Chapter 3: Introduction to SQL

Practice Exercises:

(3.1) Write the following queries in SQL, using the university schema.

(a) Find the titles courses in the Comp. Sci. department that have 3 credits.

(b) Find the ID's of all students who were taught by an instructor named Einstein, make sure there are no duplicates in the result.

(c) Find the highest salary of any instructor.

(d) Find all instructors earning the highest salary.

(e) Find the enrollment, across all sections, in Autumn, 2009.

(f) Find the maximum enrollment, across all sections in Autumn 2009.

(g) Find the sections that had the maximum enrollment in Autumn 2009.

Soln:

(a) select title from course where dept_name = 'Comp. Sci.' and credits = 3

(b) select distinct student.ID from (student join takes using (ID)) join (instructor join teaches using (ID)) using (course_id, sec_id, semester, year) where instructor.name = 'Einstein'

(c) select max (salary) from instructor

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

(e) select course_id, sec_id, count(ID) from
section natural join takes where semester = 'Autumn'
and year = 2009 group by course_id, sec_id.

Another way:

select course_id, sec_id, (select count(ID)
from takes where takes.year = section.year
and takes.semester = section.semester
and takes.course_id = section.course_id
and takes.section_id = section.section_id)
from section where semester = 'Autumn' and
year = 2009.

(f) select max(enrollment) from (select count(ID)
as enrollment from natural join takes
where semester = 'Autumn' and year = 2009
group by course_id, sec_id).

(g) With sec_enrollment as (
select course_id, sec_id, count(ID) as enrollment
from section natural join takes
where semester = 'Autumn' and year = 2009
group by course_id, sec_id)
select course_id, course_id from sec_enrollment
where enrollment = (select max(enrollment)
from sec_enrollment.

(3.2) Suppose you are given a relation grade_points (grade, points), which 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+ points" to 3.3 points, a "B" to 3 points, and so on. The grade points ~~earned~~ 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 above relation, and our university schema, write each of the following query in SQL. You can assume for simplicity that no takes tuple has the null value for grade.

(a) Find the grade-points earned by the student with ID 12345, across all courses taken by the student.

(b) Find the grade-points average (GPA) for the above student, that is, the total grade points divided by the total credits for the associated courses.

(c) Find the ID and the grade point average of every student.

Soln:

(a) (select sum (credits * points)
from (takes natural join courses) natural join grade_points where ID = '12345')

union
(select 0 from student where takes.ID '12345' and not exists (select * from takes where takes.ID = '12345'))

(b) (select sum(credits*points)/sum(credits) as GPA from (takes natural join course) natural join grade_points where ID='12345')
union
(select null as GPA from student where takes.ID = '12345' and not exists (select * from takes where takes.ID = '12345'))

(c) (select ID, sum(credits*points)/sum(credits) as GPA from (takes natural join course) natural join grade_points group by ID)
union
(select ID, null as GPA from student where not exists (select * from takes where takes.ID = student.ID))

(3.3) 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%.

(b) Delete all courses that have never been offered

(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.

(a) update instructor set salary = salary * 1.10
where dept_name = 'comp. sci'

(b) delete from course where course_id not in (select course_id from section)

(c) insert into instructor select ID, name, dept_name, 1000 from student where tot_cred > 100.

(3.4) Consider the insurance database of Figure 3.18 where the primary keys are underlined. Construct the following SQL queries for this relational database.

(a) Find the total number of people who owned cars that were involved in accidents in 1989

(b) Add a new accident to the database; assume any values for required attributes.

(c) Delete the Mazda belonging to "John Smith".

Soln:

(a) select count (distinct name)
from accident, participated, person where
accident.report_number = participated.report_number
and participated.driver_id = person.driver_id
and date between date '1989-00-00' and
date '1989-12-31'

(b) insert into accident values (4007, '2001-09-01', 'Berkeley')

insert into participated
select o.driver_id, c.license, 4007, 3500
from person p, owns o, car c where
p.name = 'Jones' and p.driver = o.driver and
o.license = c.license and c.model = 'Toyota'

(c) delete car where model = 'Mazda' and
license in (select license from person p,
owns o, where p.name = 'John Smith' and
p.driver_id = o.driver_id)

(3.5) Suppose that we have a relation marks and we wish to assign grades to student based on the score as follows: grade F if score < 40, C if score < 60, grade B if 60 ≤ score < 80 and grade A if 80 ≤ score.

(a) Display the grade for each student, based on the marks relation.

(b) Find the number of students with each grade.

Soln:

(a) select ID, case
when score < 40 then 'F'
when score < 60 then 'C'
when score < 80 then 'B'
else 'A'
end
from marks.

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

(3.6) The SQL like operator is case sensitive, but the lower() function on strings can be used to perform case insensitive matching. To show how, write a query that finds departments whose names the string "sci" as a substring. regardless of the case.

Soln: select dept_name from department
where lower (dept_name) like '%sci%'

(3.8) Consider the bank database of Figure 3.19 where the primary keys are underlined. Construct the following SQL queries for the relational database.

(a) Find all customers of the bank who have an account but not a loan.

(b) Find the names of all customers who live on the same street and in the same city as "Smith".

(c) Find the names of all branches with customers who have an account in the bank and who live in 'Harrison'.

soln :

(a) ~~Find~~
(select customer_name from depositor) except
(select customer_name from borrower)

(b) select F.customer_name from
customer F join customer S using (customer_stre
customer_city) where S.customer_name = 'Smith'.

(c) select distinct branch_name from
account natural join depositor natural join
Customer where customer_city = 'Harrison'

(3.9) Consider the employee database of Figure 3.20, where the primary keys are underlined. Give an expression SQL for each of the following queries.

employee (employee_name, street, city)
works (employee_name, company_name, salary)
company (company_name, city)
manages (employee_name, manager_name)

(a) Find the names and cities of residence of all employee who work for First Bank coorporation

(b) Find the names, street address, and cities of residence of all employees who work for First Bank Corporation and earn more than $10,000.

(c) Find all employees in the database, who do not work for First Bank Corporation.

(d) Find all employees in the database who earn more than each employee of small bank Corporation.

(e) Assume that the companies may be located in several cities. Find all companies located in every city in which Small Bank Corporation.

(f) Find the company that has most employees.

(g) Find those companies which employees earn a higher salary, on average, than the ~~others~~ average salary at First Bank Corporation

Soln:

(a) select e.employee_name, city from employee e, works w where w.company_name = 'First Bank corporation' and w.employee.name = e.employee_name.

(b) select * from employee where employee_name in (select employee_name from works where company_name = 'First Bank Corporation' and salary > 1000.

(c) select employee_name from works where company_name != 'First Bank Corporation'

(d) select employee_name from works where salary > all (select salary from works where company_name = 'small Bank Corporation'.

(e) select S.company_name from company S where not exists ((select city from company where company_name = 'Small Bank Corporation') except (select city from company T where S.company-name = T.compory_name)).

(f) select company_name from works group by company_name having count(distinct employee_name) >= all (select count(distinct employee_name) from works group by employee_name.

(g) select company_name from works group by company_name having avg(salary) > (select avg (salary) from works where company_name = 'First Bank Corporation')

(3.10) Consider the relational figure 3.20. Give an expression in SQL for each of the following queries.

(a) Modify the database so that Jones now lives in Newton.

(b) Give all managers of First Bank Corporation a 10 percent raise unless the salary becomes greater than $100,000; in such cases, give only a 3 percent raise.

Soln:

(a) update employee set city = 'Newton' where person_name = 'jones'

(b) update works T set T.salary = T.salary *
(case
when (T.salary * 1.1 > 1000) then 1.03
else 1.1
)
where T.employee_name in (select manager_name from manages) and T.company_name = 'First Bank Corporation'.

Excercies:

(3.11) Write the following queries in sql, using the university schema.

(a) Find the name of all students who have taken at least one Comp. Sci. course; make sure there are no duplicate name in the result.

(b) Find the ID's and name of all students who have not taken any course offering Spring 2009.

(c) For each department, find the maximum salary of instructors in that department for at least one instructor.

(d) Find the lowest, across all departments, of the per-department maximum salary computed by the preceding query.

Soln:

(a) select name from student natural join course where course.dept = 'Comp. Sci'

(b) select id, name from student except select id, name from student natural join takes where year < 2009

(c) select dept, max(salary) from instructor group by dept

(d) select min(maxsalary) from (select dept, max(salary) as maxsalary from instructor group by dept)

(3.12) Write the following queries in SQL, using the university schema.

(a) Create a new course "CS-101", titled "Weekly Seminar" with 0 credits.

(b) Create a section of this course in Autumn, 2009

(c) Enroll every student in the Comp. Sci. department in the above section.

(d) Delete enrollments in the above section where the student's name is Chavez.

(e) Delete the course CS-101. What will happen if you run this delete statement without first deleting offerings of this course

(f) Delete all takes tuples corresponding to any section of any course with the word "database" as a part of the title; ignore case when matching the word with the title.

Soln:

(a) insert into course values ('CS-001', 'Weekly seminar', 'Comp. Sci.', 0)

(b) insert into section values ('CS-001', 1, 'Autumn', 2009, null, null, null)

(c) insert into takes
select id, 'CS-001', 1, 'Autumn', 2009, null from student where dept_name = 'Comp. Sci'

(d) delete from takes where course_id = 'CS-001' and section_id = 1 and year = 2009 and semester = 'Autumn' and id in (select id from student where name = 'Chavez')

(e) delete from takes where course_id = 'CS-001'
delete from section where course_id = 'CS-001'
delete from course where course_id = 'CS-001'

(f) delete from takes where course_id in (select course_id from course where lower(title) like '%database%')

(3.13) Write SQL DDL corresponding to the schema in figure 3.18. Make any reasonable assumptions about data types and be sure to declare primary and foreign keys.

Soln:

(a) create table person (
driver_id varchar(50),
name varchar(50),
address varchar(50),
primary key (driver_id));

(b) create table car (
license varchar(50),
model varchar(50),
year integer,
primary key (license))

(c) create table accident (
report_number integer,
date date,
location varchar(50),
primary key (report_number))

(d) create table owns (
driverid varchar(50),
license varchar(50),
primary key (driver_id, license),
foreign key (driver_id) references person,
foreign key (license) references car)

(e) create table participated (
report_number integer,
license varchar(50),
driver_id varchar(50),
damage_amount integer,

primary key (report_number, license),
foreign key (license) references car,
foreign key (report_number) references accident))

(3.14) Consider the insurance databases of Figure 3.18, where the primary keys are underlined. Construct the following sql queries for this relational databases.

(a) Find the number of accidents in which the cars belonging to 'John Smith' were involved.

(b) Update the damage amount for the car with license number "AABB2000" in the accident with report number "AR2197" to $3000.

Soln:

(a) select count(*) from accident where exists (select * from participated, owns, person where owns.driver_id = person.driver_id, and person.name = 'John Smith' and own.license = participated.license and accident.report_number = participated.report_number)

(b) update participated set damage_amount=3000 where report_number = 'AR2197' and license = 'AABB2000')

(3.15) Consider the bank database of Figure 3.19, where the primary keys are underlined. Construct the following SQL queries for this ~~relato~~ relational database.

branch (branch_name, branch_city, assests)
customer (customer_name, customer_street, customer_city)
loan (loan_number, branch_name, amount)
borrower (customer_name, loan_number)
account (account_number, branch_name, balance),
depositor (customer_name, account_number)

Fig: 3.19

(a) Find all customers who have an account at all the branches located in "Brooklyn".

(b) Find out the total sum of all loan amounts in the bank.

(c) Find the names of all branches that have assests greater than those of at least one branch located in "Brooklyn".

Soln:

(a) with branchcount as
(select count(*) branch where branch_city = 'Brooklyn')
select customer_name from customer c

where branch count = (select count (distinct branc_name) from (customer natural join depositor natural join account natural join branch) as d
Where d.customer_name = e.customer_name)

(b) select sum(amount) from loan.

(c) select branch_name from branch
where assests > some (select assests from branc where branch_city = 'Brooklyn')

(3.16) Consider the employee database of figure 3.20, where the primary keys are underlined.

employee (employee_name, street, city)
works (employee_name, company_name, salary)
company (company_name, city)
manages (employee_name, manager_name)

(a) Find the names of all ~~department~~ employees who work for First Bank Corporation.

(b) Find all employees in the database who live in the same cities as the company for which they work.

(c) Find all employees in the database who live in the same cities and on the same streets as do their managers.

(d) Find all employees who earn more than the average salary of all employees of their company.

(e) Find the company that has the smallest payroll.

Soln:

(a) select employee_name from works where company_name = 'First Bank Corporation'

(b) select e.employee_name from employee e, works, company c
where e.employee_name = w.employee_n and e.city = c.city and w.company_name = c.company_name ;

(c) select P.employee_name from employee P, employee R, manager M
where P.employee_name = M.employee_name and M.manager_name = R.employee_name and P.street = R.street and P.city = R.city.

(d) select employee_name from works
where salary > (select avg (salary) from works S)

(e) select company_name from works group by company_name
having sum (salary) <= all (select sum (salary) from works group by company_name)

(3.17) Consider the relation of figure 3.20.

(a) Give all employees of First Bank Corporation a 10 percent raise.

(b) Give all managers of First Bank Corporation a 10 percent raise.

(c) Delete all tuples in the works relation for employees of Small Bank Corporation.

Soln:

(a) update works set salary = salary * 1.1
where company_name = 'First Bank Corporation'

(b) update works set salary = salary * 1.1
where employee_name in (select manager_name from manages) and company_name = "First Bank Corporation".

(c) delete from works where company_name = 'Small Bank Corporation'.

(3.21) Consider the library database of figure 3.21.

member (memb_no, name, age)
book (isbn, title, author, publisher)
borrowed (memb_no, isbn, date)

Fig: 3.21

(a) Print the names of members who have borrowed any book published by "McGraw-Hill".

(b) Print the names of members who have borrowed all books published by 'McGraw-Hill'

(c) For each publisher, print the names of members who have borrowed more than five books of that publisher.

(d) Print the average number of books borrowed per member. Take into account that if an member does not borrow any books, then that member does not appear in the borrowed relation at all.

Soln:

(a) Select name
from member m, book b, borrowed l
where m.memb_no = l.memb_no
and l.isbn = b.isbn and
b.publisher = 'McGraw Hill"

(b) select distinct m.name
from member m where not exists
(select isbn from book where
publisher = 'McGrawHill')
except
(select isbn from borrowed l
where l.memb_no = m.memb_no))

(c) select publisher, name
from (select publisher, name, count(isbn)
from member m, book b, borrowed l
where m.memb_no = l.memb_no,
and l.isbn = b.isbn group by
publisher, name) as
membpub (publisher, name, count_books)
where count_books > 5.

(d) With memcount as
(select count(*) from member)

select count(*)|memcount from borrowed.

Chapte 04 : Intermediate SQL

Practise Question :

(4.1) Write the following queries in sql :

(a) Display a list of all instructors, showing their ID, name and the number of sections that they were taught. Your query should use outerjoin and should not use scalar queries.

(b) Write the same query above, but using a scalar subquery, without outerjoin.

(c) Display the list of all courses sections offered in Spring, ~~2018~~ 2010 along with the names of the instructors teaching the section. If a section has more than one instructors it should appear as many times in the result as it has instructors.

(d) Display the list of all departments, with the total number of instructors in each department, without using scalar sub queries.

Soln :

(a) select ID, name, count (course_id, section_id, year semester) as number_of_sections from instructor natural left outer join teaches group by ID, name.

(b) select ID, name (select count(*) as 'Number of sections' from teaches where T.id = I.id) from instructor I.

(c) select course_id, section_id, ID,
decode (name, NULL, '—', name)
from (section natural join left outer join teaches)
natural left outer join instructor
where semester = 'Spring' and year = 2010

(d) select dept_name, count (ID)
from department natural join left outer join instructor
group by dept_name

(4.2) Outer join expression can be computed in SQL without using the SQL outer join.

(a) select * from student natural left outer join takes

(b) select * from student natural full outer join takes.

Soln:

(a) select * from student natural join takes
union
select ID, name, dept_name, tot_cred, NULL, NULL, NULL
from student S1 where not exists
(select ID takes T1 where T1.id = S1.id)

(b) (select * from student natural join takes)
union
(select ID, name, dept_name, tot_cred, NULL,
NULL, NULL, NULL, NULL) from student S1
where not exists (select ID from from takes
T1 where T1.id = S1.id)

union
(select ID, NULL, NULL, NULL, course_id, section_id,
semester, year, grade from takes T1
where not exists (select ID from student
where T1.id = S1.id))

(4.5) Show how to define the view student-grades
(ID, GPA) giving the grade-point average of each
student based on the query. Recall that
we used a relation grade_points (grade, points) to
the numeric points associated with a letter grade.

Soln: create view student_grades (ID, GPA) as
select ID, credit_points / decode (credit_sum, 0,
NULL, credit_sum)
from ((select ID, sum(decode (grade, NULL, 0, credits)
as credit_sum, sum (decode (grade, null, 0,
credits * points)) as credit_point
from (takes natural join course natural left
outerjoin grade points group by ID)
union
select ID, NULL from student
where ID not in (select ID from takes))

(4.7) Consider the relational database of Figure 4.11. Give an sql DDL definition of this database. Identify referential-integrity constraints that should hold, and include them in the DDL Definition.

soln:

```
create table employee (
    person_name  char(20),
    street  char(30),
    city  char(30),
    primary key (person_name))

create table works
    ( person_name char(20),
    company_name  char(15),
    salary  integer,
    primary key person_name,
    foreign key person_name references employee)

create table company
    ( company_name  char(15),
    city  char(20),
    primary key (company_name));

create table manages
(person_name  char(20),
 manager_name  char(20)
 primary key (person_name),
 foreign key (person_name) references employee);
```

(4.8) As discussed in section 4.4.7 complex check conditions and assertions subsection 4.4 we expect the constraints 'an instructor cannot teach sections in two different classroom in a semester in the same time slot" to hold.

(a) Write a sql query that returns all combinations that violate the constraints.

(b) Write an SQL assertion enforce this constraints.

Soln:

(a) select ID, name, section_id, semester, year, time_slot_id, count (distinct building, room_number) from instructor natural join teaches natural join section group by (ID, name, semester, year time_slot_id) having count (building, room_number) > 1

(b) create assertion check not exists (select ID, name, section_id, semester, year, time_slot_id, count (distinct building, room_number) from instructor natural join teaches natural join section group by (ID, name, section_id, semester, year, time_slot_id) having count (building, room_number) > 1

Exercises:

(4.12) For the database of figure 4.11, write a query to find those employees with no manager. Write your query using outer join ~~with~~ and then again using no outer join.

Soln:

a) select employee_name
from employee natural left outer join manages.
where manager_name is null

b) select employee_name from employee e
where not exists
(select employee_name from manages m
where e.employee_name = m.employee_name
and m.manager_name is not null)

(4.14) Show how to define a view ~~in~~ tot_credits (year, num_credits), giving the total number of credits taken by students in each year.

Soln:

create view tot_credits (year, tot_credits) as
(select year, sum(credits) ~~from~~
from takes natural join course
group by year)

(4.15) Show how to express the coalesce operation from exercise 4.10 using the case operation.

Soln:

```
select
    case Result
        when (A1 is not null) then A1,
        :
        when (An is not null) then An.
        else null
    end
from A
```

Chapter 06: Formal Relational Query Language

Practise exercises:

(6.1) Write the following queries in relational algebra, using the university schema.

(a) Find the titles of courses in the Comp. Sci. department that have 3 credits.

(b) Find the ID's of all students who were taught by an instructor named Einestine.

(c) Find the highest salary of any instructor.

(d) Find all instructors earning the highest salary.

(e) Find the enrollment of each section that was offered in Autumn, 2009.

(f) Find the maximum enrollment, across all sections, in autumn 2009.

(g) Find the sections that had the maximum enrollment in Autumn, 2009.

Soln:

(a) $\Pi_{title}(\sigma_{dept\_name = 'comp.\ sci' \wedge credits = 3}(course))$

(b) $\Pi_{ID}(\sigma_{ID='Einstein'}(takes \bowtie \rho_{t1}(ID, course\_id, section\_id, semester, year)\ teaches))$

(c) $\mathcal{G}_{max(salary)}(instructor)$

(d) $instructor \bowtie (\mathcal{G}_{max(salary)}\ as\ salary(instructor))$

(e) $course\_id, section\_id\ \mathcal{G}_{count(*)}\ as\ enrollment(\sigma_{year=2009 \wedge semester=Autumn}(takes))$

(f) $t_1 \leftarrow course\_id, section\_id\ \mathcal{G}_{count(*)}\ as\ enrollment(\sigma_{year=2009 \wedge semester=Autumn}(takes))$
$result = \mathcal{G}_{max(enrollment)}\ as\ enrollment(t_1)$

(g) $t_2 \leftarrow \mathcal{G}_{max(enrollment)}\ as\ enrollment(t1)$
$result = t1 \bowtie t_2$

(6.2) Consider the relational database of Figure 6.22, where the primary keys are underlined. Give an expression in the relational algebra to express each of the following queries:

(a) Find the names of all employees who live in the same city and on the same street as do their managers.

(b) Find the names of all employees in this database who do not work for "First Bank Corporation".

(c) Find the names of all employees who earn more than every employee of "Small Bank Corporation".

Soln:

(a) $\Pi_{person\_name}((employee \bowtie manages) \bowtie (manager\_name = employee2.person\_name \wedge employee.street = employee2.street)(\rho_{employee2}(employee)))$

(b) $\Pi_{person\_name}(\sigma_{company\_name \neq First\ Bank\ Corporation}(works))$

(c) $\Pi_{person\_name}(works) - (\Pi_{works.person\_name}(works \bowtie (works.salary \leq works2.salary \wedge works2.company\_name = "Small\ Bank\ Corporation")\ \rho_{works2}(works)))$

(6.3) The natural outer join operations extend the natural join operation so that tuples from the participating relations are not lost in the result of the join.

Describe how the theta-join operation can be extended so that tuples from the left, right, or both relations are not lost from the result of a theta join.

Soln:

(a) The left outer theta join of $r(R)$ and $s(S)$ $(r \overset{\leftarrow}{\bowtie}_\theta s)$ can be defined as $(r \bowtie_\theta s) \cup ((r - \Pi_R(r \bowtie s)) \times (null, null, \ldots null))$.

The tuple of nulls is of size equal to the number of attributes in S.

(b) The right outer theta join of $r(R)$ and $s(S)$ ($r \bowtie_\theta s$) can be defined as $(r \bowtie_\theta s) \cup ((null \ldots null) \times (s - \Pi_s (r \bowtie_\theta s)))$

(6.7) Let $R = (A, B)$ and $S = (A, C)$ and $r(R)$ and $s(S)$ be relations. Write expressions in relational algebra for each of the following queries

(a) $\{ \langle a \rangle \mid \exists b \, (\langle a, b \rangle \in r \wedge b = 17) \}$

(b) $\{ \langle a, b, c \rangle \mid \langle a, b \rangle \in r \wedge \langle a, c \rangle \in s \}$

(c) $\{ \langle a \rangle \mid \exists c \, (\langle a, c \rangle \in s \wedge \exists b_1, b_2 (\langle a, b_1 \rangle \in r \wedge \langle c, b_2 \rangle \in r \wedge b_1 > b_2)) \}$

Soln:

(a) $\Pi_A (\sigma_{B=17}(r))$

(b) $r \bowtie s$

(c) $\Pi_A (s \bowtie (\Pi_{r.A}(\sigma_{r.b > d.b}(r \times \rho_d(r)))))$

(6.8) Consider the relational database of figure 6.22 where the primary keys are underlined. Give an expression in tuple relational calculus for each of the following question.

(a) Find all employees who work directly for "Jones".

(b) Find all cities residence of all employees who work directly for 'Jones'.

(c) Find the name of the manager of the manager of 'Jones'.

Soln:

(a) $\{t \mid \exists m \in manages\ (t[person\_name] = m[person\_name] \wedge m[manager\_name] = 'Jones')\}$

(b) $\{t \mid \exists m \in manages\ \exists s \in employee\ (e[person\_name] = m[person\_name] \wedge m[manager\_name] = 'Jones' \wedge t[city] = e[city])\}$.

(c) $\{t \mid \exists m_1\ manages\ \exists m_2 \in manages\ (m_1[manager\_name] = m_2[person\_name] \wedge m_1[person\_name] = 'Jones' \wedge t[manager\_name] = m_2[manager\_name])\}$.

Exercise:

(6.10) Write the following queries in relational algebra, using the university schema.

(a) Find the names of all students who have taken at least one Comp. Sci course.

(b) Find the IDs and names of all students who have not taken any course Spring, 2009.

(c) For each department, find the maximum salary of instructor in the department.

(d) Find the lowest, across all departments, of the per-department maximum salary computed by the preceding query.

Soln:

(a) Π_{name} (student ⋈ takes ⋈ Π_{course_id} ($\sigma_{dept\_name = 'comp.sci'}$ (course)))

(b) $\Pi^{(student)}_{ID, name} - \Pi_{ID, name}$ ($\sigma_{year < 2009}$ (student ⋈ takes))

(c) $_{dept\_name}\mathcal{G}_{max(salary)}$ (instructor)

(d) $\mathcal{G}_{min(maxsal)}$ ($_{dept\_name}\mathcal{G}_{max(salary)}$ as maxsal (instructor))

(6.11) Consider the relational database 6.22, where the primary keys are underlined. Give an expression in the relational ~~database~~ algebra to express each of the following queries.

(a) employee (<u>person_name</u>, street, city)
works (<u>person_name</u>, company_name, salary)
company (<u>company_name</u>, city)
manages (<u>person_name</u>, manager_name)

Fig: 6.22

(a) Find the names of all employees who work for "First Bank Corporation".

(b) Find the names and cities of residence of all employees who work for 'First Bank Corporation'.

(b) Find the name and cities of residence of all employees who work for 'First Bank Corporation.

(c) Find the names, street address and cities of residence of all employees who work for "First Bank Corporation" and earn more than $10.000.

(d) Find the names of all employees in this database who live in the same city as the company for which they work.

(e) Assume the companies may be located in several cities. Find all companies located in every city in which 'Small Bank Corporation' is located.

Soln:

(a) $\Pi_{person\_name}(\sigma_{company\_name = \text{'First Bank Corporation'}}(works))$

(b) $\Pi_{person\_name, city}(employee \bowtie (\sigma_{company\_name = \text{'First Bank Corporation'}}(works)))$

(d) $\Pi_{person\_name}(employee \bowtie works \bowtie company)$

(c) $\Pi_{person\_name, street, city}(\sigma_{company\_name = \text{'First Bank Corporation'} \wedge salary > 10000}(works \bowtie employee))$.

(e) $\Pi_{company\_name}(company \div (\Pi_{city}(\sigma_{company\_name='Small\ Bank\ Corporation'}(company))))$

(6.12) Using the university example, write relational-algebra queries to find the course sections taught by more than one instructor in the following ways:

(a) Using an aggregate function.

(b) Without using any aggregate function.

Soln:

(a) $\sigma_{instnt > 1}(course\_id, section\_id, year, semester\ \mathcal{G}_{count(*)}\ as\ instnt(teaches))$

(b) $\Pi_{course\_id, section\_id, year, semester}(\sigma_{ID <> ID2}(takes \bowtie \rho_{takes1(ID2, course\_id, section\_id, year, semester)}(takes)))$

(6.13) Consider the relational database of Figure 6.22. Give a relational-algebra expression for each of the following queries:

(a) Find the company with the most employees.

(b) Find the company with smallest payroll.

(c) Find those companies whose employees earn a higher salary, on average at First Bank Corporation.

Soln:

(a) $t_1 \leftarrow {}_{company\_name}\mathcal{G}_{count\text{-}distinct(person\_name)}(works)$

$t_2 \leftarrow \mathcal{G}_{max(max\_employees)}(\rho_{company\text{-}strength(company\_name, num\_employees)}(t_1))$

$\Pi_{company\_name}(\rho_{t3(company\_name, num\_employees)}(t_1) \bowtie \rho_{t4(num\_employees)}(t_2))$

(b) $t_1 \leftarrow {}_{company\_name}\mathcal{G}_{sum(salary)}(works)$

$t_2 \leftarrow \mathcal{G}_{min(payroll)}(\rho_{company\text{-}payroll(company\_name, payroll)}(t_1))$

$\Pi_{company\_name}(\rho_{t3(company\_name, payroll)}(t_1) \bowtie \rho_{t4(Payroll)}(t_2))$

(c) $t_1 \leftarrow {}_{company\_name}\mathcal{G}_{avg(salary)}(works)$

$t_2 \leftarrow \sigma_{company\_name = 'First\ Bank\ Corporation'}(t_1)$

$\Pi_{t3.company\_name}(\rho_{t3(company\_name, avg\_salary)}(t_1) \bowtie_{t3.avg\_salary > first\_bank\_avg\_salary}(\rho_{first\_bank(company\_name, avg\_salary)}(t_2)))$

(6.14) Consider the following relational schema for a library:

member (memb_no, name, dob)

books (isbn, title, authors, publisher)

borrowed (memb_no, isbn, date)

(a) Find the names of members who have borrowed any book published by 'McGraw-Hills'

(b) Find the name of members who have borrowed all books published by 'McGraw-Hills'.

(c) Find the name and membership number of members who have borrowed more than five different books published by McGraw-Hills'

(d) For each publisher, find the name of books borrowed per members who have borrowed more than five books of that publisher.

Soln:

(a) $t1 \leftarrow \Pi_{isbn}(\sigma_{publisher='McGraw-Hills'}(books))$

$\Pi_{name}((member \bowtie borrowed) \bowtie t1))$

(b) $t1 \leftarrow \Pi_{isbn}(\sigma_{publisher='McGraw-Hills'}(books))$

$\Pi_{name,isbn}(member \bowtie borrowed) \div t_1$

(c) $t1 \leftarrow member \bowtie borrowed \bowtie (\sigma_{publishe='McGraw-Hill}(books))$

$\Pi_{name}(\sigma_{countisbn > 5}\, \mathcal{G}_{count\text{-}distinct(isbn)\ as\ countisbn(t1)}))$

(d) $t_1 \leftarrow member \bowtie borrowed \bowtie books$

$\Pi_{publisher,name}(\sigma_{countisbn > 5}((publisher, memb\_no, \mathcal{G}_{count\text{-}distinct\ as\ countisbn}(t1)))$

(6.16) Let R = (A, B) and S = (A, C) and let r(R) and s(S) be relations. Write relational-algebra expressions equivalent to the following domain-relational-calculus:

(a) $\{\langle a \rangle \mid \exists\, b (\langle a, b \rangle \in r \wedge b = 17)\}$

(b) $\{\langle a, b, c \rangle \mid \langle a, b \rangle \in r \wedge \langle a, c \rangle \in s\}$

(c) $\{\langle a \rangle \mid \exists\, b (\langle a, b \rangle \in r) \vee \forall\, c (\exists\, d (\langle d, c \rangle \in s) \Rightarrow \langle a, c \rangle \in s)\}$

(d) $\{\langle a \rangle \mid \exists\, c (\langle a, c \rangle \in s \wedge \exists\, b_1, b_2 (\langle a, b_1 \rangle \in r \wedge \langle c, b_2 \rangle \in r \wedge b_1 > b_2))\}$

Soln:

(a) $\Pi_A (\sigma_{B = 17}(r))$

(b) $r \bowtie s$

(c) $\Pi_A(r) \cup (r \div \sigma_B (\Pi_C (s)))$

(d) $\Pi_{r.A} ((r \bowtie s) \text{ as } r_2 \; r_2.A \wedge r.B \wedge r_2.B \; (r_{r_2}(r)))$

(6.17) Repeat exercise 6.16, writing SQL queries instead of relational-algebra expressions.

Soln:

(a) select a from r where b = 17

(b) select a, b, c from r, s where r.a = s.a

(c) (select a from r) union (select a from s)

(d) select a from r as r1, r as r2, s
where r1.a = s.a and r2.a = s.c and r1.b > r2.b.

(6.18) Let R=(A,B) and S=(A,C) and let r(R) and s(S) be relations. Using the special constants null, write tuple-relational-calculus expressions equivalent to each of the following:

(a) $r \bowtie s$

(b) $r \rightarrow\!\!\!\bowtie s$

(c) $r \leftarrow\!\!\!\bowtie s$

Soln:

(a) $\{t \mid \exists r \in R \; \exists s \in S (r[A] = s[A] \wedge t[A] = r[A] \wedge t[B] = r[B] \wedge t[C] = s[C]) \vee \exists s \in S (\neg \exists r \in R (r[A] = s[A]) \wedge t[A] = s[A] \wedge t[C] = s[C] \wedge t[B] = null)\}$

(b) $\{t \mid \exists r \in R \; \exists s \in S (r[A] = s[A] \wedge t[A] = r[A] \wedge t[B] = r[B] \wedge t[C] = s[C]) \vee \exists r \in R (\neg \exists s \in S (r[A] = s[A]) \wedge t[A] = r[A] \wedge t[B] = r[B] \wedge t[C] = null) \vee \exists s \in S (\neg \exists r \in R (r[A] = s[A]) \wedge t[A] = s[A] \wedge t[C] = s[C] \wedge t[B] = null)\}$

(c) $\{t \mid \exists r \in R \; \exists s \in S (r[A] = s[A] \wedge t[A] = r[A] \wedge t[B] = r[B] \wedge t[C] = s[C]) \vee \exists r \in R (\neg \exists s \in S (r[A] = s[A]) \wedge t[A] = r[A] \wedge t[B] = r[B] \wedge t[C] = null)\}$

(6.19) Give a tuple-relational-calculus expressions to find the maximum value in relational R(A).

Soln: $\{ \langle a \rangle \mid \langle a \rangle \in r \wedge \forall \langle b \rangle \in R\, a >= b \}$