, sec_id, semester, year),     foreign key (course_id, sec_id,
semester, year) references section,     foreign key (ID) references instructor )  0
row(s) affected 0.028 sec
```

```
-- use university
use university;

-- select name from instructor
select name from instructor;

-- 'Lembr'
-- 'Bawa'
-- 'Yazdi'
-- 'Wieland'
-- 'DAgostino'
-- 'Liley'
-- 'Kean'
-- 'Atanassov'
-- 'Moreira'
-- 'Gustafsson'
-- 'Bourrier'
-- 'Bondi'
-- 'Soisalon-Soininen'
-- 'Morris'
-- 'Arias'
-- 'Murata'
-- 'Tung'
-- 'Luo'
-- 'Vicentino'
-- 'Romero'
-- 'Lent'
-- 'Sarkar'
-- 'Shuming'
-- 'Konstantinides'
-- 'Bancilhon'
-- 'Hau'
-- 'Dusserre'
-- 'Desyl'
-- 'Jaekel'
-- 'McKinnon'
-- 'Gutierrez'
-- 'Mingoz'
-- 'Pimenta'
-- 'Yin'
-- 'Sullivan'
-- 'Voronina'
-- 'Kenje'
-- 'Mahmoud'
-- 'Pingr'
-- 'Ullman '
-- 'Levine'
-- 'Queiroz'
-- 'Valtchev'
-- 'Bietzk'
-- 'Choll'
-- 'Arinb'
-- 'Sakurai'
-- 'Mird'
-- 'Bertolino'
```

```
-- 'Dale'

-- select departments from instructor
select dept_name from instructor;

-- 'Accounting'
-- 'Accounting'
-- 'Accounting'
-- 'Accounting'
-- 'Astronomy'
-- 'Athletics'
-- 'Athletics'
-- 'Athletics'
-- 'Athletics'
-- 'Athletics'
-- 'Biology'
-- 'Biology'
-- 'Comp. Sci.'
-- 'Comp. Sci.'
-- 'Cybernetics'
-- 'Cybernetics'
-- 'Cybernetics'
-- 'Cybernetics'
-- 'Elec. Eng.'
-- 'Elec. Eng.'
-- 'Elec. Eng.'
-- 'Elec. Eng.'
-- 'English'
-- 'English'
-- 'English'
-- 'English'
-- 'Finance'
-- 'Geology'
-- 'Languages'
-- 'Languages'
-- 'Languages'
-- 'Marketing'
-- 'Marketing'
-- 'Marketing'
-- 'Marketing'
-- 'Mech. Eng.'
-- 'Mech. Eng.'
-- 'Physics'
-- 'Physics'
-- 'Pol. Sci.'
-- 'Pol. Sci.'
-- 'Pol. Sci.'
-- 'Psychology'
-- 'Psychology'
-- 'Statistics'
-- 'Statistics'
-- 'Statistics'
-- 'Statistics'
-- 'Statistics'
-- 'Statistics'
```

```sql
-- select department names but with all
select all dept_name from instructor;

-- 'Accounting'
-- 'Accounting'
-- 'Accounting'
-- 'Accounting'
-- 'Astronomy'
-- 'Athletics'
-- 'Athletics'
-- 'Athletics'
-- 'Athletics'
-- 'Athletics'
-- 'Biology'
-- 'Biology'
-- 'Comp. Sci.'
-- 'Comp. Sci.'
-- 'Cybernetics'
-- 'Cybernetics'
-- 'Cybernetics'
-- 'Cybernetics'
-- 'Elec. Eng.'
-- 'Elec. Eng.'
-- 'Elec. Eng.'
-- 'Elec. Eng.'
-- 'English'
-- 'English'
-- 'English'
-- 'English'
-- 'Finance'
-- 'Geology'
-- 'Languages'
-- 'Languages'
-- 'Languages'
-- 'Marketing'
-- 'Marketing'
-- 'Marketing'
-- 'Marketing'
-- 'Mech. Eng.'
-- 'Mech. Eng.'
-- 'Physics'
-- 'Physics'
-- 'Pol. Sci.'
-- 'Pol. Sci.'
-- 'Pol. Sci.'
-- 'Psychology'
-- 'Psychology'
-- 'Statistics'
-- 'Statistics'
-- 'Statistics'
-- 'Statistics'
-- 'Statistics'
-- 'Statistics'

-- select only distinct departments from instructor
select distinct dept_name from instructor;
```

```
-- 'Accounting'
-- 'Astronomy'
-- 'Athletics'
-- 'Biology'
-- 'Comp. Sci.'
-- 'Cybernetics'
-- 'Elec. Eng.'
-- 'English'
-- 'Finance'
-- 'Geology'
-- 'Languages'
-- 'Marketing'
-- 'Mech. Eng.'
-- 'Physics'
-- 'Pol. Sci.'
-- 'Psychology'
-- 'Statistics'

-- select all districts from instructor
select distinct * from instructor;

-- '14365', 'Lembr', 'Accounting', '32241.56'
-- '15347', 'Bawa', 'Athletics', '72140.88'
-- '16807', 'Yazdi', 'Athletics', '98333.65'
-- '19368', 'Wieland', 'Pol. Sci.', '124651.41'
-- '22591', 'DAgostino', 'Psychology', '59706.49'
-- '25946', 'Liley', 'Languages', '90891.69'
-- '28097', 'Kean', 'English', '35023.18'
-- '28400', 'Atanassov', 'Statistics', '84982.92'
-- '31955', 'Moreira', 'Accounting', '71351.42'
-- '3199', 'Gustafsson', 'Elec. Eng.', '82534.37'
-- '3335', 'Bourrier', 'Comp. Sci.', '80797.83'
-- '34175', 'Bondi', 'Comp. Sci.', '115469.11'
-- '35579', 'Soisalon-Soininen', 'Psychology', '62579.61'
-- '36897', 'Morris', 'Marketing', '43770.36'
-- '37687', 'Arias', 'Statistics', '104563.38'
-- '4034', 'Murata', 'Athletics', '61387.56'
-- '41930', 'Tung', 'Athletics', '50482.03'
-- '4233', 'Luo', 'English', '88791.45'
-- '42782', 'Vicentino', 'Elec. Eng.', '34272.67'
-- '43779', 'Romero', 'Astronomy', '79070.08'
-- '48507', 'Lent', 'Mech. Eng.', '107978.47'
-- '48570', 'Sarkar', 'Pol. Sci.', '87549.80'
-- '50330', 'Shuming', 'Physics', '108011.81'
-- '50885', 'Konstantinides', 'Languages', '32570.50'
-- '52647', 'Bancilhon', 'Pol. Sci.', '87958.01'
-- '57180', 'Hau', 'Accounting', '43966.29'
-- '58558', 'Dusserre', 'Marketing', '66143.25'
-- '59795', 'Desyl', 'Languages', '48803.38'
-- '63287', 'Jaekel', 'Athletics', '103146.87'
-- '63395', 'McKinnon', 'Cybernetics', '94333.99'
-- '64871', 'Gutierrez', 'Statistics', '45310.53'
-- '6569', 'Mingoz', 'Finance', '105311.38'
-- '65931', 'Pimenta', 'Cybernetics', '79866.95'
-- '72553', 'Yin', 'English', '46397.59'
```

```
-- '73623', 'Sullivan', 'Elec. Eng.', '90038.09'
-- '74420', 'Voronina', 'Physics', '121141.99'
-- '74426', 'Kenje', 'Marketing', '106554.73'
-- '77346', 'Mahmoud', 'Geology', '99382.59'
-- '78699', 'Pingr', 'Statistics', '59303.62'
-- '79081', 'Ullman ', 'Accounting', '47307.10'
-- '79653', 'Levine', 'Elec. Eng.', '89805.83'
-- '80759', 'Queiroz', 'Biology', '45538.32'
-- '81991', 'Valtchev', 'Biology', '77036.18'
-- '90376', 'Bietzk', 'Cybernetics', '117836.50'
-- '90643', 'Choll', 'Statistics', '57807.09'
-- '95030', 'Arinb', 'Statistics', '54805.11'
-- '95709', 'Sakurai', 'English', '118143.98'
-- '96895', 'Mird', 'Marketing', '119921.41'
-- '97302', 'Bertolino', 'Mech. Eng.', '51647.57'
-- '99052', 'Dale', 'Cybernetics', '93348.83'

-- list all properties of instructor
describe instructor;

-- 'ID', 'varchar(5)', 'NO', 'PRI', NULL, ''
-- 'name', 'varchar(20)', 'NO', '', NULL, ''
-- 'dept_name', 'varchar(20)', 'YES', 'MUL', NULL, ''
-- 'salary', 'decimal(8,2)', 'YES', '', NULL, ''

-- select all from instructor but with their salaries multiplied by 0
select name, dept_name, salary * 0 from instructor;

-- 'Lembr', 'Accounting', '0.00'
-- 'Bawa', 'Athletics', '0.00'
-- 'Yazdi', 'Athletics', '0.00'
-- 'Wieland', 'Pol. Sci.', '0.00'
-- 'DAgostino', 'Psychology', '0.00'
-- 'Liley', 'Languages', '0.00'
-- 'Kean', 'English', '0.00'
-- 'Atanassov', 'Statistics', '0.00'
-- 'Moreira', 'Accounting', '0.00'
-- 'Gustafsson', 'Elec. Eng.', '0.00'
-- 'Bourrier', 'Comp. Sci.', '0.00'
-- 'Bondi', 'Comp. Sci.', '0.00'
-- 'Soisalon-Soininen', 'Psychology', '0.00'
-- 'Morris', 'Marketing', '0.00'
-- 'Arias', 'Statistics', '0.00'
-- 'Murata', 'Athletics', '0.00'
-- 'Tung', 'Athletics', '0.00'
-- 'Luo', 'English', '0.00'
-- 'Vicentino', 'Elec. Eng.', '0.00'
-- 'Romero', 'Astronomy', '0.00'
-- 'Lent', 'Mech. Eng.', '0.00'
-- 'Sarkar', 'Pol. Sci.', '0.00'
-- 'Shuming', 'Physics', '0.00'
-- 'Konstantinides', 'Languages', '0.00'
-- 'Bancilhon', 'Pol. Sci.', '0.00'
-- 'Hau', 'Accounting', '0.00'
-- 'Dusserre', 'Marketing', '0.00'
-- 'Desyl', 'Languages', '0.00'
```

```
-- 'Jaekel', 'Athletics', '0.00'
-- 'McKinnon', 'Cybernetics', '0.00'
-- 'Gutierrez', 'Statistics', '0.00'
-- 'Mingoz', 'Finance', '0.00'
-- 'Pimenta', 'Cybernetics', '0.00'
-- 'Yin', 'English', '0.00'
-- 'Sullivan', 'Elec. Eng.', '0.00'
-- 'Voronina', 'Physics', '0.00'
-- 'Kenje', 'Marketing', '0.00'
-- 'Mahmoud', 'Geology', '0.00'
-- 'Pingr', 'Statistics', '0.00'
-- 'Ullman ', 'Accounting', '0.00'
-- 'Levine', 'Elec. Eng.', '0.00'
-- 'Queiroz', 'Biology', '0.00'
-- 'Valtchev', 'Biology', '0.00'
-- 'Bietzk', 'Cybernetics', '0.00'
-- 'Choll', 'Statistics', '0.00'
-- 'Arinb', 'Statistics', '0.00'
-- 'Sakurai', 'English', '0.00'
-- 'Mird', 'Marketing', '0.00'
-- 'Bertolino', 'Mech. Eng.', '0.00'
-- 'Dale', 'Cybernetics', '0.00'


-- selecting instructor whose dept name is comp sci and salary is 70k
select name from instructor where dept_name = 'Comp. Sci.' and salary > 70000;

-- 'Bourrier'
-- 'Bondi'

-- custom names support
select T.name, S.course_id
from instructor as T, teaches as S
where T .ID= S.ID;

-- # name, course_id
-- 'Romero', '105'
-- 'Romero', '105'
-- 'Mingoz', '137'
-- 'Bietzk', '158'
-- 'Dale', '158'
-- 'Gustafsson', '169'
-- 'Gustafsson', '169'
-- 'Liley', '192'
-- 'Lembr', '200'
-- 'Ullman ', '200'
-- 'Dale', '237'
-- 'Dale', '237'
-- 'Voronina', '239'
-- 'Morris', '242'
-- 'Sakurai', '258'
-- 'Sakurai', '270'
-- 'Bondi', '274'
-- 'Mingoz', '304'
-- 'Morris', '313'
-- 'Mingoz', '319'
```

```
-- 'Jaekel', '334'
-- 'DAgostino', '338'
-- 'DAgostino', '338'
-- 'Ullman ', '345'
-- 'Mingoz', '349'
-- 'DAgostino', '352'
-- 'Mingoz', '362'
-- 'Mingoz', '362'
-- 'Mingoz', '362'
-- 'Kean', '366'
-- 'Voronina', '376'
-- 'DAgostino', '400'
-- 'DAgostino', '400'
-- 'Tung', '401'
-- 'Ullman ', '408'
-- 'Ullman ', '408'
-- 'Valtchev', '415'
-- 'Tung', '421'
-- 'Mingoz', '426'
-- 'Voronina', '443'
-- 'Voronina', '443'
-- 'Mingoz', '445'
-- 'Bawa', '457'
-- 'Choll', '461'
-- 'Sakurai', '468'
-- 'Shuming', '468'
-- 'Romero', '476'
-- 'DAgostino', '482'
-- 'Mahmoud', '486'
-- 'Romero', '489'
-- 'Mahmoud', '493'
-- 'Dale', '496'
-- 'Mingoz', '527'
-- 'Wieland', '545'
-- 'Queiroz', '559'
-- 'Gustafsson', '561'
-- 'Bondi', '571'
-- 'Wieland', '581'
-- 'Wieland', '591'
-- 'DAgostino', '599'
-- 'Atanassov', '603'
-- 'Atanassov', '604'
-- 'Voronina', '612'
-- 'Lent', '626'
-- 'Dale', '629'
-- 'Gustafsson', '631'
-- 'DAgostino', '642'
-- 'DAgostino', '663'
-- 'Luo', '679'
-- 'Tung', '692'
-- 'Sullivan', '694'
-- 'Morris', '696'
-- 'Valtchev', '702'
-- 'Mahmoud', '704'
-- 'Mahmoud', '735'
-- 'Mahmoud', '735'
```

```
-- 'Bondi', '747'
-- 'Dale', '748'
-- 'Ullman ', '760'
-- 'Morris', '791'
-- 'Vicentino', '793'
-- 'Morris', '795'
-- 'Dale', '802'
-- 'Kean', '808'
-- 'Lembr', '843'
-- 'Jaekel', '852'
-- 'Mahmoud', '864'
-- 'DAgostino', '867'
-- 'Sarkar', '867'
-- 'Pimenta', '875'
-- 'Dale', '893'
-- 'Dale', '927'
-- 'Bourrier', '949'
-- 'Voronina', '959'
-- 'Bourrier', '960'
-- 'Sakurai', '960'
-- 'DAgostino', '962'
-- 'DAgostino', '972'
-- 'Ullman ', '974'
-- 'DAgostino', '991'

select distinct T.name
from instructor as T , instructor as S
where T.salary > S.salary and S.dept_name = 'Biology';

-- # name
-- 'Bawa'
-- 'Yazdi'
-- 'Wieland'
-- 'DAgostino'
-- 'Liley'
-- 'Atanassov'
-- 'Moreira'
-- 'Gustafsson'
-- 'Bourrier'
-- 'Bondi'
-- 'Soisalon-Soininen'
-- 'Arias'
-- 'Murata'
-- 'Tung'
-- 'Luo'
-- 'Romero'
-- 'Lent'
-- 'Sarkar'
-- 'Shuming'
-- 'Bancilhon'
-- 'Dusserre'
-- 'Desyl'
-- 'Jaekel'
-- 'McKinnon'
-- 'Mingoz'
-- 'Pimenta'
```

```
-- 'Yin'
-- 'Sullivan'
-- 'Voronina'
-- 'Kenje'
-- 'Mahmoud'
-- 'Pingr'
-- 'Ullman '
-- 'Levine'
-- 'Valtchev'
-- 'Bietzk'
-- 'Choll'
-- 'Arinb'
-- 'Sakurai'
-- 'Mird'
-- 'Bertolino'
-- 'Dale'

-- selecting building with custom like
select distinct dept_name, building
from department
where building like '%auc_n%';

-- # dept_name, building
-- 'Accounting', 'Saucon'

-- select all order by salary descending and name ascending
select *
from instructor
order by salary desc, name asc;

-- # ID, name, dept_name, salary
-- '19368', 'Wieland', 'Pol. Sci.', '124651.41'
-- '74420', 'Voronina', 'Physics', '121141.99'
-- '96895', 'Mird', 'Marketing', '119921.41'
-- '95709', 'Sakurai', 'English', '118143.98'
-- '90376', 'Bietzk', 'Cybernetics', '117836.50'
-- '34175', 'Bondi', 'Comp. Sci.', '115469.11'
-- '50330', 'Shuming', 'Physics', '108011.81'
-- '48507', 'Lent', 'Mech. Eng.', '107978.47'
-- '74426', 'Kenje', 'Marketing', '106554.73'
-- '6569', 'Mingoz', 'Finance', '105311.38'
-- '37687', 'Arias', 'Statistics', '104563.38'
-- '63287', 'Jaekel', 'Athletics', '103146.87'
-- '77346', 'Mahmoud', 'Geology', '99382.59'
-- '16807', 'Yazdi', 'Athletics', '98333.65'
-- '63395', 'McKinnon', 'Cybernetics', '94333.99'
-- '99052', 'Dale', 'Cybernetics', '93348.83'
-- '25946', 'Liley', 'Languages', '90891.69'
-- '73623', 'Sullivan', 'Elec. Eng.', '90038.09'
-- '79653', 'Levine', 'Elec. Eng.', '89805.83'
-- '4233', 'Luo', 'English', '88791.45'
-- '52647', 'Bancilhon', 'Pol. Sci.', '87958.01'
-- '48570', 'Sarkar', 'Pol. Sci.', '87549.80'
-- '28400', 'Atanassov', 'Statistics', '84982.92'
-- '3199', 'Gustafsson', 'Elec. Eng.', '82534.37'
-- '3335', 'Bourrier', 'Comp. Sci.', '80797.83'
```

```
-- '65931', 'Pimenta', 'Cybernetics', '79866.95'
-- '43779', 'Romero', 'Astronomy', '79070.08'
-- '81991', 'Valtchev', 'Biology', '77036.18'
-- '15347', 'Bawa', 'Athletics', '72140.88'
-- '31955', 'Moreira', 'Accounting', '71351.42'
-- '58558', 'Dusserre', 'Marketing', '66143.25'
-- '35579', 'Soisalon-Soininen', 'Psychology', '62579.61'
-- '4034', 'Murata', 'Athletics', '61387.56'
-- '22591', 'DAgostino', 'Psychology', '59706.49'
-- '78699', 'Pingr', 'Statistics', '59303.62'
-- '90643', 'Choll', 'Statistics', '57807.09'
-- '95030', 'Arinb', 'Statistics', '54805.11'
-- '97302', 'Bertolino', 'Mech. Eng.', '51647.57'
-- '41930', 'Tung', 'Athletics', '50482.03'
-- '59795', 'Desyl', 'Languages', '48803.38'
-- '79081', 'Ullman ', 'Accounting', '47307.10'
-- '72553', 'Yin', 'English', '46397.59'
-- '80759', 'Queiroz', 'Biology', '45538.32'
-- '64871', 'Gutierrez', 'Statistics', '45310.53'
-- '57180', 'Hau', 'Accounting', '43966.29'
-- '36897', 'Morris', 'Marketing', '43770.36'
-- '28097', 'Kean', 'English', '35023.18'
-- '42782', 'Vicentino', 'Elec. Eng.', '34272.67'
-- '50885', 'Konstantinides', 'Languages', '32570.50'
-- '14365', 'Lembr', 'Accounting', '32241.56'

-- select instructor name with defined salary
select name
from instructor
where salary between 90000 and 100000;

-- # name
-- 'Yazdi'
-- 'Liley'
-- 'McKinnon'
-- 'Sullivan'
-- 'Mahmoud'
-- 'Dale'

-- union to combine 2 diff tables
(select course_id, semester
from section
where semester = 'Fall' and year= 2007)
union
(select course_id, semester
from section
where semester = 'Spring' and year= 2008);

-- # course_id, semester
-- '893', 'Fall'
-- '489', 'Fall'
-- '612', 'Fall'
-- '258', 'Fall'
-- '468', 'Fall'
-- '949', 'Fall'
-- '362', 'Spring'
```

```
-- '852', 'Spring'
-- '991', 'Spring'
-- '962', 'Spring'
-- '237', 'Spring'
-- '349', 'Spring'
-- '345', 'Spring'
-- '158', 'Spring'
-- '704', 'Spring'

-- union to combine 2 diff tables but with duplicates
(select course_id, semester
from section
where semester = 'Fall' and year= 2007)
union all
(select course_id, semester
from section
where semester = 'Spring' and year= 2008);

-- # course_id, semester
-- '893', 'Fall'
-- '489', 'Fall'
-- '612', 'Fall'
-- '258', 'Fall'
-- '468', 'Fall'
-- '949', 'Fall'
-- '362', 'Spring'
-- '852', 'Spring'
-- '991', 'Spring'
-- '962', 'Spring'
-- '237', 'Spring'
-- '349', 'Spring'
-- '345', 'Spring'
-- '158', 'Spring'
-- '704', 'Spring'

-- same thing just course_id
(select course_id
from section
where semester = 'Fall')
union all
(select course_id
from section
where semester = 'Spring');

-- # course_id
-- '893'
-- '489'
-- '612'
-- '258'
-- '468'
-- '949'
-- '362'
-- '852'
-- '991'
-- '962'
-- '237'
```

```
  -- '349'
  -- '345'
  -- '158'
  -- '704'

  -- intersecting two tables
  (select course_id
  from section
  where semester = 'Fall' order by course_id)
  intersect all
  (select course_id
  from section
  where semester = 'Spring' order by course_id);

  -- # course_id
  -- '362'
  -- '200'
  -- '169'
  -- '237'
  -- '400'
  -- '158'

  describe instructor;

  -- finds the average salary
  select avg (salary)
  from instructor
  where dept_name = 'Comp. Sci.';

  -- # avg (salary)
  -- '98133.470000'

  SELECT sum(value)
  FROM (
    SELECT 2 AS value
    UNION ALL
    SELECT 1
    UNION ALL
    SELECT 3
  ) AS temp_table;
```

7. Lab day 4

```
use university;

-- select instructor's dept name and building from 2 tables,
-- where these 2 are same
select name, instructor.dept_name, building
from instructor, department
where instructor.dept_name= department.dept_name;

-- 'Lembr', 'Accounting', 'Saucon'
-- 'Bawa', 'Athletics', 'Bronfman'
-- 'Yazdi', 'Athletics', 'Bronfman'
-- 'Wieland', 'Pol. Sci.', 'Whitman'
-- 'DAgostino', 'Psychology', 'Thompson'
-- 'Liley', 'Languages', 'Linderman'
```

```
-- 'Kean', 'English', 'Palmer'
-- 'Atanassov', 'Statistics', 'Taylor'
-- 'Moreira', 'Accounting', 'Saucon'
-- 'Gustafsson', 'Elec. Eng.', 'Main'
-- 'Bourrier', 'Comp. Sci.', 'Lamberton'
-- 'Bondi', 'Comp. Sci.', 'Lamberton'
-- 'Soisalon-Soininen', 'Psychology', 'Thompson'
-- 'Morris', 'Marketing', 'Lambeau'
-- 'Arias', 'Statistics', 'Taylor'
-- 'Murata', 'Athletics', 'Bronfman'
-- 'Tung', 'Athletics', 'Bronfman'
-- 'Luo', 'English', 'Palmer'
-- 'Vicentino', 'Elec. Eng.', 'Main'
-- 'Romero', 'Astronomy', 'Taylor'
-- 'Lent', 'Mech. Eng.', 'Rauch'
-- 'Sarkar', 'Pol. Sci.', 'Whitman'
-- 'Shuming', 'Physics', 'Wrigley'
-- 'Konstantinides', 'Languages', 'Linderman'
-- 'Bancilhon', 'Pol. Sci.', 'Whitman'
-- 'Hau', 'Accounting', 'Saucon'
-- 'Dusserre', 'Marketing', 'Lambeau'
-- 'Desyl', 'Languages', 'Linderman'
-- 'Jaekel', 'Athletics', 'Bronfman'
-- 'McKinnon', 'Cybernetics', 'Mercer'
-- 'Gutierrez', 'Statistics', 'Taylor'
-- 'Mingoz', 'Finance', 'Candlestick'
-- 'Pimenta', 'Cybernetics', 'Mercer'
-- 'Yin', 'English', 'Palmer'
-- 'Sullivan', 'Elec. Eng.', 'Main'
-- 'Voronina', 'Physics', 'Wrigley'
-- 'Kenje', 'Marketing', 'Lambeau'
-- 'Mahmoud', 'Geology', 'Palmer'
-- 'Pingr', 'Statistics', 'Taylor'
-- 'Ullman ', 'Accounting', 'Saucon'
-- 'Levine', 'Elec. Eng.', 'Main'
-- 'Queiroz', 'Biology', 'Candlestick'
-- 'Valtchev', 'Biology', 'Candlestick'
-- 'Bietzk', 'Cybernetics', 'Mercer'
-- 'Choll', 'Statistics', 'Taylor'
-- 'Arinb', 'Statistics', 'Taylor'
-- 'Sakurai', 'English', 'Palmer'
-- 'Mird', 'Marketing', 'Lambeau'
-- 'Bertolino', 'Mech. Eng.', 'Rauch'
-- 'Dale', 'Cybernetics', 'Mercer'

select T.name, S.course_id
from instructor as T , teaches as S
where T.ID= S.ID;

-- # name, course_id
-- 'Lembr', '200'
-- 'Lembr', '843'
-- 'Bawa', '457'
-- 'Wieland', '545'
-- 'Wieland', '581'
-- 'Wieland', '591'
```

```
-- 'DAgostino', '338'
-- 'DAgostino', '338'
-- 'DAgostino', '352'
-- 'DAgostino', '400'
-- 'DAgostino', '400'
-- 'DAgostino', '482'
-- 'DAgostino', '599'
-- 'DAgostino', '642'
-- 'DAgostino', '663'
-- 'DAgostino', '867'
-- 'DAgostino', '962'
-- 'DAgostino', '972'
-- 'DAgostino', '991'
-- 'Liley', '192'
-- 'Kean', '366'
-- 'Kean', '808'
-- 'Atanassov', '603'
-- 'Atanassov', '604'
-- 'Gustafsson', '169'
-- 'Gustafsson', '169'
-- 'Gustafsson', '561'
-- 'Gustafsson', '631'
-- 'Bourrier', '949'
-- 'Bourrier', '960'
-- 'Bondi', '274'
-- 'Bondi', '571'
-- 'Bondi', '747'
-- 'Morris', '242'
-- 'Morris', '313'
-- 'Morris', '696'
-- 'Morris', '791'
-- 'Morris', '795'
-- 'Tung', '401'
-- 'Tung', '421'
-- 'Tung', '692'
-- 'Luo', '679'
-- 'Vicentino', '793'
-- 'Romero', '105'
-- 'Romero', '105'
-- 'Romero', '476'
-- 'Romero', '489'
-- 'Lent', '626'
-- 'Sarkar', '867'
-- 'Shuming', '468'
-- 'Jaekel', '334'
-- 'Jaekel', '852'
-- 'Mingoz', '137'
-- 'Mingoz', '304'
-- 'Mingoz', '319'
-- 'Mingoz', '349'
-- 'Mingoz', '362'
-- 'Mingoz', '362'
-- 'Mingoz', '362'
-- 'Mingoz', '426'
-- 'Mingoz', '445'
-- 'Mingoz', '527'
```

```
-- 'Pimenta', '875'
-- 'Sullivan', '694'
-- 'Voronina', '239'
-- 'Voronina', '376'
-- 'Voronina', '443'
-- 'Voronina', '443'
-- 'Voronina', '612'
-- 'Voronina', '959'
-- 'Mahmoud', '486'
-- 'Mahmoud', '493'
-- 'Mahmoud', '704'
-- 'Mahmoud', '735'
-- 'Mahmoud', '735'
-- 'Mahmoud', '864'
-- 'Ullman ', '200'
-- 'Ullman ', '345'
-- 'Ullman ', '408'
-- 'Ullman ', '408'
-- 'Ullman ', '760'
-- 'Ullman ', '974'
-- 'Queiroz', '559'
-- 'Valtchev', '415'
-- 'Valtchev', '702'
-- 'Bietzk', '158'
-- 'Choll', '461'
-- 'Sakurai', '258'
-- 'Sakurai', '270'
-- 'Sakurai', '468'
-- 'Sakurai', '960'
-- 'Dale', '158'
-- 'Dale', '237'
-- 'Dale', '237'
-- 'Dale', '496'
-- 'Dale', '629'
-- 'Dale', '748'
-- 'Dale', '802'
-- 'Dale', '893'
-- 'Dale', '927'

select distinct T .name
from instructor as T , instructor as S
where T.salary > S.salary and S.dept_name = 'Biology';

-- # name
-- 'Bawa'
-- 'Yazdi'
-- 'Wieland'
-- 'DAgostino'
-- 'Liley'
-- 'Atanassov'
-- 'Moreira'
-- 'Gustafsson'
-- 'Bourrier'
-- 'Bondi'
-- 'Soisalon-Soininen'
-- 'Arias'
```

```
-- 'Murata'
-- 'Tung'
-- 'Luo'
-- 'Romero'
-- 'Lent'
-- 'Sarkar'
-- 'Shuming'
-- 'Bancilhon'
-- 'Dusserre'
-- 'Desyl'
-- 'Jaekel'
-- 'McKinnon'
-- 'Mingoz'
-- 'Pimenta'
-- 'Yin'
-- 'Sullivan'
-- 'Voronina'
-- 'Kenje'
-- 'Mahmoud'
-- 'Pingr'
-- 'Ullman '
-- 'Levine'
-- 'Valtchev'
-- 'Bietzk'
-- 'Choll'
-- 'Arinb'
-- 'Sakurai'
-- 'Mird'
-- 'Bertolino'
-- 'Dale'

select dept_name
from department
where building like '%a%';

-- # dept_name
-- 'Accounting'
-- 'Astronomy'
-- 'Athletics'
-- 'Biology'
-- 'Civil Eng.'
-- 'Comp. Sci.'
-- 'Elec. Eng.'
-- 'English'
-- 'Finance'
-- 'Geology'
-- 'History'
-- 'Languages'
-- 'Marketing'
-- 'Math'
-- 'Mech. Eng.'
-- 'Pol. Sci.'
-- 'Statistics'

select count (distinct_ID)
from teaches
```

```sql
where semester = 'Spring';

describe teaches;


select course_ID
from teaches
where semester = 'Spring';

select count(*)
from course;

select dept_name, AVG(salary) as avg_salary
  from instructor
    group by dept_name;

select dept_name, avg_salary
from (select dept_name, AVG(salary) as avg_salary
      from instructor
      group by dept_name) as T(dept_name, avg_salary)
      where avg_salary>42000;

-- select all distinct course_id
-- where semester is Fall
select distinct course_id
from section
where semester = 'Fall' and
course_id not in (select course_id
from section
where semester = 'Spring');

-- # course_id
-- '694'
-- '105'
-- '313'
-- '476'
-- '242'
-- '843'
-- '893'
-- '421'
-- '468'
-- '415'
-- '559'
-- '867'
-- '960'
-- '304'
-- '489'
-- '612'
-- '626'
-- '274'
-- '461'
-- '258'
-- '561'
-- '192'
-- '808'
-- '974'
```

```
-- '376'
-- '527'
-- '642'
-- '401'
-- '545'
-- '748'
-- '927'
-- '949'
-- '959'
-- '366'
-- '239'
-- '334'
-- '496'
-- '603'
-- '486'
-- '482'

--
select distinct course_id
from section
where semester = 'Fall' and
course_id not in (select course_id
from section
where semester = 'Spring');

-- # course_id
-- '694'
-- '105'
-- '313'
-- '476'
-- '242'
-- '843'
-- '893'
-- '421'
-- '468'
-- '415'
-- '559'
-- '867'
-- '960'
-- '304'
-- '489'
-- '612'
-- '626'
-- '274'
-- '461'
-- '258'
-- '561'
-- '192'
-- '808'
-- '974'
-- '376'
-- '527'
-- '642'
-- '401'
-- '545'
-- '748'
```

```
-- '927'
-- '949'
-- '959'
-- '366'
-- '239'
-- '334'
-- '496'
-- '603'
-- '486'
-- '482'

select * from section;

select distinct course_id
from section
where semester = 'Fall' and
course_id in (select course_id from section where semester = 'Spring');

select S.ID, S.name
from student as S
where exists ((select course_id
from course
where dept_name = 'Biology')
except
(select T.course_id
from takes as T
where S.ID = T.ID));
```

8. Lab day 5

```
use universipedia;
use university;

-- simple instruction to list all instructors
select name
from instructor;

SELECT
    name, course_id
FROM
    student,
    takes
WHERE
    student.ID = takes.ID;

SELECT
    name, course_id
FROM
    student
        NATURAL JOIN
    takes;

-- Find the names of those departments whose budget is higher than that of Astronomy.
List them
-- in alphabetic order.

SELECT
```

```
    dept_name
FROM
    department
WHERE
    budget > (SELECT
            budget
        FROM
            department
        WHERE
            dept_name = 'Astronomy');

-- # dept_name
-- 'Athletics'
-- 'Biology'
-- 'Cybernetics'
-- 'Finance'
-- 'History'
-- 'Math'
-- 'Physics'
-- 'Psychology'

SELECT
    *
FROM
    instructor;

SELECT
    I.ID, COUNT(*)
FROM
    instructor AS I,
    teaches AS T
WHERE
    I.ID = T.ID
GROUP BY I.ID
ORDER BY I.ID;

SELECT
    I.ID, COUNT(T.ID) as number_of_sections
FROM
    instructor AS I
        NATURAL LEFT JOIN
    teaches AS T
GROUP BY I.ID
ORDER BY number_of_sections;

-- # ID, number_of_sections
-- '35579', '0'
-- '52647', '0'
-- '50885', '0'
-- '57180', '0'
-- '58558', '0'
-- '59795', '0'
-- '63395', '0'
-- '64871', '0'
-- '72553', '0'
-- '4034', '0'
```

```
-- '37687', '0'
-- '74426', '0'
-- '78699', '0'
-- '79653', '0'
-- '31955', '0'
-- '95030', '0'
-- '96895', '0'
-- '16807', '0'
-- '97302', '0'
-- '15347', '1'
-- '73623', '1'
-- '65931', '1'
-- '80759', '1'
-- '90376', '1'
-- '90643', '1'
-- '48570', '1'
-- '25946', '1'
-- '50330', '1'
-- '4233', '1'
-- '42782', '1'
-- '48507', '1'
-- '14365', '2'
-- '63287', '2'
-- '3335', '2'
-- '28400', '2'
-- '81991', '2'
-- '28097', '2'
-- '41930', '3'
-- '19368', '3'
-- '34175', '3'
-- '43779', '4'
-- '95709', '4'
-- '3199', '4'
-- '36897', '5'
-- '77346', '6'
-- '79081', '6'
-- '74420', '6'
-- '99052', '9'
-- '6569', '10'
-- '22591', '13'

-- For each student who has retaken a course at least twice (i.e., the student has
taken the course at
-- least three times), show the course ID and the student&#39;s ID. Please display
your results in order
-- of course ID and do not display duplicate rows.

describe takes;

select distinct takes.course_id, D.ID
from
  ( select distinct S.ID, count(T.ID) as cnt
    from student as S natural left join takes as T
    group by S.ID ) as D natural inner join takes
where D.cnt>1 ;
```

```sql
select distinct course_id, ID
  from takes
    group by ID, course_id having count(*) > 2
    order by course_id;

-- # course_id, ID
-- '362', '16480'
-- '362', '16969'
-- '362', '27236'
-- '362', '39925'
-- '362', '39978'
-- '362', '44881'
-- '362', '49611'
-- '362', '5414'
-- '362', '69581'
-- '362', '9993'



-- select distinct S.ID, count(T.ID) as cnt
-- from student as S natural left join takes as T
-- group by S.ID;

show tables;
describe course;
select * from course;
describe student;

-- Find the names of Biology students who have taken at least 3 Accounting courses.

select name
from student natural inner join (
select ID, course_id
from takes
group by id, course_id having count(*) > 2 ) as T
where T.course_id in ( select course_id from course
  where dept_name = "Accounting" );

select ID from takes
where course_id in ( select course_id from course
  where dept_name = "Accounting")
    group by ID having count(*) > 2
    ;

    describe student;
select ID, course_id
from takes
group by id, course_id having count(*) > 2;

SELECT
    name
FROM
    student
        NATURAL JOIN
    (SELECT
        ID
```

```
    FROM
        takes
    WHERE
        course_id IN (SELECT
                course_id
            FROM
                course
            WHERE
                dept_name = 'Accounting')
    GROUP BY ID
    HAVING COUNT(*) > 2) AS T
WHERE
    dept_name = 'Biology';

SELECT s.name
FROM student s
JOIN takes t ON s.ID = t.ID
JOIN course c ON t.course_id = c.course_id
WHERE s.dept_name = 'Biology'
AND c.dept_name = 'Accounting'
GROUP BY s.ID
HAVING COUNT(*) > 2;

-- # name
-- 'Michael'
-- 'Dalton'
-- 'Shoji'
-- 'Wehen'
-- 'Uchiyama'
-- 'Schill'
-- 'Kaminsky'
-- 'Giannoulis'

-- Find the sections that had maximum enrollment in Fall 2010.
show tables;
desc takes;
select * from section;


select max(cnt) from (
  select count(*) as cnt from takes as T2
  group by course_id, sec_id
) as D;

select count(*) as cnt from takes as T2
  group by course_id, sec_id
    order by cnt;

select course_id, sec_id from takes as T1
where semester = "Fall" and year = 2010
group by course_id, sec_id
having count(*) = ( select max(cnt) from (
  select count(*) as cnt from takes as T2
    where semester = "Fall" and year = 2010
  group by course_id, sec_id
) as D );
```

```sql
SELECT course_id, sec_id
FROM takes
WHERE semester = 'Fall' AND year = 2010
GROUP BY course_id, sec_id
HAVING COUNT(ID) = (
    SELECT MAX(enrollment_count)
    FROM (
        SELECT COUNT(ID) AS enrollment_count
        FROM takes
        WHERE semester = 'Fall' AND year = 2010
        GROUP BY course_id, sec_id
    ) AS subquery
);

-- # course_id, sec_id
-- '867', '2'

--  Find student names and the number of law courses taken for students who have
taken at least half of the available law courses.
-- (These courses are named things like 'Tort Law' or 'Environmental Law').

describe student;
describe takes;
describe course;

select name, count(*)
from student as S
join takes T on S.ID = T.ID
where T.course_id in (
  select course_id
    from course
    where title like "%Law%" )
group by S.ID
having count(*) >= (
  select count(course_id) / 2
  from (
    select course_id
    from course
    where title like "%Law%" )
        as D
  );

-- # name, count(*)
-- 'Nakajima', '4'
-- 'Nikut', '4'
-- 'Hahn-', '4'
-- 'Nanda', '4'
-- 'Schinag', '4'

-- Find the rank and name of the 10 students who earned the most A grades (A-, A,
A+).
-- Use alphabetical order by name to break ties. Note: the browser SQLite does not
support window functions.
show tables;
describe student;
```

```sql
describe takes;

select * from student;
select * from takes;

select name, grade
from student S
join takes T on S.ID = T.ID
where grade in ("A+", "A", "A-")
;

select row_number() over () as rnk, P.name from (
  select name, count(*) as cnt
  from student S
  join takes T on S.ID = T.ID
  where T.grade in ("A+", "A", "A-")
  group by T.ID
    ) as P
order by P.cnt desc, P.name
limit 10;

WITH StudentAGrades AS (
    SELECT s.ID, s.name, COUNT(*) AS a_count
    FROM student s
    JOIN takes t ON s.ID = t.ID
    WHERE t.grade IN ('A', 'A-', 'A+')
    GROUP BY s.ID, s.name
)
-- SELECT RANK() OVER (ORDER BY a_count DESC, name ASC) AS rnk, name
-- FROM StudentAGrades
-- ORDER BY rank
-- LIMIT 10
select ID, name, a_count
from StudentAGrades
order by a_count desc;

-- weirdness overloaded
SELECT s.ID, s.name, COUNT(*) AS a_count
    FROM student s
    JOIN takes t ON s.ID = t.ID
    WHERE t.grade IN ('A', 'A-', 'A+')
    GROUP BY s.ID, s.name
    order by a_count desc;

FROM student s
    JOIN takes t ON s.ID = t.ID
    WHERE t.grade IN ('A', 'A-', 'A+')
    GROUP BY s.ID, s.name
)
SELECT name, a_count
FROM (
    SELECT name, a_count, @curRank := @curRank + 1 AS rnk
    FROM StudentAGrades, (SELECT @curRank := 0) r
    ORDER BY a_count DESC, name ASC
) ranked
WHERE rnk <= 10
```

```
ORDER BY rnk;

-- # rnk, name
-- '1', 'Lepp'
-- '2', 'Eller'
-- '3', 'Masri'
-- '4', 'Vries'
-- '5', 'Åström'
-- '6', 'Gandhi'
-- '7', 'Greene'
-- '8', 'Haigh'
-- '9', 'McCarter'
-- '10', 'Sanchez'
```

9. Lab day 6

```
create database small_uni;
use small_uni;
use university;

show tables;

desc instructor;
select ID, salary from instructor;

-- small database
-- # ID, salary
-- '10101', '65000.00'
-- '12121', '90000.00'
-- '15151', '40000.00'
-- '22222', '95000.00'
-- '32343', '60000.00'
-- '33456', '87000.00'
-- '45565', '75000.00'
-- '58583', '62000.00'
-- '76543', '80000.00'
-- '76766', '72000.00'
-- '83821', '92000.00'
-- '98345', '80000.00'

-- use big database
-- # ID, salary
-- '14365', '32241.56'
-- '15347', '72140.88'
-- '16807', '98333.65'
-- '19368', '124651.41'
-- '22591', '59706.49'
-- '25946', '90891.69'
-- '28097', '35023.18'
-- '28400', '84982.92'
-- '31955', '71351.42'
-- '3199', '82534.37'
-- '3335', '80797.83'
-- '34175', '115469.11'
-- '35579', '62579.61'
-- '36897', '43770.36'
-- '37687', '104563.38'
```

```
-- '4034', '61387.56'
-- '41930', '50482.03'
-- '4233', '88791.45'
-- '42782', '34272.67'
-- '43779', '79070.08'
-- '48507', '107978.47'
-- '48570', '87549.80'
-- '50330', '108011.81'
-- '50885', '32570.50'
-- '52647', '87958.01'
-- '57180', '43966.29'
-- '58558', '66143.25'
-- '59795', '48803.38'
-- '63287', '103146.87'
-- '63395', '94333.99'
-- '64871', '45310.53'
-- '6569', '105311.38'
-- '65931', '79866.95'
-- '72553', '46397.59'
-- '73623', '90038.09'
-- '74420', '121141.99'
-- '74426', '106554.73'
-- '77346', '99382.59'
-- '78699', '59303.62'
-- '79081', '47307.10'
-- '79653', '89805.83'
-- '80759', '45538.32'
-- '81991', '77036.18'
-- '90376', '117836.50'
-- '90643', '57807.09'
-- '95030', '54805.11'
-- '95709', '118143.98'
-- '96895', '119921.41'
-- '97302', '51647.57'
-- '99052', '93348.83'

-- big database
-- # ID, salary
-- '14365', '32241.56'
-- '15347', '72140.88'
-- '16807', '98333.65'
-- '19368', '124651.41'
-- '22591', '59706.49'
-- '25946', '90891.69'
-- '28097', '35023.18'
-- '28400', '84982.92'
-- '31955', '71351.42'
-- '3199', '82534.37'
-- '3335', '80797.83'
-- '34175', '115469.11'
-- '35579', '62579.61'
-- '36897', '43770.36'
-- '37687', '104563.38'
-- '4034', '61387.56'
-- '41930', '50482.03'
-- '4233', '88791.45'
```

```
-- '42782', '34272.67'
-- '43779', '79070.08'
-- '48507', '107978.47'
-- '48570', '87549.80'
-- '50330', '108011.81'
-- '50885', '32570.50'
-- '52647', '87958.01'
-- '57180', '43966.29'
-- '58558', '66143.25'
-- '59795', '48803.38'
-- '63287', '103146.87'
-- '63395', '94333.99'
-- '64871', '45310.53'
-- '6569', '105311.38'
-- '65931', '79866.95'
-- '72553', '46397.59'
-- '73623', '90038.09'
-- '74420', '121141.99'
-- '74426', '106554.73'
-- '77346', '99382.59'
-- '78699', '59303.62'
-- '79081', '47307.10'
-- '79653', '89805.83'
-- '80759', '45538.32'
-- '81991', '77036.18'
-- '90376', '117836.50'
-- '90643', '57807.09'
-- '95030', '54805.11'
-- '95709', '118143.98'
-- '96895', '119921.41'
-- '97302', '51647.57'
-- '99052', '93348.83'

-- Find out the ID and salary of the instructor who gets more than $85,000.
select ID, salary from instructor
where salary > 85000;

-- small db
-- # ID, salary
-- '12121', '90000.00'
-- '22222', '95000.00'
-- '33456', '87000.00'
-- '83821', '92000.00'

-- big db
-- # ID, salary
-- '16807', '98333.65'
-- '19368', '124651.41'
-- '25946', '90891.69'
-- '34175', '115469.11'
-- '37687', '104563.38'
-- '4233', '88791.45'
-- '48507', '107978.47'
-- '48570', '87549.80'
-- '50330', '108011.81'
-- '52647', '87958.01'
```

```
-- '63287', '103146.87'
-- '63395', '94333.99'
-- '6569', '105311.38'
-- '73623', '90038.09'
-- '74420', '121141.99'
-- '74426', '106554.73'
-- '77346', '99382.59'
-- '79653', '89805.83'
-- '90376', '117836.50'
-- '95709', '118143.98'
-- '96895', '119921.41'
-- '99052', '93348.83'

-- Find out the department names and their budget at the university.
show tables;
desc department;

select dept_name, budget from department;

-- List out the names of the instructors from Computer Science who have more than
$70,000.
desc instructor;
select name from instructor
where salary > 70000;

-- For all instructors in the university who have taught some course, find their
names and the course ID of
-- all courses they taught.
desc instructor;
desc teaches;

select I.name, T.course_id
from instructor I
natural join teaches T
order by I.name;

-- Find the names of all instructors whose salary is greater than at least one
instructor in the Biology
-- department.
select name
from instructor
where dept_name = "Biology"
and salary > ( select min(S.salary) from (
  select salary
    from instructor
    where dept_name = "Biology"
) as S );

select min(S.salary) from ( select salary
    from instructor
    where dept_name = "Biology") as S;

select * from instructor;

-- Find the advisor of the student with ID 12345
select * from advisor
```

```sql
where s_ID = 12345;

-- Find the average salary of all instructors.
select avg(I.salary) average_salary from
(select salary from instructor) as I;

-- Find the names of all departments whose building name includes the substring
'Watson'.
select dept_name from department
where building like "%Watson%";

-- Find the names of instructors with salary amounts between $90,000 and $100,000.
select name from instructor
where salary between 90000 and 100000;

-- Find the instructor names and the courses they taught for all instructors in the
Biology department who
-- have taught some course.
select name, course_id
from instructor
natural left join teaches
where dept_name = "Biology";

-- Find the courses taught in Fall-2009 semester.
select course_id from teaches
where semester = "Fall" and year = 2009;

-- Find the set of all courses taught either in Fall-2009 or in Spring-2010.
select course_id from teaches
where (semester = "Fall" and year = 2009) or (semester = "Spring" and year = 2010);

( select course_id from teaches
where semester = "Fall" and year = 2009 )
union ( select course_id from teaches
where semester = "Spring" and year = 2010 );

-- Find the set of all courses taught in the Fall-2009 as well as in Spring-2010.
select course_id from teaches
where (semester = "Fall" and year = 2009) and (semester = "Spring" and year = 2010);

( select course_id from teaches
where semester = "Fall" and year = 2009 )
intersect ( select course_id from teaches
where semester = "Spring" and year = 2010 );

-- Find all courses taught in the Fall-2009 semester but not in the Spring-2010
semester.
select course_id from teaches
where (semester = "Fall" and year = 2009) and not (semester = "Spring" and year =
2010);

( select course_id from teaches
where semester = "Fall" and year = 2009 )
except ( select course_id from teaches
where semester = "Spring" and year = 2010 );
```

```
-- Find all instructors who appear in the instructor relation with null values for
salary.
select * from instructor
where salary = NULL;

select * from instructor;

-- Find the average salary of instructors in the Finance department.
select avg(T.salary) from
( select salary from instructor
where dept_name = "Finance" ) as T;

-- Find the total number of instructors who teach a course in the Spring-2010
semester.
describe teaches;
select  count(T.id) from
( select id from instructor
  natural join teaches
    where semester = "Spring"
    and year = 2010 ) as T ;

-- Find the average salary in each department.
select dept_name, avg(salary)
from department
natural join instructor
group by dept_name;

-- Find the number of instructors in each department who teach a course in the
Spring-2010 semester.
select dept_name, count(ID)
from department
natural join instructor
where ID in (
  select ID from teaches
    where semester = "Spring"
    and year = 2010
)
group by dept_name;

-- List out the departments where the average salary of the instructors is more than
$42,000.
select dept_name
from department D
where ( select avg(salary) from (
  select salary
    from instructor I
    where D.dept_name = I.dept_name
) as T ) > 42000;

select distinct dept_name
from instructor
group by dept_name
having avg(salary) > 42000;

-- For each course section offered in 2009, find the average total credits (tot cred)
of all students enrolled
```

```sql
-- in the section, if the section had at least 2 students.
select course_id, sec_id, avg(tot_cred)
from takes
natural join student
where year = 2009
group by sec_id, course_id
having count(*) > 1;

-- Find all the courses taught in both the Fall-2009 and Spring-2010 semesters.
select course_id from teaches
where (semester = "Fall" and year = 2009) and (semester = "Spring" and year = 2010);

( select course_id from teaches
where semester = "Fall" and year = 2009 )
intersect ( select course_id from teaches
where semester = "Spring" and year = 2010 );

-- Select the names of instructors whose names are neither 'Mozart' nor 'Einstein'.
select name from instructor
where name not in ("Mozart", "Einstein");

select name from instructor;

-- Find the total number of (distinct) students who have taken course sections taught
by the instructor
-- with ID 110011.
select count(T.ID)
from takes T
natural join section S
where "110011" in (
  select ID
    from teaches ST
    where T.course_id = ST.course_id
    and T.sec_id = ST.sec_id
    and T.semester = ST.semester
    and T.year = ST.year
);

SELECT COUNT(DISTINCT t.ID) AS total_students
FROM takes t
JOIN teaches te ON t.course_id = te.course_id
               AND t.sec_id = te.sec_id
               AND t.semester = te.semester
               AND t.year = te.year
WHERE te.ID = '110011';


select * from teaches;

-- Find the ID and names of all instructors whose salary is greater than at least one
instructor in the History
-- department.
desc instructor;

select ID, name
from instructor
```

```sql
where salary > ( select min(T.salary) from (
  select salary
    from instructor
    where dept_name = "History"
) as T );

-- Find the names of all instructors that have a salary value greater than that of
each instructor in the
-- Biology department.
select name
from instructor
where salary > ( select max(T.salary) from (
  select salary
    from instructor
    where dept_name = "Biology"
) as T );

-- Find the departments that have the highest average salary.
select dept_name
from department
natural join instructor
group by dept_name
order by avg(salary) desc
limit 1;

select dept_name
from instructor
group by dept_name
order by avg(salary) desc
limit 1;

-- Find all courses taught in both the Fall 2009 semester and in the Spring-2010
semester.

select course_id from teaches
where (semester = "Fall" and year = 2009) and (semester = "Spring" and year = 2010);

( select course_id from teaches
where semester = "Fall" and year = 2009 )
intersect ( select course_id from teaches
where semester = "Spring" and year = 2010 );

-- Find all students who have taken all the courses offered in the Biology
department.
with cnt as (
  select count(course_id) as ct
    from course
    where dept_name = "Biology"
)
select ID
from takes
where course_id in (
  select course_id
    from course
    where dept_name = "Biology"
)
```

```sql
group by ID
having count(*) = (
  select ct from cnt
);

-- Find all courses that were offered at most once in 2009.
select course_id
from takes
where year = 2009
group by course_id
having count(*) = 1;

-- Find all courses that were offered at least twice in 2009.
select course_id
from takes
where year = 2009
group by course_id
having count(*) > 1;

-- Find the average instructors' salaries of those departments where the average
salary is greater than
-- $42,000.

-- Find the maximum across all departments of the total salary at each department.
select sum(salary)
from department
natural join instructor
group by dept_name
order by sum(salary) desc
limit 1;

SELECT MAX(total_salary) AS max_total_salary
FROM (
    SELECT dept_name, SUM(salary) AS total_salary
    FROM instructor
    GROUP BY dept_name
) AS dept_salaries;

-- List all departments along with the number of instructors in each department.
select dept_name, count(ID)
from department
natural left join instructor
group by dept_name;

SELECT d.dept_name, COUNT(i.ID) AS num_instructors
FROM department d
LEFT JOIN instructor i ON d.dept_name = i.dept_name
GROUP BY d.dept_name;

-- Find the titles of courses in the Comp. Sci. department that have 3 credits.
select title
from course
where dept_name = "Comp. Sci."
and credits = 3;

-- Find the IDs of all students who were taught by an instructor named Einstein; make
```

```sql
-- sure there are no duplicates in the result.
select ID from instructor
where name = "Einstein";

select takes_table.ID
from takes takes_table
join teaches teaches_table
  on takes_table.course_id = teaches_table.course_id
    and takes_table.sec_id = teaches_table.sec_id
    and takes_table.semester = teaches_table.semester
    and takes_table.year = teaches_table.year
where teaches_table.ID = ( select ID from instructor
where name = "Einstein" ) ;

select * from teaches;

-- Find the ID and name of each student who has taken at least one Comp. Sci. course;
-- make sure there are no duplicate names in the result.
select distinct ID, name
from student
natural join takes
where takes.course_id in (
  select course_id
    from course
    where dept_name = "Comp. Sci."
);

-- Find the course id, section id, and building for each section of a Biology course.
select course_id, sec_id, building
from section
natural join course
where dept_name = "Biology";

-- Output instructor names sorted by the ratio of their salary to their department's
budget
-- (in ascending order).
select name
from instructor
natural left join department
order by (salary/ budget) asc;

-- Output instructor names and buildings for each building an instructor has taught
in.
-- Include instructor names who have not taught any classes (the building name should
-- be NULL in this case).

select name, building
from instructor
natural left join teaches
natural left join section;
```

10. Lab day 7

```sql
use university;

SELECT
    title
```

```
FROM
    course
WHERE
    dept_name = 'Comp. Sci.' AND credits = 3;

select distinct takes.ID
from takes, instructor, teaches
where takes.course_id = teaches.course_id and
takes.sec_id = teaches.sec_id and
takes.semester = teaches.semester and
takes.year = teaches.year and
teaches.id = instructor.id and
instructor.name = 'Einstein';

desc instructor;
select salary
from instructor
order by salary desc
limit 1;

select max(salary)
from
 instructor;

-- a. Find the titles of courses in the Comp. Sci. department that have 3 credits.
select title
from course
where dept_name = 'Comp. Sci.' and credits = 3;
-- b. Find the IDs of all students who were taught by an instructor named Einstein;
make sure there are no duplicates in the result.
select takes_table.ID
from takes takes_table
join teaches teaches_table
  on takes_table.course_id = teaches_table.course_id
    and takes_table.sec_id = teaches_table.sec_id
    and takes_table.semester = teaches_table.semester
    and takes_table.year = teaches_table.year
where teaches_table.ID = ( select ID from instructor
where name = "Einstein" ) ;
-- c. Find the highest salary of any instructor.
desc instructor;
select salary
from instructor
order by salary desc
limit 1;

-- Find all instru tors earning the highest salary (there may be more than
-- one with the same salary).

select ID, name
from instructor
where salary = (
  select salary
  from instructor
  order by salary desc
  limit 1
```

```
);

select ID, name
from instructor
where salary = (select max(salary) from instructor);

-- Find the enrollment of each section that was offered in Fall 2017.
desc takes;

select course_id, sec_id, count(*)
from takes
where year = 2017 and
semester = "Fall"
group by course_id, sec_id;

select takes.course_id, takes.sec_id, count(ID)
from section, takes
where takes.course_id= section.course_id
  and takes.sec_id = section.sec_id
  and takes.semester = section.semester
  and takes.year = section.year
    and takes.semester = 'Fall'
    and takes.year = 2017
group by takes.course_id, takes.sec_id ;

-- Find the maximum enrollment, a ross all se tions, in Fall 2017.
SELECT course_id, sec_id
FROM takes
WHERE semester = 'Fall' AND year = 2017
GROUP BY course_id, sec_id
HAVING COUNT(ID) = (
    SELECT MAX(enrollment_count)
    FROM (
        SELECT COUNT(ID) AS enrollment_count
        FROM takes
        WHERE semester = 'Fall' AND year = 2017
        GROUP BY course_id, sec_id
    ) AS subquery
);

with sec_enrollment as (
  select takes.course_id, takes.sec_id, count(ID) as enrollment
    from section, takes
    where takes.year = section.year
    and takes.semester = section.semester
    and takes.course_id = section.course_id
        and takes.sec_id = section.sec_id
        and takes.semester = 'Fall' and takes.year = 2017
  group by takes.course_id, takes.sec_id)
select course_id, sec_id
from sec_enrollment
where enrollment = (select max(enrollment) from sec_enrollment) ;

-- Find the names of Biology students who have taken at least 3 Accounting courses.
select name
from student natural join (
```

```
  select ID from takes
  where course_id in ( select course_id from course
    where dept_name = "Accounting")
    group by ID having count(*) > 2
    ) as T where dept_name = "Biology";

SELECT s.name
FROM student s
JOIN takes t ON s.ID = t.ID
JOIN course c ON t.course_id = c.course_id
WHERE s.dept_name = 'Biology'
AND c.dept_name = 'Accounting'
GROUP BY s.ID
HAVING COUNT(*) > 2;

-- Find the sections that had the maximum enrollment in Fall 2017.
desc section;
desc takes;

select sec_id, course_id
from takes
where year = 2017
and semester = "Fall"
group by sec_id, course_id
having count(*)
order by count(*) desc
limit 1;

select * from section;

WITH sec_enrollment AS (
    SELECT
        t.course_id, t.sec_id, COUNT(t.ID) AS enrollment
    FROM section s
    JOIN takes t
        ON t.year = s.year
        AND t.semester = s.semester
        AND t.course_id = s.course_id
        AND t.sec_id = s.sec_id
    WHERE t.semester = 'Fall'
      AND t.year = 2017
    GROUP BY t.course_id, t.sec_id
)
SELECT course_id, sec_id
FROM sec_enrollment
WHERE enrollment = (SELECT MAX(enrollment) FROM sec_enrollment);

-- Find the total grade points earned by the student with ID '12345', across all
courses taken by the student.

desc course;

select sum(credits * points)
from takes, course, grade_points
where takes.grade = grade_points.grade
and takes.course_id = course.course_id
```

```sql
and ID = '12345';

select
 sum(credits * points)/sum(credits) as GPA
from
 takes, course, grade_points
where
 takes.grade = grade_points.grade
and takes.course_id = course.course_id
and ID= '12345';

select
 ID, sum(credits * points)/sum(credits) as GPA
from
 takes, ourse, grade_points
where
 takes.grade = grade_points.grade
and takes.course_id = course.course_id
group by ID;

--  In rease the salary of ea h instru tor in the Comp. S i. department by
-- 10%.

update instructor
set salary = salary * 1
where dept_name = "Comp. Sci.";

-- b. Delete all courses that have never been offered (i.e., do not occur in the
section relation).
delete from course
where course_id not in (
  select course_id from section
);

-- c. Insert every student whose tot_cred attribute is greater than 100 as an
instructor in the same department, with a salary of $10,000.
desc instructor;
desc student;

insert into instructor
select ID, name, dept_name, 100000
from student
where tot_cred > 100;

select
 count(distinct person.driver_id)
from
 accident, participated, person, owns
where
 accident.report_number = participated.report_number
and owns.driver_id = person.driver_id
and owns.license_plate = participated.license_plate
and year = 2017;

select * from student where ID = "12345";
create index studentID_index on student(ID);
```

```
desc student;

  select ID,
    case
      when name=score < 40 then "F"
      when name=score < 60 then "C"
      when name=score < 80 then "B"
      else "A"
    end
  from marks;

with grades as (
select ID,
  case
    when name=score < 40 then "F"
        when name=score < 60 then "C"
        when name=score < 80 then "B"
        else "A"
    end
from marks )
select grade, count(ID)
from grades
group by grade;

select dept_name
from department
where lower(dept_name) like "%sci%";

-- a. Find the ID, name, and city of residence of each employee who works for
-- “First Bank Corporation”.
select e.ID, e.person name, city
from employee as e, works as w
where w.company_name = "First Bank Corporation" and
  w.ID = e.ID;

-- b. Find the ID, name, and city of residence of each employee who works for “First
Bank Corporation” and earns more than $10000.
select e.ID, e.person name, city
from employee as e, works as w
where w.company_name = "First Bank Corporation" and
  w.salary > 10000 and
  w.ID = e.ID;

describe instructor;

with tb as (
  select salary
    from instructor
)
select salary from tb
;

describe student;
SELECT DISTINCT student.ID, student.name
FROM student
```

```sql
JOIN takes ON student.ID = takes.ID
JOIN course ON takes.course_id = course.course_id
WHERE course.dept_name = 'Comp. Sci.';

    -- 11. Write the following queries in SQL, using the university schema:
-- a. Find the ID and name of each student who has taken at least one Comp. Sci.
course; make sure there are no duplicate names in the result.
SELECT DISTINCT student.ID, student.name
FROM student
JOIN takes ON student.ID = takes.ID
JOIN course ON takes.course_id = course.course_id
WHERE course.dept_name = 'Comp. Sci.';

-- b. Find the ID and name of each student who has not taken any course offered
before 2017.
SELECT student.ID, student.name
FROM student
WHERE student.ID NOT IN (
    SELECT takes.ID
    FROM takes
    WHERE takes.year < 2017
);

-- c. For each department, find the maximum salary of instructors in that department.
You may assume that every department has at least one instructor.
SELECT dept_name, MAX(salary) AS max_salary
FROM instructor
GROUP BY dept_name;
-- d. Find the lowest, across all departments, of the per-department maximum salary
computed by the preceding query.
SELECT MIN(max_salary) AS lowest_max_salary
FROM (
    SELECT dept_name, MAX(salary) AS max_salary
    FROM instructor
    GROUP BY dept_name
) AS dept_max_salaries;

-- Write the SQL statements using the university schema to perform the following
operations:
-- a. Create a new course “CS-001”, titled “Weekly Seminar”, with 0 credits.
desc course;
show create table course;
SHOW COLUMNS FROM course;
INSERT INTO course (course_id, title, dept_name, credits)
VALUES ('CS-001', 'Weekly Seminar', 'Comp. Sci.', 0);
-- b. Create a section of this course in Fall 2017, with sec_id of 1, and with the
location of this section not yet specified.
desc section;
INSERT INTO section (course_id, sec_id, semester, year)
VALUES ('CS-101', 1, 'Fall', 2017);
select * from course;
-- c. Enroll every student in the Comp. Sci. department in the above section.
INSERT INTO takes (ID, course_id, sec_id, semester, year)
SELECT student.ID, 'CS-001', 1, 'Fall', 2017
FROM student
WHERE student.dept_name = 'Comp. Sci.';
```

```sql
-- d. Delete enrollments in the above section where the student’s ID is 12345.
DELETE FROM takes
WHERE course_id = 'CS-001'
  AND sec_id = 1
  AND semester = 'Fall'
  AND year = 2017
  AND ID = '12345';
-- e. Delete the course CS-001. What will happen if you run this delete statement
without first deleting offerings (sections) of this course?
DELETE FROM course
WHERE course_id = 'CS-001';
-- f. Delete all takes tuples corresponding to any section of any course with the
word “advanced” as a part of the title; ignore case when matching the word with the
title.
DELETE FROM takes
WHERE course_id IN (
    SELECT course_id
    FROM course
    WHERE LOWER(title) LIKE '%advanced%'
);

-- Write SQL DDL corresponding to the schema in Figure 3.17. Make any reasonable
assumptions about data types, and be sure to declare primary and foreign keys.
CREATE DATABASE InsuranceDB;
USE InsuranceDB;

-- Drop things a bit
DROP TABLE person;
DROP TABLE car;
DROP TABLE accident;
DROP TABLE owns;
DROP TABLE participated;

-- Person Table
CREATE TABLE person (
    driver_id INT PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    address VARCHAR(100)
);

-- Car Table
CREATE TABLE car (
    license_plate VARCHAR(20) PRIMARY KEY,
    model VARCHAR(50),
    year YEAR
);

-- Accident Table
CREATE TABLE accident (
    report_number INT PRIMARY KEY,
    year YEAR,  -- Changed from VARCHAR(10) to YEAR
    location VARCHAR(100)
);

-- Owns Table
-- (Relationship between person and car)
```

```sql
CREATE TABLE owns (
    driver_id INT,
    license_plate VARCHAR(20),
    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
);

-- Participated Table
-- (Records cars and drivers involved in accidents)
CREATE TABLE participated (
    report_number INT,
    license_plate VARCHAR(20),
    driver_id INT,
    damage_amount DECIMAL(10,2) DEFAULT 0.00,
    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 SET NULL,
    FOREIGN KEY (driver_id) REFERENCES person(driver_id) ON DELETE SET NULL
);

SELECT employee.ID, employee.name
FROM employee
JOIN works ON employee.ID = works.ID
WHERE works.salary > (
    SELECT AVG(salary)
    FROM works w2
    WHERE w2.company_name = works.company_name
);

SELECT AVG(borrowed_count) AS avg_books_borrowed
FROM (
    SELECT m.memb_no, COUNT(b.isbn) AS borrowed_count
    FROM member m
    LEFT JOIN borrowed b ON m.memb_no = b.memb_no
    GROUP BY m.memb_no
) AS member_borrow_counts;

SELECT DISTINCT t.course_id, t.ID
FROM takes t
WHERE t.grade IS NOT NULL
GROUP BY t.ID, t.course_id
HAVING COUNT(t.course_id) >= 3
ORDER BY t.course_id;

SELECT t.ID
FROM takes t
WHERE t.grade IS NOT NULL
AND 2 <= (
  select count(course_id)
    from takes t2
    where t2.ID = t.ID
    and t2.course_id = t.course_id
)
GROUP BY t.ID
```

```sql
HAVING COUNT(DISTINCT t.course_id) >= 3;

SELECT i.name, i.ID
FROM instructor i
WHERE NOT EXISTS (
    SELECT c.course_id
    FROM course c
    WHERE c.dept_name = i.dept_name
    EXCEPT
    SELECT t.course_id
    FROM teaches t
    WHERE t.ID = i.ID
)
ORDER BY i.name;

SELECT i.name, i.ID
FROM instructor i
WHERE NOT EXISTS (
    SELECT c.course_id
    FROM course c
    WHERE c.dept_name = i.dept_name
    AND NOT EXISTS (
        SELECT t.course_id
        FROM teaches t
        WHERE t.ID = i.ID AND t.course_id = c.course_id
    )
)
ORDER BY i.name;

SELECT s.ID, s.name
FROM student s
WHERE s.dept_name = 'History'
AND s.name LIKE 'D%'
AND (
    SELECT COUNT(*)
    FROM takes t
    WHERE t.ID = s.ID
    AND t.course_id IN (
        SELECT c.course_id
        FROM course c
        WHERE c.dept_name = 'Music'
    )
) < 5;

  SELECT AVG(salary) - (SUM(salary) / COUNT(*))
  FROM instructor;

SELECT distinct i.ID, i.name
FROM instructor i
LEFT JOIN teaches t ON i.ID = t.ID
LEFT JOIN takes tk ON t.course_id = tk.course_id
WHERE tk.grade != 'A' AND EXISTS (
  select tk2.grade
    from takes tk2
    where t.course_id = tk2.course_id
    and tk2.grade is not null
```

```
);

SELECT i.ID, i.name
FROM instructor i
LEFT JOIN teaches t ON i.ID = t.ID
LEFT JOIN takes tk ON t.course_id = tk.course_id
WHERE tk.grade != 'A' OR tk.grade IS NULL
GROUP BY i.ID;

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

SELECT s.course_id, s.sec_id, s.year, s.semester, COUNT(t.ID) AS num
FROM section s
LEFT JOIN takes t ON s.course_id = t.course_id
AND s.sec_id = t.sec_id
GROUP BY course_id, sec_id, year, semester
HAVING num > 0;

WITH section_enrollment AS (
    SELECT s.course_id, s.sec_id, s.year, s.semester, COUNT(t.ID) AS num
    FROM section s
    LEFT JOIN takes t ON s.course_id = t.course_id AND s.sec_id = t.sec_id
    GROUP BY course_id, sec_id, year, semester
)
SELECT course_id, sec_id, year, semester, num
FROM section_enrollment
WHERE num = (SELECT MAX(num) FROM section_enrollment);

SELECT instructor.name
FROM instructor
WHERE instructor.dept_name = 'History';

SELECT instructor.ID, department.dept_name
FROM instructor,department
WHERE instructor.dept_name = department.dept_name
AND department.budget > 95000;
```

11. Lab day 8

```
use small_uni;

-- Create a relational database for employee salary maintenance with attributes
EmployeeID (PK), EmployeeName, Department, Salary, Month.
-- From Employee Table, transform rows into columns in MySQL


CREATE TABLE employee (
    employee_id INT,
    employee_name VARCHAR(50),
    department VARCHAR(50),
    salary INT,
    month VARCHAR(50),
    PRIMARY KEY (employee_id, month)
```

```
);

truncate employee;

INSERT INTO employee (employee_id, employee_name, department, salary, month)
VALUES
(1, 'Alice', 'HR', 1, 'January'),
(1, 'Alice', 'HR', 2, 'February'),
(1, 'Alice', 'HR', 3, 'March'),
(2, 'BOB', 'IT', 4, 'January'),
(2, 'BOB', 'IT', 5, 'February'),
(2, 'BOB', 'IT', 6, 'March');

INSERT INTO employee (employee_id, employee_name, department, salary, month)
VALUES
(3, 'BOB', 'IT', 5555, 'March');

select * from employee;

SELECT
    employee_id,
    employee_name,
    department,
    MAX(CASE WHEN Month = 'January' THEN Salary END) AS January,
    MAX(CASE WHEN Month = 'February' THEN Salary END) AS February,
    MAX(CASE WHEN Month = 'March' THEN Salary END) AS March
FROM employee
GROUP BY employee_id, employee_name, department;

SELECT
    employee_id,
    employee_name,
    department
FROM employee
GROUP BY employee_id, employee_name, department;

SELECT
    Employee_ID,
    Employee_Name,
    Department,
    MAX(CASE WHEN Month = 'January' THEN Salary END) AS January,
    MAX(CASE WHEN Month = 'February' THEN Salary END) AS February,
    MAX(CASE WHEN Month = 'March' THEN Salary END) AS March
FROM employee
GROUP BY Employee_ID, Employee_Name, Department;

SELECT e.Employee_ID, e.Employee_Name, e.Department, s.Salary, s.Month
FROM employee e
INNER JOIN employee s ON e.Employee_ID = s.Employee_ID;

DELETE FROM employee
WHERE Employee_ID NOT IN (
    SELECT MIN(Employee_ID) FROM employee
    GROUP BY Employee_ID, Month
);
```

```sql
SELECT MIN(Employee_ID) FROM employee
    GROUP BY Employee_ID, Month;

SELECT course.DEPT_NAME ,instructor.SALARY
FROM course ,instructor
WHERE SALARY > all (SELECT SALARY FROM instructor WHERE DEPT_NAME = 'Astronomy')
ORDER BY DEPT_NAME;

select * from instructor;
```

PATUAKHALI SCIENCE AND TECHNOLOGY UNIVERSITY

COURSE CODE CCE-224

SUBMITTED TO:

Prof. Dr. Md Samsuzzaman

Department of Computer and Communication Engineering
Faculty of Computer Science and Engineering

SUBMITTED BY:

Md. Sharafat Karim

ID: **2102024**,
Registration No: **10151**
Faculty of Computer Science and Engineering

Date of submission: **29 Wed, 2025**
Assignment 02
Assignment title: Chapter 01 (Ramez)

Chapter 1 | Review Questions

1. Define the following terms:

1. **Data**
   Data is referred as known facts that can be recorded and have some implicit meaning.
2. **Database**
   Database is a collection of related data.
3. **DBMS**
   A DBMS is a computerized system that enables user to create and maintain a database.
4. **Database system**
   Database system referes to the combination of a database, a DBMS and application that uses the data.
5. **Database catalog**
   Database catalog referes to the directory that stores metadata like table structures and constrains.
6. **Program-data independence**
   Program data independence refers to the ability where we can change the database structure without affecting the programs that use it.
7. **User view**
   User view refers to the part where an user interacts with the database.
8. **DBA**
   DBA or database administrator refers to the role where they assign the roles of other database users.
9. **End user**
   End users are the people whose jobs require access to the database for querying, updating, and generating reports; the database primarily exists for their use.
10. **Canned transaction**
    Canned transaction refers to the standard types of queries and updates which are used constantly.
11. **Deductive database system**
    Deductive database system is a system where data is stored with logical reasoning capabilities.
12. **Persistent object**
    Persistent object refers to the object which remains stored, even after closing the application that created the object.
13. **Meta-data**
    Meta data is the information that is stored in the DBMS catalog, where structure of all files, file-type and constrains are stored.
14. **Transaction-processing application**
    Transaction processing application refers to the real time applications that needs to update the data frequently.

2. What four main types of actions involve databases? Briefly discuss each.

Four types of actions are,

1. **Create:** We need to insert new information.
2. **Read:** We may need to read the existing data.
3. **Update:** We may need to update some information in our database.
4. **Delete:** We may also need to delete some entry from our database if necessary.

3. Discuss the main characteristics of the database approach and how it differs from traditional file systems.

Main characteristics of database includes central file processing. Here all applications can use the central database for data. Besides data is stored as program independent manner. So any changes in the DBMS or the application won't break the data.

In the other way, traditional file systems use multiple files for storing data. So duplicates of the same data may occur. And we also have to separately write our applications for accessing each of these files, which may make the entire process more time-consuming.

4. What are the responsibilities of the DBA and the database designers?

DBA or database administrator refers to the role where they assign the roles of other database users. They usually maintain the whole database system by allocating each users roles.

In the other hand, **database designers** usually design the whole data scheme and relations concept which will be used by the database DDL. As it is a bit complicated to change the system after once creating the data definition structure, so this process requires a lot of domain knowledge.

5. What are the different types of database end users? Discuss the main activities of each.

End users are the people whose jobs require access to the database for querying, updating, and generating reports; the database primarily exists for their use. Some of them are discussed below:

1. **Causal end users:** Causal end users access the database occasionally. But they may need different types of data each time.
2. **Naive end users:** Naive users are users who constantly update or query the database. They don't have to have much knowledge about the database.
3. **Sophisticated end users:** Sophisticated end users refers to the people who thoroughly study the database, so that they can implement it in their application.
4. **Standalone users:** Standalone users refers to the personal databases by using ready-made programs that provide user-friendly interface.

6. Discuss the capabilities that should be provided by a DBMS.

A DBMS should provide following capabilities:

1. Data definition (DDL)
2. CRUD operation
3. Data integrity and constrains auto checkup
4. Security and user roles
5. Concurrency control
6. Transaction Management
7. Backup and restore support
8. Performance and optimization
9. Data sharing and appropriate language support
10. Scalability and the ability to deploy in network

7. Discuss the differences between database systems and information retrieval systems.

Database system	Information retrieval system
Manages structure data	Data may be unstructured
Needs SQL or predefined rule	Keyword based search
Uses SQL for generating the same results	Uses ranking system for retrieving information
Uses tables, constrains	Uses files for linking
Focuses on accuracy	Focuses on precision
Can be used for data storing	Can be used for information recommendation system
For example, MySQL, MongoDB, Oracle DB	For example, Google or bing or any modern search engine

8. Identify some informal queries and update operations that you would expect to apply to the database shown in Figure 1.2.

Some informal queries that I may apply are,

1. Getting the grade of a particular student
2. Generating a report of a grade sheet of all students with same class
3. Listing the prerequisites of a particular course (if required)
4. List all instructor and courses of a particular class

9. What is the difference between controlled and uncontrolled redundancy? Illustrate with examples.

In a **controlled redundancy system** instead of duplicating a data in RDBMS, we use reference to the parent data, so that we don't need to store multiple value of the same data. On the other hand, in **uncontrolled redundancy** same data is copied multiple times, leading to much more storage and no auto update support.

For example, in a library management system, we have to put the book name on two tables, one bookshelf and one another is to the loan table. But if controlled redundancy is used then, changing the title of the book will update the other tables as well. On the other hand if uncontrolled redundancy is used then, we may need to manually update the name of the book in all tables, which may lead to further problems.

10. Specify all the relationships among the records of the database shown in Figure 1.2.

Here among multiple tables, we can construct some relations like,

1. *Course_number* in prerequisite table refers to the *Course_number* of section table.
2. *Prerequisite_number* in prerequisite table refers to the *Course_number* of section table.
3. *Student_number* in grade report refers to the *Student_number* in Student table.
4. *Section_number* in grade report refers to the *Section_number* in Section table.
5. *Course_number* in section table refers to the *Course_number* of section table.

11. Give some additional views that may be needed by other user groups for the database shown in Figure 1.2.

Some additional views that we can implement are,

1. Grade report per student
2. Instructors list per class
3. All courses list per year or class

12. Cite some examples of integrity constraints that you think can apply to the database shown in Figure 1.2.

Some constrains that we can apply to the database are,

1. Year can be between 1 and 8
2. Course number must have 6 alphanumeric digit
3. Semester should be either “Fall” or “Spring”
4. Major must be “CS” or “MATH” or in a set of values.
5. Name’s length should be within a certain range.
6. Credit hour’s value should be between 0 and an integer, like 4.

13. Give examples of systems in which it may make sense to use traditional file processing instead of a database approach.

Some examples where file processing instead of a databse approach is much efficient are,

1. **Media storage:** In media storage, generally files don’t change that much frequently. And mostly they are stored as BLOB in the databases. So in this case file storage is much more efficient.
2. **Single user application:** In single user applications like portable applications, concurrency is not generally used, so we can use single files instead of database. In this way we can also back up our data more accurately.

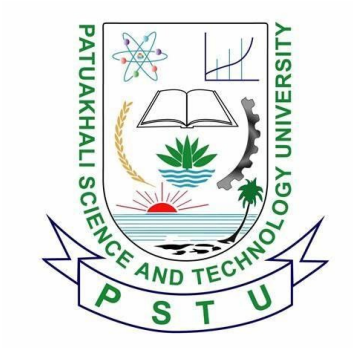
14. Consider Figure 1.2.

a. If the name of the ‘CS’ (Computer Science) Department changes to ‘CSSE’ (Computer Science and Software Engineering) Department and the corresponding prefix for the course number also changes, identify the columns in the database that would need to be updated.

If the department name changes then we have to update the *Department* column of COURSE table. And we also have to update the Course_number in COURSE, SECTION and PREREQUISITE table.

b. Can you restructure the columns in the COURSE, SECTION, and PREREQUISITE tables so that only one column will need to be updated?

We can implement primary key in COURSE table and corresponding foreign key in SECTION and PREREQUISITE table, so that we only need to update the COURSE table.



PATUAKHALI SCIENCE AND TECHNOLOGY UNIVERSITY

COURSE CODE CCE-224

SUBMITTED TO:

Prof. Dr. Md Samsuzzaman
Department of Computer and Communication Engineering
Faculty of Computer Science and Engineering

SUBMITTED BY:

Md. Sharafat Karim
ID: **2102024**,
Registration No: **10151**
Faculty of Computer Science and Engineering

Date of submission: **29 Wed, 2025**
Assignment 02
Assignment title: Chapter 01 (silberchatz)

Chapter 1 | Exercises

1. This chapter has described several major advantages of a database system. What are two disadvantages?

Some disadvantages,

1. Increased complexity
2. Poor performance

2. List five ways in which the type declaration system of a language such as "Java" or "C++" differs from the data definition language used in a database.

DDL	Programming language
Creates new object	Doesn't create new object, rather it creates an abstraction
It can contain consistency constrains to check a value before assigning	No constrain is used, so any values within the same type can be applied
Authorization → everyone can't access everything	No authorization exists
Only basic data types	Can contain advanced data types like array
Can contain relation between data	Doesn't contain relation between

3. List six major steps that you would take in setting up a database for a particular enterprise.

1. System requirement specification
2. Data model (data type)
3. Define integrity constrains
4. Define physical layer
5. Write application programs (user interface)
6. Create/ initialize the database

4. Suppose you want to build a video site similar to YouTube. Consider each of the points listed in Section 1.2 as disadvantages of keeping data in a file-processing system. Discuss the relevance of each of these points to the storage of actual video data, and to metadata about the video, such as title, the user who uploaded it, tags, and which users viewed it.

Disadvantage	Video	metadata
Data redundancy and inconsistency	No effect	It can be issue, if multiple metadata is stored per one video
Difficulty in accessing data	Less relevant	It can be issue, if search filter is used
Data isolation	Not much affect	Can be in seperate files

		leading to isolation
Integrity problems	Not much problem	Constrains can be difficult to implement like unique ID
Atomicity problems	If a video is uploaded and it's metadata isn't added then it may cause atomicity	If a video is uploaded and it's metadata isn't added then it may cause atomicity
Concurrent-access anomalies	Won't affect because video is not updated often	Won't affect because video is not updated often
Security problems	May cause problem if sustem has authorization	May cause problem if sustem has authorization

5. Keyword queries used in web search are quite different from database queries. List key differences between the two, in terms of the way the queries are specified and in terms of what is the result of a query.

Keyword query	**Database query**
No specific syntax	Has a specific syntax
Result is an unordered list of URL	Result is a table

6. List four applications you have used that most likely employed a database system to store persistent data.

Some apps that I used might have used databases are below,

1. **Facebook**: Facebook stores metadata about videos in it's database which is persistent and doesn't change frequently.
2. **Bangla Dictionary**: Dictionary apps uses database for storing words which are preety much persistant and doesn't get updated frequently.
3. **Gboard**: Gboard, a keyboard type application stores keyboard layouts, and other attributes like preferences are saved in persistant database.
4. **Daraz**: Daraz stores product information in persistant database. Because product information is not pretty much updated or rarely updated.

7. List four significant differances between a file-processing system and a DBMS.

File processing	**DBMS**
No centralized control of data	Data is controlled in one or multiple instance by one system
Data redundancy can be occurred by different files are used	Data redundancy is used to link multiple occurrence of same database
Change in one table doesn't change other same fields	Change in one occurrence applies in all other occurrences

Doesn’t provide security, because authentication can’t be applied	Provide authentication, so everyone can’t access everything

8. Explain the concept of physical data independence and its importance in database systems.

Physical data independence is the case when it is separated from the logical view to ensure logical view is not affected whenever a change is made to physical data.

9. List five responsibilities of a database-management system. For each responsibility, explain the problems that would arise if the responsibility were not discharged.

Some responsibility contains,

1. Redundancy minimize – without it, our database will use much more storage volume
2. Integrity maintain – without integrity, when one occurrence of data will be updated, it won’t update other occuraces automatically
3. Concurrency support – without concurrency if multiple users try to do same operation, system may not work in the expected way
4. Authentication – without it anyone can access or edit data which is out of the scope of his
5. Backup and restore – without it, we may lose data permanently

10. List at least two reasons why database systems support data manipulation using a declarative query language such as SQL, instead of just providing a library of C or C++ functions to carry out data manipulation.

Firstly to make DBMS more easy and simple, SQL is used. C or C++ is much harder to write manually, and different people can use different types of code for doing the same thing, but their complexity will differ which will make it difficult to work on the same codebase.

Secondly, SQL can provide a standardization, where all of its code is highly optimized. Otherwise, programming languages could be implemented in different ways which will lack a default standard and optimization.

11. Assume that two students are trying to register for a course in which there is only one open seat. What component of a database system prevents both students from being given that last seat?

Transaction management is responsible for handling this case. This part of the database deals with consistency.

12. Explain the difference between two-tier and three-tier application architectures. Which is better suited for web applications? Why?

Currently, **three-tier is better suited for web applications**.

In two tier applications, user can directly interact with the database through front end which can cause many issues like performance. Every calculation is done

in the frontend part. And the users can also database through frontend which can also cause security concerns.

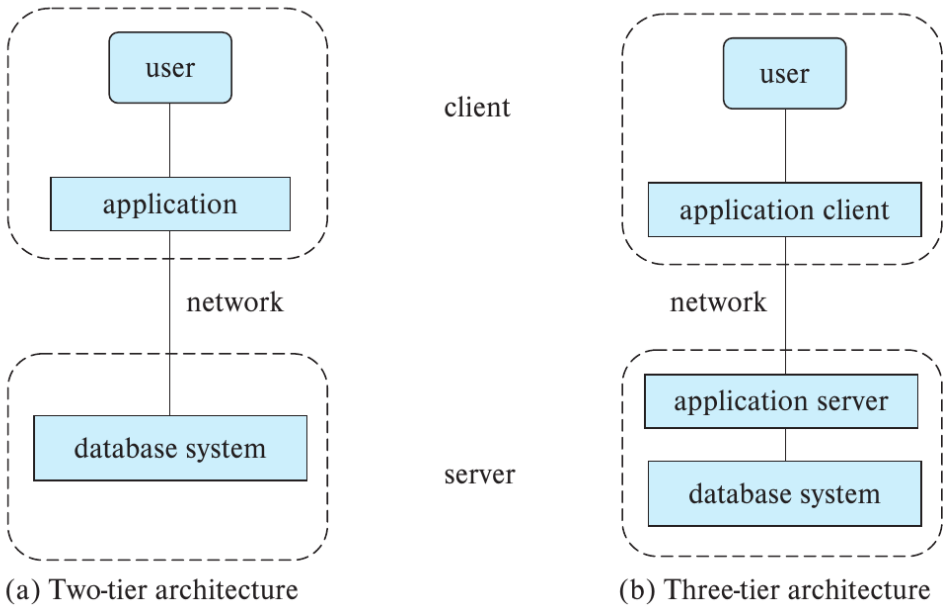


Figure 1.4 Two-tier and three-tier architectures.

On the other hand, in the three tier application, we have a backend server in between the database and the frontend. The frontend or application server directly communicates with the database, and provide services for the frontend which can gives its users more performance alongside security and authentication support.

13. List two features developed in the 2000s and that help database systems handle data-analytics workloads.

Firstly, graph databases were introduced, mainly used for social networks. Because social networks can't be described in tables efficiently. It made the social networking more practical.

Secondly, NoSQL were also created to facillate lightweight form of data management. It was also needed for data-intensive applications. At that time data mining became ubiquitous and new frameworks were beign developed.

14. Explain why NoSQL systems emerged in the 2000s, and briefy contrast their features with traditional database systems.

NoSQL systems were mainly emerged to solve some leggings of RDBMS systems. Some of the problems contain solving big data, scalability, real-time processing issues. Some of the features are,

1. **Scalability**: Traditional databases couldn't scale horizontally because it has a fixed data type. On the other hand in NoSQL data type is quite flexible.
2. **Performance**: Modern applications may need intensive use of data. For example data mining applications.

3. **Big data**: Social networks doesn't have a fixed pattern so it's hard to implement in the relation table structure.

15. Describe at least three tables that might be used to store information in a socialnetworking system such as Facebook.

For facebook, we may use three tables, 'user' table, 'friend' table and 'posts' table. Here's a short description,

1. **User table**:
   In user table we can put,
   1. unique id
   2. user name
   3. email
   4. password (encrypted)
   5. data of birth
   6. profile picture
   7. gender
2. **Friend table**:
   In friend table, we can link to another user table,
   1. unique id
   2. first friend's unique id
   3. second friend's unique id
   4. status
3. **posts table**:
   this table may contain posts of a user,
   1. unique id
   2. user id of the post creator
   3. content
   4. date

PATUAKHALI SCIENCE AND TECHNOLOGY UNIVERSITY

COURSE CODE CCE-224

SUBMITTED TO:

Prof. Dr. Md Samsuzzaman

Department of Computer and Communication Engineering
Faculty of Computer Science and Engineering

SUBMITTED BY:

Md. Sharafat Karim

ID: **2102024**,
Registration No: **10151**
Faculty of Computer Science and Engineering

Date of submission: **28 Fri, 2025**
Assignment 04
Assignment title: Chapter 02 (silberchatz)

Chapter 2 | Practice Exercises

1. Consider the employee database of Figure 2.17. What are the appropriate primary keys?

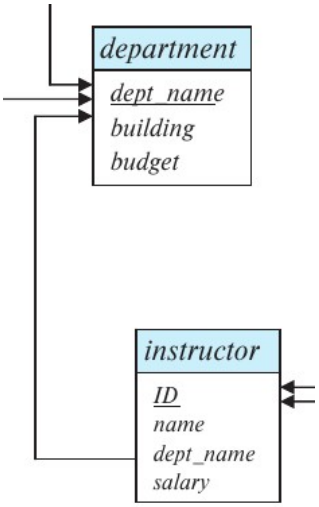
employee (*person_name*, *street*, *city*)
works (*person_name*, *company_name*, *salary*)
company (*company_name*, *city*)

Figure 2.17 Employee database.

The appropriate primary keys are shown below:

employee (person_name, street, city)
works (person_name, company_name, salary)
company (company_name, city)

2. Consider the foreign-key constraint from the dept name attribute of instructor to the department relation. Give examples of inserts and deletes to these relations that can cause a violation of the foreign-key constraint.



If we try to insert the following data into the instructor relation we will break the integration,

- (1024, “Sharafat”, “CSEES”, 0)

because the “CSEES” department doesn’t exist in the department relation.

And if we try to remove the following entry from the department relation, then it can break the integrity if CIT department has any instructor,

- (“CIT”, “Library”, 10000)

3. Consider the time slot relation. Given that a particular time slot can meet more than once in a week, explain why day and start time are part of the primary key of this relation, while end time is not.

The attributes day and start time are part of the primary key sin e a particular class will most likely meet on several different days and may even meet more than on e in a day. However, end time is not part of the primary key sin e a particular class that starts at a particular time on a particular day cannot end at more than one time.

4. In the instance of instructor shown in Figure 2.1, no two instructors have the same name. From this, can we conclude that name can be used as a superkey (or primary key) of instructor?

No, because even if in this database we don't have multiple teachers who have same name, unless in the university explicitly defines that multiple instructor can't have same name, we can't conclude that "name" can be a super key.

5. What is the result of first performing the Cartesian product of *student* and *advisor*, and then performing a selection operation on the result with the predicate s_id = ID? (Using the symbolic notation of relational algebra, this query can be written as $\sigma_{s\_id=ID}(student \times advisor)$.)

In this case first attributes of student and advisors will be combined. Then based on "s_id = ID". So only attributes with identical s_id and ID will be shown.

Students who have no advisor, will not be shown. Then if any student has multiple advisor then multiple times it'll be shown.

6. Consider the employee database of Figure 2.17. Give an expression in the relational algebra to express each of the following queries:

employee (*person_name*, *street*, *city*)
works (*person_name*, *company_name*, *salary*)
company (*company_name*, *city*)

Figure 2.17 Employee database.

a. **Find the name of each employee who lives in city "Miami".**

$\Pi_{person\_name}(\sigma_{city = \text{"Miami"}}(employee))$

b. **Find the name of each employee whose salary is greater than $100000.**

$\Pi_{person\_name}(\sigma_{salary > 10000}(employee \bowtie_{person\_name} works))$

c. **Find the name of each employee who lives in "Miami" and whose salary is greater than $100000.**

$\Pi_{person\_name}(\sigma_{city = \text{"Miami"} \wedge salary > 10000}(employee \bowtie_{person\_name} works))$

7. Consider the bank database of Figure 2.18. Give an expression in the relational algebra for each of the following queries:

branch(*branch_name*, *branch_city, assets*)
customer (*ID*, *customer_name*, *customer_street, customer_city*)
loan (*loan_number*, *branch_name, amount*)
borrower (*ID*, *loan_number*)
account (*account_number*, *branch_name, balance*)
depositor (*ID*, *account_number*)

Figure 2.18 Bank database.

a. **Find the name of each branch located in "Chicago".**

$\Pi_{\text{branch\_name}} (\sigma_{\text{branch\_city = "Chicago"}} (\text{branch}))$

b. **Find the ID of each borrower who has a loan in branch "Downtown".**

$\Pi_{\text{ID}} (\sigma_{\text{branch\_name= "Downtown"}} (\text{borrower} \bowtie_{\text{borrower.loan\_number=loan.loan\_number}} \text{loan}))$

8. Consider the employee database of Figure 2.17. Give an expression in the relational algebra to express each of the following queries:

a. **Find the ID and name of each employee who does not work for "BigBank".**

$\Pi_{\text{ID, person\_name}} (\text{employee})$

$- \Pi_{\text{ID, person\_name}} (\text{employee} \bowtie_{\text{employee.ID = works.ID}} \sigma_{(\text{company\_name= "BigBank"}}(\text{works})))$

b. **Find the ID and name of each employee who earns at least as much as every employee in the database.**

$\Pi_{\text{ID, person\_name}} (\text{employee})$

$- \Pi_{\text{A.ID, A.person\_name}} (\rho_A(\text{employee}) \bowtie_{\text{A.salary<B.salary}} \rho_B(\text{employee}))$

9. The division operator of relational algebra, "÷", is defined as follows. Let r(R) and s(S) be relations, and let $S \subseteq R$; that is, every attribute of schema S is also in schema R. Given a tuple t, let t[S] denote the projection of tuple t on the attributes in S. Then r ÷ s is a relation on schema R – S (that is, on the schema containing all attributes of schema R that are not in schema S). A tuple t is in r ÷ s if and only if both of two conditions hold:

- t is in Π_{R-S} (r)
- For every tuple t_s in s, there is a tuple t_r in r satisfying both of the following:
  - a. $t_r [S] = t_s [S]$
  - b. $t_r [R – S] = t$

Given the above definition:

a. **Write a relational algebra expression using the division operator to find the IDs of all students who have taken all Comp. Sci. courses. (Hint: project takes to just ID and course_id, and generate the set of all Comp. Sci. course_ids using a select expression, before doing the division.)**

$\Pi_{\text{ID}} (\Pi_{\text{ID, course\_id}} (\text{takes})$
$\div \Pi_{\text{ID, course\_id}} (\sigma_{\text{dept\_name='Comp. Sci'}} (\text{course})))$

b. **Show how to write the above query in relational algebra, without using division. (By doing so, you would have shown how to define the division operation using the other relational algebra operations.)**

$$\Pi_{ID}(\Pi_{ID, course\_id}(takes) - (\Pi_{ID, course\_id}(takes) \bowtie_{takes.ID=course.course\_id} (\Pi_{ID, course\_id}(\sigma_{dept\_name='Comp.\ Sci'}(course)))))$$

Chapter 2 | Exercises

10. Describe the differences in meaning between the terms relation and relation schema.

Relation refers to the tables in a database. And relation schema refers to the structure of relations and the logical connection in between them.

11. Consider the advisor relation shown in the schema diagram in Figure 2.9, with s_id as the primary key of advisor. Suppose a student can have more than one advisor. Then, would s_id still be a primary key of the advisor relation? If not, what should the primary key of advisor be?

If a student can have more than one advisor, then both the s_id and i_id will be primary key. Because in this case we will have multiple students with same student id. So in order to uniquely identify we will need both s_id and i_id to be primary key.

12. Consider the bank database of Figure 2.18. Assume that branch names and customer names uniquely identify branches and customers, but loans and accounts can be associated with more than one customer.

a. What are the appropriate primary keys?

The appropriate primary keys will be,

branch(<u>branch name</u>, branch city, assets)
customer (<u>ID</u>, customer name, customer street, customer city)
loan (<u>loan number</u>, branch name, amount)
borrower (<u>ID</u>, loan number)
account (<u>account number</u>, branch name, balance)
depositor (<u>ID</u>, account number)

primary keys are underlined

b. Given your choice of primary keys, identify appropriate foreign keys.

The appropriate foreign keys will be,

branch(*branch name*, branch city, assets)
customer (*ID*, customer name, customer street, customer city)
loan (*loan number*, <u>branch name</u>, amount)
borrower (*ID*, <u>loan number</u>)
account (*account number*, <u>branch name</u>, balance)
depositor (*ID*, <u>account number</u>)

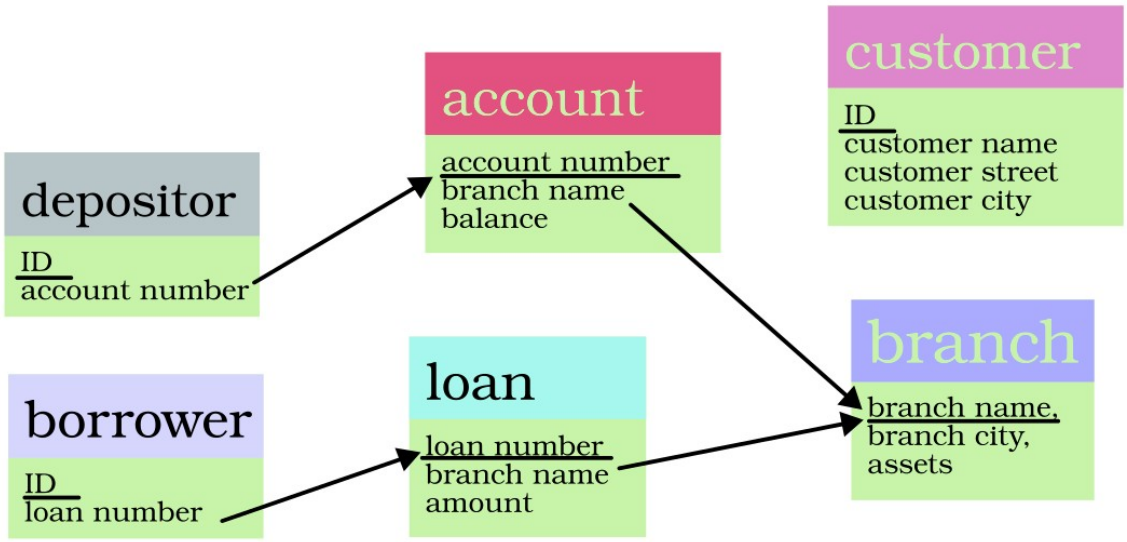
primary keys are marked as italic and secondary keys are underlined

13. Construct a schema diagram for the bank database of Figure 2.18.

branch(*branch_name*, *branch_city*, *assets*)
customer (*ID*, *customer_name*, *customer_street*, *customer_city*)
loan (*loan_number*, *branch_name*, *amount*)
borrower (*ID*, *loan_number*)
account (*account_number*, *branch_name*, *balance*)
depositor (*ID*, *account_number*)

Figure 2.18 Bank database.

Here's a schema diagram for the above diagram,



14. Consider the employee database of Figure 2.17. Give an expression in the relational algebra to express each of the following queries:

a. **Find the ID and name of each employee who works for "BigBank".**

$\Pi_{\text{employee.ID, employee.person\_name}}$ (employee $\bowtie_{\text{employee.person\_name=company.person\_name}}$ $\sigma_{\text{compnay\_name="BigBank"}}$(company))

b. **Find the ID, name, and city of residence of each employee who works for "BigBank".**

$\Pi_{\text{employee.ID, employee.person\_name,city}}$ (employee $\bowtie_{\text{employee.person\_name=company.person\_name}}$ $\sigma_{\text{compnay\_name="BigBank"}}$(company))

c. **Find the ID, name, street address, and city of residence of each employee who works for "BigBank" and earns more than $10000.**

$\Pi_{\text{employee.ID, employee.person\_name,street,city}}$ ((employee $\bowtie_{\text{employee.person\_name=company.person\_name}}$ $\sigma_{\text{compnay\_name="BigBank"}}$(company)) $\bowtie_{\text{employee.person\_name=works.person\_name}}$ $\sigma_{\text{salary>10000}}$(works))

d. **Find the ID and name of each employee in this database who lives in the same city as the company for which she or he works.**

$\Pi_{\text{employee.ID, employee.person\_name}}$(employee $\bowtie_{\text{employee.city=company.city}}$ company)

15. Consider the bank database of Figure 2.18. Give an expression in the relational algebra for each of the following queries:

a. **Find each loan number with a loan amount greater than $10000.**

$\Pi_{\text{loan\_number}}(\sigma_{\text{amount}>10000}(\text{loan}))$

b. **Find the ID of each depositor who has an account with a balance greater than $6000.**

$\Pi_{\text{ID}}(\sigma_{\text{balance}>10000}(\text{depositor} \bowtie_{\text{account\_number}} \text{account}))$

c. **Find the ID of each depositor who has an account with a balance greater than $6000 at the "Uptown" branch.**

$\Pi_{\text{ID}}(\sigma_{\text{balance}>6000 \wedge \text{branch\_name}=\text{"Uptown"}} (\text{depositor} \bowtie_{\text{account\_number}} \text{account}))$

16. List two reasons why null values might be introduced into a database.

To imply that a value of a database may not exist or left intentionally null value is introduced to the database.

17. Discuss the relative merits of imperative, functional, and declarative languages.

The relative merits are,

1. For imperative languages we give the direct instruction to do the task. So it has more performance.
2. For functional languages we think about what elements we may need. Then we make the function which will give the same time always. So it has less bugs.
3. In declarative languages, system does the logical part. We just have to tell the system what to do without thinking about time complexity, and the system will figure out how to pull the job.

18. Write the following queries in relational algebra, using the university schema.

a. **Find the ID and name of each instructor in the Physics department.**

$\Pi_{\text{ID, dept\_name}}(\sigma_{\text{dept\_name}=\text{"Physics"}}(\text{department}))$

b. **Find the ID and name of each instructor in a department located in the building "Watson".**

$\Pi_{\text{ID, instructor.name}}(\sigma_{\text{building}=\text{"Watson"}}((\text{department}) \bowtie_{\text{dept\_name}} \text{instructor}))$

c. **Find the ID and name of each student who has taken at least one course in the "Comp. Sci." department.**

$\Pi_{\text{ID, student.name}}(\text{student}) \cap$

$\Pi_{\text{ID, student.name}}(\sigma_{\text{dept\_name}=\text{"Comp. Sci."}}((\text{student}) \bowtie_{\text{ID}} (\text{takes})) \bowtie_{\text{course\_id}} (\text{course}))$

d. **Find the ID and name of each student who has taken at least one course section in the year 2018.**

$\Pi_{\text{ID, student.name}}(\text{student}) \cap$

$\Pi_{\text{ID, student.name}}(\sigma_{\text{section.year}=2018}(((\text{student}) \bowtie_{\text{ID}} (\text{takes})) \bowtie_{\text{course\_id}} (\text{section})))$

e. **Find the ID and name of each student who has not taken any course section in the year 2018.**

$\Pi_{\text{ID, student.name}}(\text{student}) \cap$

$\Pi_{\text{ID, student.name}}(\sigma_{\text{section.year}\neq2018}(((\text{student}) \bowtie_{\text{ID}} (\text{takes})) \bowtie_{\text{course\_id}} (\text{section})))$

PATUAKHALI SCIENCE AND TECHNOLOGY UNIVERSITY

COURSE CODE CCE-224

SUBMITTED TO:

Prof. Dr. Md Samsuzzaman

Department of Computer and Communication Engineering
Faculty of Computer Science and Engineering

SUBMITTED BY:

Md. Sharafat Karim

ID: **2102024**,
Registration No: **10151**
Faculty of Computer Science and Engineering

Date of submission: **08 Sat, 2025**
Assignment 03
Assignment title: Chapter 03 (silberchatz)

Chapter 3 | Practice Exercises

1. Write the following queries in SQL, using the university schema. (We suggest you actually run these queries on a database, using the sample data that we provide on the website of the book, db-book.com. Instructions for setting up a database, and loading sample data, are provided on the above website.)

a. Find the titles of courses in the Comp. Sci. department that have 3 credits.

```
select title
from course
where dept_name = 'Comp. Sci.' and credits = 3;
```

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

```
select takes_table.ID
from takes takes_table
join teaches teaches_table
     on takes_table.course_id = teaches_table.course_id
    and takes_table.sec_id = teaches_table.sec_id
    and takes_table.semester = teaches_table.semester
    and takes_table.year = teaches_table.year
where teaches_table.ID = ( select ID from instructor
where name = "Einstein" ) ;
```

c. Find the highest salary of any instructor.

```
desc instructor;
select salary
from instructor
order by salary desc
limit 1;
```

d. Find all instructors earning the highest salary (there may be more than one with the same salary).

```
select ID, name
from instructor
where salary = (
     select salary
     from instructor
     order by salary desc
     limit 1
);
```

e. Find the enrollment of each section that was offered in Fall 2017.

```
select course_id, sec_id, count(*)
from takes
where year = 2017 and
semester = "Fall"
group by course_id, sec_id;
```

f. Find the maximum enrollment, a ross all se tions, in Fall 2017.

```sql
SELECT course_id, sec_id
FROM takes
WHERE semester = 'Fall' AND year = 2017
GROUP BY course_id, sec_id
HAVING COUNT(ID) = (
    SELECT MAX(enrollment_count)
    FROM (
        SELECT COUNT(ID) AS enrollment_count
        FROM takes
        WHERE semester = 'Fall' AND year = 2017
        GROUP BY course_id, sec_id
    ) AS subquery
);
```

g. Find the sections that had the maximum enrollment in Fall 2017.

```sql
select sec_id, course_id
from takes
where year = 2017
and semester = "Fall"
group by sec_id, course_id
having count(*)
order by count(*) desc
limit 1;
```

2. Suppose you are given a relation grade points(grade, points) that provides a conversion from letter grades in the takes relation to numeric scores; for example, an "A" grade could be specified to correspond to 4 points, an "A–" to 3.7 points, a "B+" to 3.3 points, a "B" to 3 points, and so on. The grade points earned by a student for a course offering (section) is defined as the number of credits for the course multiplied by the numeric points for the grade that the student received. Given the preceding relation, and our university schema, write each of the following queries in SQL. You may assume for simplicity that no takes tuple has the null value for grade.

a. Find the total grade points earned by the student with ID '12345', across all courses taken by the student.

```sql
select sum(credits * points)
from takes, course, grade_points
where takes.grade = grade_points.grade
and takes.course_id = course.course_id
and ID = '12345';
```

b. Find the grade point average (GPA) for the above student, that is, the total grade points divided by the total credits for the associated courses.

```
select
 sum(credits * points)/sum(credits) as GPA
from
 takes, course, grade_points
where
 takes.grade = grade_points.grade
and takes.course_id = course.course_id
and ID= '12345';
```

c. Find the ID and the grade-point average of each student.

```
select
 ID, sum(credits * points)/sum(credits) as GPA
from
 takes, ourse, grade_points
where
 takes.grade = grade_points.grade
and takes.course_id = course.course_id
group by ID;
```

d. Now reconsider your answers to the earlier parts of this exercise under the assumption that some grades might be null. Explain whether your solutions still work and, if not, provide versions that handle nulls properly.

Above solutions won't work if the answer is null. To fix it, we can use join operations or union operation with *null* filtering.

3. Write the following inserts, deletes, or updates in SQL, using the university schema.

a. Increase the salary of each instructor in the Comp. Sci. department by 10%.

```
update instructor
set salary = salary * 1
where dept_name = "Comp. Sci.";
```

b. Delete all courses that have never been offered (i.e., do not occur in the section relation).

```
delete from course
where course_id not in (
     select course_id from section
);
```

c. Insert every student whose tot_cred attribute is greater than 100 as an instructor in the same department, with a salary of $10,000.

```
insert into instructor
select ID, name, dept_name, 100000
from student
where tot_cred > 100;
```

4. Consider the insurance database of Figure 3.17, where the primary keys are underlined. Construct the following SQL queries for

person (*driver_id*, *name*, *address*)
car (*license_plate*, *model*, *year*)
accident (*report_number*, *year*, *location*)
owns (*driver_id*, *license_plate*)
participated (*report_number*, *license_plate*, *driver_id*, *damage_amount*)

Figure 3.17 Insurance database

this relational database.

a. Find the total number of people who owned cars that were involved in accidents in 2017.

```
select
 count(distinct person.driver_id)
from
 accident, participated, person, owns
where
 accident.report_number = participated.report_number
and owns.driver_id = person.driver_id
and owns.license_plate = participated.license_plate
and year = 2017;
```

b. Delete all year-2010 cars belonging to the person whose ID is '12345'.

```
delete car
where year = 2010 and license_plate in
     (select license plate
     from owns o
     where o.driver id = '12345')
```

5. Suppose that we have a relation marks(ID, score) and we wish to assign grades to students based on the score as follows: grade F if score < 40, grade C if 40 <= score < 60, grade B if 60 <= score < 80, and grade A if 80 <= score. Write SQL queries to do the following:

Display the grade for each student, based on the marks relation.

```
select ID,
     case
          when name=score < 40 then "F"
                  when name=score < 60 then "C"
                  when name=score < 80 then "B"
             else "A"
     end
from marks;
```

Find the number of students with each grade.

```
with grades as (
select ID,
     case
          when name=score < 40 then "F"
        when name=score < 60 then "C"
        when name=score < 80 then "B"
        else "A"
    end
from marks )
select grade, count(ID)
from grades
group by grade;
```

6. The SQL like operator is case sensitive (in most systems), but the lower() funtion on strings can be used to perform case insensitive matching. To show how, write a query that finds departments whose names contain the string “sci” as a substring, regardless of the case.

```
select dept_name
from department
where lower(dept_name) like "%sci%";
```

7. Consider the SQL query:

```
select p.a1
from p, r1, r2
where p.a1 = r1.a1 or p.a1 = r2.a1
```

Under what conditions does the preceding query select values of p.a1 that are either in r1 or in r2? Examine carefully the cases where either r1 or r2 may be empty.

The query selects those values of p.a1 that are equal to some value of r1.a1 or r2.a1 if and only if both r1 and r2 are non-empty. If one or both of r1 and r2 are empty, the Cartesian product of p, r1 and r2 is empty, hen e the result of the query is empty. If p itself is empty, the result is empty.

8. Consider the bank database of Figure 3.18, where the primary keys are underlined. Construct the following SQL queries for this

branch(*branch_name*, *branch_city*, *assets*)
customer (*ID*, *customer_name*, *customer_street*, *customer_city*)
loan (*loan_number*, *branch_name*, *amount*)
borrower (*ID*, *loan_number*)
account (*account_number*, *branch_name*, *balance*)
depositor (*ID*, *account_number*)

Figure 3.18 Banking database.

relational database.

a. Find the ID of each customer of the bank who has an account but not a loan.

```
(select ID
from
 depositor)
except
(select ID
from
 borrower)
```

b. Find the ID of each customer who lives on the same street and in the same city as customer '12345'.

```
select F.ID
from customer as F, customer as S
where
      F.customer_street = S.customer_street
     and F.customer_city = S.customer_city
     and S.customer_id = '12345';
```

c. Find the name of each branch that has at least one customer who has an account in the bank and who lives in "Harrison".

```
select distinct branch name
from account, depositor, customer
where  ustomer.id = depositor.id
     and depositor.account_number = account.account_number
     and customer_city = 'Harrison';
```

9. Consider the relational database of Figure 3.19, where the primary keys are underlined. Give an expression in SQL for each of the following queries.

---

employee (*ID*, *person_name*, *street*, *city*)
works (*ID*, *company_name*, *salary*)
company (*company_name*, *city*)
manages (*ID*, *manager_id*)

Figure 3.19 Employee database.

a. Find the ID, name, and city of residence of each employee who works for "First Bank Corporation".

```
select e.ID, e.person name, city
from employee as e, works as w
where w.company name = "First Bank Corporation" and
     w.ID = e.ID
```

b. Find the ID, name, and city of residence of each employee who works for "First Bank Corporation" and earns more than $10000.

```
select e.ID, e.person name, city
from employee as e, works as w
where w.company_name = "First Bank Corporation" and
     w.salary > 10000 and
     w.ID = e.ID;
```

c. Find the ID of each employee who does not work for "First Bank Corporation".

```
     select ID
     from works
     where
          company name <> "First Bank Corporation";
```

d. Find the ID of each employee who earns more than every employee of "Small Bank Corporation".

```
SELECT ID
FROM works
WHERE salary > (
    SELECT MAX(salary)
    FROM works
    WHERE company_name = 'Small Bank Corporation'
);
```

e. Assume that companies may be located in several cities. Find the name of each company that is located in every city in which "Small Bank Corporation" is located.

```
SELECT S.company_name
```

```
FROM company AS S
WHERE NOT EXISTS (
    SELECT city
    FROM company
    WHERE company_name = 'Small Bank Corporation'
      AND city NOT IN (
          SELECT city
          FROM company AS T
          WHERE S.company_name = T.company_name
      )
);
```

f. Find the name of the company that has the most employees (or companies, in the case where there is a tie for the most).

```
select company_name
from works
group by company_name
having count (distinct ID) >= all
     (select count (distinct ID)
     from works
     group by company_name)
```

g. Find the name of each company whose employees earn a higher salary, on average, than the average salary at “First Bank Corporation”.

```
SELECT company_name
FROM works
GROUP BY company_name
HAVING AVG(salary) > (
     SELECT AVG(salary)
     FROM works
     WHERE company_name = 'First Bank Corporation'
);
```

10. Consider the relational database of Figure 3.19. Give an expression in SQL for each of the following:

a. Modify the database so that the employee whose ID is “12345” now lives in “Newtown”.

employee (*ID*, *person_name*, *street*, *city*)
works (*ID*, *company_name*, *salary*)
company (*company_name*, *city*)
manages (*ID*, *manager_id*)

Figure 3.19 Employee database.

```
update employee
```

```
set city = “Newtown”
where ID = “12345”
```

b. Give each manager of “First Bank Corporation” a 10 percent raise unless the salary becomes greater than $100000; in such cases, give only a 3 percent raise.

```
-- First update: Increase salary by 3% for managers earning more than 100,000 after a 10% raise
UPDATE works T
SET T.salary = T.salary * 1.03
WHERE T.ID IN (SELECT manager_id FROM manages)
  AND T.salary * 1.1 > 100000
  AND T.company_name = 'First Bank Corporation';

-- Second update: Increase salary by 10% for managers earning 100,000 or less after a 10% raise
UPDATE works T
SET T.salary = T.salary * 1.1
WHERE T.ID IN (SELECT manager_id FROM manages)
  AND T.salary * 1.1 <= 100000
  AND T.company_name = 'First Bank Corporation';
```

11. Write the following queries in SQL, using the university schema:

a. Find the ID and name of each student who has taken at least one Comp. Sci. course; make sure there are no duplicate names in the result.

```sql
SELECT DISTINCT student.ID, student.name
FROM student
JOIN takes ON student.ID = takes.ID
JOIN course ON takes.course_id = course.course_id
WHERE course.dept_name = 'Comp. Sci.';
```

b. Find the ID and name of each student who has not taken any course offered before 2017.

```sql
SELECT student.ID, student.name
FROM student
WHERE student.ID NOT IN (
    SELECT takes.ID
    FROM takes
    WHERE takes.year < 2017
);
```

c. For each department, find the maximum salary of instructors in that department. You may assume that every department has at least one instructor.

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

d. Find the lowest, across all departments, of the per-department maximum salary computed by the preceding query.

```sql
SELECT MIN(max_salary) AS lowest_max_salary
FROM (
    SELECT dept_name, MAX(salary) AS max_salary
    FROM instructor
    GROUP BY dept_name
) AS dept_max_salaries;
```

12. Write the SQL statements using the university schema to perform the following operations:

a. Create a new course “CS-001”, titled “Weekly Seminar”, with 0 credits.

```sql
INSERT INTO course (course_id, title, dept_name, credits)
VALUES ('CS-001', 'Weekly Seminar', 'Comp. Sci.', 0);
```

b. Create a section of this course in Fall 2017, with sec_id of 1, and with the location of this section not yet specified.

```sql
INSERT INTO section (course_id, sec_id, semester, year)
VALUES ('CS-101', 1, 'Fall', 2017);
```

c. Enroll every student in the Comp. Sci. department in the above section.

```sql
INSERT INTO takes (ID, course_id, sec_id, semester, year)
SELECT student.ID, 'CS-001', 1, 'Fall', 2017
```

```
FROM student
WHERE student.dept_name = 'Comp. Sci.';
```

d. Delete enrollments in the above section where the student's ID is 12345.

```
DELETE FROM takes
WHERE course_id = 'CS-001'
  AND sec_id = 1
  AND semester = 'Fall'
  AND year = 2017
  AND ID = '12345';
```

e. Delete the course CS-001. What will happen if you run this delete statement without first deleting offerings (sections) of this course?

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

Note: If you run this delete statement without first deleting the sections of this course, it will fail due to foreign key constraints (if the database enforces referential integrity).

f. Delete all `takes` tuples corresponding to any section of any course with the word "advanced" as a part of the title; ignore case when matching the word with the title.

```
DELETE FROM takes
WHERE course_id IN (
    SELECT course_id
    FROM course
    WHERE LOWER(title) LIKE '%advanced%'
);
```

13. Write SQL DDL corresponding to the schema in Figure 3.17. Make any reasonable assumptions about data types, and be sure to declare primary and foreign keys.

person (*driver_id*, *name*, *address*)
car (*license_plate*, *model*, *year*)
accident (*report_number*, *year*, *location*)
owns (*driver_id*, *license_plate*)
participated (*report_number*, *license_plate*, *driver_id*, *damage_amount*)

Figure 3.17 Insurance database

```
-- Person Table
CREATE TABLE person (
    driver_id INT PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    address VARCHAR(100)
);

-- Car Table
```

```sql
CREATE TABLE car (
    license_plate VARCHAR(20) PRIMARY KEY,
    model VARCHAR(50),
    year YEAR
);

-- Accident Table
CREATE TABLE accident (
    report_number INT PRIMARY KEY,
    year YEAR,
    location VARCHAR(100)
);

-- Owns Table
-- (Relationship between person and car)
CREATE TABLE owns (
    driver_id INT,
    license_plate VARCHAR(20),
    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
);

-- Participated Table
-- (Records cars and drivers involved in accidents)
CREATE TABLE participated (
    report_number INT,
    license_plate VARCHAR(20),
    driver_id INT,
    damage_amount DECIMAL(10,2) DEFAULT 0.00,
    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 SET
NULL,
    FOREIGN KEY (driver_id) REFERENCES person(driver_id) ON DELETE SET NULL
);
```

14. Consider the insurance database of Figure 3.17, where the primary keys are underlined. Construct the following SQL queries for this relational database.

a. Find the number of accidents involving a car belonging to a person named “John Smith”.

```sql
SELECT COUNT(DISTINCT accident.report_number) AS num_accidents
FROM accident
JOIN participated ON accident.report_number = participated.report_number
JOIN person ON participated.driver_id = person.driver_id
WHERE person.name = 'John Smith';
```

b. Update the damage amount for the car with license plate “AABB2000” in the accident with report number “AR2197” to $3000.

```sql
UPDATE participated
SET damage_amount = 3000
WHERE report_number = 'AR2197'
  AND license_plate = 'AABB2000';
```

15. Consider the bank database of Figure 3.18, where the primary keys are underlined. Construct the following SQL queries for this relational database.

branch(*branch_name*, *branch_city*, *assets*)
customer (*ID*, *customer_name*, *customer_street*, *customer_city*)
loan (*loan_number*, *branch_name*, *amount*)
borrower (*ID*, *loan_number*)
account (*account_number*, *branch_name*, *balance*)
depositor (*ID*, *account_number*)

Figure 3.18 Banking database.

a. Find each customer who has an account at every branch located in “Brooklyn”.

```sql
SELECT customer_name
FROM customer C
WHERE NOT EXISTS (
    SELECT branch_name
    FROM branch
    WHERE branch_city = 'Brooklyn'
    EXCEPT
    SELECT A.branch_name
    FROM account A
    JOIN depositor D ON A.account_number = D.account_number
    WHERE D.ID = C.ID
);
```

b. Find the total sum of all loan amounts in the bank.

```sql
SELECT SUM(amount) AS total_loan_amount
FROM loan;
```

c. Find the names of all branches that have assets greater than those of at least one branch located in “Brooklyn”.

```sql
SELECT branch_name
FROM branch
WHERE assets > (
    SELECT MIN(assets)
    FROM branch
    WHERE branch_city = 'Brooklyn'
);
```

16. Consider the employee database of Figure 3.19, where the primary keys are underlined. Give an expression in SQL for each of the following queries.

employee (*ID*, *person_name*, *street*, *city*)
works (*ID*, *company_name*, *salary*)
company (*company_name*, *city*)
manages (*ID*, *manager_id*)

Figure 3.19 Employee database.

a. Find ID and name of each employee who lives in the same city as the location of the company for which the employee works.

```sql
SELECT employee.ID, employee.person_name
FROM employee
JOIN works ON employee.ID = works.ID
JOIN company ON works.company_name = company.company_name
WHERE employee.city = company.city;
```

b. Find ID and name of each employee who lives in the same city and on the same street as does her or his manager.

```sql
SELECT e1.ID, e1.person_name
FROM employee e1
JOIN manages ON e1.ID = manages.ID
JOIN employee e2 ON manages.manager_id = e2.ID
WHERE e1.city = e2.city AND e1.street = e2.street;
```

c. Find ID and name of each employee who earns more than the average salary of all employees of her or his company.

```sql
SELECT employee.ID, employee.person_name
FROM employee
JOIN works ON employee.ID = works.ID
WHERE works.salary > (
    SELECT AVG(salary)
    FROM works w2
    WHERE w2.company_name = works.company_name
);
```

d. Find the company that has the smallest payroll.

```sql
SELECT company_name
FROM works
GROUP BY company_name
ORDER BY SUM(salary) ASC
LIMIT 1;
```

17. Consider the employee database of Figure 3.19. Give an expression in SQL for each of the following queries.

a. Give all employees of “First Bank Corporation” a 10 percent raise.

```sql
UPDATE works
SET salary = salary * 1.10
WHERE company_name = 'First Bank Corporation';
```

b. Give all managers of “First Bank Corporation” a 10 percent raise.

```sql
UPDATE works
SET salary = salary * 1.10
WHERE company_name = 'First Bank Corporation'
  AND ID IN (SELECT manager_id FROM manages);
```

c. Delete all tuples in the `works` relation for employees of “Small Bank Corporation”.

```sql
DELETE FROM works
WHERE company_name = 'Small Bank Corporation';
```

18. Give an SQL schema definition for the employee database of Figure 3.19. Choose an appropriate domain for each attribute and an appropriate primary key for each relation schema. Include any foreign-key constraints that might be appropriate.

employee (*ID*, *person_name*, *street*, *city*)
works (*ID*, *company_name*, *salary*)
company (*company_name*, *city*)
manages (*ID*, *manager_id*)

Figure 3.19 Employee database.

```sql
CREATE TABLE employee (
    ID INT PRIMARY KEY,
    person_name VARCHAR(100) NOT NULL,
    street VARCHAR(100),
    city VARCHAR(100)
);

CREATE TABLE company (
    company_name VARCHAR(100) PRIMARY KEY,
    city VARCHAR(100)
);

CREATE TABLE works (
    ID INT,
    company_name VARCHAR(100),
    salary DECIMAL(10, 2),
    PRIMARY KEY (ID, company_name),
    FOREIGN KEY (ID) REFERENCES employee(ID) ON DELETE CASCADE,
    FOREIGN KEY (company_name) REFERENCES company(company_name)
);

CREATE TABLE manages (
    ID INT,
    manager_id INT,
    PRIMARY KEY (ID),
    FOREIGN KEY (ID) REFERENCES employee(ID) ON DELETE CASCADE,
    FOREIGN KEY (manager_id) REFERENCES employee(ID) ON DELETE SET NULL
);
```

19. List two reasons why null values might be introduced into the database.

1. **Missing or Unknown Data**: A value may not be available or unknown at the time of data entry (e.g., a customer's middle name).
2. **Optional Attributes**: Some attributes may not apply to all entities (e.g., a `end_date` for an employee who is still employed).

20. Show that, in SQL, `<> ALL` is identical to `NOT IN`.

```sql
-- Using <> ALL
SELECT *
FROM table
WHERE column <> ALL (SELECT value FROM other_table);

-- Using NOT IN
SELECT *
FROM table
WHERE column NOT IN (SELECT value FROM other_table);
```

Both queries return the same result: rows where `column` does not match any value in the sub-query.

21. Consider the library database of Figure 3.20. Write the following queries in SQL.

member(*memb_no*, *name*)
book(*isbn*, *title*, *authors*, *publisher*)
borrowed(*memb_no*, *isbn*, *date*)

Figure 3.20 Library database.

a. Find the member number and name of each member who has borrowed at least one book published by "McGraw-Hill".

```sql
SELECT DISTINCT m.memb_no, m.name
FROM member m
JOIN borrowed b ON m.memb_no = b.memb_no
JOIN book bk ON b.isbn = bk.isbn
WHERE bk.publisher = 'McGraw-Hill';
```

b. Find the member number and name of each member who has borrowed every book published by "McGraw-Hill".

```sql
SELECT m.memb_no, m.name
FROM member m
WHERE NOT EXISTS (
    SELECT bk.isbn
    FROM book bk
    WHERE bk.publisher = 'McGraw-Hill'
    EXCEPT
    SELECT b.isbn
    FROM borrowed b
```

```
    WHERE b.memb_no = m.memb_no
);
```

c. For each publisher, find the member number and name of each member who has borrowed more than five books of that publisher.

```
SELECT bk.publisher, m.memb_no, m.name
FROM member m
JOIN borrowed b ON m.memb_no = b.memb_no
JOIN book bk ON b.isbn = bk.isbn
GROUP BY bk.publisher, m.memb_no, m.name
HAVING COUNT(b.isbn) > 5;
```

d. Find the average number of books borrowed per member. Take into account that if a member does not borrow any books, then that member does not appear in the borrowed relation at all, but that member still counts in the average.

```
SELECT AVG(borrowed_count) AS avg_books_borrowed
FROM (
    SELECT m.memb_no, COUNT(b.isbn) AS borrowed_count
    FROM member m
    LEFT JOIN borrowed b ON m.memb_no = b.memb_no
    GROUP BY m.memb_no
) AS member_borrow_counts;
```

22. Rewrite the `where` clause without using the `unique` construct.

```
where unique (select title from course)
```

```
-- We can use a equivalent test with distinct value and all values instead
of where clause

WHERE (SELECT COUNT(DISTINCT title) FROM course) = (SELECT COUNT(*) FROM
course);
```

23. Rewrite the following query without using the `with` construct.

Original Query:

```
WITH dept_total (dept_name, value) AS (
    SELECT dept_name, SUM(salary)
    FROM instructor
    GROUP BY dept_name
),
dept_total_avg(value) AS (
    SELECT AVG(value)
    FROM dept_total
)
SELECT dept_name
FROM dept_total, dept_total_avg
WHERE dept_total.value >= dept_total_avg.value;
```

Rewritten Query:

```
SELECT dept_name
FROM (
    SELECT dept_name, SUM(salary) AS total_salary
```

```
    FROM instructor
    GROUP BY dept_name
) AS dept_total
WHERE dept_total.total_salary >= (
    SELECT AVG(total_salary)
    FROM (
        SELECT SUM(salary) AS total_salary
        FROM instructor
        GROUP BY dept_name
    ) AS avg_salaries
);
```

24. Using the university schema, write an SQL query to find the name and ID of those Accounting students advised by an instructor in the Physics department.

```
SELECT s.name, s.ID
FROM student s
JOIN advisor a ON s.ID = a.s_ID
JOIN instructor i ON a.i_ID = i.ID
WHERE s.dept_name = 'Accounting' AND i.dept_name = 'Physics';
```

25. Using the university schema, write an SQL query to find the names of those departments whose budget is higher than that of Philosophy. List them in alphabetic order.

```
SELECT
    dept_name
FROM
    department
WHERE
    budget > (SELECT
            budget
        FROM
            department
        WHERE
            dept_name = 'Philosophy');
```

26. Using the university schema, use SQL to do the following: For each student who has retaken a course at least twice (i.e., the student has taken the course at least three times), show the course ID and the student's ID. Please display your results in order of course ID and do not display duplicate rows.

```
select distinct course_id, ID
        from takes
    group by ID, course_id having count(*) > 2
    order by course_id;
```

27. Using the university schema, write an SQL query to 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 two times).

```
SELECT t.ID
```

```sql
FROM takes t
WHERE t.grade IS NOT NULL
AND 2 <= (
     select count(course_id)
    from takes t2
    where t2.ID = t.ID
    and t2.course_id = t.course_id
)
GROUP BY t.ID
HAVING COUNT(DISTINCT t.course_id) >= 3;
```

28. Using the university schema, write an SQL query to find the names and IDs of those instructors who teach every course taught in his or her department (i.e., every course that appears in the course relation with the instructor's department name). Order the result by name.

```sql
SELECT i.name, i.ID
FROM instructor i
WHERE NOT EXISTS (
    SELECT c.course_id
    FROM course c
    WHERE c.dept_name = i.dept_name
    EXCEPT
    SELECT t.course_id
    FROM teaches t
    WHERE t.ID = i.ID
)
ORDER BY i.name;
```

29. Using the university schema, write an SQL query to find the name and ID of each History student whose name begins with the letter 'D' and who has not taken at least five Music courses.

```sql
SELECT s.ID, s.name
FROM student s
WHERE s.dept_name = 'History'
AND s.name LIKE 'D%'
AND (
    SELECT COUNT(*)
    FROM takes t
    WHERE t.ID = s.ID
    AND t.course_id IN (
        SELECT c.course_id
        FROM course c
        WHERE c.dept_name = 'Music'
    )
) < 5;
```

30. Consider the following SQL query on the university schema:

```
SELECT AVG(salary) - (SUM(salary) / COUNT(*))
FROM instructor
```

We might expect that the result of this query is zero since the average of a set of numbers is defined to be the sum of the numbers divided by the number of numbers. Indeed, this is true for the example instructor relation in Figure 2.1. However, there are other possible instances of that relation for which the result would not be zero. Give one such instance, and explain why the result would not be zero.

The result might not be zero if there are instructors with missing or null salary data. The `AVG()` function ignores nulls, while `SUM(salary)` and `COUNT(*)` count all rows, potentially leading to discrepancies.

31. Using the university schema, write an SQL query to 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 i.ID, i.name
FROM instructor i
LEFT JOIN teaches t ON i.ID = t.ID
LEFT JOIN takes tk ON t.course_id = tk.course_id
WHERE tk.grade != 'A' OR tk.grade IS NULL
GROUP BY i.ID;
```

32. 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 i
LEFT JOIN teaches t ON i.ID = t.ID
LEFT JOIN takes tk ON t.course_id = tk.course_id
WHERE tk.grade != 'A' AND EXISTS (
     select tk2.grade
    from takes tk2
    where t.course_id = tk2.course_id
    and tk2.grade is not null
);
```

33. Using the university schema, write an SQL query to 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 c
JOIN section s ON c.course_id = s.course_id
```

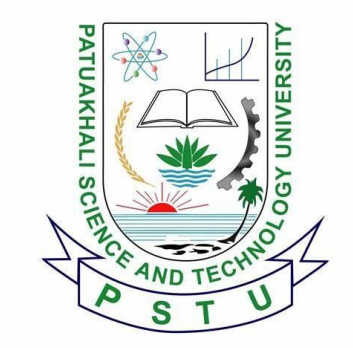
```
JOIN time_slot t ON s.time_slot_id = t.time_slot_id
WHERE t.end_hr >= 12 AND c.dept_name = 'Comp. Sci.';
```

34. Using the university schema, write an SQL query to find the number of students in each section. The result columns should appear in the order "courseid, secid, year, semester, num." You do not need to output sections with 0 students.

```
SELECT s.course_id, s.sec_id, s.year, s.semester,
COUNT(t.ID) AS num
FROM section s
LEFT JOIN takes t ON s.course_id = t.course_id
AND s.sec_id = t.sec_id
GROUP BY course_id, sec_id, year, semester
HAVING num > 0;
```

35. Using the university schema, write an SQL query to find section(s) with maximum enrollment. The result columns should appear in the order "courseid, secid, year, semester, num." (It may be convenient to use the WITH construct.)

```
WITH section_enrollment AS (
    SELECT s.course_id, s.sec_id, s.year, s.semester,
COUNT(t.ID) AS num
    FROM section s
    LEFT JOIN takes t ON s.course_id = t.course_id
    AND s.sec_id = t.sec_id
    GROUP BY course_id, sec_id, year, semester
)
SELECT course_id, sec_id, year, semester, num
FROM section_enrollment
WHERE num = (SELECT MAX(num) FROM section_enrollment);
```



PATUAKHALI SCIENCE AND TECHNOLOGY UNIVERSITY

COURSE CODE CCE-224

SUBMITTED TO:

Prof. Dr. Md Samsuzzaman
Department of Computer and Communication Engineering
Faculty of Computer Science and Engineering

SUBMITTED BY:

Md. Sharafat Karim
ID: **2102024**,
Registration No: **10151**
Faculty of Computer Science and Engineering

Date of submission: **Fri 28, March 2025**
Assignment 04
Assignment title: Chapter 04 (silberchatz)

Chapter 4 | Practice Exercises

1. Consider the following SQL query that seeks to find a list of titles of all courses taught in Spring 2017 along with the name of the instructor.

```
select name, title
from instructor natural join teaches natural join section natural join course
where semester ='Spring' and year = 2017
```

What is wrong with this query?

Here instructor and course share the same attribute, named “dept_name” and as natural join is used, they will be joined only if the value is same, skipping other possible columns. So basically every teacher will be listed in his own department only.

A better approach would be to specify the join conditions explicitly to ensure only the intended columns are matched.

2. Write the following queries in SQL:

a. Display a list of all instructors, showing each instructor's ID and the number of sections taught. Make sure to show the number of sections as 0 for instructors who have not taught any section. Your query should use an outer join, and should not use subqueries.

```
select ID, count(sec_id) as Number_of_sections
from instructor natural left outer join teaches
group by ID;
```

b. Write the same query as in part a, but using a scalar subquery and not using outer join.

```
select ID,
        (select count(*) as Number_of_sections
        from teaches T where T.id = I.id)
from instructor I;
```

c. Display the list of all course sections offered in Spring 2018, along with the ID and name of each instructor teaching the section. If a section has more than one instructor, that section should appear as many times in the result as it has instructors. If a section does not have any instructor, it should still appear in the result with the instructor name set to "-".

```
select course_id, sec_id, ID,
        coalesce(name,'--') as name
from (section natural left outer join teaches)
natural left outer join instructor
where semester='Spring' and year = 2018;
```

d. Display the list of all departments, with the total number of instructors in each department, without using subqueries. Make sure to show departments that have no instructors, and list those departments with an instructor count of zero.

```
SELECT d.dept_name, COUNT(i.ID) AS num_instructors
FROM department d
LEFT OUTER JOIN instructor i ON d.dept_name = i.dept_name
GROUP BY d.dept_name;
```

3. Outer join expressions can be computed in SQL without using the SQL outer join operation. To illustrate this fact, show how to rewrite each of the following SQL queries without using the outer join expression.

a. select * from student natural left outer join takes

```
SELECT *
FROM student
NATURAL JOIN takes
UNION
SELECT student.ID, name, dept_name, tot_cred,
    NULL, NULL, NULL, NULL, NULL
FROM student
WHERE student.ID NOT IN (SELECT ID FROM takes);
```

b. select * from student natural full outer join takes

```
SELECT *
```

```
FROM student
NATURAL JOIN takes
UNION
SELECT student.ID, name, dept_name, tot_cred,
    NULL, NULL, NULL, NULL, NULL
FROM student
WHERE ID NOT IN (SELECT ID FROM takes)
UNION
SELECT NULL, NULL, NULL, NULL,
    takes.ID, course_id, sec_id, semester, year
FROM takes
WHERE ID NOT IN (SELECT ID FROM student);
```

4. Suppose we have three relations r(A, B), s(B, C), and t(B, D), with all attributes declared as not null.

a. Give instances of relations r, s, and t such that in the result of (r natural left outer join s) natural left outer join t attribute C has a null value but attribute D has a non-null value.

Yes, it is possible that C has a null value, but not D. Let's consider,

r(A,B) = {(1, 1)}
s(B,C) = {(2, 1)}
t(B,D) = {(1, 3)}

And the result of the operation will be, {(1, 1, Null, 3)}

b. Are there instances of r, s, and t such that the result of r natural left outer join (s natural left outer join t) has a null value for C but a non-null value for D? Explain why or why not.

This is not possilbe. If we are to join s and t first then obviously there will be a value of C, to join with the corresponding D via B. And we know that all attributes are declared as not null, from the question.

Let's consider,

r(A,B) = {(1, 1)}
s(B,C) = {(2, 1)}
t(B,D) = {(1, 3)}

And the result of the operation will be, {(1, 1, Null, Null)}

If we need D, then without matching with a C won't make sense.

5. Testing SQL queries: To test if a query specified in English has been correctly written in SQL, the SQL query is typically executed on multiple test databases, and a human checks if the SQL query result on each test database matches the intention of the specification in English.

a. In Section 4.1.1 we saw an example of an erroneous SQL query which was intended to find which courses had been taught by each instructor; the query computed the natural join of instructor, teaches, and course, and as a result it unintentionally equated the dept name attribute of instructor and course. Give an example of a dataset that would help catch this particular error.

Let's consider a scenario,

instructor = {('12345','Gauss', 'Physics', 10000)}
teaches = {('12345', 'EE321', 1, 'Spring', 2017)}
course = {('EE321', 'Magnetism', 'Ele . Eng.', 6)}

Here it won't be combined because instructor's *dept_name* and course's *dept_name* are different, hence will produce an error.

b. When creating test databases, it is important to create tuples in referenced relations that do not have any matching tuple in the referencing relation for each foreign key. Explain why, using an example query on the university database.

If we create etries in the referenced relation without any childrens, then it'll be a good test subject to check SQL queries like left join or right join or outer full join.

c. When creating test databases, it is important to create tuples with null values for foreign-key attributes, provided the attribute is nullable (SQL allows foreign-key attributes to take on null values, as long as they are not part of the primary key and have not been declared as not null). Explain why, using an example query on the university database.

This kind of tuple is important as in SQL, Null is not equal to Null. So values with Null value will be totally skipped in operations like,

```
SELECT * FROM teaches NATURAL JOIN instructor;


-- (will appear in results)

INSERT INTO teaches VALUES ('101', 'CS-101', '1', 'Fall', 2023);


-- (will disappear from results)

INSERT INTO teaches VALUES (NULL, 'CS-101', '2', 'Fall', 2023);
```

6. Show how to define the view student grades (ID, GPA) giving the grade-point average of each student, based on the query in Exercise 3.2; recall that we used a relation grade points(grade, points) to get the numeric points associated with a letter grade. Make sure your view definition correctly handles the case of null values for the grade attribute of the takes relation.

```
create view student_grades(ID, GPA) as
select
 ID, sum(credits * points)/sum(credits) as GPA
from
 takes, course, grade_points
where
 takes.grade = grade_points.grade
and takes.course_id = course.course_id
group by ID;
```

7. Consider the employee database of Figure 4.12. Give an SQL DDL definition of this database. Identify referential-integrity constraints that should hold, and include them in the DDL definition.

employee (*ID*, *person_name*, *street*, *city*)
works (*ID*, *company_name*, *salary*)
company (*company_name*, *city*)
manages (*ID*, *manager_id*)

Figure 4.12 Employee database.

```
CREATE TABLE employee (
        ID NUMERIC(5,0),
  person_name VARCHAR(255) NOT NULL,
  street VARCHAR(255),
  city VARCHAR(255),
  PRIMARY KEY (ID)
```

```
);

CREATE TABLE company (
   company_name VARCHAR(255) NOT NULL,
   city VARCHAR(255),
   PRIMARY KEY (company_name)
);

CREATE TABLE works (
        ID NUMERIC(5,0),
   person_name VARCHAR(255) NOT NULL,
   company_name VARCHAR(255),
   salary DECIMAL(10,2),
   PRIMARY KEY (ID),
   FOREIGN KEY (ID) REFERENCES employee(ID),
   FOREIGN KEY (company_name) REFERENCES company(company_name)
);

CREATE TABLE manages (
        ID NUMERIC(5,0),
   manager_id NUMERIC(5,0),
   PRIMARY KEY (ID),
   FOREIGN KEY (ID) REFERENCES employee(ID),
   FOREIGN KEY (manager_id) REFERENCES employee(ID)
);
```

8. As discussed in Section 4.4.8, we expect the constraint "an instructor cannot teach sections in two different classrooms in a semester in the same time slot" to hold.

a. Write an SQL query that returns all (instructor, section) combinations that violate this constraint.

```
SELECT
```

```
    instructor.ID,
    instructor.name,
    teaches.sec_id,
    teaches.semester,
    teaches.year,
    section.time_slot_id,
    COUNT(DISTINCT section.building, section.room_number) AS location_count
FROM
    instructor
    NATURAL JOIN teaches
    NATURAL JOIN section
GROUP BY
    instructor.ID,
    instructor.name,
    teaches.sec_id,
    teaches.semester,
    teaches.year,
    section.time_slot_id
HAVING
    COUNT(DISTINCT section.building, section.room_number) > 1;
```

b. Write an SQL assertion to enforce this constraint.

```
create assertion check not exists
(SELECT
    instructor.ID,
    instructor.name,
    teaches.sec_id,
    teaches.semester,
    teaches.year,
    section.time_slot_id,
    COUNT(DISTINCT section.building, section.room_number) AS location_count
FROM
```

```
  instructor
  NATURAL JOIN teaches
  NATURAL JOIN section
GROUP BY
  instructor.ID,
  instructor.name,
  teaches.sec_id,
  teaches.semester,
  teaches.year,
  section.time_slot_id
HAVING
  COUNT(DISTINCT section.building, section.room_number) > 1);
```

9. SQL allows a foreign-key dependency to refer to the same relation, as in the following example:

```
create table manager
(employee ID char(20),
manager ID char(20),
primary key employee ID,
foreign key (manager ID) references manager(employee ID)
on delete cascade )
```

Here, employee ID is a key to the table manager, meaning that each employee has at most one manager. The foreign-key clause requires that every manager also be an employee. Explain exactly what happens when a tuple in the relation manager is deleted.

The tuples of all employees of the manager, at all levels, get deleted as well! This deletion works recursively and thus further relations will also be deleted.

10. Given the relations a(name, address, title) and b(name, address, salary), show how to express a natural full outer join b using the full outer-join operation with an on condition rather than using the natural join syntax. This can be done using the coalesce operation. Make sure that the result relation does not contain two copies of the attributes name and address and that the solution is correct even if some tuples in a and b have null values for attributes name or address.

```
SELECT
  COALESCE(a.name, b.name) AS name,
  COALESCE(a.address, b.address) AS address,
  a.title,
  b.salary
FROM
  a FULL OUTER JOIN b
    ON a.name = b.name
    AND a.address = b.address;
```

11. Operating systems usually offer only two types of authorization control for data files: read access and write access. Why do database systems offer so many kinds of authorization?

Database systems offer more authorization types because:

1. Databases have more complex operations than simple file read/write (SELECT, INSERT, UPDATE, DELETE, etc.)
2. Granular control is needed for security in multi-user environments
3. Different operations have different risk profiles (e.g., SELECT vs DELETE)
4. Need to control schema modification (ALTER, REFERENCES) separately from data access
5. Support for row-level and column-level security requirements
6. Need to manage privileges for views, stored procedures, and other database objects
7. Requirement to delegate privileges (GRANT OPTION)

12. Suppose a user wants to grant select access on a relation to another user. Why should the user include (or not include) the clause granted by current role in the grant statement?

If an user grants permission though a role then, even if the user leaves, the permission still persists. That's why it's a good practice in the long run.

13. Consider a view v whose definition references only relation r.

• If a user is granted select authorization on v, does that user need to have select authorization on r as well? Why or why not?

No, the user doesn't need SELECT on the base table (r).

• If a user is granted update authorization on v, does that user need to have update authorization on r as well? Why or why not?

Yes, the user also needs update permission on r as well, because this update operation is performed there.

• Give an example of an insert operation on a view v to add a tuple t that is not visible in the result of select * from v. Explain your answer.

CREATE VIEW v AS SELECT * FROM r WHERE rating > 5;

INSERT INTO v VALUES (101, 'John', 3); -- Tuple won't appear in SELECT * FROM v

Explanation: The inserted tuple has rating=3 which doesn't satisfy the view's condition (rating > 5), so it won't be visible in the view even though it exists in the base table.

14. Consider the query:

```
select course_id, semester, year, sec_id, avg(tot_cred)
from takes natural join student
where year = 2017
group by course_id, semester, year, sec_id
having count(ID) >= 2;
```

Explain why appending natural join section in the from clause would not change the result.

The query already groups by course_id, semester, year, and sec_id. Here sec_id is the key attributes of section. So the section relation doesn't contain any attributes that would affect the average of tot_cred.

15. Rewrite the query:

```
select *
from section natural join classroom
```

without using a natural join but instead using an inner join with a using condition.

```
select * from section join classroom using (building, room_number);
```

16. Write an SQL query using the university schema to find the ID of each student who has never taken a course at the university. Do this using no subqueries and no set operations (use an outer join).

```
select s.ID
from student s left outer join takes t on s.ID = t.ID
where t.ID is null
```

17. Express the following query in SQL using no subqueries and no set operations:

```
select ID
from student
except
select s_id
from advisor
where i_ID is not null
```

```
select s.ID
from student s left outer join advisor a on s.ID = a.s_id
where a.s_id is null or a.i_ID is null
```

18. For the database of Figure 4.12, write a query to find the ID of each employee with no manager. Note that an employee may simply have no manager listed or may have a null manager. Write your query using an outer join and then write it again using no outer join at all.

employee (*ID*, *person_name*, *street*, *city*)
works (*ID*, *company_name*, *salary*)
company (*company_name*, *city*)
manages (*ID*, *manager_id*)

Figure 4.12 Employee database.

With outer join,

```
select e.person_name
from employee e left outer join manages m on e.ID = m.ID
where m.manager_id is null;
```

Without outer join,

```
SELECT e.person_name
FROM employee e
WHERE NOT EXISTS (
  SELECT 1
  FROM manages m
  WHERE e.ID = m.ID
);
```

19. Under what circumstances would the query:

```
select *
from student natural full outer join takes
natural full outer join course
```

include tuples with null values for the title attribute?

The query would include tuples with null title values when:

1. There are students who haven't taken any courses (their takes attributes would be null)
2. There are courses that no students have taken (their student attributes would be null)

The natural join fails to match on dept_name between student and course (if student's dept_name doesn't match course's dept_name)

20. Show how to define a view tot_credits (year, num_credits), giving the total number of credits taken in each year.

```
create view tot_credits (year, num_credits) as
select year, sum(credits)
```

```
from takes natural join course
where grade is not null and grade <> 'F'
group by year;
```

21. For the view of Exercise 4.20, explain why the database system would not allow a tuple to be inserted into the database through this view.

We will get the following error,
Error Code: 1471. The target table tot_credits of the INSERT is not insertable-into

The view is not updatable because:

1. It contains aggregation (SUM)
2. It involves a GROUP BY clause
3. It joins multiple tables

So it's really not ideal to pull update operations through view.

22. Show how to express the coalesce function using the case construct.

```
-- COALESCE(a,b,c) is equivalent to:
case when a is not null then a
   when b is not null then b
   else c
end
```

23. Explain why, when a manager, say Satoshi, grants an authorization, the grant should be done by the manager role, rather than by the user Satoshi.

If grant is done by a role then if Satoshi leaves the organization, privileges granted by their role will persist as it maintains separation between personal and role-based privileges. It allows for proper privilege inheritance and revocation

24. Suppose user A, who has all authorization privileges on a relation r, grants select on relation r to public with grant option. Suppose user B then grants select on r to A. Does this cause a cycle in the authorization graph? Explain why.

As A has all the priviledges beforehand B, B won't be able to grant select on r to A again. So it wont' create a cycle.

25. Suppose a user creates a new relation r1 with a foreign key referencing another relation r2. What authorization privilege does the user need on r2? Why should this not simply be allowed without any such authorization?

The user needs REFERENCES privilege on r2 because,

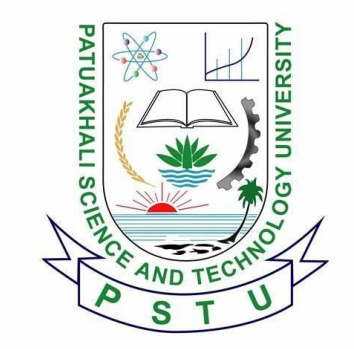
1. The foreign key creates a dependency between the relations.
2. It prevents the referenced rows in r2 from being deleted while referenced.

3. Without this requirement, users could create constraints on data they don't own

Suppose an user creates a foreign key in a relation r1 referencing the dept name attribute of the r2 relation and then inserts a tuple into r pertaining to the Geology department. It is no longer possible to delete the Geology department from the department relation without also modifying relation r1. Thus, the definition of a foreign key by an user restricts future activity by other users; therefore, there is a need for the references privilege.

26. Explain the difference between integrity constraints and authorization constraints.

Feature	Integrity Constraints	Authorization Constraints
Definition	Rules enforced to ensure **data correctness** and **consistency** within the database.	Restrictions that control **who can access or modify** data.
Purpose	Prevent invalid or inconsistent data from being stored.	Restrict unauthorized users from performing certain actions.
Scope	Applied to **data values and relationships** between tables.	Applied to **users and their privileges** on database objects.
Enforcement	Enforced **automatically** by the database system.	Enforced **based on user roles and privileges**.
Examples	- **Primary Key** (ensures uniqueness) - **Foreign Key** (maintains referential integrity) - **Check Constraints** (enforces specific conditions) - **Not Null** (ensures presence of values)	- **GRANT/REVOKE statements** (assign/restrict permissions) - **Access control on tables/views** (read/write restrictions) - **REFERENCES privilege** (needed for foreign key creation)
Focus	**Data correctness and logical consistency.**	**User access control and security.**
Modification Impact	Prevents changes that would violate constraints.	Prevents unauthorized users from making changes.



PATUAKHALI SCIENCE AND TECHNOLOGY UNIVERSITY

COURSE CODE CCE-224

SUBMITTED TO:

Prof. Dr. Md Samsuzzaman

Department of Computer and Communication Engineering
Faculty of Computer Science and Engineering

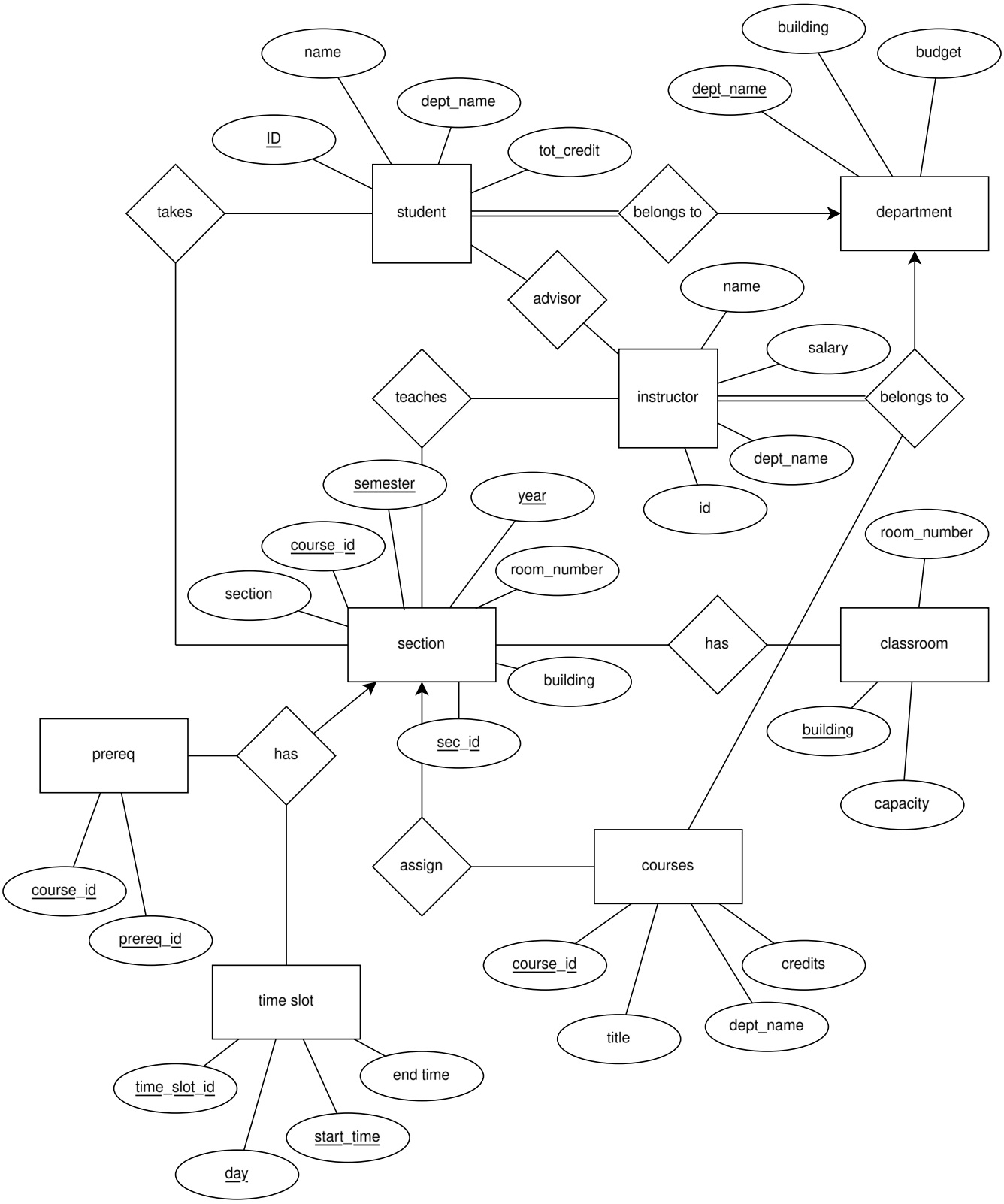
SUBMITTED BY:

Md. Sharafat Karim

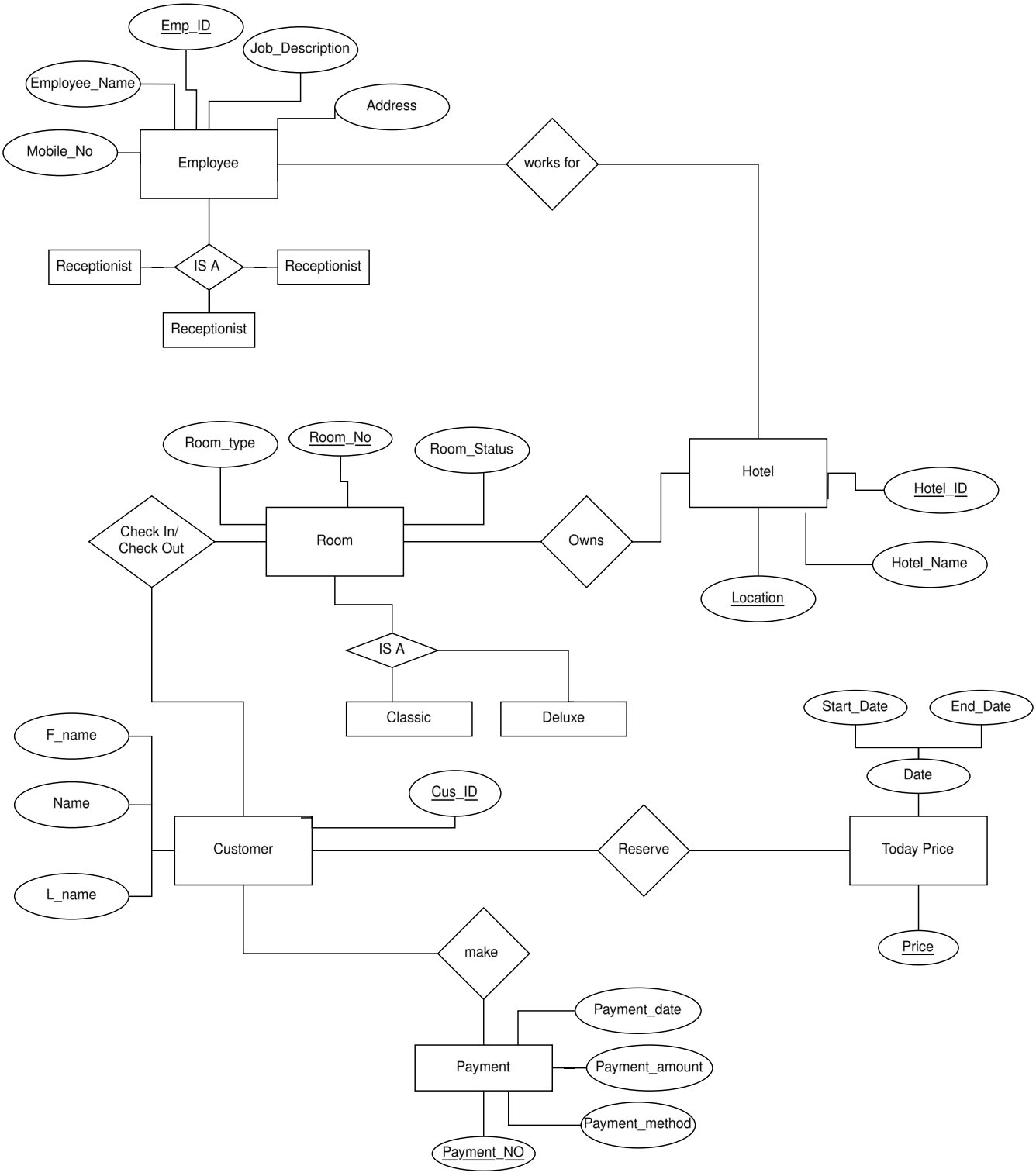
ID: **2102024**,
Registration No: **10151**
Faculty of Computer Science and Engineering

Date of submission: **06 Thu, 2025**
Assignment 03
Assignment title: ER diagrams

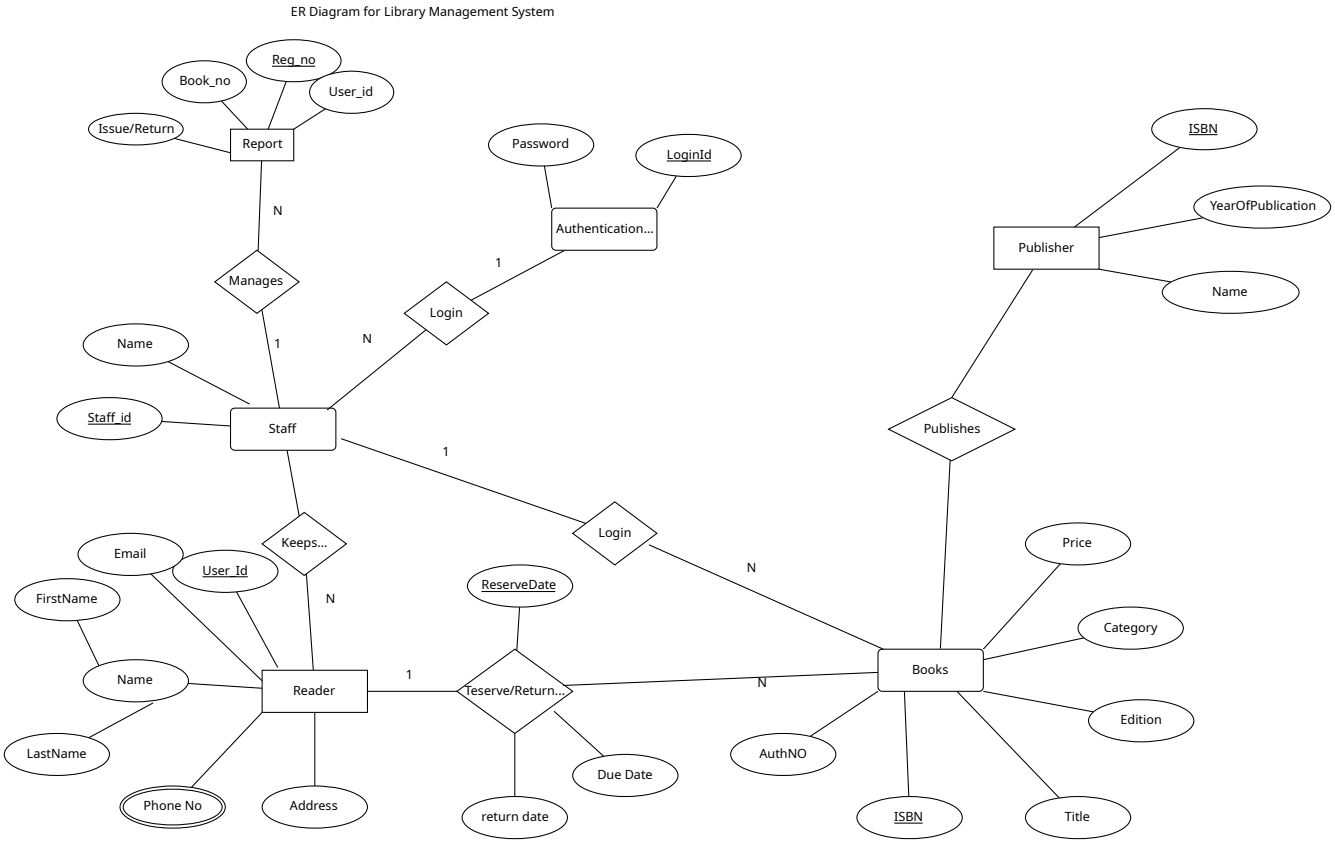
- University Management system



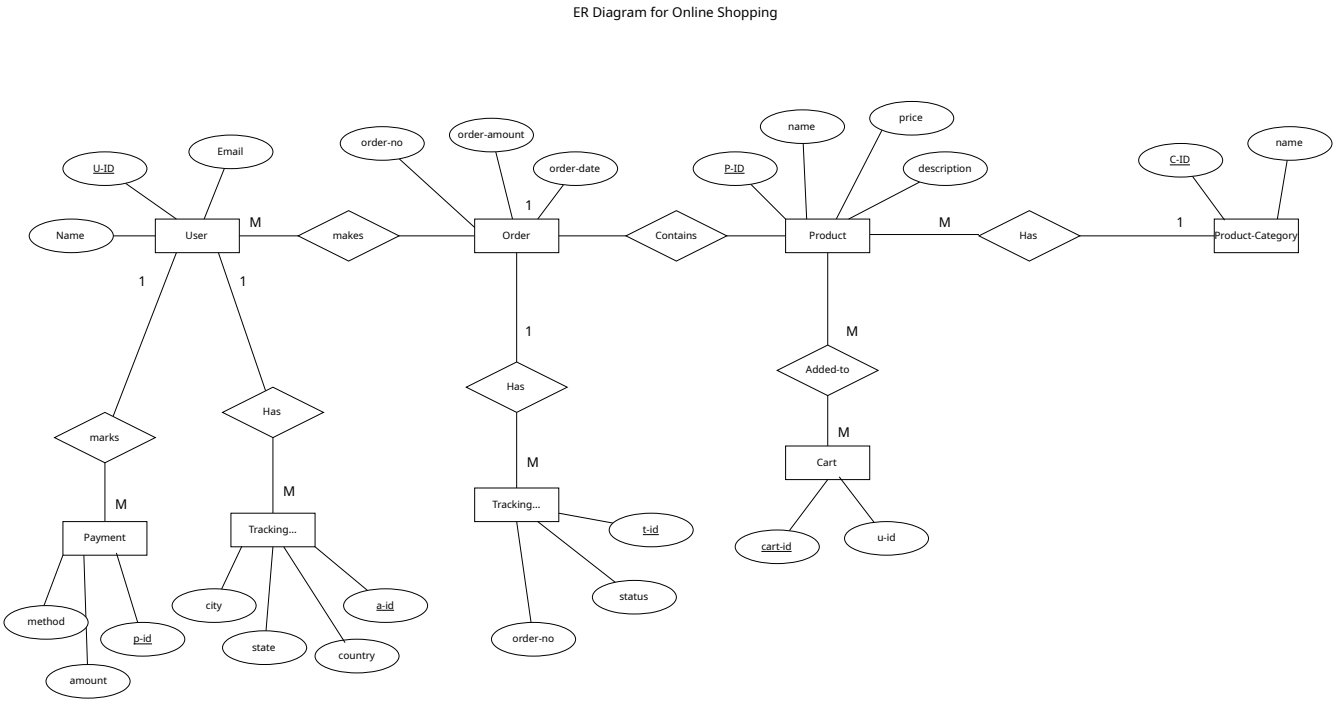
- ER Diagram of Hotel Management System



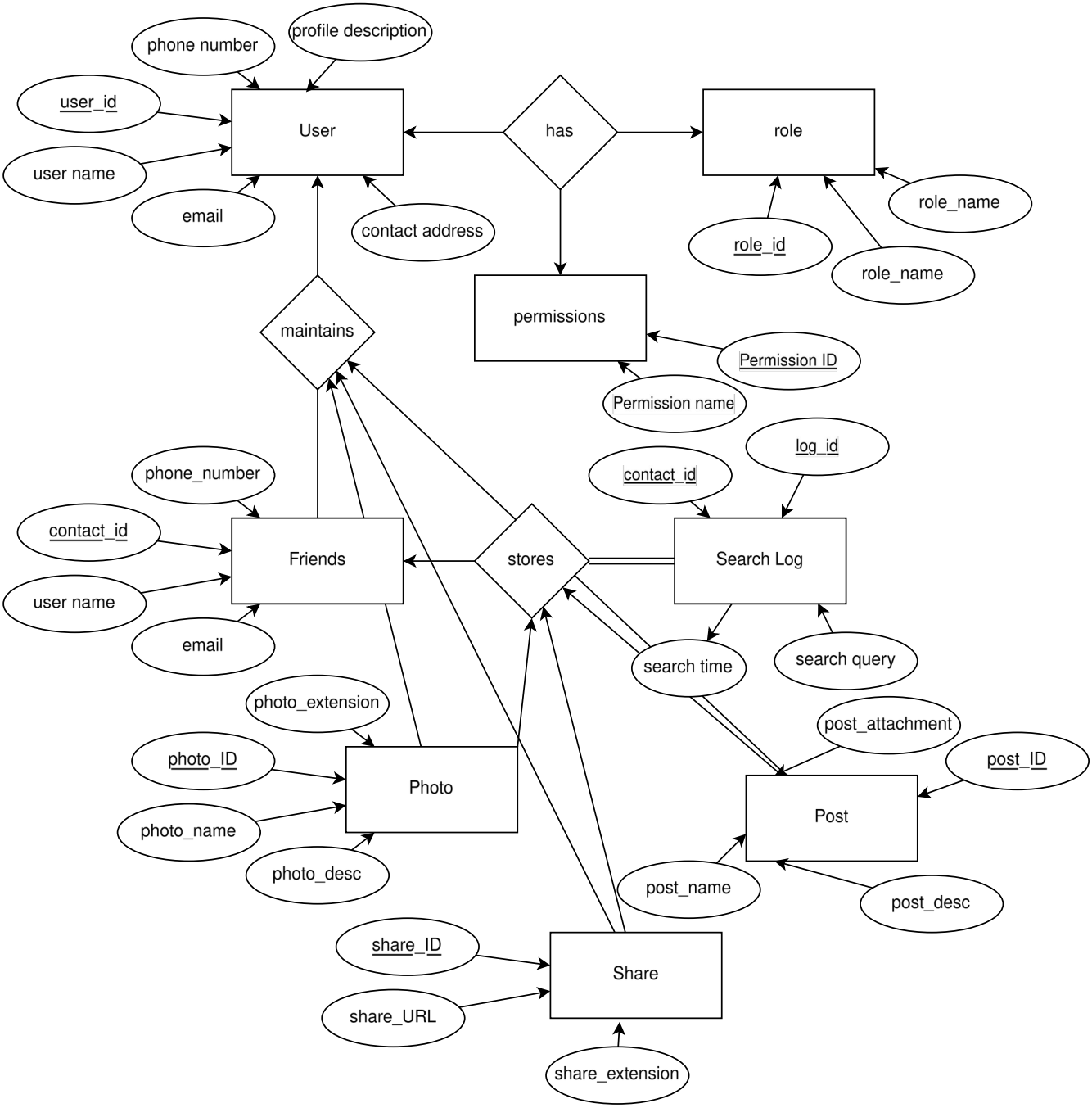
- ER Diagram of Library Management System



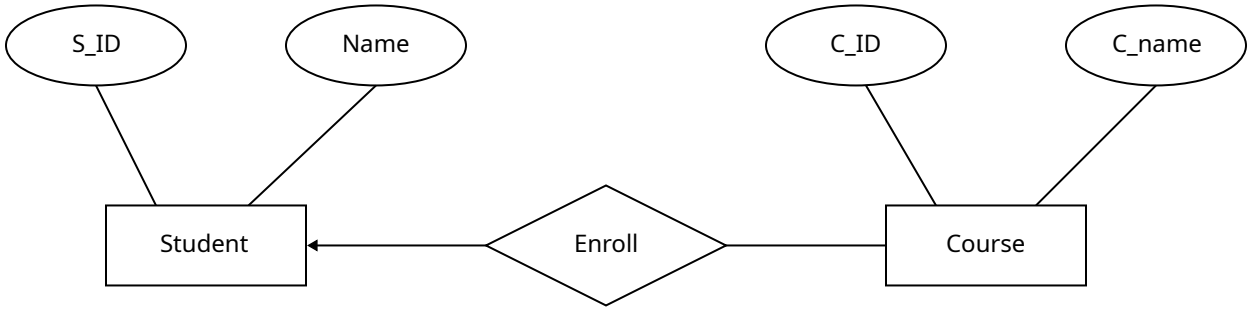
- ER Diagram of Online Shopping System



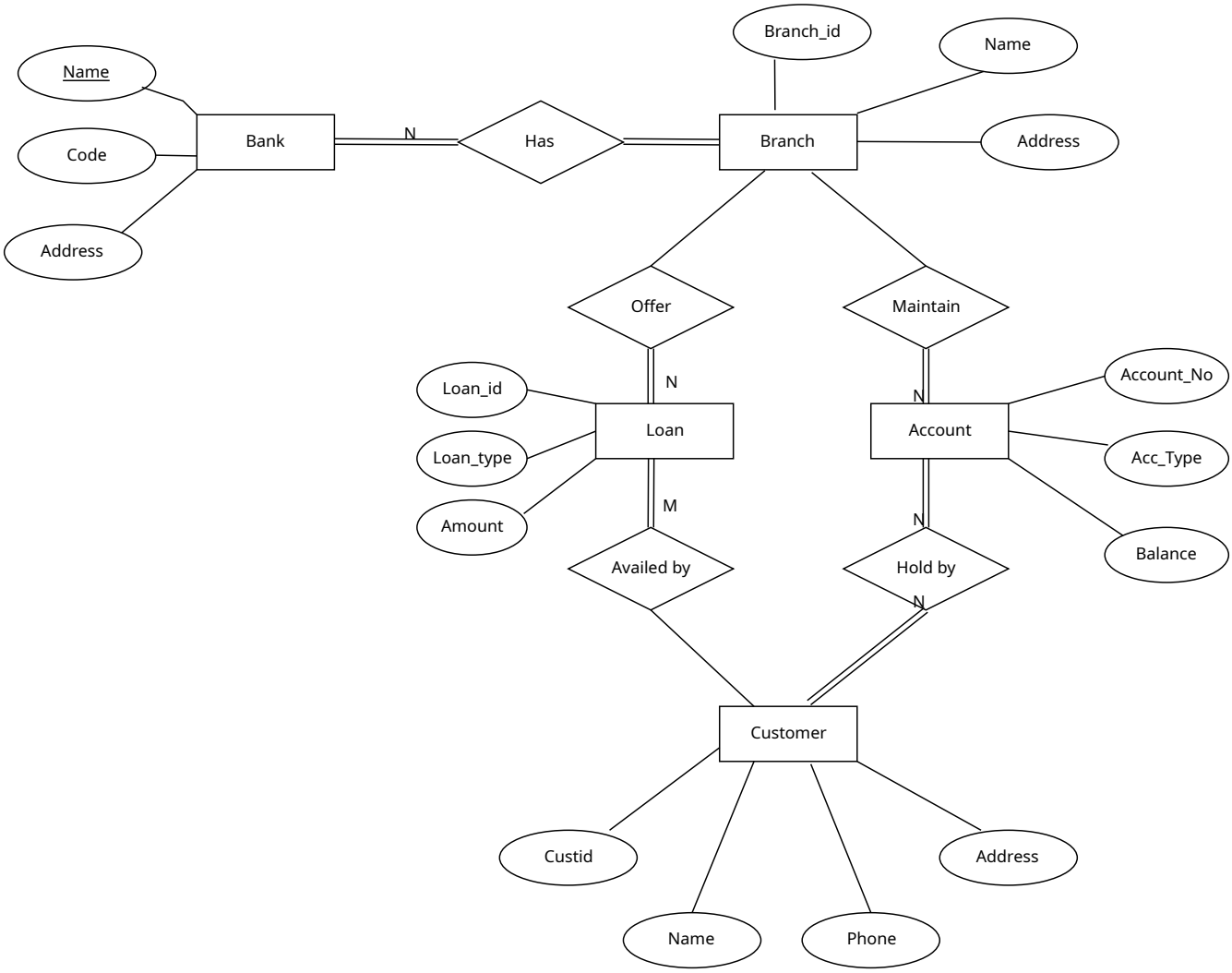
- ER Diagram of NoSQL Database



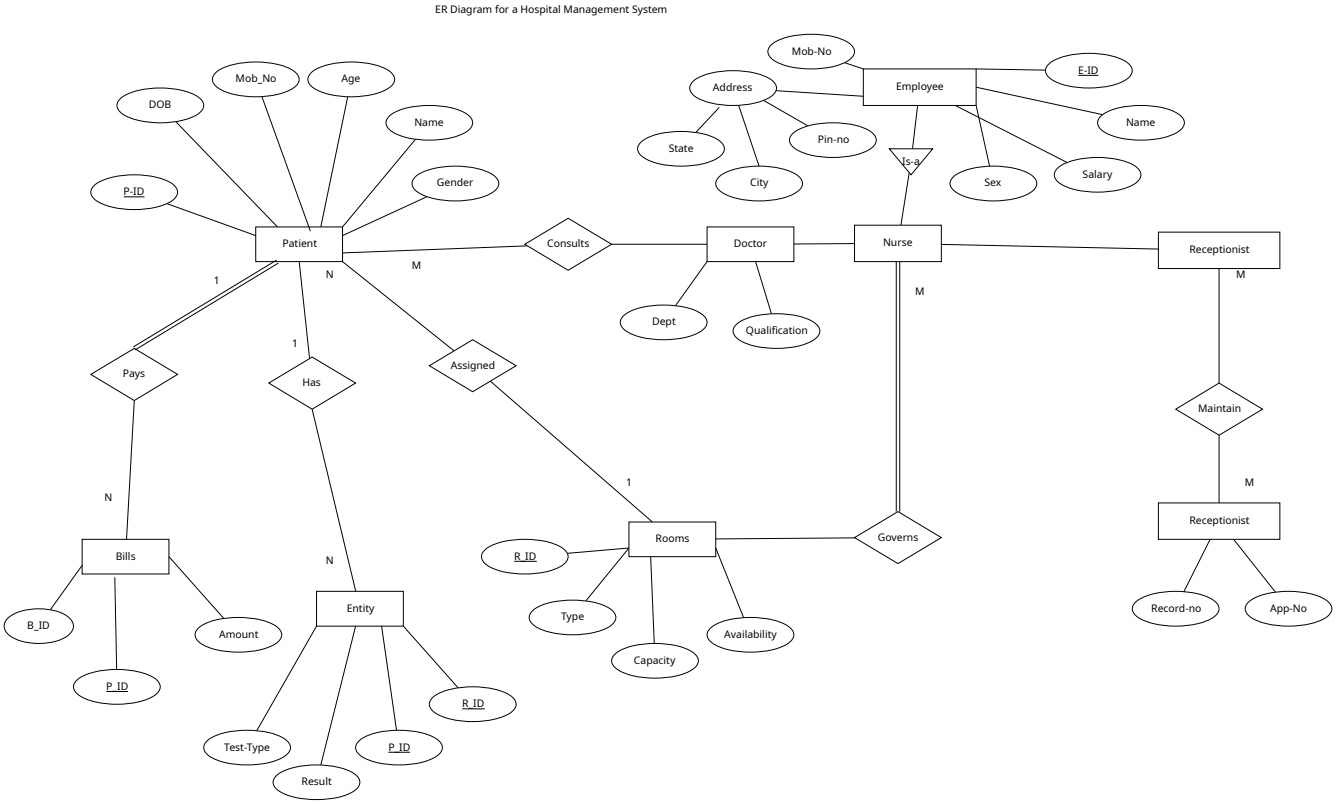
- ER Diagram of One-to-Many Relationship



- ER Diagram of Banking System



- ER Diagram of Hospital Management System



- Telephone directory Management system

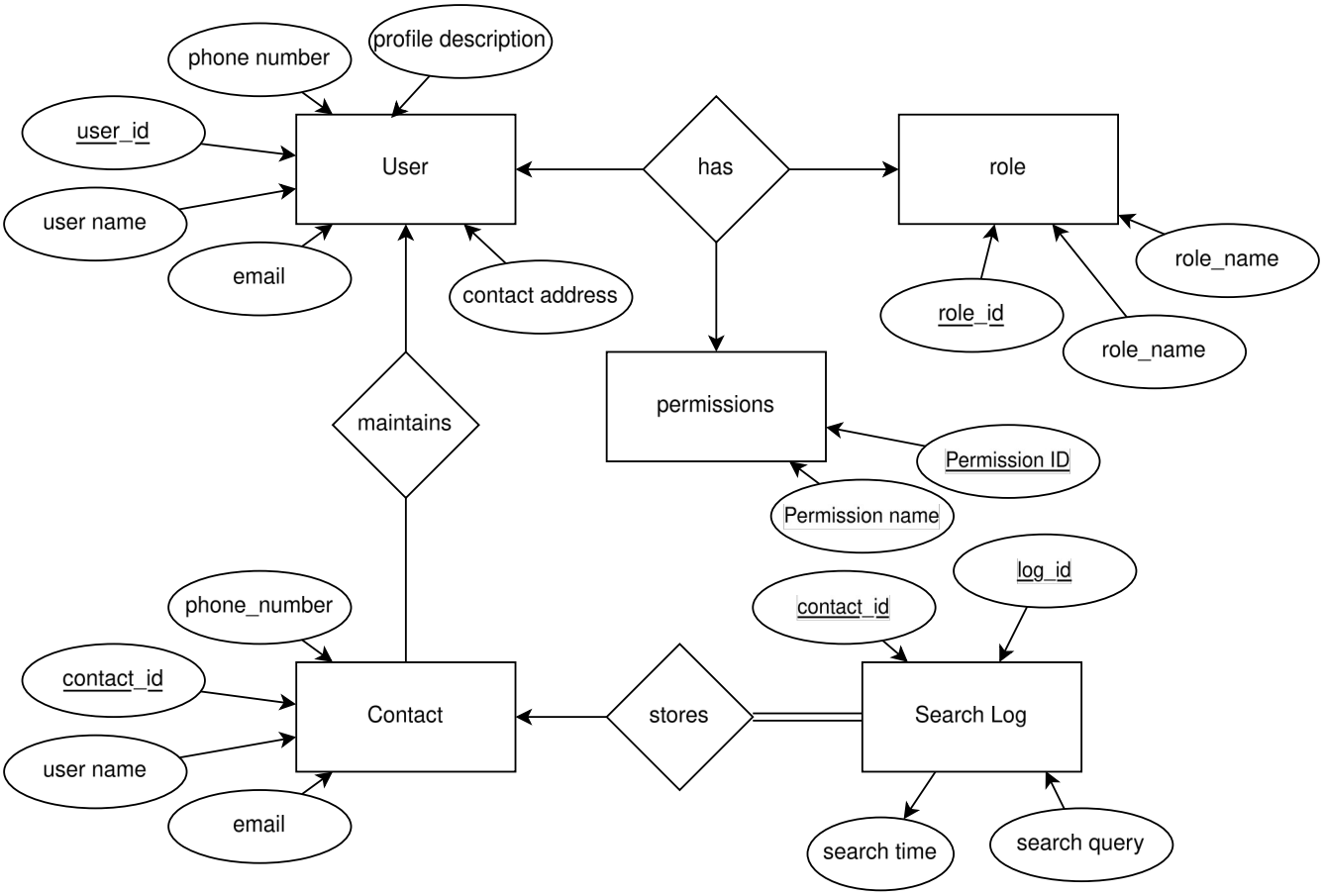


Table of contents

1 blogs

Creation: May 01, 2025 at 04:12 AM

Column	Type	Attributes	Null	Default	Extra	Links to	Comments	MIME
ID	int		No		auto_increment			
author_id	int		No			-> users.ID ON UPDATE RESTRICT ON DELETE RESTRICT		
title	varchar(255)		No					
content	text		No					
created_at	datetime		Yes	CURRENT_TIMESTAMP	DEFAULT_GENERATED			
updated_at	datetime		Yes	NULL	on update CURRENT_TIMESTAMP			
is_published	tinyint(1)		Yes	0				

2 blog_comments

Creation: May 12, 2025 at 06:33 PM

Column	Type	Attributes	Null	Default	Extra	Links to	Comments	MIME
ID	int		No		auto_increment			
blog_id	int		No			-> blogs.ID ON UPDATE RESTRICT ON DELETE RESTRICT		
user_id	int		No			-> users.ID ON UPDATE RESTRICT ON DELETE RESTRICT		
comment	text		No					
created_at	datetime		Yes	CURRENT_TIMESTAMP	DEFAULT_GENERATED			

3 blog_reactions

Creation: May 12, 2025 at 06:23 PM

Column	Type	Attributes	Null	Default	Extra	Links to	Comments	MIME
blog_id	int		No			-> blogs.ID ON UPDATE RESTRICT ON DELETE RESTRICT		
user_id	int		No			-> users.ID ON UPDATE RESTRICT ON DELETE RESTRICT		
reaction	enum('like', 'dislike')		No					
created_at	datetime		Yes	CURRENT_TIMESTAMP	DEFAULT_GENERATED			

4 contests

Creation: Jun 13, 2025 at 04:20 AM

Column	Type	Attributes	Null	Default	Extra	Links to	Comments	MIME
ID	int		No		auto_increment			
title	varchar(100)		No					
description	text		Yes	NULL				
start_time	datetime		No					
end_time	datetime		No					
is_public	tinyint(1)		Yes	1				
created_by	int		No			-> users.ID ON UPDATE RESTRICT ON DELETE RESTRICT		
created_at	datetime		Yes	CURRENT_TIMESTAMP	DEFAULT_GENERATED			

5 contest_registrations

Creation: Jun 13, 2025 at 09:43 AM

Column	Type	Attributes	Null	Default	Extra	Links to	Comments	MIME
user_id	int		No			-> users.ID ON UPDATE RESTRICT ON DELETE RESTRICT		
contest_id	int		No			-> contests.ID ON UPDATE RESTRICT ON DELETE RESTRICT		
registered_at	datetime		Yes	CURRENT_TIMESTAMP	DEFAULT_GENERATED			

6 feedback

Creation: Apr 06, 2025 at 03:59 AM

Column	Type	Attributes	Null	Default	Extra	Links to	Comments	MIME
ID	int		No		auto_increment			
name	varchar(50)		No					
email	varchar(100)		Yes	NULL				
feedback	text		No					
website	varchar(255)		Yes	NULL				
created_at	datetime		Yes	CURRENT_TIMESTAMP	DEFAULT_GENERATED			

7 newsletters

Creation: May 24, 2025 at 05:27 AM

Column	Type	Attributes	Null	Default	Extra	Links to	Comments	MIME
ID	int		No		auto_increment			
email	varchar(100)		No					
subscribed_at	datetime		Yes	CURRENT_TIMESTAMP	DEFAULT_GENERATED			

8 problems

Creation: Jun 13, 2025 at 09:43 AM

Column	Type	Attributes	Null	Default	Extra	Links to	Comments	MIME
ID	int		No		auto_increment			
contest_id	int		Yes	NULL		-> contests.ID ON UPDATE RESTRICT ON DELETE RESTRICT		
title	varchar(100)		No					
description	text		No					
difficulty	enum('easy', 'medium', 'hard')		Yes	medium				
time_limit	int		Yes	2				
memory_limit	int		Yes	256				
created_at	datetime		Yes	CURRENT_TIMESTAMP	DEFAULT_GENERATED			

9 submissions

Creation: Jun 13, 2025 at 04:21 PM

Column	Type	Attributes	Null	Default	Extra	Links to	Comments	MIME
ID	int		No		auto_increment			
user_id	int		No			-> users.ID ON UPDATE RESTRICT ON DELETE RESTRICT		
problem_id	int		No			-> problems.ID ON UPDATE RESTRICT ON DELETE RESTRICT		
code	text		No					
status	enum('pending', 'running', 'accepted', 'wrong_answer', 'time_limit_exceeded', 'runtime_error')		Yes	pending				
execution_time	float		Yes	NULL				
memory_used	int		Yes	NULL				
submitted_at	datetime		Yes	CURRENT_TIMESTAMP	DEFAULT_GENERATED			

10 test_cases

Creation: Jun 13, 2025 at 03:22 PM

Column	Type	Attributes	Null	Default	Extra	Links to	Comments	MIME
ID	int		No		auto_increment			
problem_id	int		No			-> problems.ID ON UPDATE RESTRICT ON DELETE RESTRICT		
input	text		Yes	NULL				
expected_output	text		Yes	NULL				
is_hidden	tinyint(1)		Yes	0				

11 users

Creation: Apr 08, 2025 at 06:43 AM

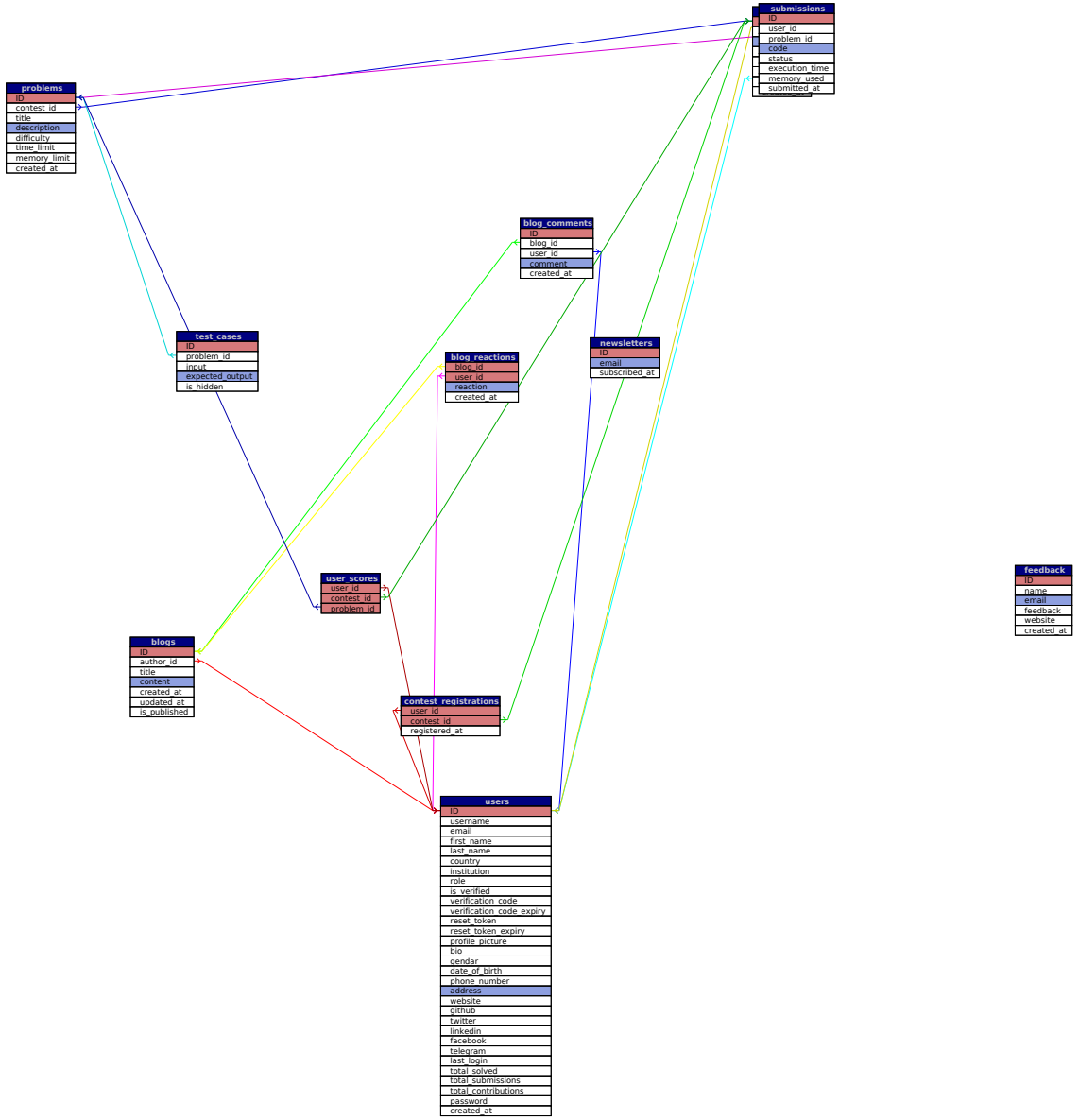
Column	Type	Attributes	Null	Default	Extra	Links to	Comments	MIME
ID	int		No		auto_increment			
username	varchar(50)		No					
email	varchar(100)		No					
first_name	varchar(50)		No					
last_name	varchar(50)		No					
country	varchar(50)		Yes	NULL				
institution	varchar(100)		Yes	NULL				
role	enum('admin', 'user')		No	user				
is_verified	tinyint(1)		Yes	0				
verification_code	varchar(255)		Yes	NULL				
verification_code_expiry	datetime		Yes	NULL				
reset_token	varchar(255)		Yes	NULL				
reset_token_expiry	datetime		Yes	NULL				
profile_picture	varchar(255)		Yes	NULL				
bio	text		Yes	NULL				
gendar	varchar(10)		Yes	NULL				
date_of_birth	date		Yes	NULL				
phone_number	varchar(20)		Yes	NULL				
address	varchar(255)		Yes	NULL				
website	varchar(255)		Yes	NULL				
github	varchar(255)		Yes	NULL				
twitter	varchar(255)		Yes	NULL				
linkedin	varchar(255)		Yes	NULL				
facebook	varchar(255)		Yes	NULL				

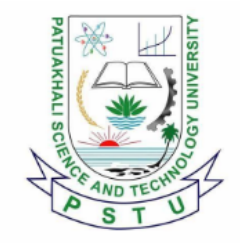
telegram	varchar(255)		Yes	NULL				
last_login	datetime		Yes	NULL				
total_solved	int		Yes	0				
total_submissions	int		Yes	0				
total_contributions	int		Yes	0				
password	varchar(255)		No					
created_at	datetime		Yes	CURRENT_TIMESTAMP	DEFAULT_GENERATED			

12 user_scores

Creation: Jun 14, 2025 at 03:56 AM

Column	Type	Attributes	Null	Default	Extra	Links to	Comments	MIME
user_id	int		No			-> users.ID ON UPDATE RESTRICT ON DELETE RESTRICT		
contest_id	int		No			-> contests.ID ON UPDATE RESTRICT ON DELETE RESTRICT		
problem_id	int		No			-> problems.ID ON UPDATE RESTRICT ON DELETE RESTRICT		


problems
ID
contest_id
title
description
difficulty
time_limit
memory_limit
created_at
submissions
ID
user_id
problem_id
code
status
execution_time
memory_used
submitted_at
blog_comments
ID
blog_id
user_id
comment
created_at
test cases
ID
problem_id
input
expected_output
is_hidden
blog_reactions
blog_id
user_id
reaction
created_at
newsletters
ID
email
subscribed_at
user_scores
user_id
contest_id
problem_id
blogs
ID
author_id
title
content
created_at
updated_at
is_published
contest_registrations
user_id
contest_id
registered_at
users
ID
username
email
first_name
last_name
country
institution
role
is_verified
verification_code
verification_code_expiry
reset_token
reset_token_expiry
profile_picture
bio
gendar
date_of_birth
phone_number
address
website
github
twitter
linkedin
facebook
telegram
last_login
total_solved
total_submissions
total_contributions
password
created_at
feedback
ID
name
email
feedback
website
created_at



Patuakhali Science and Technology University
Faculty of Computer Science and Engineering

CCE 224 :: Database System Sessional

Sessional Project Report

Project Title : SQL Judge
Submission Date : Sat 14, June 2025

Submitted to,

Prof. Dr. Md Samsuzzaman
Professor,
Department of Computer and Communication Engineering,
Patuakhali Science and Technology University.

Submitted by,

Md. Sharafat Karim
ID : 2102024,
Reg: 10151

Contents

SQL Judge

1. Introduction

An SQL learning platform that allows users to learn and practice SQL queries. It it provides a set of features including user registration, problem submission, and a leaderboard. And last but not least, it has built in Blog and chatsheet.

2. Objective

- To create a platform that allows users to learn and practice SQL queries in a fun and interactive way.
- To provide a set of features that will help users to learn and practice SQL queries.
- To create a platform that will help mentors and teachers to help spreading the knowledge of SQL and database management.
- To enable users to share their learningings through blogs and discussions.
- A quick way to find chatsheet and resources related to SQL and database management.

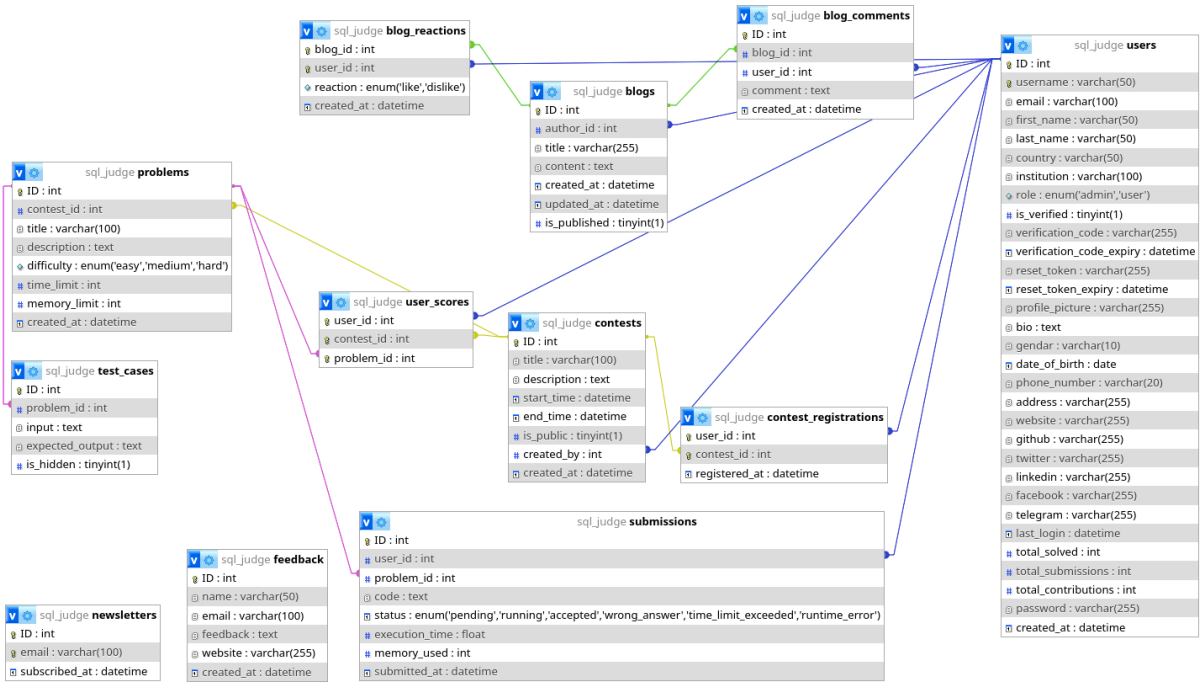
3. Technology

Layer	**Technology**
Frontend	HTML, CSS & JavaScript
Backend	PHP
Database	MySQL
Authentication	Session storage
Hosting	Localhost, infinityfree
Version control	Git
CI/ CD	GitHub

4. Database Characteristics

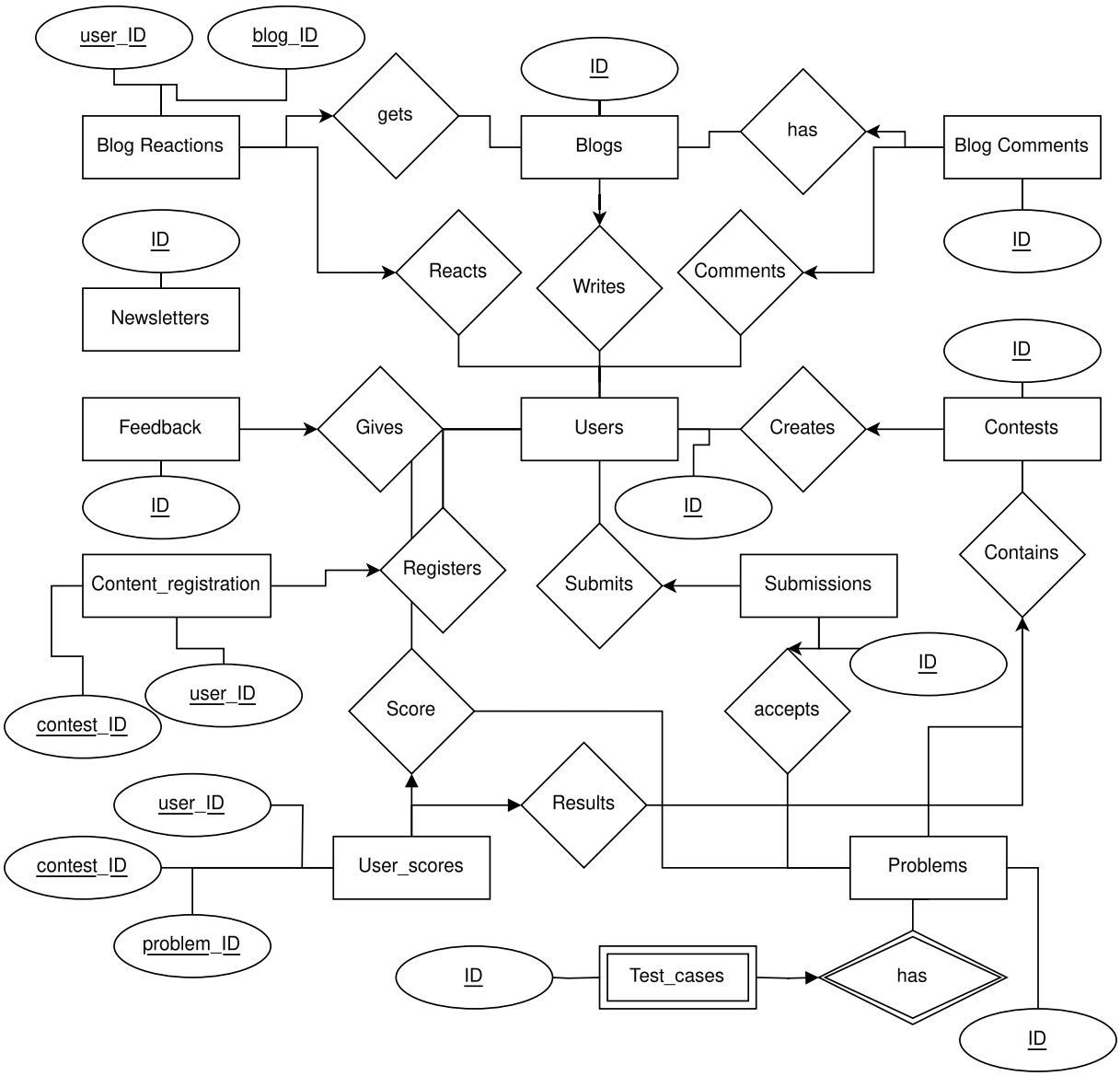
- **CRUD Operations** = Create, Read, Update, Delete
- **Data Integrity** is Enforced through foreign keys and constraints
- **Normalization**: Applied to reduce redundancy
- **Auth Security**: Implemented through user authentication and authorization. Mainly session storage is used for user authentication.
- **Php PDO driver** is used for database interactions, so that it can also connect to other databases like PostgreSQL, SQLite, etc.
- **Parameterized arguments** were used to prevent SQL injection attacks.
- **Database triggers** were used to automatically update total_contribution, total_submission and total_solved per user.

4.1. Schema Diagram

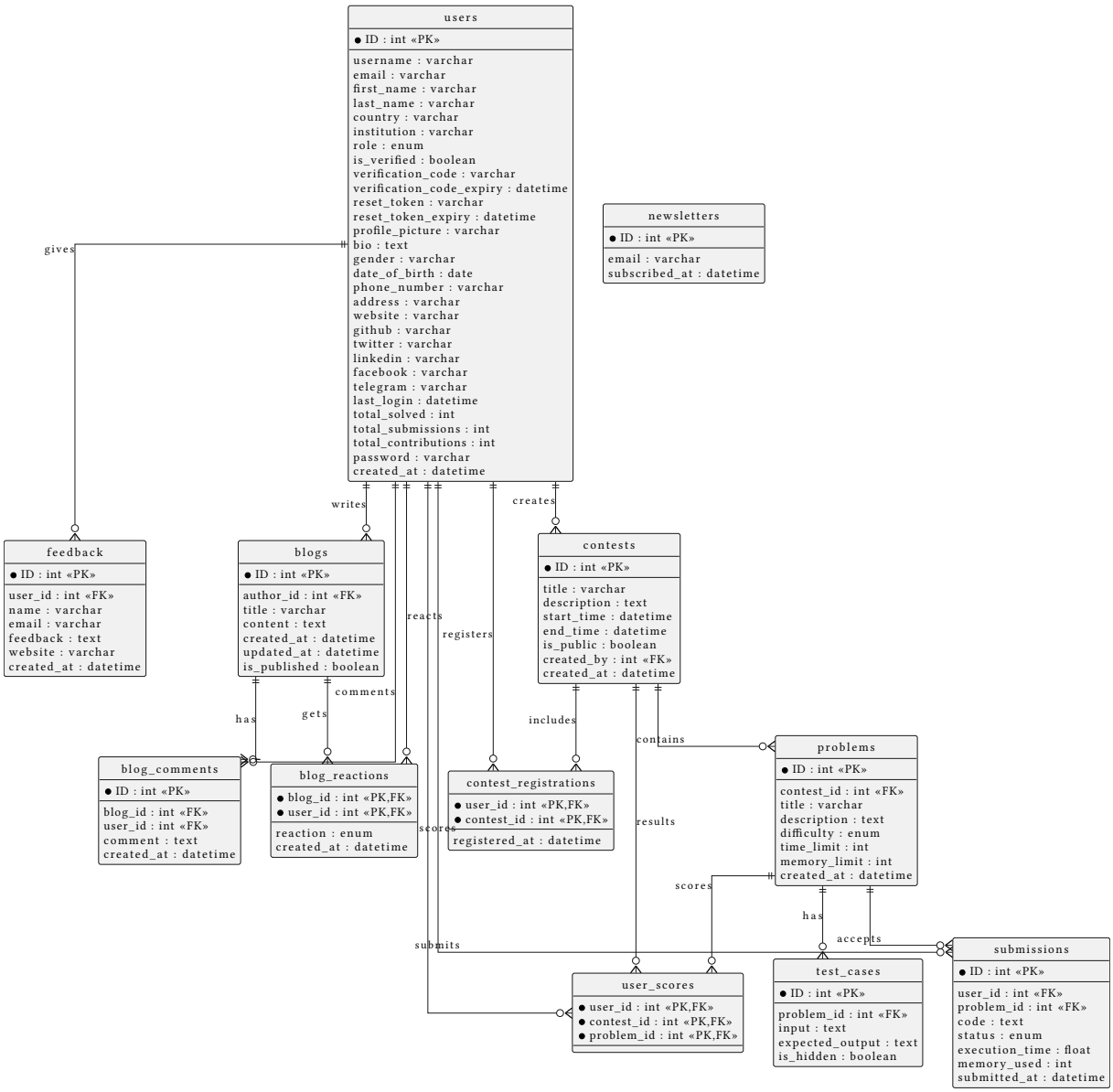


4.2. E-R Diagram

4.2.1. Without attributes



4.2.2. With all attributes



5. Database Implementation

5.1. DDL

Data definition language statements,

5.1.1. Database Creation

```
CREATE DATABASE IF NOT EXISTS sql_judge;
USE sql_judge;
```

5.1.2. Table Creation

5.1.2.1. Users Table

```
DROP TABLE IF EXISTS users;
CREATE TABLE users (
    ID INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
    username VARCHAR(50) NOT NULL UNIQUE,
    email VARCHAR(100) NOT NULL,
    first_name VARCHAR(50) NOT NULL,
    last_name VARCHAR(50) NOT NULL,
    country VARCHAR(50),
    institution VARCHAR(100),
    role ENUM('admin', 'user') NOT NULL DEFAULT 'user',
    is_verified BOOLEAN DEFAULT FALSE,
    verification_code VARCHAR(255),
    verification_code_expiry DATETIME,
    reset_token VARCHAR(255),
    reset_token_expiry DATETIME,
    profile_picture VARCHAR(255),
    bio TEXT,
    gender VARCHAR(10),
    date_of_birth DATE,
    phone_number VARCHAR(20),
    address VARCHAR(255),
    website VARCHAR(255),
    github VARCHAR(255),
    twitter VARCHAR(255),
    linkedin VARCHAR(255),
    facebook VARCHAR(255),
    telegram VARCHAR(255),
    last_login DATETIME,
    total_solved INT DEFAULT 0,
    total_submissions INT DEFAULT 0,
    total_contributions INT DEFAULT 0,
    password VARCHAR(255) NOT NULL,
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP
);
```

5.1.2.2. Feedback Table

```
DROP TABLE IF EXISTS feedback;
CREATE TABLE feedback (
    ID INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
    user_id INT,
    name VARCHAR(50) NOT NULL,
    email VARCHAR(100),
    feedback TEXT NOT NULL,
    website VARCHAR(255),
```

```
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (user_id) REFERENCES users(ID)
);
```

5.1.2.3. Blogs and Comments

```
DROP TABLE IF EXISTS blogs;
CREATE TABLE blogs (
    ID INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
    author_id INT NOT NULL,
    title VARCHAR(255) NOT NULL,
    content TEXT NOT NULL, -- HTML content
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
    updated_at DATETIME ON UPDATE CURRENT_TIMESTAMP,
    is_published BOOLEAN DEFAULT FALSE,
    FOREIGN KEY (author_id) REFERENCES users(ID)
);

DROP TABLE IF EXISTS blog_comments;
CREATE TABLE blog_comments (
    ID INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
    blog_id INT NOT NULL,
    user_id INT NOT NULL,
    comment TEXT NOT NULL,
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (blog_id) REFERENCES blogs(ID),
    FOREIGN KEY (user_id) REFERENCES users(ID)
);

DROP TABLE IF EXISTS blog_reactions;
CREATE TABLE blog_reactions (
    blog_id INT,
    user_id INT,
    reaction ENUM('like', 'dislike') NOT NULL,
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
    PRIMARY KEY (blog_id, user_id),
    FOREIGN KEY (blog_id) REFERENCES blogs(ID),
    FOREIGN KEY (user_id) REFERENCES users(ID)
);
```

5.1.2.4. Newsletters Table

```
DROP TABLE IF EXISTS newsletters;
CREATE TABLE newsletters (
    ID INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
    email VARCHAR(100) NOT NULL UNIQUE,
    subscribed_at DATETIME DEFAULT CURRENT_TIMESTAMP
);
```

5.1.2.5. Contests, Problems

```
DROP TABLE IF EXISTS contests;
CREATE TABLE contests (
    ID INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
    title VARCHAR(100) NOT NULL,
    description TEXT,
    start_time DATETIME NOT NULL,
    end_time DATETIME NOT NULL,
    is_public BOOLEAN DEFAULT TRUE,
```

```
    created_by INT NOT NULL,
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (created_by) REFERENCES users(ID)
);

DROP TABLE IF EXISTS contest_registrations;
CREATE TABLE contest_registrations (
    user_id INT NOT NULL,
    contest_id INT NOT NULL,
    registered_at DATETIME DEFAULT CURRENT_TIMESTAMP,
    PRIMARY KEY (user_id, contest_id),
    FOREIGN KEY (user_id) REFERENCES users(ID),
    FOREIGN KEY (contest_id) REFERENCES contests(ID)
);

DROP TABLE IF EXISTS problems;
CREATE TABLE problems (
    ID INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
    contest_id INT,
    title VARCHAR(100) NOT NULL,
    description TEXT NOT NULL,
    difficulty ENUM('easy', 'medium', 'hard') DEFAULT 'medium',
    time_limit INT DEFAULT 2, -- in seconds
    memory_limit INT DEFAULT 256, -- in MB
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (contest_id) REFERENCES contests(ID)
);

DROP TABLE IF EXISTS test_cases;
CREATE TABLE test_cases (
    ID INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
    problem_id INT NOT NULL,
    input TEXT,
    expected_output TEXT,
    is_hidden BOOLEAN DEFAULT FALSE,
    FOREIGN KEY (problem_id) REFERENCES problems(ID)
);
```

5.1.2.6. Submissions and User Scores

```
DROP TABLE IF EXISTS submissions;
CREATE TABLE submissions (
    ID INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
    user_id INT NOT NULL,
    problem_id INT NOT NULL,
    code TEXT NOT NULL,
    status ENUM('pending', 'running', 'accepted', 'wrong_answer',
'time_limit_exceeded', 'runtime_error') DEFAULT 'pending',
    execution_time FLOAT,
    memory_used INT,
    submitted_at DATETIME DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (user_id) REFERENCES users(ID),
    FOREIGN KEY (problem_id) REFERENCES problems(ID)
);

DROP TABLE IF EXISTS user_scores;
CREATE TABLE user_scores (
```

```
    user_id INT NOT NULL,
    contest_id INT NOT NULL,
    problem_id INT NOT NULL,
    PRIMARY KEY (user_id, contest_id, problem_id),
    FOREIGN KEY (user_id) REFERENCES users(ID),
    FOREIGN KEY (contest_id) REFERENCES contests(ID),
    FOREIGN KEY (problem_id) REFERENCES problems(ID)
);
```

5.1.3. Triggers

```
-- A trigger to increment total_contributions for the author when a new blog is
published
DELIMITER $$
CREATE TRIGGER increment_contributions_after_insert
AFTER INSERT ON blogs
FOR EACH ROW
BEGIN
    IF NEW.is_published = TRUE THEN
        UPDATE users
        SET total_contributions = total_contributions + 5
        WHERE ID = NEW.author_id;
    END IF;
END$$
DELIMITER ;

-- Trigger to decrement total_contributions when a blog is updated from published to
draft
DELIMITER $$
CREATE TRIGGER decrement_contributions_after_update_to_draft
AFTER UPDATE ON blogs
FOR EACH ROW
BEGIN
    IF OLD.is_published = TRUE AND NEW.is_published = FALSE THEN
        UPDATE users
        SET total_contributions = total_contributions - 5
        WHERE ID = NEW.author_id;
    END IF;
END$$
DELIMITER ;

-- Trigger to increment total_contributions when a blog is updated from draft to
published
DELIMITER $$
CREATE TRIGGER increment_contributions_after_update_to_publish
AFTER UPDATE ON blogs
FOR EACH ROW
BEGIN
    IF OLD.is_published = FALSE AND NEW.is_published = TRUE THEN
        UPDATE users
        SET total_contributions = total_contributions + 5
        WHERE ID = NEW.author_id;
    END IF;
END$$
DELIMITER ;

-- Trigger to increment total_contributions by 1 when a new comment is added
```

```
DELIMITER $$
CREATE TRIGGER increment_contributions_after_comment
AFTER INSERT ON blog_comments
FOR EACH ROW
BEGIN
    UPDATE users
    SET total_contributions = total_contributions + 1
    WHERE ID = NEW.user_id;
END$$
DELIMITER ;

-- Trigger to increment total_submissions by 1 when a new submission is added
DELIMITER $$
CREATE TRIGGER increment_total_submissions_after_insert
AFTER INSERT ON submissions
FOR EACH ROW
BEGIN
    UPDATE users
    SET total_submissions = total_submissions + 1
    WHERE ID = NEW.user_id;
END$$
DELIMITER ;

-- Trigger to increment total_solved by 1 when a new user_scores entry is added
DELIMITER $$
CREATE TRIGGER increment_total_solved_after_user_score_insert
AFTER INSERT ON user_scores
FOR EACH ROW
BEGIN
    UPDATE users
    SET total_solved = total_solved + 1
    WHERE ID = NEW.user_id;
END$$
DELIMITER ;
```

5.1.4. Views

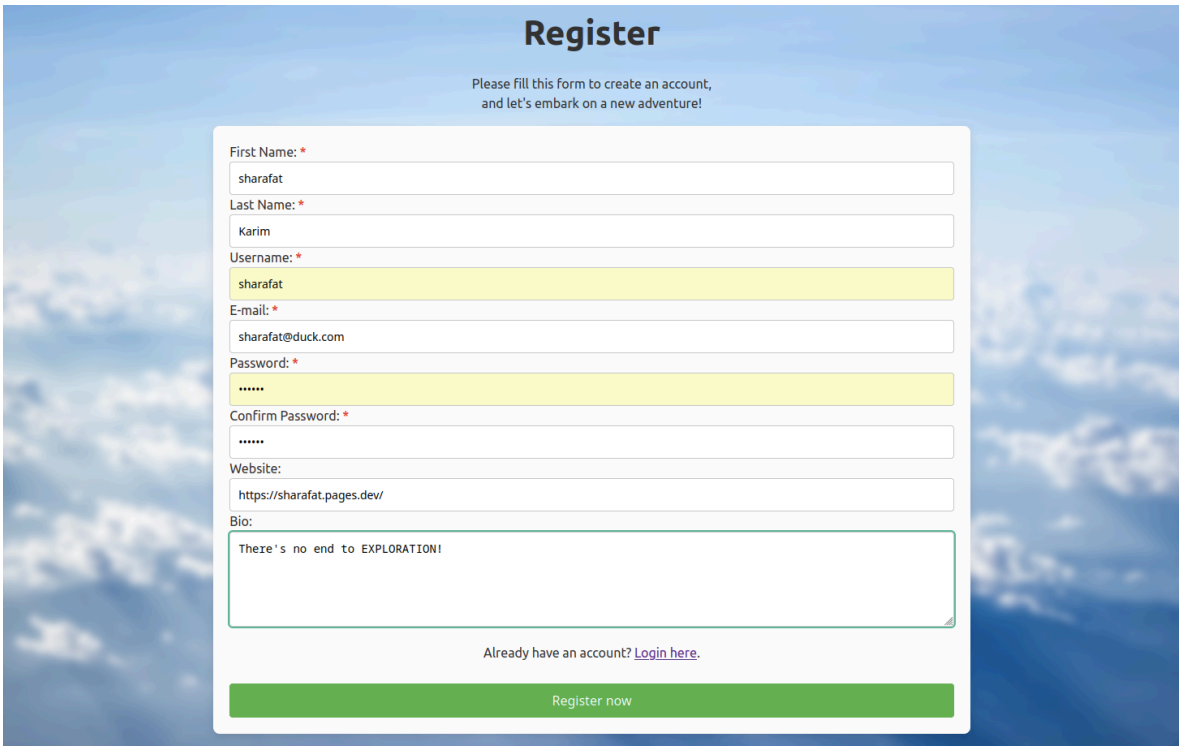
```
-- View to get the top 5 users based on total_solved
CREATE VIEW top_rated_5 as
SELECT username, first_name, last_name, total_solved
FROM users
ORDER BY total_solved DESC LIMIT 5;

-- View to get the top 5 users based on total_contributions
CREATE VIEW top_contributors_5 as
SELECT username, first_name, last_name, total_contributions
FROM users
ORDER BY total_contributions DESC LIMIT 5;
```

5.2. DML (SQL Queries)

5.2.1. Authentication

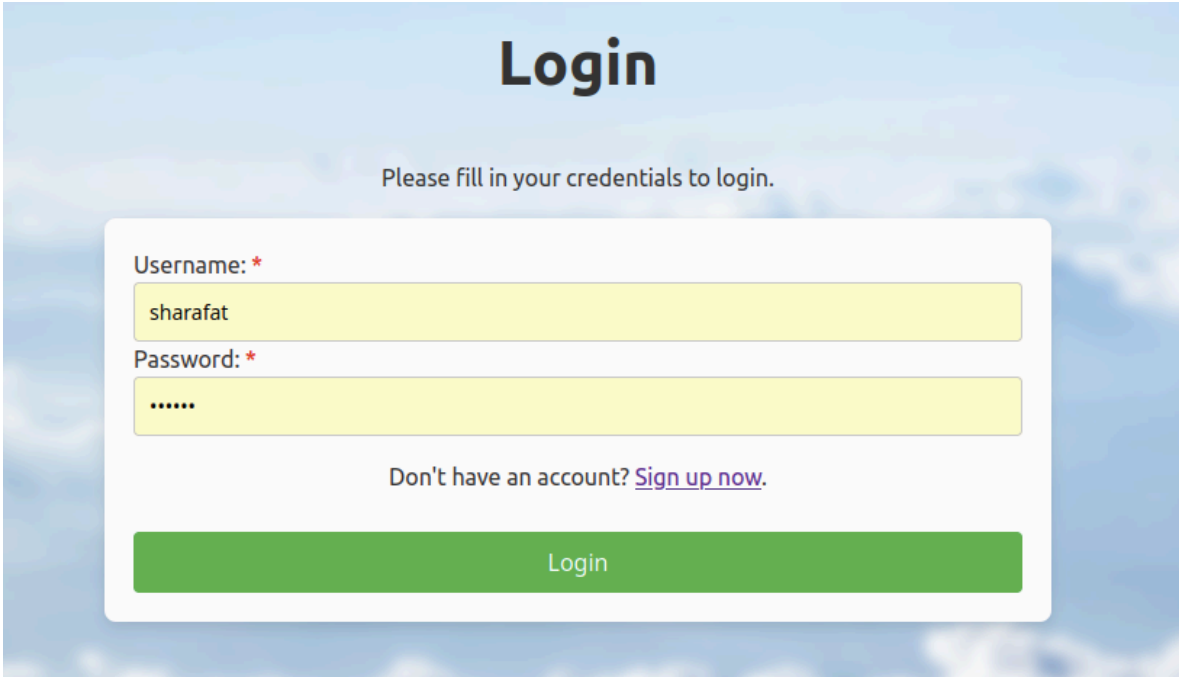
1. **User Registration**

```
INSERT INTO users (username, email, first_name, last_name, password, website, bio)
VALUES (:username, :email, :first_name, :last_name, :password, :website, :bio)
```



2. **User Login**

```
SELECT id, username, password FROM users WHERE username = :username"
```



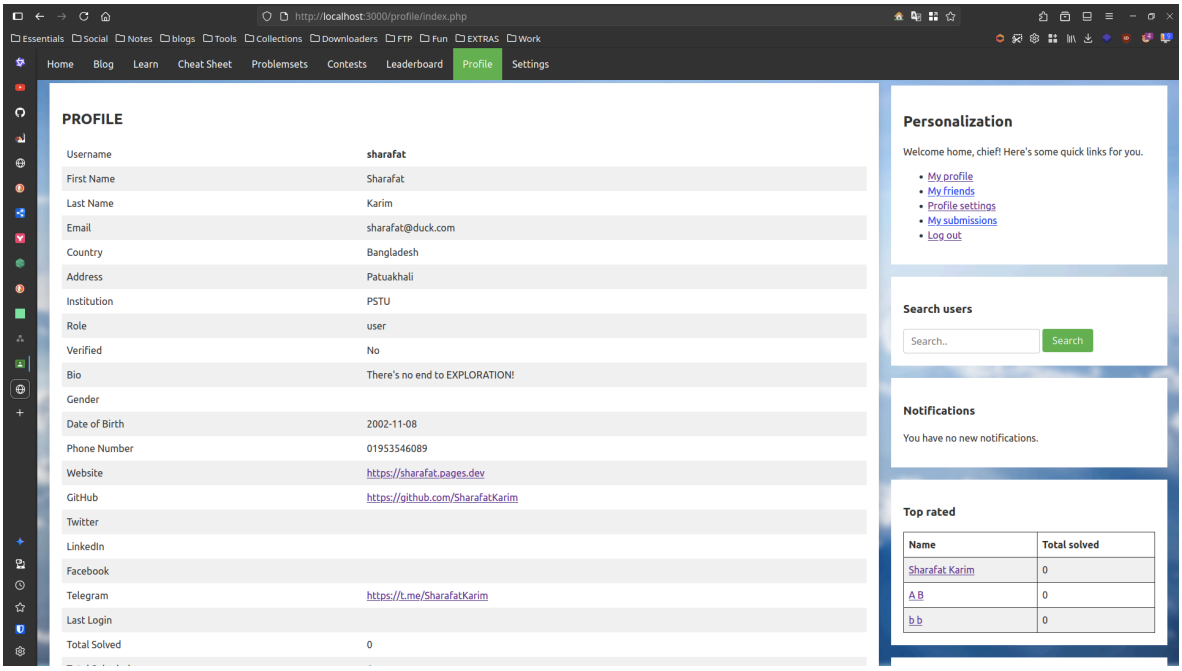
3. **Check if user already exists**

```
SELECT id FROM users WHERE username = :username"
```

5.2.2. User Profile

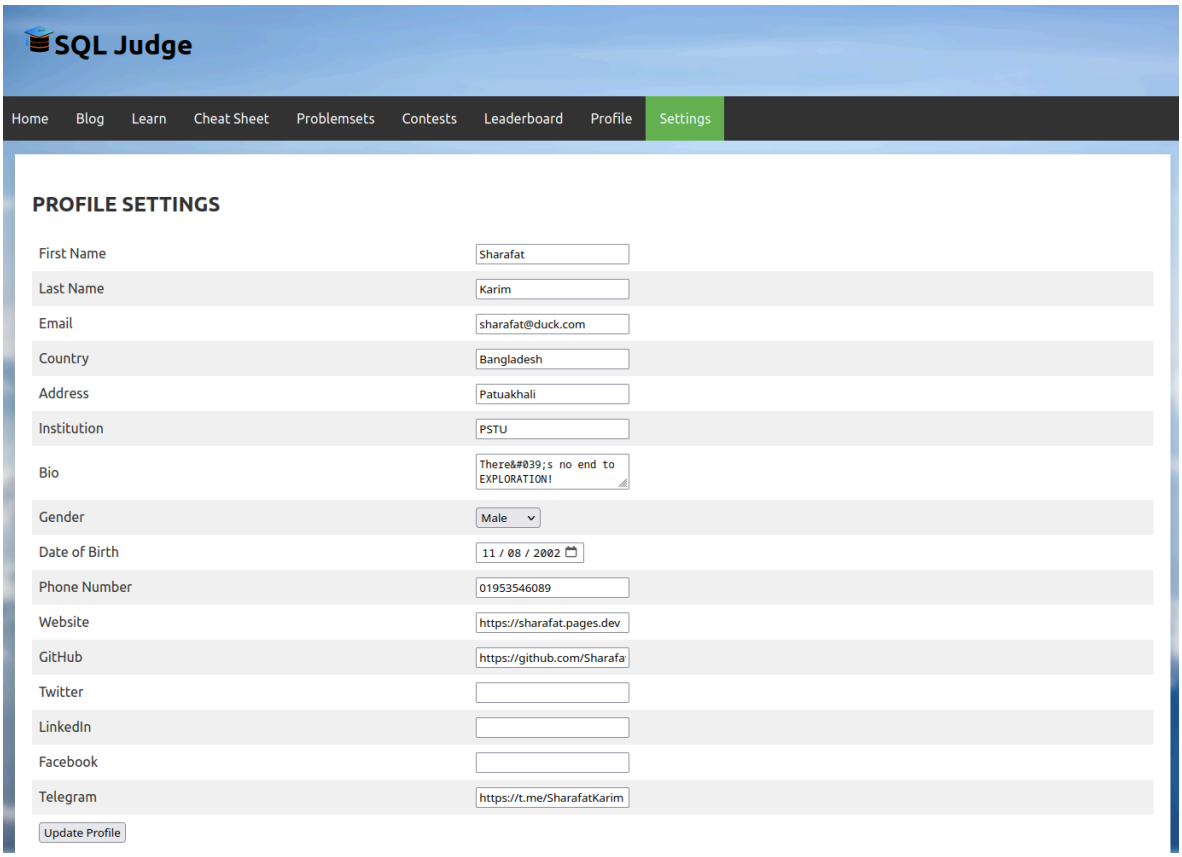
1. **Get user profile**

```
SELECT * FROM users WHERE username = :username

  # ID, username, email, first_name, last_name, country, institution, role,
is_verified, verification_code, verification_code_expiry, reset_token,
reset_token_expiry, profile_picture, bio, gendar, date_of_birth, phone_number,
address, website, github, twitter, linkedin, facebook, telegram, last_login,
total_solved, total_submissions, total_contributions, password, created_at
  1, sharafat, sharafat@duck.com, Sharafat, Karim, Bangladesh, PSTU, user,
0, , , , , , There&#039;s no end to EXPLORATION!, Male, 2002-11-08, 01953546089,
Patuakhali, https://sharafat.pages.dev, https://github.com/SharafatKarim, , , ,
https://t.me/SharafatKarim, , 0, 1, 32, $2y$12$CceqDu/
Ww9T44k2SdgT5DuzeeyR2ZanlSD8rvZlA/MXcGd3iC2Gbe, 2025-04-08 06:44:18
```



2. **Update user profile**

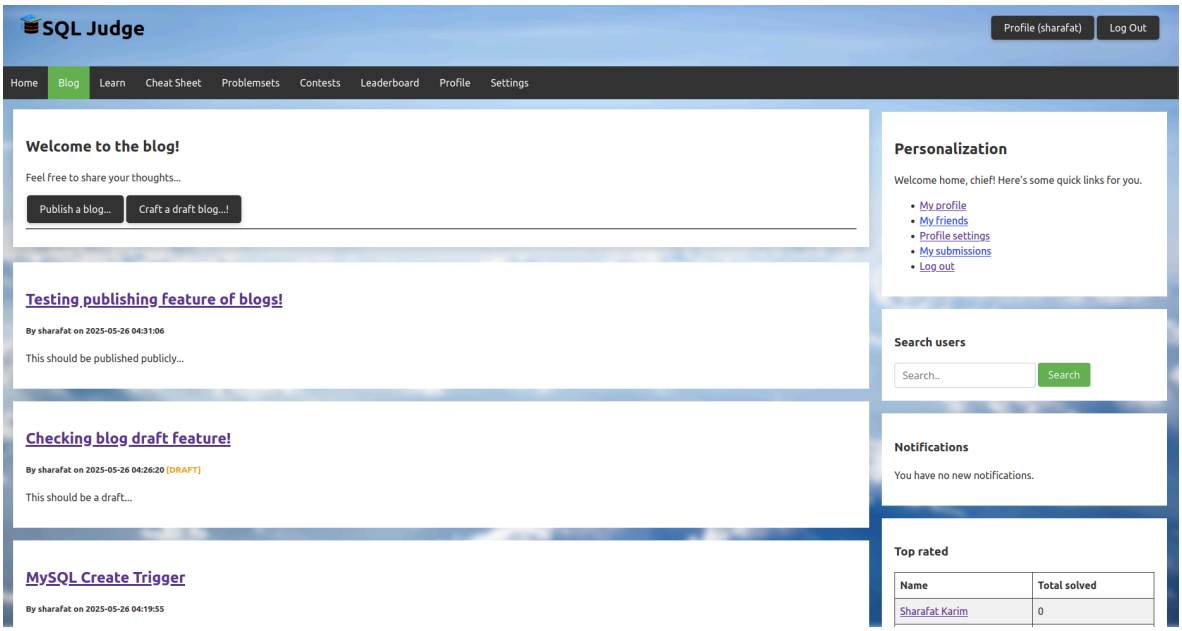
```
UPDATE users SET
          first_name = :first_name,
          last_name = :last_name,
          email = :email,
          country = :country,
          address = :address,
          institution = :institution,
          bio = :bio,
          gender = :gender,
          date_of_birth = :date_of_birth,
          phone_number = :phone_number,
          website = :website,
          github = :github,
          twitter = :twitter,
          linkedin = :linkedin,
          facebook = :facebook,
          telegram = :telegram
        WHERE username = :username"
```



5.2.3. Blog

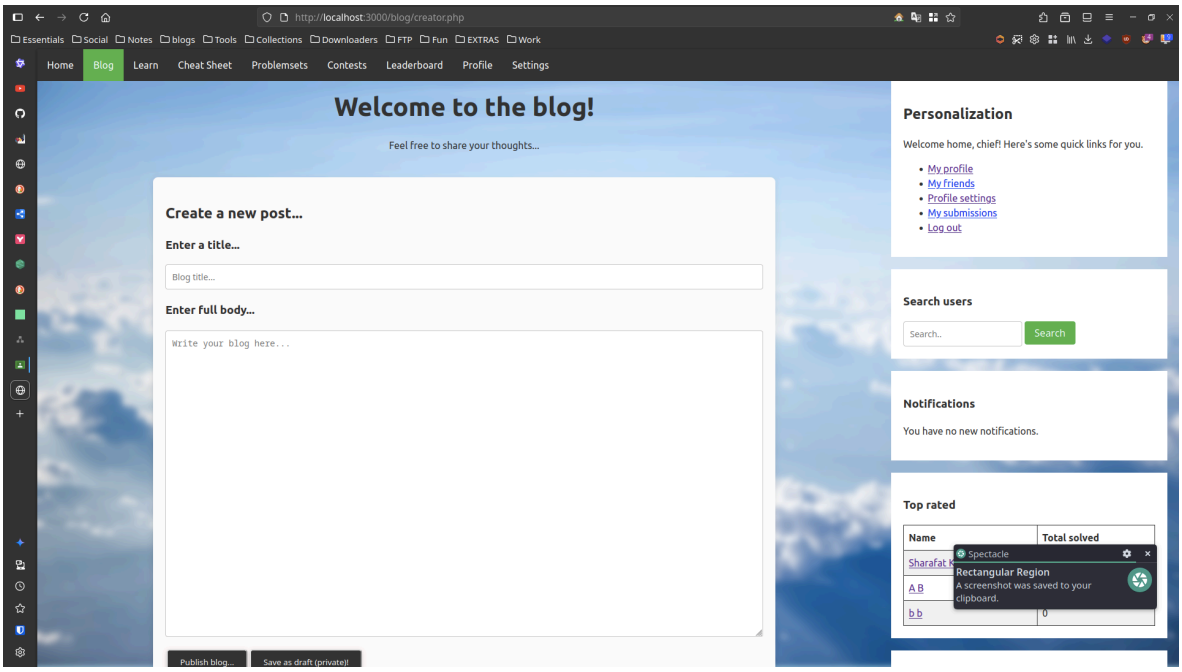
1. **Get all blogs**

```
  SELECT blogs.ID, blogs.title, blogs.content, blogs.created_at, users.username,
blogs.is_published
  FROM blogs
  JOIN users ON blogs.author_id = users.ID
  WHERE blogs.is_published = 1 OR blogs.author_id = 1
  ORDER BY blogs.created_at DESC

  # ID, title, content, created_at, username, is_published
'9', 'Testing publishing feature of blogs!', 'This should be published
publicly...', '2025-05-26 04:31:06', 'sharafat', '1'
```



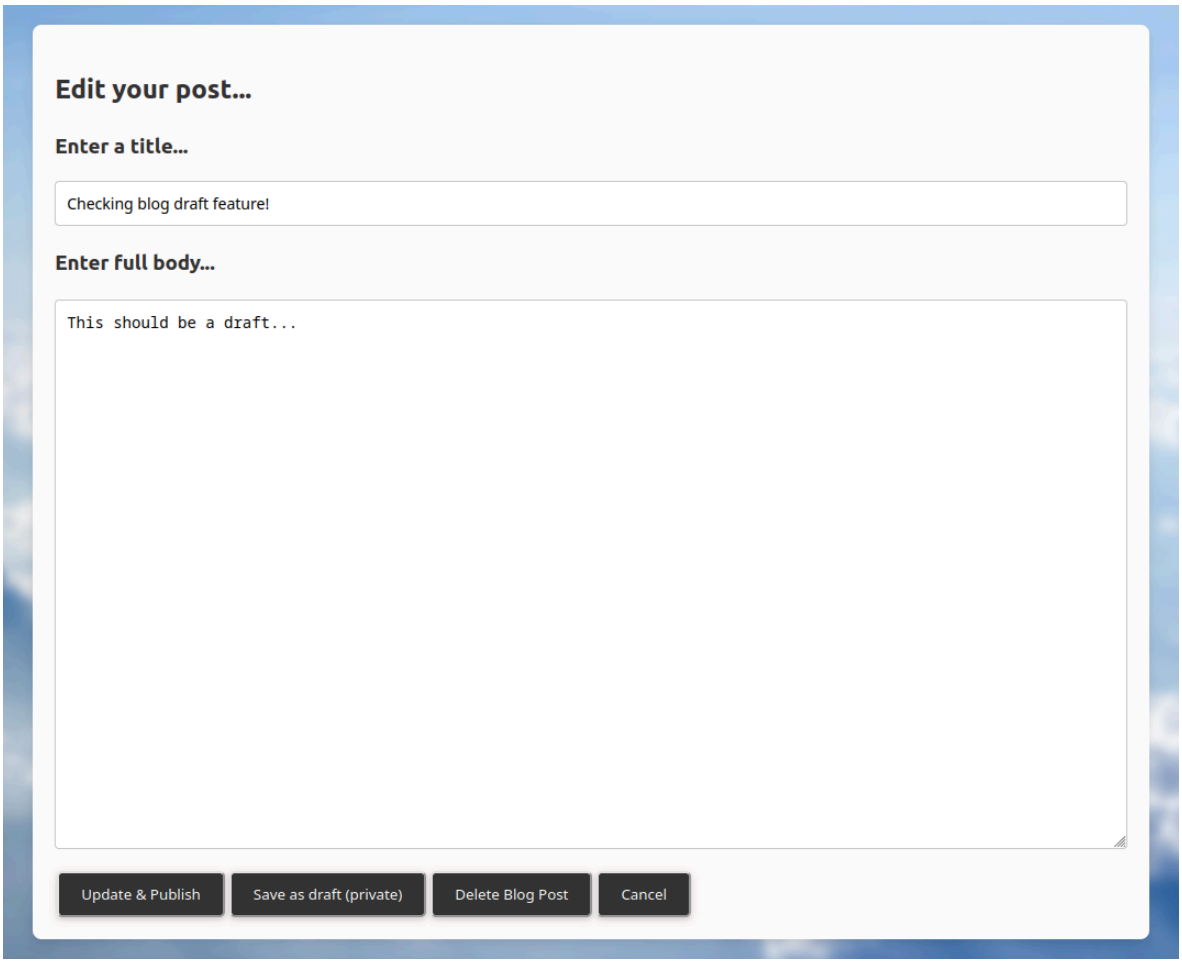
2. **Insert a new data (blog)**

```
INSERT INTO blogs (author_id, title, content, is_published) VALUES
(:author_id, :title, :content, :is_published)
```



3. **Update a blog**

```
UPDATE blogs SET
          title = :title,
          content = :content,
          is_published = :is_published
        WHERE ID = :blog_id"
```



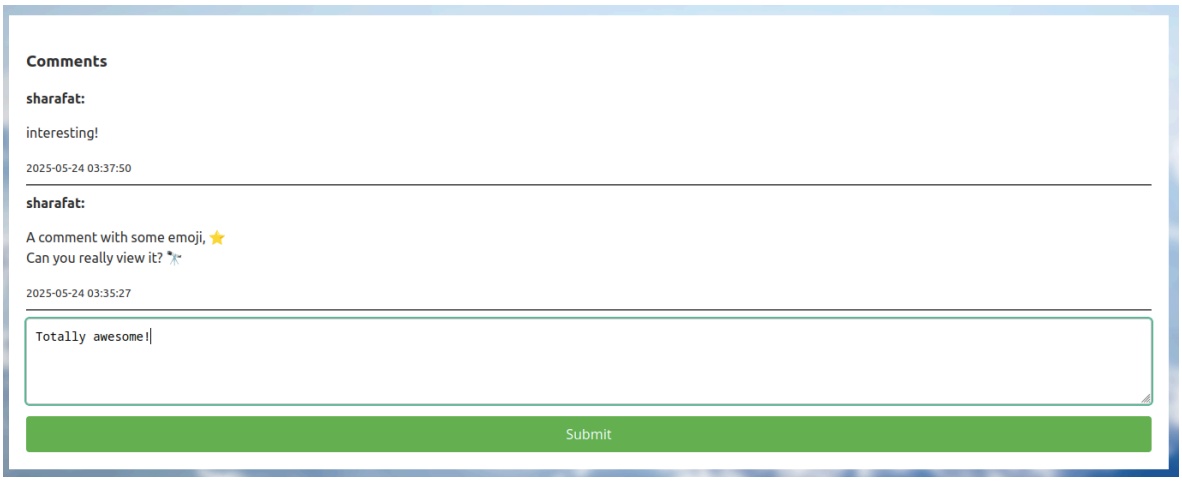
4. **Delete a blog**

```
DELETE FROM blogs WHERE ID = :blog_id
```

5.2.4. Comment & React

1. **Insert a new comment**

```
INSERT INTO blog_comments (blog_id, user_id, comment) VALUES
(:blog_id, :user_id, :comment)
```



2. **Get all comments for a blog**

```
  SELECT blog_comments.comment, blog_comments.created_at, users.username
  FROM blog_comments
  JOIN users
  ON blog_comments.user_id = 1
  WHERE blog_comments.blog_id = 1
  ORDER BY blog_comments.created_at DESC

  # comment, created_at, username
'interesting!', '2025-05-24 03:37:50', 'a'
```

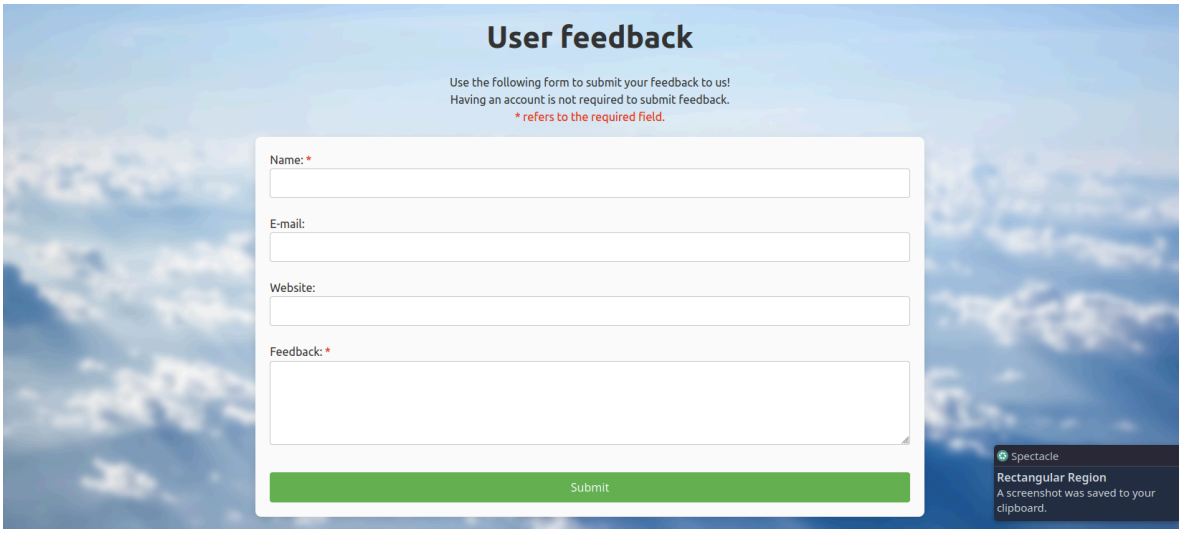
3. **Fetch reactions**

```
SELECT reaction, COUNT(*) as count
FROM blog_reactions
WHERE blog_id = 1 GROUP BY reaction
```

5.2.5. feedback

1. **Insert a new feedback**

```
INSERT INTO feedback (user_id, name, email, feedback, website) VALUES
(:user_id, :name, :email, :feedback, :website)
```



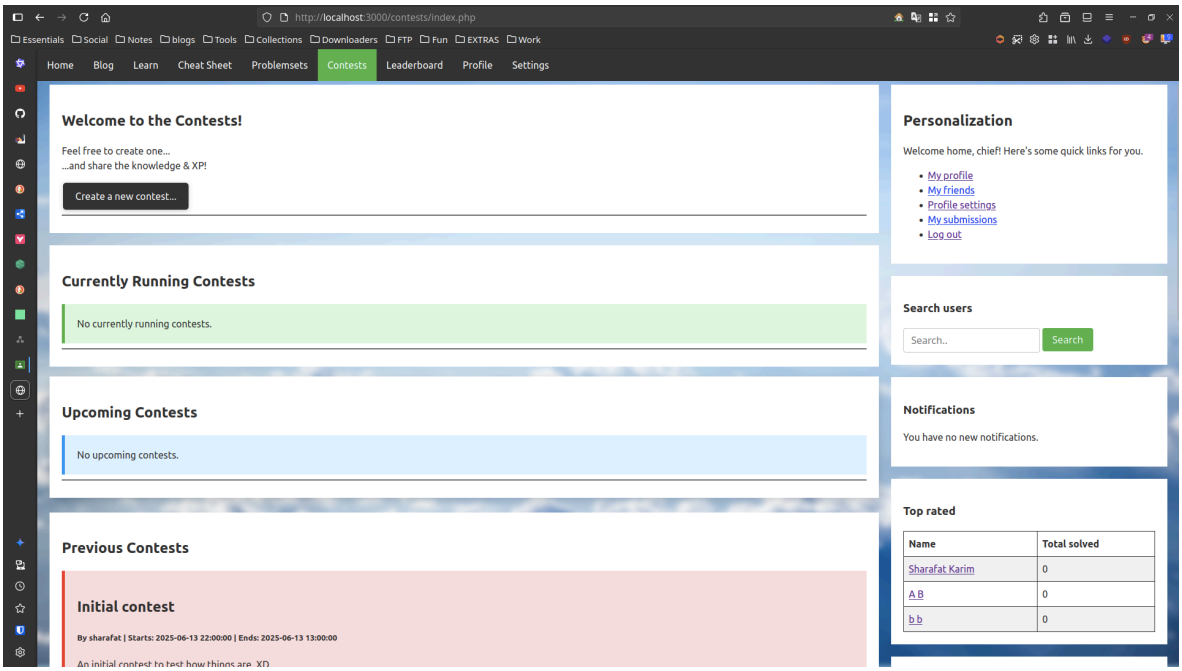
5.2.6. Contest

1. **Get all upcoming contests**

```
  SELECT contests.*, users.username
  FROM contests
  JOIN users
    ON contests.created_by = users.ID
  WHERE contests.start_time > NOW()
  ORDER BY contests.start_time ASC;
```

2. **Get all ongoing contests**

```
SELECT contests.*, users.username
FROM contests
JOIN users
  ON contests.created_by = users.ID
WHERE contests.start_time <= NOW()
  AND contests.end_time > NOW()
ORDER BY contests.start_time ASC;
```



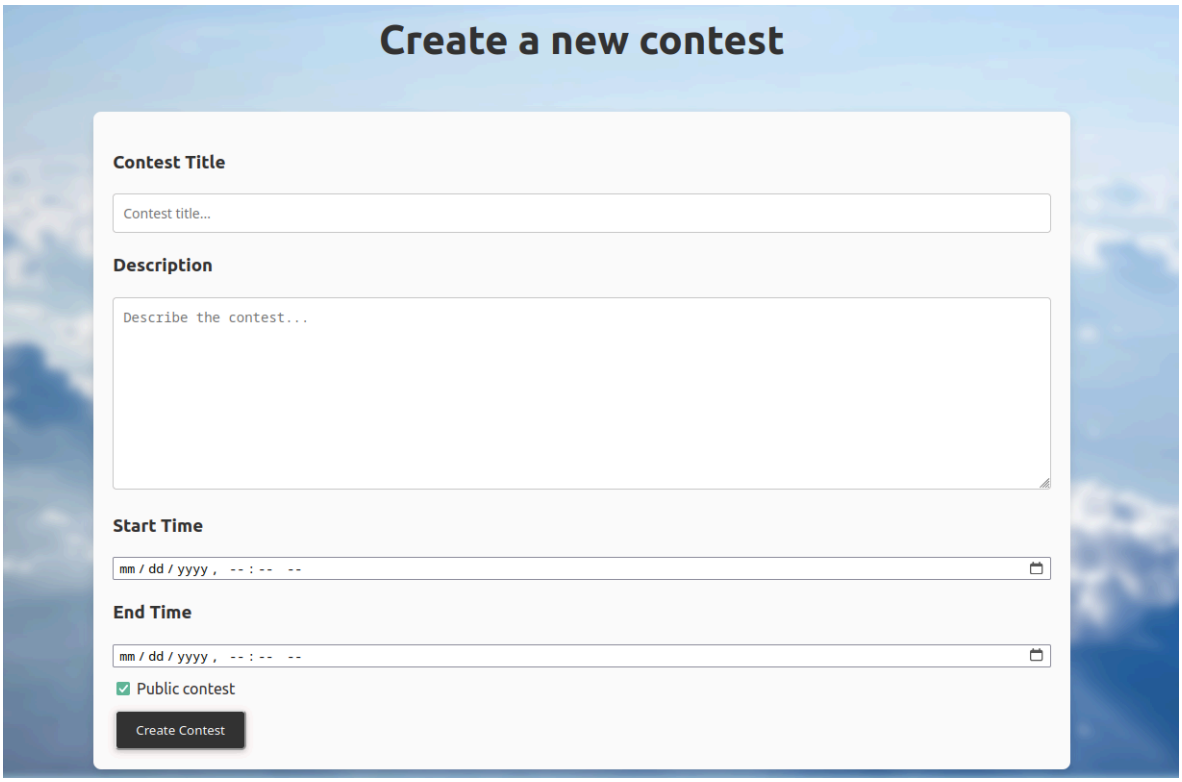
3. **Get all previous contests**

```sql
SELECT contests.*, users.username
FROM contests
JOIN users
  ON contests.created_by = users.ID
WHERE contests.end_time <= NOW()
ORDER BY contests.start_time ASC;
```

```
# ID, title, description, start_time, end_time, is_public, created_by, created_at,
username
'3', 'A past title!', 'A title happended to be happed in the past.', '1992-06-13
00:00:00', '2000-06-21 00:00:00', '1', '1', '2025-06-13 05:03:46', 'sharafat'
'2', 'Second one...', 'A second contest!', '2025-06-13 10:00:00', '2025-06-13
11:00:00', '1', '1', '2025-06-13 04:28:17', 'sharafat'
'1', 'Initial contest', 'An initial contest to test how things are, XD',
'2025-06-13 22:00:00', '2025-06-13 13:00:00', '1', '1', '2025-06-13 04:22:07',
'sharafat'
```

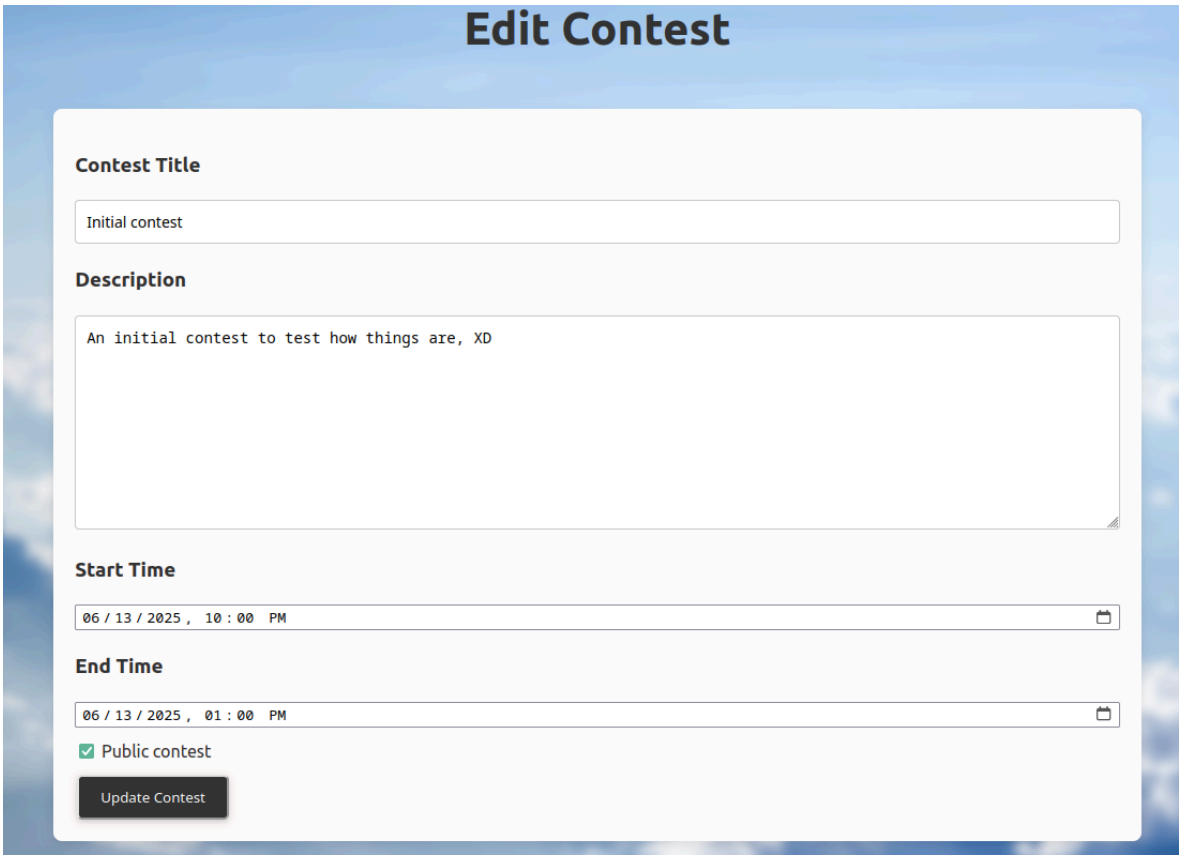
4. **Add new contest**

```sql
INSERT INTO contests (title, description, start_time, end_time, is_public,
created_by)
VALUES (:title, :description, :start_time, :end_time, :is_public, :created_by)
```



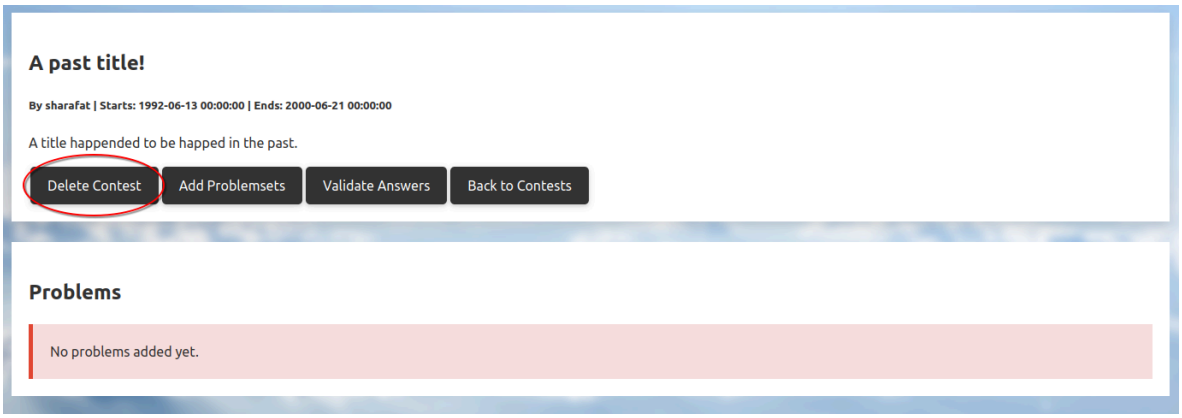
5. **Update contest**

```
UPDATE contests SET
          title = :title,
          description = :description,
          start_time = :start_time,
          end_time = :end_time,
          is_public = :is_public
        WHERE ID = :contest_id"
```



6. **Delete contest**

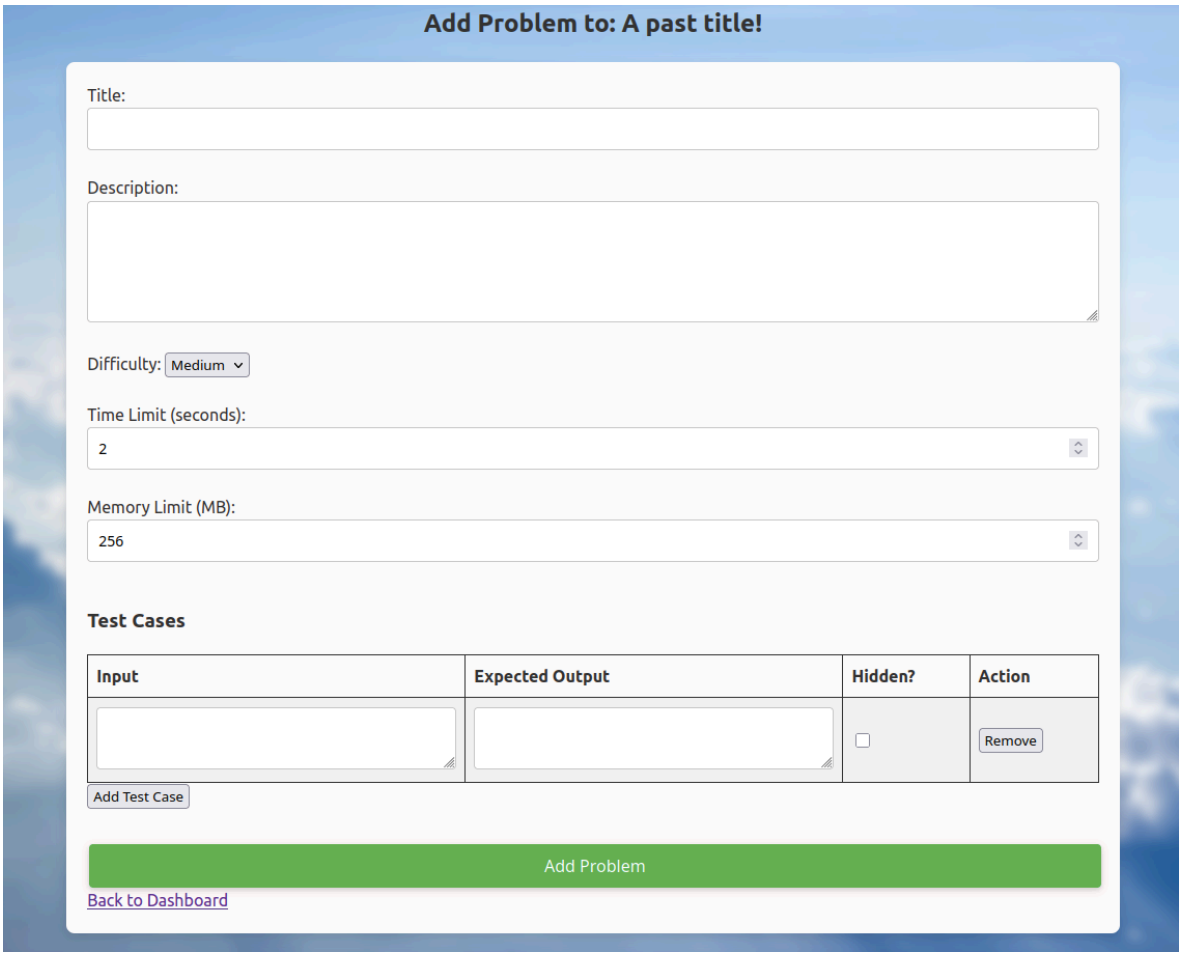
```
DELETE FROM contests WHERE ID = :contest_id
```



5.2.7. Problemsets

1. **Add a new problem**

```
INSERT INTO problems (contest_id, title, description, difficulty, time_limit,
memory_limit)
VALUES (:contest_id, :title, :description, :difficulty, :time_limit, :memory_limit)
```

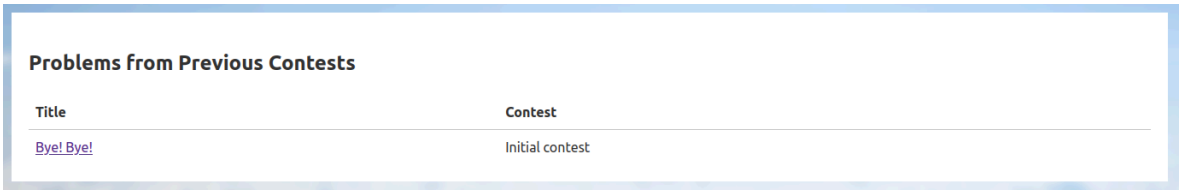


2. **Delete a problem**

```
DELETE FROM problems WHERE ID = :problem_id
```

3. **Get all problems from previous contests**

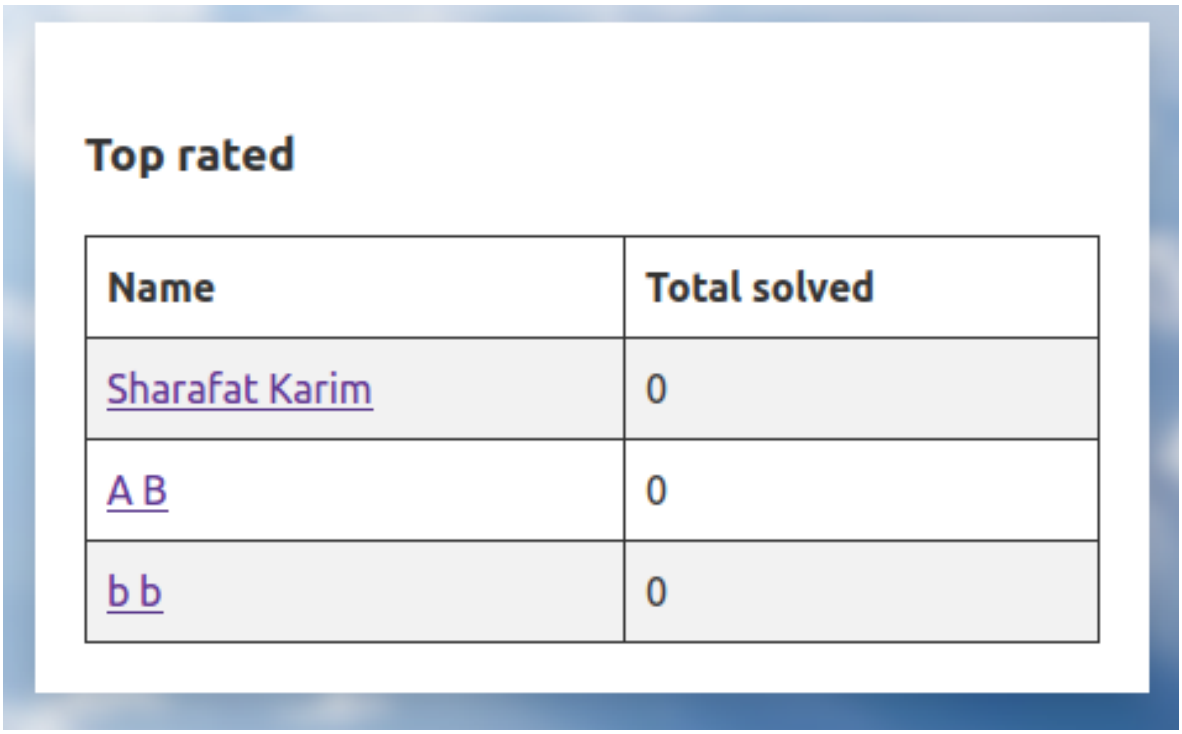
```
  SELECT problems.*, contests.title AS contest_title
  FROM problems
  JOIN contests
  ON problems.contest_id = contests.ID
  WHERE contests.end_time <= NOW()
  ORDER BY contests.end_time DESC, problems.ID ASC;

  # ID, contest_id, title, description, difficulty, time_limit, memory_limit,
created_at, contest_title
'2', '1', 'Bye! Bye!', 'Print the title!', 'easy', '2', '256', '2025-06-13
15:42:11', 'Initial contest'
```



5.2.8. Leaderboard

1. **Get top 5 users based on total_solved**

```
  SELECT *
  FROM top_rated_5

# username, first_name, last_name, total_solved
'sharafat', 'Sharafat', 'Karim', '1'
'a', 'A', 'B', '0'
'b', 'b', 'b', '0'
```



Top rated

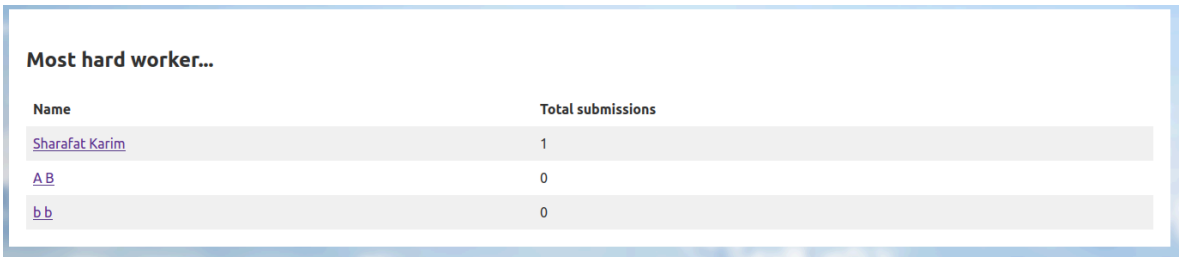
Name	Total solved
Sharafat Karim	0
A B	0
b b	0

2. **Get 50 user's rank based on total_contribution**

```
SELECT first_name, last_name, username, total_contributions
                  FROM users
                  ORDER BY total_contributions DESC
                  LIMIT 50"

# first_name, last_name, username, total_solved
'Sharafat', 'Karim', 'sharafat', '1'
'A', 'B', 'a', '2'
'b', 'b', 'b', '0'
```

3. **Get 50 user's rank based on total_submission**

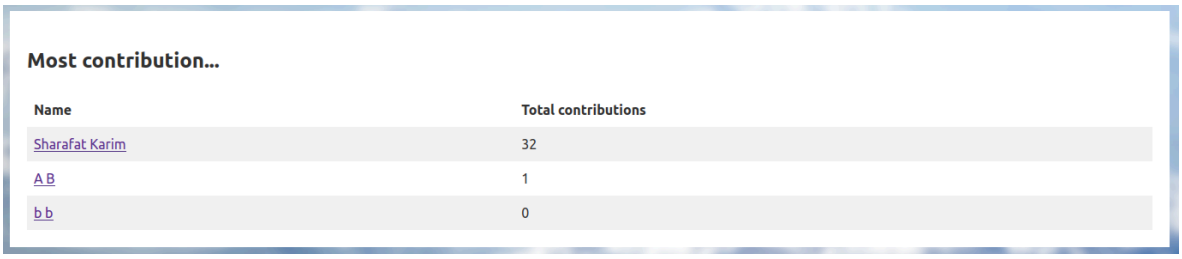
```
SELECT first_name, last_name, username, total_submissions
                  FROM users
                  ORDER BY total_submissions DESC
                  LIMIT 50

# first_name, last_name, username, total_submissions
'Sharafat', 'Karim', 'sharafat', '1'
'A', 'B', 'a', '2'
'b', 'b', 'b', '0'
```



4. **Get 50 user's rank based on total_contribution**

```
SELECT first_name, last_name, username, total_contributions
                FROM users
                ORDER BY total_contributions DESC
                LIMIT 50
```

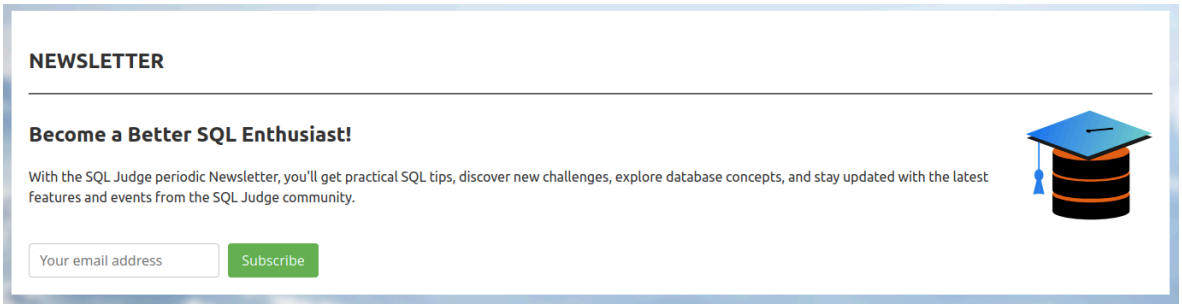
```
# first_name, last_name, username, total_contributions
'Sharafat', 'Karim', 'sharafat', '32'
'A', 'B', 'a', '1'
'b', 'b', 'b', '0'
```



5.2.9. Newsletters

1. **Insert a new newsletter subscription**

```
INSERT INTO newsletters (email) VALUES (:email)
```



6. Limitations

- The platform is currently hosted on a free hosting service, which may have limitations on performance and uptime.
- Currenty the creator of the contest has to manually review the submissions and update the user scores. This can be automated in the future.
- Markdown editor is not implemented yet, so users cannot format their blogs and comments using markdown syntax everywhere.

7. Conclusion

Finally we can conclude that, SQL Judge platform will help mentors and teachers to help spreading the knowledge of SQL and database management. It will also help students to learn and practice SQL queries in a fun and interactive way. The platform is designed to be user-friendly and easy to navigate, making it accessible to users of all skill levels.

8. References

8.1. Documentations

- https://www.w3schools.com/html/ [**W3Schools HTML**]
- https://www.w3schools.com/css/ [**W3Schools CSS**]
- https://www.w3schools.com/js/ [**W3Schools JavaScript**]
- https://www.w3schools.com/sql/ [**W3Schools SQL**]
- https://www.php.net/manual/en/ [**PHP Manual**]
- https://www.w3schools.com/php/default.asp [**W3Schools PHP**]

THE END