COMP3311 19T3 Database Systems

Exercises 04
SQL Queries and Views

Notation: in the relational schemas below, primary key attributes are shown in **`bold`** font, foreign key attributes are shown in *`italic`* font, and primary key attributes that are also foreign keys are shown in ***`bold italic`*** font.

Note1: all of these solutions have alternative formulations. If you think you have a better solution than the one(s) presented here, let me know.
Note2: a useful strategy, when developing an SQL solution to an information request, is to express intermediate results as views; this has been done in a few solutions here, but you might like to consider reformulating more of them with views, for clarity.

1. Consider the following relational schemas:

```
Suppliers(sid, sname, address)
Parts(pid, pname, colour)
Catalog(sid, pid, cost)
```

Assume that the ids are integers, that `cost` is a real number, that all other attributes are strings, that the ***`supplier`*** field is a foreign key containing a supplier id, and that the ***`part`*** field is a foreign key containing a part id.

Write SQL statements to answer each of the following queries:

a. Find the *names* of suppliers who supply some red part.

Answer:

```
select S.sname
from   Suppliers S, Parts P, Catalog C
where  P.colour='red' and C.pid=P.pid and C.sid=S.sid
```

or

```
select sname
from   Suppliers natural join Catalog natural join Parts
where  P.colour='red'
```

b. Find the *sids* of suppliers who supply some red or green part.

Answer:

```
select C.sid
from   Parts P, Catalog C
where  (P.colour='red' or P.colour='green') and C.pid=P.pid
```

or

```
select sid
from   Catalog C natural join Parts P
where  (P.colour='red' or P.colour='green')
```

c. Find the *sids* of suppliers who supply some red part or whose address is 221 Packer Street.

Answer:

```
select S.sid
from   Suppliers S
where  S.address='221 Packer Street'
       or S.sid in (select C.sid
                    from   Parts P, Catalog C
                    where  P.color='red' and P.pid=C.pid
                   )
```

d. Find the *sids* of suppliers who supply some red part and some green part.

Answer:

```
select C.sid
from   Parts P, Catalog C
where  P.color='red' and P.pid=C.pid
       and exists (select P2.pid
```

```
                   from   Parts P2, Catalog C2
                   where  P2.color='green' and C2.sid=C.sid and P2.pid=C2.pid
                  )
```

or

```
(select C.sid
 from Parts P, Catalog C
 where P.color='red' and P.pid=C.pid
)
intersect
(select C.sid
 from Parts P, Catalog C
 where P.color='green' and P.pid=C.pid
)
```

e. Find the *sids* of suppliers who supply every part.

Answer:

```
select S.sid
from   Suppliers S
where  not exists((select P.pid from Parts P)
                  except
                  (select C.pid from Catalog C where C.sid=S.sid)
                 )
```

or

```
select C.sid
from   Catalog C
where  not exists(select P.pid
                  from   Part P
                  where  not exists(select C1.sid
                                    from   Catalog C1
                                    where  C1.sid=C.sid and C1.pid=P.pid
                                   )
                 )
```

f. Find the *sids* of suppliers who supply every red part.

Answer:

```
select S.sid
from   Suppliers S
where  not exists((select P.pid from Parts P where P.color='red')
                  except
                  (select C.pid from Catalog C where C.sid=S.sid)
                 )
```

or

```
select C.sid
from   Catalog C
where  not exists(select P.pid
                  from   Part P
                  where  P.colour='red' and
                         not exists(select C1.sid
                                    from   Catalog C1
                                    where  C1.sid=C.sid and C1.pid=P.pid
                                   )
                 )
```

g. Find the *sids* of suppliers who supply every red or green part.

Answer:

```
select S.sid
from   Suppliers S
where  not exists((select P.pid from Parts P
                   where P.color='red' or P.color='green')
                  except
                  (select C.pid from Catalog C where C.sid=S.sid)
                 )
```

or

```
select C.sid
from   Catalog C
where  not exists(select P.pid
                  from   Part P
                  where  (P.colour='red' or P.colour='green') and
                         not exists(select C1.sid
                                    from   Catalog C1
                                    where  C1.sid=C.sid and C1.pid=P.pid
                                   )
                 )
```

h. Find the *sids* of suppliers who supply every red part or supply every green part.

Answer:

```
(select S.sid
 from   Suppliers S
 where  not exists((select P.pid from Parts P where P.color='red')
                   except
                   (select C.pid from Catalog C where C.sid=S.sid)
                  )
)
union
(select S.sid
 from   Suppliers S
 where  not exists((select P.pid from Parts P where P.color='green')
                   except
                   (select C.pid from Catalog C where C.sid=S.sid)
                  )
)
```

i. Find pairs of *sids* such that the supplier with the first *sid* charges more for some part than the supplier with the second *sid*.

Answer:

```
select C1.sid, C2.sid
from   Catalog C1, Catalog C2
where  C1.pid = C2.pid and C1.sid != C2.sid and C1.cost > C2.cost
```

j. Find the *pids* of parts that are supplied by at least two different suppliers.

Answer:

```
select C.pid
from   Catalog C
where  exists(select C1.sid
              from   Catalog C1
              where  C1.pid = C.pid and C1.sid != C.sid
             )
```

k. Find the *pids* of the most expensive part(s) supplied by suppliers named "Yosemite Sham".

Answer:

```
select C.pid
from   Catalog C, Suppliers S
where  S.sname='Yosemite Sham' and C.sid=S.sid and
       C.cost >= all(select C2.cost
                     from   Catalog C2, Suppliers S2
                     where  S2.sname='Yosemite Sham' and C2.sid=S2.sid
                    )
```

l. Find the *pids* of parts supplied by every supplier at a price less than 200 dollars (if any supplier either does not supply the part or charges more than 200 dollars for it, the part should not be selected).

Answer:

```
select C.pid
from   Catalog C
```

```
where  C.price < 200.00
group by C.pid
having count(*) = (select count(*) from Suppliers);
```

2. Consider a schema that represents information about a University:

```
Student(id, name, major, stage, age)
Class(name, meetsAt, room, lecturer)
Enrolled(student, class, mark)
Lecturer(id, name, department)
Department(id, name)
```

Using this schema, write SQL queries to answer the following:

a. Find the names of all first-year (stage 1) students who are enrolled in a class taught by Andrew Taylor.

Answer:

```
select S.name
from   Student S, Class C, Enrolled E, Lecturer L
where  S.id = E.student and E.class = C.name
       and C.lecturer = L.id and L.name = 'Andrew Taylor' and S.stage = 1;
```

b. Find the age of the oldest student enrolled in any of John Shepherd's classes.

Answer:

```
select max(age)
from   Student
where  id in (
          select student
          from  Class C, Enrolled E, Lecturer L
          where E.class = C.name and C.lecturer = L.id
                and L.name = 'John Shepherd'
       );
```

This is done in a different style to the first query. It more closely matches the english expression of the query but is most likely rather less efficient.

Here is the join-based version:

```
select max(age)
from   Student S, Class C, Enrolled E, Lecturer L
where  S.id = E.student and E.class = C.name
       and C.lecturer = L.id and L.name = 'John Shepherd';
```

c. Find the names of all classes that have more than 100 students enrolled.

Answer:

```
select   class
from     Enrolled
group by class
having   count(*) > 100;
```

d. Find the names of all students who are enrolled in two classes that meet at the same time.

Answer:

```
select name
from   Student
where  id in (
          select e1.student
          from   Enrolled e1, Enrolled e2, Class c1, Class c2
          where  e1.student = e2.student and e1.class != e2.class
                 and e1.class = c1.name and e2.class = c2.name
                 and c1.meetsAt = c2.meetsAt
       );
```

The subquery gets a list of ids for all students who are enrolled in two classes that meet at the same time. The outer query then uses this list to access the names from the `Student` relation. As before, this expression of the query more closely matches the english version, but is not particularly efficient.

The join-based version:

```
select name
from   Student S, Enrolled e1, Enrolled e2, Class c1, Class c2
where  S.id = e1.student and
       e1.student = e2.student and e1.class != e2.class
       and e1.class = c1.name and e2.class = c2.name
       and c1.meetsAt = c2.meetsAt;
```

e. Find the names of faculty members for whom the combined enrollment of the courses they teach is less than five.

Answer:

```
select name
from   Lecturer
where  5 > (select count(id)
            from   Class, Enrolled
            where  name = class and lecturer = id);
```

Alternative solution:

```
select l.name
from   Lecturer l, Class c, Enrolled e
where  l.id = c.lecturer and c.name = e.class
group by l.name
having count(student) < 5;
```

f. For each stage, print the stage and average age of students.

Answer:

```
select   stage, avg(age)
from     Student
group by stage;
```

g. Find the names of students who are not enrolled in any class.

Answer:

```
select name
from   Student S
where  S.id not in (select E.student from Enrolled E);
```

(Note that it is not necessary to qualify the attribute names here, so we could omit the tuple variables S and E).

Alternative solution:

```
create view StudentClasses(id,name,nclasses) as
select s.id, s.name, count(class)
from   Student s left out join Enrolled e on (s.id=e.student)
group by s.id, s.name;

select name from StudentClasses where nclasses = 0;
```

3. Consider the following relations for keeping track of airline flights information:

```
Flights(flno, from, to, distance, departs, arrives, price)
Aircraft(aid, aname, cruisingRange)
Certified(employee, aircraft)
Employees(eid, ename, salary)
```

Using this schema, write SQL queries to answer the following:

a. Find the names of aircraft such that all pilots certified to operate them earn more than 80,000.

Answer:

```
-- the set of pilots certified for this aircraft is a subset of the set
-- of pilots who earn more than 80K
select aname
from   Aircraft a
```

```
where  ((select employee from Certified where c.aircraft = a.aid)
        intersect
        (select eid from Employee where salary > 80000))
        =
       (select employee from Certified where c.aircraft = a.aid)

-- there are no certified pilots for this aircraft earning less than 80K
select aname
from   Aircraft a
where  not exists(
         select eid
         from   Certified c, Employee e
         where  c.aircraft=a.id and c.employee=e.eid and e.salary<=80000
       )
```

b. For each pilot who is certified for more than 3 aircraft, find the pilot's id and the maximum cruising range of the aircraft that he (or she) is certified for.

Answer:

```
select   C.employee, max(A.cruisingrange)
from     Certified C, Aircraft A
where    C.aircraft = A.aid
group by C.employee
having   count(*) > 3
```

c. Find the names of pilots whose salary is less than the price of the cheapest route from Los Angeles to Honolulu.

Answer:

```
select distinct E.aname
from   Employee E
where  E.salary < (select min(F.price)
                   from   Flights F
                   where  F.from = 'Los Angeles' and F.to = 'Honolulu');
```

d. For all aircraft with cruising range over 1000km, find the manufacturer of the aircraft, and the average salary of all pilots certified for this aircraft.

Answer:

```
select A.aname as name, AVG(E.salary) as AvaSalary
from   Aircraft A, Certified C, Employees E
where  A.aid = C.aircraft and C.employee = E.eid and A.cruisingrange > 1000
group  by A.aname
```

e. Find the names of pilots certified for some "Boeing" aircraft.

Answer:

```
select distinct E.ename
from   Employees E, Certified C, Aircraft A
where  E.eid = C.employee and C.aircraft = A.aid and A.aname = 'Boeing'
```

f. Find the id's of all aircraft that can be used on routes from Los Angeles to Chicago.

Answer:

```
select A.aid
from   Aircraft A
where  A.cruisingrange > (select min(F.distance)
                          from Flights F
                          where F.from = 'Los Angeles' and F.to = 'Chicago')
```

g. Identify the routes that can be flown by every pilot who makes more than $100000.

Answer:

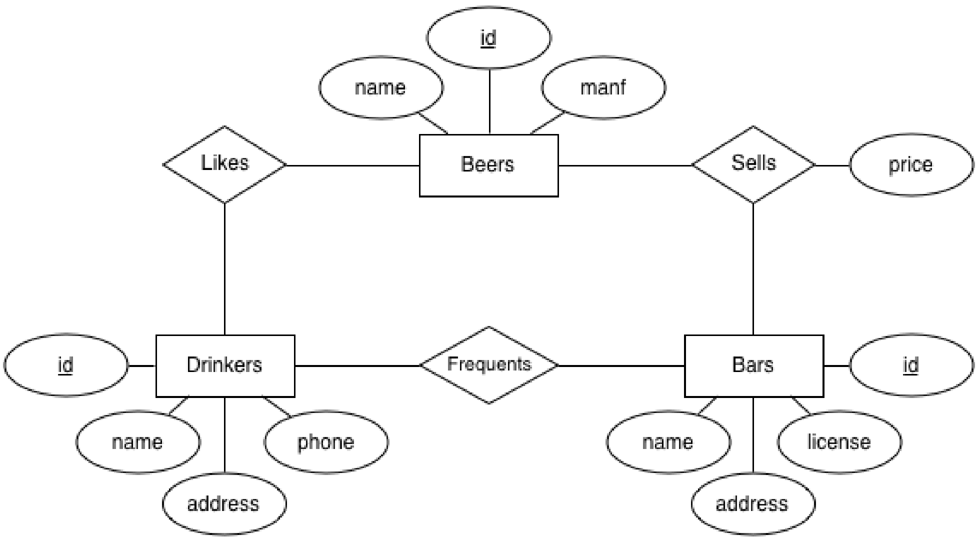
```
select distinct F.from, F.to
from   Flights F
where  not exists (select *
                   from Employees E
                   where E.salary > 100000
                         and
                         not exists (select *
                                     from   Aircraft A, Certified C
                                     where  A.cruisingrange > F.distance
                                            and E.eid = C.employee
                                            and A.eid = C.aircraft
                                    )
                  )
```

h. Print the ename's of pilots who can operate planes with cruising range greater than 3000 miles, but are not certified on any "Boeing" aircraft.

Answer:

```
select distinct E.ename
from   Employees E, Certified C, Aircraft A
where  C.employee = E.eid
       and C.aircraft=A.aid
       and A.cruisingrange > 3000
       and E.eid NOT IN (select C1.eid
                         from   Certified C1, Aircraft A1
                         where  C1.aircraft = A1.aid and A1.aname ='Boeing');
```

4. Consider the following variation on the Beer database schema used in lectures:



and a relational schema derived from it:

```
create table Beers (
    id      integer primary key,
    name    varchar(30),
    manf    varchar(20)
);
create table Bars (
    id      integer primary key,
    name    varchar(30),
    addr    varchar(20),
    license integer
);
create table Drinkers (
    id      integer primary key,
    name    varchar(20),
    addr    varchar(30),
    phone   char(10)
);
create table Sells (
    bar     integer references Bars(id),
    beer    integer references Beers(id),
```

```
    price   real,
    primary key (bar,beer)
);
create table Likes (
    drinker integer references Drinkers(id),
    beer    integer references Beers(id),
    primary key (drinker,beer)
);
create table Frequents (
    drinker integer references Drinkers(id),
    bar     integer references Bars(id),
    primary key (drinker,bar)
);
```

This databse differs from the one in lectures because it uses numeric IDs, rather than names, as primary keys. It also has a different set of data than the database used in lectures, so the results of the queries may be different.

A loadable schema and data is available in this directory.

Give SQL queries to answer the following questions on this database:

a. What beers are made by Toohey's?

Answer:

```
select name from Beers where manf = 'Toohey''s';
```

b. Show beers with headings "Beer", "Brewer".

Answer:

```
select name as "Beer", manf as "Brewer"
from   Beers;
```

c. Find the brewers whose beers John likes.

Answer:

```
select distinct b.manf
from   Beers b
        join Likes L on (b.id = L.beer)
        join Drinkers d on (L.drinker = d.id)
where  d.name = 'John';

-- alternatively

select distinct b.manf
from   Beers b, Likes L, Drinkers d
where  d.name = 'John' and d.id = L.drinker
        and L.beer = b.id;
```

d. Find pairs of beers by the same manufacturer.

Answer:

```
select b1.name, b2.name
from   Beers b1, Beers b2
where  b1.manf = b2.manf and b1.name < b2.name;
```

e. Find beers that are the only one by their brewer.

Answer:

```
select name, manf
from   Beers b
where  not exists (select * from Beers
                   where  manf = b.manf and name != b.name);

-- alternatively

select name, manf
from   Beers
```

```
where  manf in (select manf from Beers
                group by manf having count(name) = 1);
```

f. Find the beers sold at bars where John drinks.

Answer:

```
select distinct b.name
from   Beers b, Sells s, Frequents f, Drinkers d
where  d.name = 'John' and d.id = f.drinker
        and f.bar = s.bar and s.beer = b.id;
```

g. How many different beers are there?

Answer:

```
select count(*) from Beers;
```

h. How many different brewers are there?

Answer:

```
select count(distinct manf) from Beers;
```

i. How many beers does each brewer make?

Answer:

```
select manf,count(*) from Beers group by manf;
```

j. Which brewer makes the most beers?

Answer:

```
create view NumBeers as
   select manf as brewer, count(*) as nbeers
   from   Beers
   group  by manf;

select brewer
from   NumBeers
where  nbeers = (select max(nbeers) from NumBeers);
```

k. Bars where either Gernot or John drink.

Answer:

```
select distinct b.name
from   Bars b join Frequents f on (b.id = f.bar)
        join Drinkers d on (f.drinker=d.id)
where  d.name = 'John' or d.name = 'Gernot';

-- or

create view JohnsBars as
select distinct b.name as bar
from   Bars b join Frequents f on (b.id = f.bar)
        join Drinkers d on (f.drinker=d.id)
where  d.name = 'John';

create view GernotsBars(bar) as
select distinct b.name
from   Bars b join Frequents f on (b.id = f.bar)
        join Drinkers d on (f.drinker=d.id)
where  d.name = 'Gernot';

(select bar from JohnsBars)
union
(select bar from GernotsBars)
```

l. Bars where both Gernot and John drink.

Answer:

```
-- cannot use something like the first solution to the previous question

-- however, a variant of the second solution works just fine

(select bar from JohnsBars)
intersect
(select bar from GernotsBars);

-- alternatively, but less efficiently

select b.name as bar
from   Bars b
        join Frequents f on (b.id = f.bar)
        join Drinkers d on (d.id = f.drinker)
where  d.name = 'John' and
       exists (select *
               from Frequents ff
                       join Drinkers d on (ff.drinker = d.id)
               where ff.bar = f.bar and d.name = 'Gernot');
```

m. Find bars that serve New at the same price as the Coogee Bay Hotel charges for VB.

Answer:

```
create view CBHpriceForVB as
select s.price
from   Sells s
        join Beers b on (s.beer = b.id)
        join Bars r on (s.bar = r.id)
where  b.name = 'Victoria Bitter'
        and r.name = 'Coogee Bay Hotel';

select r.name
from   Sells s
        join Beers b on (s.beer = b.id)
        join Bars r on (s.bar = r.id)
where  b.name = 'New' and
       s.price = (select price from CBHpriceForVB);
```

n. Find the average price of common beers (where "common" means those that are served in more than two hotels).

Answer:

```
select b.name,avg(s.price)
from   Sells s
        join Beers b on (s.beer = b.id)
group  by b.name
having count(s.bar) > 2;
```

o. Which bar sells 'New' cheapest?

Answer:

By now, we're sick of typing the join on Bars-Sells-Beers so we use a view to re-create the table from the original beers database.

```
create view BeerSales(beer,bar,price)
as
select b.name, r.name, s.price
from   Beers b join Sells s on (b.id=s.beer)
        join Bars r on (s.bar=r.id);

select bar
from   BeerSales
where  beer = 'New' and
       price = (select min(price) from BeerSales where beer='New');
```

p. Which bar is the most popular? (i.e. most drinkers)

Answer:

```
create view NumDrinkers(bar,nDrinkers)
as
select b.name, count(*)
from   Frequents f join Bars b on (f.bar=b.id)
group by b.name;

select bar
from   NumDrinkers
where  nDrinkers = (select max(nDrinkers) from NumDrinkers);
```

q. Which bar is the most expensive? (i.e. highest average price)

Answer:

```
create view AveragePrices(bar,avgPrice)
as
select b.name, avg(price)
from   Sells s join Bars b on (s.bar = b.id)
group by b.name;

select bar
from   AveragePrices
where  avgPrice = (select max(avgPrice) from AveragePrices);
```

r. Which beers are sold at all bars?

Answer:

This uses the SQL version of the relational algebra division operator

```
select distinct b.name as beer
from   Sells s join Beers b on (s.beer = b.id)
where  not exists (
        (select id as bar from Bars)
        except
        (select bar from Sells where beer = s.beer)
       );
```

s. What is the price of the cheapest beer at each bar?

Answer:

```
select bar,min(price) from BeerSales group by bar;

-- or, without using the BeerSales view

select b.name as bar, min(price)
from   Sells s join Bars b on (s.bar = b.id)
group  by b.name;
```

t. What is the name of the cheapest beer at each bar?

Answer:

```
create view Bargains as
   select bar,min(price) as cheapest from BeerSales group by bar;

select s.bar,s.beer
from   BeerSales s
where  s.price = (select cheapest from Bargains where bar=s.bar)
order by bar;

-- alternatively

select s.bar,s.beer,s.price
from   BeerSales s, Bargains p
where  s.bar=p.bar and s.price=p.min;
```

u. How many drinkers are there in each suburb?

Answer:

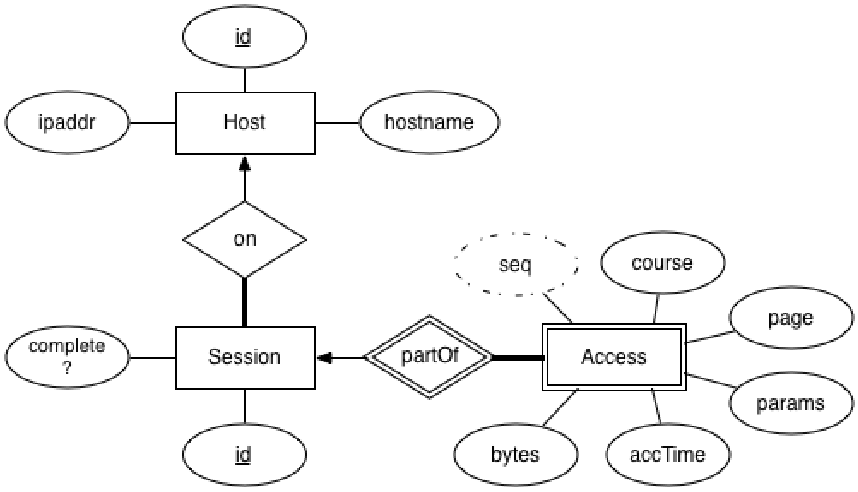
```
select address,count(*) from Drinkers group by address;
```

v. How many bars are there in suburbs where drinkers live?
(Must include suburbs where there are no bars)

Answer:

```
select d.addr, count(b.name)
from   Drinkers d left outer join Bars b using (addr)
group  by d.addr;
```

5. Consider the following database which tracks references to pages in course websites:



and a relational schema derived from it:

```
create table Host (
        id          integer,
        ipaddr      varchar(20) unique,
        hostname    varchar(100),
        primary key (id)
);

create table Session (
        id          int,
        host        integer,
        complete    boolean,
        primary key (id),
        foreign key (host) references Host(id)
);

create table Access (
        session     int,
        seq         int,
        course      int,
        page        varchar(200),
        params      varchar(200),
        accTime     timestamp,
        nbytes      integer,
        primary key (session,seq),
        foreign key (session) references Session(id)
);
```

A looadable schema and data is available in this directory.

Give SQL queries to answer the following questions on this schema:

a. How many sessions are there?

Answer:

```
select count(*) from session;
```

b. How many page accesses are there?

Answer:

```
select count(*) from access;
```

c. What is the average number of accesses per session?

Answer:

```
create view sessLength as
select session, count(*) as length from Access group by session;

select avg(length) from sessLength;
```

d. What are the most common session lengths?

Answer:

```
create view sessLengthFreqs as
select length, count(*) as freq from sessLength group by length;

select length
from   sessLengthFreqs
where  freq = (select max(freq) from sessLengthFreqs);
```

e. What is the longest session?

Answer:

```
select session
from   sessLength
where  length = (select max(length) from sessLength);
```

f. How long was the longest session?

Answer:

```
select max(length) from sessLength;
```

g. What is the most common first page in a session?

Answer:

```
create view firstPages as
select page, count(*) as nAccesses from Access where seq=1 group by page;

select page
from   firstPages
where  nAccesses = (select max(nAccesses) from firstPages);
```

h. What is the most common first page after the 3-page startup?

Answer:

```
-- Assumes that all sessions start with 3-page initial sequence
create view fourthPages as
select page, count(*) as nAccesses from Access where seq=4 group by page;

select page
from   fourthPages
where  nAccesses = (select max(nAccesses) from fourthPages);
```

i. What is the most frequently accessed page?

Answer:

```
create view pageFreq as
select page, count(*) as freq from Access group by page;

select page
```

```
from   PageFreq
where  freq = (select max(freq) from pageFreq);
```

j. What is the most frequently accessed module?

Answer:

```
create view ModuleAccess as
select session, seq, substring(page from '^[^/]+') as module from access;

create view ModuleFreq as
select module, count(*) as freq from Access group by module;

select module
from   ModuleFreq
where  freq = (select max(freq) from ModuleFreq);
```

6. Consider the following relational schema. An employee can work in more than one department; the pct_time field of the Works relation shows the percentage of time that a given employee works in a given department.

```
Emp(eid, emane, age, salary)
Works(eid, did, pct_time)
Dept(did, budget, managerid)
```

Write SQL queries to answer the following:

a. Print the names and ages of each employee who works in both the Hardware and Software department.

Answer:

```
select e.ename, e.age
from   Emp e, Works w1, Works w2
where  e.eid = w1.eid and w1.did = 'Hardware'
       and e.eid = w2.eid and w2.did = 'Software'
```

b. For each department with more than 20 full-time-equivalent employees (i.e., where the part-time and full-time employees add up to at least that many full-time employees), print the department id together with the number of employees that work in that department.

Answer:

```
select w.did, count(w.eid)
from   Works w
group by w.did
having 20 < (select sum(w1.pic_time)
             from   Works w1
             where  w1.did = w.did);
```

c. Print the names of each employee whose salary exceeds the budget of all of the departments that he/she works in.

Answer:

```
select e.ename
from   Emp e
where  e.salary > all (select d.budget
                       from   Dept d, Works w
                       where  e.eid=w.eid and d.did = w.did);
```

d. Find the managerids of managers who only manage departments with budgets over $1,000,000.

Answer:

```
select distinct d.managerid
from   Dept d
where  1000000 < all (select d2.budget
                      from   Dept d2
                      where  d2.managerid = d.managerid)
```

Note the correlated sub-query.

e. Find the names of the manager(s) who manage the department(s) with the largest budget.

Answer:

```
select E.ename
from   Emp E, Dept D
where  E.eid = D.managerid
       and D.budget = (select max(D2.budget) from Dept D2))
```

f. If a manager manages more than one department, he/she is said to "control" the sum of all the budgets for those departments. Find the managerids of managers who control more than 5,000,000.

Answer:

```
select d.managerid
from   Dept d
where  5000000 < (select sum(d2.budget))
                  from Dept d2
                  where d2.managerid = d.managerid)
```

Alternative solution (using a view):

```
create view ManagerBudget(managerid,totBudget) as
        select distinct d.managerid, sum(d.budget)
        from   Dept d
        group by d.managerid;

select managerid
from   ManagerBudgets
where  totBudget > 5000000;
```

g. Find the managerids of managers who "control" the largest amount.

Answer:

```
-- assuming the ManagerBudget view from the previous question

select managerid
from   ManagerBudget
where  totBudget = (select max(totBudget) from ManagerBudget);
```

7. (Based on [E&N 8.18]) Define views for the following in SQL:

a. A view called `DeptManagers` that has department name, manager name and manager salary for each department.

Answer:

```
CREATE VIEW DeptManagers AS
   SELECT d.dname, e.ename, e.salary
   FROM   Dept d, Emp e
   WHERE  d.manager = e.eid;
```

b. A view called `EmpManagers` that has the employee name, manager name for each employee.

Answer:

```
CREATE VIEW EmpManagers AS
   SELECT e.ename as employee, m.ename as manager
   FROM   Emp e, WorksIn w, Dept d, Emp m
   WHERE  e.eid = w.eid AND w.did = d.did
          and d.manager = m.eid AND m.eid != e.eid
   ORDER BY e.ename;
```

c. A view called `DeptEmployees` that has the department name, the employee name, the fraction of time that the employee works for that department, and the amount of their salary that is contributed by that department (assume each department contributes precisely according to the proportion of time that the employee works in the department).

Answer:

```
CREATE VIEW DeptEmployees AS
   SELECT e.ename as employee, d.dname as department,
          w.pct_time as percent, e.salary*w.pct_time as salary
```

```
   FROM   Emp e, WorksIn w, Dept d
   WHERE  e.eid = w.eid AND w.did = d.did
   ORDER BY e.ename;
```

8. (Based on [E&N 8.19]) Consider the following view on the above database (which uses a more simplistic view of salaries than the last part of the previous question):

```
CREATE VIEW DeptSummary(did,nemps,totsal,avgsal) AS
   SELECT did,count(*),sum(salary),avg(salary)
   FROM   Emp e, Works w
   WHERE  w.eid = e.eid
   GROUP BY w.did;
```

State which of the following queries and updates would be allowed on the view. If a query or update would be allowed, show what the corresponding query or update on the tables above would look like:

a.
```
select * from DeptSummary;
```

Answer:

```
 did | nemps | totsal | avgsal
-----+-------+--------+--------
   1 |     2 |  90000 |  45000
   2 |     3 | 135000 |  45000
   3 |     2 | 125000 |  62500
```

b.
```
select did,nemps from DeptSummary;
```

Answer:

```
 did | nemps
-----+-------
   1 |     2
   2 |     3
   3 |     2
```

c.
```
select did,avgsal
from   DeptSummary
where  nemps > (select nemps from DeptSummary where did=1);
```

Answer:

```
 did | avgsal
-----+--------
   2 |  45000
```

d.
```
update DeptSummary set did=4 where did=3;
```

Answer:

In principle, this update would not lose any tuples from the view. However, it involves changing the primary key for the `Departments` table, which could have flow-on effects in referential integrity constraints (e.g., foreign keys in the `Employees` and `WorksIn` tables). Presumably, these kinds of issues could be resolved in an implementation, but it would take some considerable effort, so real DBMS's would probably disallow it. For example, PostgreSQL says:

```
ERROR:  Cannot update a view
        You need an unconditional ON UPDATE DO INSTEAD rule
```

In other words, if you want to do updates on views in PosgreSQL, you need to specify explicitly how you want all of the cascaded updates done.

e.
```
delete from DeptSummary where did>2;
```

Answer:

Similarly to the previous question, this does not create any problems in principle for the view, but practical problems may prevent many DBMS's from implementing it automatically. PostgreSQL says:

```
ERROR:  Cannot delete from a view
        You need an unconditional ON DELETE DO INSTEAD rule
```

In other words, you need to implement the semantics of delete yourself.

9. Suppose you have a view `SeniorEmp` defined as follows:

```
CREATE VIEW SeniorEmp (ename, age, alary) AS
   SELECT E.ename, E.age, E.salary
   FROM   Emp E
   WHERE  E.age > 50
```

Explain how a database system would process the following query:

```
SELECT S.ename
FROM   SeniorEmp S
WHERE  S.salary > 100000
```

Answer:

The database system will transform the query into something like the following:

```
SELECT S.ename
FROM   (SELECT E.ename, E.age, E.salary
        FROM   Emp E
        WHERE  E.age > 50) AS S
WHERE S.salary > 100000
```

or

```
SELECT E.ename
FROM   Emp E
WHERE  E.salary > 100000 AND E.age > 50;
```

The latter would be based on mapping the SQL to relational algebra and then transforming via various algebraic equivalences, e.g.

```
   SeniorEmp = proj[ename,age,salary]( sel[age>50]( Emp ) )

   Query
   = proj[ename]( sel[salary>10000]( SeniorEmp ) )
     (expand definition of SeniorEmp)
   = proj[ename]( sel[salary>10000]( proj[ename,age,salary]( sel[age>50]( Emp ) ) ) )
     (swap selection and projection)
   = proj[ename]( proj[ename,age,salary]( sel[salary>10000]( sel[age>50]( Emp ) ) ) )
     (drop redundant projection)
   = proj[ename]( sel[salary>10000]( sel[age>50]( Emp ) ) )
     (convert nested selection to conjunction)
   = proj[ename]( sel[salary>10000 & age>50]( Emp ) ) )
     (which is the same as the SQL query ...)
   = SELECT ename FROM Emp WHERE salary>100000 AND age>50
```

10. Give an example of an updatable view on the `Employees` relation (i.e. a view such that if we perform an update on the view, the `Employees` relation will be updated appropriately, and the new tuple will appear next time the view is used).

Answer:

The following view can be updated, because each update on the view produces a well-defined update on the base table:

```
CREATE VIEW SeniorEmp (eid, name, age, salary) AS
   SELECT E.eid, E.ename, E.age, E.salary
   FROM   Employees E
   WHERE  E.age>50
```

Consider updates such as:

```
insert into SeniorEmp values (6,'Jack Black',55,125000)
update SeniorEmp set age=age+1 where eid=3
```

On the other hand, consider an update such as

```
insert into SeniorEmp values (7,'Joe Blow',45,35000)
```

It is easy to define the corresponding `INSERT` operation on the base table, but when the tuple is added, it will not appear in the view (because Joe Blow is not older than 50). The SQL standard allows you to specify that you want this kind of thing to be checked for and have such updates rejected by adding a `WTIH CHECK OPTION` clause to the view definition.

11. Give an example of a view on the `Employees` relation that is not updatable. Explain why your example presents the update problem that it does.

Answer:

This view cannot be updated because any view update does not lead to a single well-defined update on the base table:

```
CREATE VIEW AveSalaryByAge (age, avgSalary) AS
   SELECT E.age, AVG(E.salary)
   FROM   Employees E
   GROUP BY  E.age
```

For example, an update such as

```
UPDATE AveSalaryByAge
SET    avgSalary=60000
WHERE  age=30
```

could be achieved in many ways (e.g., increasing employee A's salary and decreasing employee B's salary, or increasing A's salary and decreasing B's salary, etc.)

Any query that does not satisfy the following conditions is not updatable:

- the `FROM` clause contains only a single table
- neither `GROUP BY` nor `HAVING` clauses are used
- the `DISTINCT` qualifier is not used
- the `WHERE` clause does not contain any sub-selects
- all result attributes are derived from single attribute values
  (i.e. no aggregates or computed values)

This is different to the set of conditions given in lecture notes. Other alternative definitions exist, depending on the author's interpretation of "updatable view". As noted above, many RDBMS's avoid the issue and don't allow any views to be updated, or require the user to specify explicitly how the updates on the base tables should be done.