CSE 414 14wi Final Exam Sample Solution

Question 1. (21 points, 7 each) SQL. We have a small database with information about people and events.

Person(pid, name)
Event(eid, name, start_time, end_time)
Invited(pid, eid, going)

Where:

- *Person* contains information about all people in the database. Each person has a unique integer id (*pid*), which is the key.
- *Event* is a list of all events. Each event has a unique integer event id (*eid*), which is the key. The start and end time of each event is given as an integer, which represents the number of seconds that have elapsed since 00:00:00 on January 1, 1970 (i.e., times are given as the number of seconds from a specific time in the past.) You may assume that the *start_time* of each event is always less than the event *end_time*.
- Each tuple in *Invited* indicates that the Person identified by *pid* has been invited to the Event identified by *eid*. The attribute *going* has the value 1 if the person will attend the event and is 0 if not. Attributes *pid* and *eid* are foreign keys referencing *Person* and *Event* respectively.

Write SQL queries that return the information described below.

Grading note: in many cases there are other possible queries that produce the requested answers. Those also received full credit if they were correct.

(a) (7 points) Write a SQL query that returns the number of people who are invited to and are going to attend the event "Film Festival".

```
SELECT COUNT(*)
FROM Invited i, Event e
WHERE i.eid = e.eid AND e.name = 'Film Festival' AND i.going = 1;
```

(b) (7 points) Write a SQL query that finds the event or events that the most people have been invited to attend. The query should return the event id(s) (eid) and event name(s) (name) of the event(s) that have the most persons invited, regardless of whether those people plan to attend.

```
SELECT e.eid, e.name
FROM Event e, Invited i
WHERE e.eid = i.eid
GROUP BY e.eid, e.name
HAVING COUNT(i.pid) >= ALL (SELECT COUNT(i2.pid)
                            FROM Invited i2, Event e2
                            WHERE i2.eid = e2.eid
                            GROUP BY e2.eid);
```

Person(pid, name)
Event(eid, name, start_time, end_time)
Invited(pid, eid, going)

Question 1. (cont) (c) (7 points) Write a SQL query that returns the ids (pid) and names of all persons who have been invited to two distinct events that overlap in time. Two events overlap if one starts while the other is in progress. If the previous event ends at exactly the same time that a later one starts, the two events do not overlap. The query result should include each person (pid and name) only once, even if they have been invited to more than one overlapping event. Persons should be included in the result regardless of whether or not they plan to attend either or both events.

```
SELECT DISTINCT p.pid, p.name
FROM Person p, Invited i1, Invited i2, Event e1, Event e2
WHERE p.pid = i1.pid AND p.pid = i2.pid
      AND i1.eid = e1.eid AND i2.eid = e2.eid AND e1.eid <> e2.eid          -- distinct events
      AND e1.start_time <= e2.start_time AND e2.start_time < e1.end_time;
```

Question 2. (9 points, 3 each) Relational algebra. Suppose we have two relations A and B. Relation A has *na* tuples and relation B has *nb* tuples. For each of the following relational algebra expressions, give the minimum and maximum number of tuples that can appear in the result.

(You do not need to show your work, but it might be helpful in case we need to assign partial credit to a not-quite-right answer.)

(a) $A \cap B$

max = min(*na*, *nb*)
min = 0

(b) $\sigma_c(A) - B$ (where *c* is some logical (Boolean) condition)

max = *na*
min = 0

(c) $A \ltimes B$ (recall that $\ltimes$ is the left-outer-join operator)

max = *na* * *nb*
min = *na*

Question 3. (12 points, 4 each) XML. Suppose we have the following XML document describing a music catalog. The information is organized by label and lists for each label the artists and albums that have been released on that label. A single artist may have released albums on more than one label, so an artist name may appear under multiple label listings (A DTD for this data is given in the next problem. Feel free to reference that if it is useful. You also can remove this page for reference if you wish.)

```
<Catalog>
    <Label> <Name> Swan Song </Name>
        <Artist>
            <Name> Led Zeppelin </Name>
            <Album>
                <Title> Physical Graffiti </Title>
                <Song> Custard Pie </Song>
                <Song> Kashmir </Song>
                <Price> 19.99 </Price>
                <Year> 1975 </Price>
            </Album>
        </Artist>
    </Label >
    <Label>  <Name> Albert </Name>
        <Artist>
            <Name> AC/DC </Name>
            <Album>
                <Title> T.N.T. </Title>
                <Song> Live Wire </Song>
                <Song> T.N.T. </Song>
                <Song> High Voltage </Song>
                <Price> 19.99 </Price>
                <Year> 1975 </Year>
            </Album>
            <Album>
                <Title> Back in Black </Title>
                <Song> Shoot to Thrill </Song>
                <Song> Back in Black </Song>
                <Song> Rock and Roll Ain't Noise Pollution </Song>
                <Price> 25.99 </Price>
                <Year> 1980 </Year>
            </Album>
        </Artist>
    </Label>
    <Label>  <Name> RCA </Name>
        <Artist>
            <Name> Justin Timberlake </Name>
            <Album>
                <Title> The 20/20 Experience </Title>
                <Song> Suit & Tie </Song>
                <Song> Mirrors </Song>
                <Price> 12.99 </Price>
                <Year> 2013 </Year>
            </Album>
        </Artist>
    </Label>
</Catalog>
```

(continued on next page)

Question 3. (cont.) Give the results of the following XPath expressions when they are applied to the XML document on the previous page.

(a) //Price

<Price> 19.99 </Price>
<Price> 19.99 </Price>
<Price> 25.99 </Price>
<Price> 12.99 </Price>

(b) Catalog/Label//Album[//Year/text() >= 1980]/Title

<Title> Back in Black </Title>
<Title> The 20/20 Experience </Title>

(c) Catalog/Label[count(//Album/Song) > 2]/Name

<Name> Albert </Name>

Question 4. (12 points, 6 each) For reference, the DTD of the previous question's XML data is:

```
< !DOCTYPE Catalog [
<!ELEMENT Catalog (Label+)>
<!ELEMENT Label (Name, Artist+)>
<!ELEMENT Artist (Name, Album+)>
<!ELEMENT Album (Title, Song+, Price, Year)>
<!ELEMENT Name(#PCDATA)>
<!ELEMENT Title(#PCDATA)>
<!ELEMENT Song(#PCDATA)>
<!ELEMENT Price(#PCDATA)>
<!ELEMENT Year(#PCDATA)>
]>
```

Below, give XQuery expressions that return the specified results. The queries should work on any valid XML document described by the above DTD, not just the sample data given in the previous problem. The results should be well-formed XML. If needed, you can assume the data is in a file named "catalog.xml".

Grading note: As with the SQL queries, there are other answers that produce the requested results. As long as they worked properly, they received full credit.

(a) Return a list giving the names of every artist included in the catalog and, for each artist, the titles of all of the albums they have released on any label.

```
<result>
for $artist in distinct-values(doc("catalog.xml")//Artist/Name)
return <Artist> {$artist/text()}{
for $album in doc("catalog.xml")//Label/Artist[/Name = $artist]//Title
return <Album> {$album/text()} </Album>
} </Artist>
</result>
```

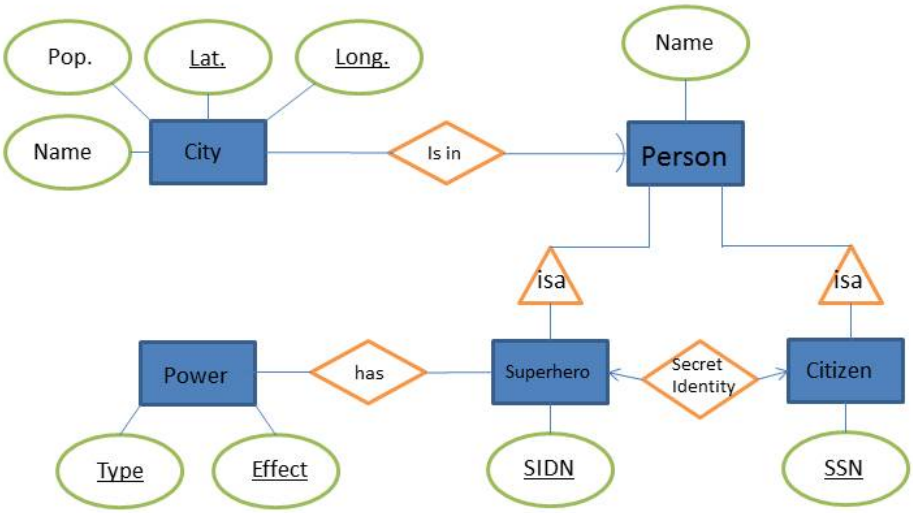
(b) Return a list showing for each year the names of artists who released an album during that year.

```
<result>
for $year in distinct-values(doc("catalog.xml")//Year)
return <Year> {$year/text()} {
        for $artist in distinct-values(doc("catalog.xml")//Artist[//Year = $year])
        return <Artist> {$artist/Name/text()} </Artist>
} </Year>
</result>
```

Question 5. (20 points) Database design. We would like to design a database to describe the members of the Marvel/DC comic book and movie universe. This universe has the following properties that we want to model.

- A citizen has a name and unique social security number (ssn)
- A superhero has a name and a unique superhero id number (sidn)
- A superhero power has a type and an effect, and that combination is unique
- A city has a name, population, latitude and longitude
- A superhero may have a secret identity as a citizen
- A superhero may have one or more powers
- Citizens and superheroes reside within a city (i.e. Gotham, Metropolis)

(a) (10 points) Draw an E/R diagram that captures this information.



There are obviously several possible variations on this diagram and as long as your solution was a reasonable design for the given constraints it received credit. For instance a solution could use Name as the key for the City entities instead of latitude/longitude. Or it might be reasonable to restrict a Superhero to having a single Power, or allow a single Citizen to be the Secret Identity of more than one Superhero.

(continued next page)

Question 5. (cont.) (b) (10 points) Give a series of SQL Create Table statements to create tables to hold the information in the E/R diagram from part (a) of this question. Your SQL statements should identify the key or keys of each table and include any appropriate constraints, including foreign key constraints.

```
Create table City(name varchar(50), population INTEGER, latitude INTEGER, longitude INTEGER,
        primary key(latitude, longitude)
);

Create table Person(name varchar(50), latitude INTEGER, longitude INTEGER, foreign key
        (latitude, longitude) references City(latitude, longitude)
);

Create table Superhero(SIDN INTEGER primary key, foreign key (latitude, longitude) references
        Person(latitude, longitude)
);

Create table Citizen(SSN INTEGER primary key, foreign key (latitude, longitude) references
        Person(latitude, longitude)

Create table Power(type varchar(20), effect varchar(50), SIDN INTEGER references
        Superhero(SIDN), primary key(type, effect)
);

Create table secretIdentity(
SIDN INTEGER references Superhero(SIDN), SSN INTEGER references Citizen(SSN), primary
        key(SIDN, SSN)
);
```

Grading note: We allowed any reasonable definitions that matched the E/R diagram given as an answer to the previous part of the question.

Question 6. (15 points) Database design. We have a database with information about university courses. Unfortunately, it was designed by Mr. Frumble, and it has only one table with this schema:

Course(CourseID, Course_name, Instructor, Instructor_email, Max_students, Room)

These functional dependencies exist between attributes in this table:

CourseID -> Course_name
CourseID -> Instructor
CourseID -> Room
Instructor -> Instructor_email
Room -> Max_students

Is this schema in BCNF? If so, give a justification for your answer and identify the key(s) in the schema. If not, identify the bad functional dependencies and decompose the schema into two or more tables that are in BCNF. Identify the key(s) of each table.

No, it is not in BCNF. One bad dependency is Instructor -> Instructor_email because Instructor+ = {Instructor, Instructor_email}, which does not include all of the attributes. The other bad dependency is Room -> Max_students because Room+ = {Room, Max_students}, which also is not the entire set of attributes.

First decomposition: break the original table into:

I(Instructor, Instructor_email) and
C2(CourseID, Course_name, Instructor, Max_students, Room)

Instructor is the key of the I relation.

C2 still has the bad dependency Room -> Max_students, so we break that into two tables:

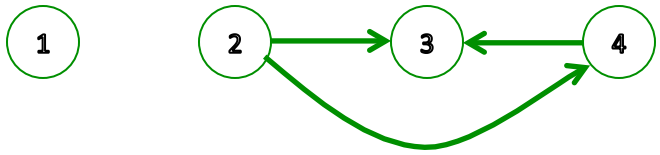
R(Room, Max_students)
C(CourseID, Course_name, Instructor, Room)

The three tables I, R, and C are in BCNF because they have no bad dependencies.

Another solution would be to use the Room -> Max_students dependency for the first decomposition, then break apart the resulting tables using the Instructor -> Instructor_email one.

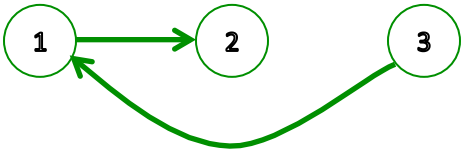
Question 7. (18 points, 6 each) Serializibility. For each of the following schedules, draw the precedence (conflict) graph and decide if the schedule is conflict-serializable. If the schedule is conflict-serializable, give an equivalent serial schedule by listing the transactions in an order they could occur (you do not need to write down all of the read-write operations, just give the order of the transactions). If the schedule is not conflict-serializable, explain why not.

(a) R1(A) W4(C) R3(C) W2(B) R4(B) R3(B)



Yes this is serializable and one possible order is T1, T2, T4, T3. T1 can happen anywhere, but the ordering T2, T4, T3 is the only possible ordering of those three transactions.

(b) R3(B) R1(A) R1(B) W1(C) R2(A) R1(D) R2(C) W1(B)



This is also serializable. The only possible order is T3, T1, T2.

(c) R1(A) R2(A) W2(A) R3(B) R2(B) W3(B) W2(B) W1(A)

Not serializable. There is a cycle between T1 and T2 on A, and a cycle between T2 and T3 on B.

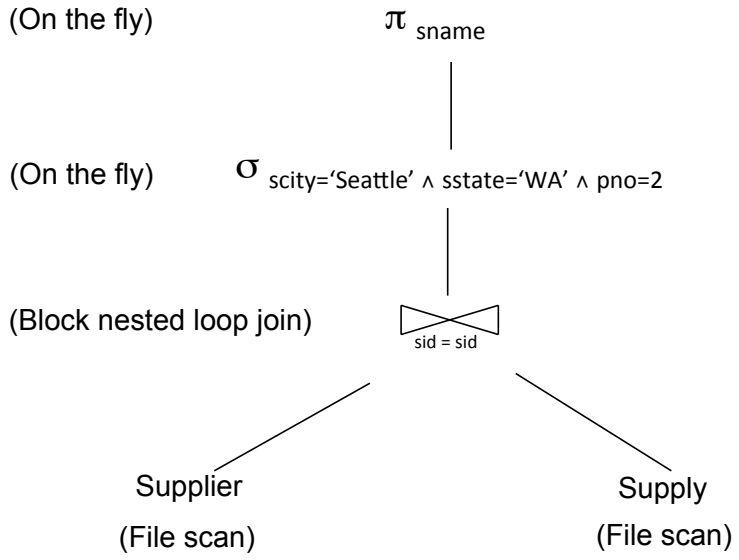
Question 8. (16 points) Query estimation. One of our favorite queries involves tables of suppliers and parts. The two tables have the following schemas:

Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)

We assume that these tables have the following characteristics:

T(Supplier) = 1000	T(Supply) = 10000
B(Supplier) = 100	B(Supply) = 100
V(Supplier, scity) = 20	V(Supply, pno) = 2500
V(Supplier, state) = 10	

Now, consider the following query, where the physical operators to be used are shown in parentheses.

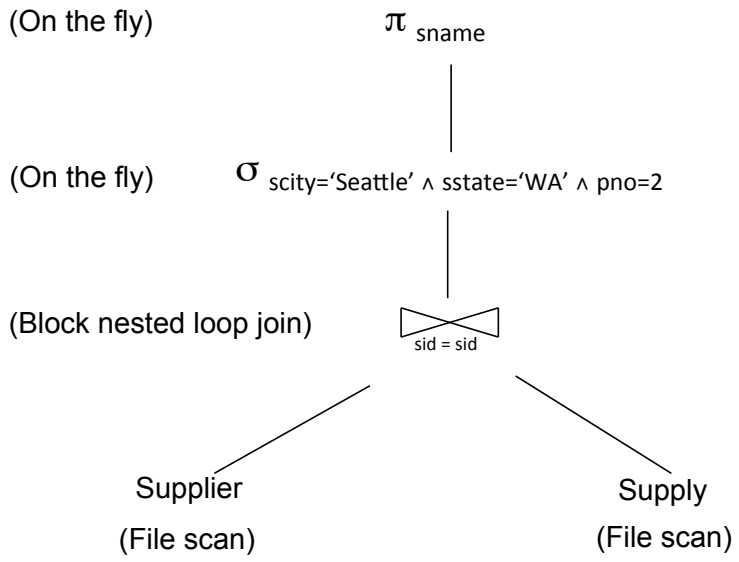


(a) (4 points) Translate this query into SQL.

SELECT sname
FROM Supplier x, Supply y
WHERE x.sid = y.sid AND scity = 'Seattle' AND sstate = 'WA' AND pno = 2

Question 8 (cont.) (b) (6 points) What is the expected cost of this physical query? Remember that the cost of a query is measured in terms of the expected number of I/O operations, not including writing the result. Also, you should assume that there is enough main memory to hold all necessary data, if that is a consideration.

Query diagram and relation information repeated for convenience:



Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)

T(Supplier) = 1000
B(Supplier) = 100
V(Supplier, scity) = 20
V(Supplier, state) = 10

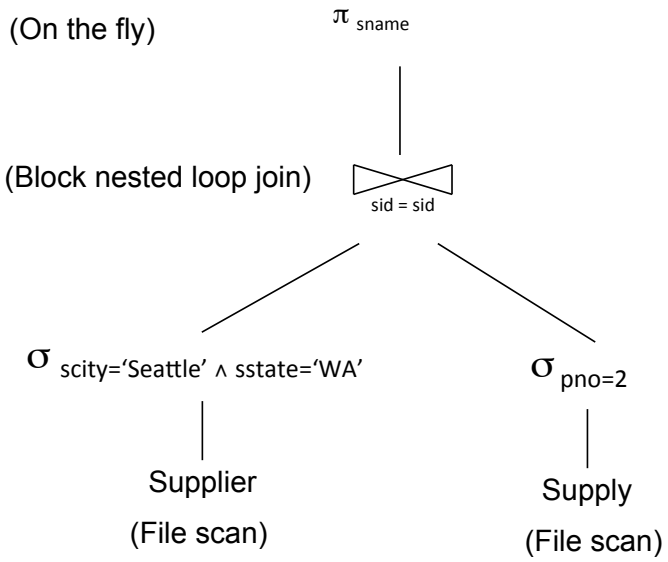
T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500

Write your cost estimate below. It would help to show enough of your work so we can follow it, in case it is necessary to award partial credit.

The cost of the block nested loop join is B(Supplier) + B(Supplier)*B(Supply) = 100 + 100*100 = 10100. There is no additional cost for the select and project operations since these don't involve additional disk I/O operations.

(continued next page)

Question 8. (cont) (c) (6 points) Now suppose we use this different plan for the same SQL query.



Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)

T(Supplier) = 1000
B(Supplier) = 100
V(Supplier, scity) = 20
V(Supplier, state) = 10

T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500

What is the cost estimate for this different plan? (Again, it would help to show enough of your work so we can follow it, but be sure the final answer is clearly indicated.)

The cost estimate for scanning the two files is B(Supplier) + B(Supply) = 200.

The project operation on Supply will yield an estimated 10000/2500 = 4 tuples, which is far less than one block. The project operation on Supplier involves cities, which individually yields an estimated 1000/20 = 50 tuples and states, which is estimated to yield 1000/10 = 100 tuples. In any case, the project operation on Supplier will yield at most 5 blocks worth of data to be streamed to the join operation to be joined with the block of data from Supply. All of that can be streamed in memory with no additional disk operations, so the total cost estimate is 200.

Question 9. (12 points) Map-Reduce. One of the original applications of the map-reduce framework was to compute the *page rank* of each page on the web. The basic idea behind page rank is that pages are more likely to contain useful information if many other pages on the web contain links to them.

For this problem we'd like to compute a simplified version of page rank by counting the number of other pages that link to each page in our data. The input is a large collection of key-value data pairs (pageURL, linkURL) where pageURL is the key of the input data file and linkURL is the associated value. A (pageURL, linkURL) pair indicates that the page identified by pageURL contains a link to the page identified by linkURL. We assume that the data contains no duplicates, i.e., there is only one key-value pair for links from one page to another even if the page identified by pageURL contains multiple links to linkURL.

Describe a sequence of one or more map-reduce jobs (not Pig programs) that will report the number of pages that link to each page that appears as a linkURL in the input data. The output should be a collection of key-value pairs (linkURL, count), where count is the number of pages that contain a link to linkURL.

You need to clearly describe the (key, value) pairs that are input to and output from each map and reduce phase of the map-reduce job(s) needed. Be sure to clearly describe the input and output (key, value) pairs from each map and reduce stage, and explain (concisely) how the output of each map or reduce stage is computed from its input. If it takes more than one map-reduce job to compute the final result you should show how the output of each job is used as input to subsequent ones.

This can be done in a single map-reduce pass since we are basically counting the number of times each linkURL appears in the second part of an input tuple.

Map: input: (pageURL, linkURL) tuples
output: (linkURL, 1) for each linkURL read in an input tuple

Reduce: input: (linkURL, set of {1,1,1,1…} with a 1 for each copy of linkURL found in the map phase)
output: (linkURL, n) where n is the number of 1's associated with linkURL in the input

To cut down on the communication overhead, the each worker in the map phase could store the linkURLs it discovers in a hash table and count how many times each linkURL is seen, then emit a single (linkURL, c) tuple for each linkURL, where c is the number of occurrences discovered by that map worker. The reduce phase would then have as input (linkURL, {c1, c2, …, cn}) where the ci numbers are the individual map counts. The output would be (linkURL, n) as before except that n would be the sum of c1+c2+…+cn.