CSE 344 14au Final Exam Sample Solution

Reference Information - This information may be useful during the exam. Feel free to use it or not as you wish. You can remove this page from the exam if that is convenient.

Reference for SQL Syntax

Outer Joins

```
-- left outer join with two selections:
select *
from R left outer join S on R.x=55 and R.y=S.z and S.u=99
```

The UNION Operation

```
select R.k from R union select S.k from S
```

The CASE Statement

```
select R.name, (case when R.rating=1 then 'like it'
                when R.rating=0 then 'do not like it'
                when R.rating is null then 'do not know'
                else 'unknown' end)
                as a_rating
from R;
```

The WITH Statement

Note: with is not supported in sqlite, but it is supported SQL Server and in postgres.

```
with T as (select * from R where R.K>10)
select * from T where T.K<20
```

Relational Algebra

Name	Symbol
Selection	σ
Projection	π
Join	⋈
Group By	γ
Set Difference	–
Duplicate Elimination	δ

XQuery example (from lecture slides) (a reminder of XQuery syntax)

```
FOR $b in doc("bib.xml")/bib
LET $a:=avg($b/book/price/text())
FOR $x in $b/book
WHERE $x/price/text() > $a
RETURN $x
```

CSE 344 14au Final Exam Sample Solution

The holidays are coming and that means that Santa Claus is preparing to distribute toys to boys and girls of all ages around the world. Santa is joining the modern age and wants to use databases to keep track of his operation, but needs some help.

Several questions in this exam involve the following database schema, used to keep track of toys, clients, and requests.

Toys (id, name, color, min_age, weight, number_available)
Client (id, name, age, address, city)
Request (client_id, toy_id)

The underlined attributes are keys for each relation. The tables contain the following information:

- *Toys* records information about all of the toys in Santa's workshops. Each Toy has a unique integer *id*. Attributes *name* and *color* are strings that describe the toy; *min_age, weight,* and *number_available* are integers giving the minimum age a child should be to use the toy, the weight of the toy in pounds, and how many are currently available in Santa's workshops.

- *Client* stores information about each of the boys and girls in Santa's database. Each client has a unique integer *id*, strings with the client's *name*, *address*, and *city*, and an integer *age*. (To simplify things for the exam, we will ignore issues about different cities that might have the same name and assume that all cities have a unique name.)

- *Request* has a pair of integers indicating that a particular client has requested a particular toy.

You should assume that all entries in all of the tables are not null.

Answer the questions about this database and other topics on the following pages. You may remove this page from the exam for reference if that is convenient.

CSE 344 14au Final Exam Sample Solution

Question 1. (20 points) SQL queries. Write SQL queries to retrieve the requested information from the database tables described on the previous page. The queries you write must be proper SQL that would be accepted by SQL Server or any other SQL implementation. You should not use incorrect SQL, even if sqlite might produce some sort of answer from the buggy SQL.

(a) (10 points) List all of the requests where *age* of the client is less than the *min_age* listed for the Toy. The answer should include the client id, client name, and the toy id. The results should be sorted by client name.

```
SELECT c.id, c.name, t.id
FROM Request r, Client c, Toys t
WHERE c.id = r.client_id AND t.id = r.toy_id AND c.age < t.min_age
ORDER BY c.name;
```

(b) (10 points) Give the id and name of the "most popular" toy. The most popular toy is the one that has more requests than all others. If there is a tie so there is more than one toy with the maximum number of requests, give the id's and names of all of these toys. They do not need to be in any particular order.

```
SELECT t.id, t.name
FROM Toys t, Request r
WHERE t.id = r.toy_id
GROUP BY t.id, t.name
HAVING count(*) = (SELECT max(cnt)
                   FROM (SELECT count(*) as cnt
                         FROM Request r
                         GROUP BY r.toy_id));
```

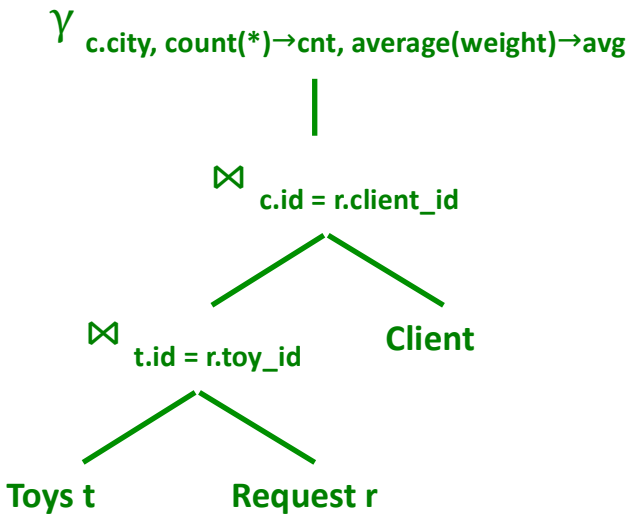
Notes: Both t.id and t.name have to appear in the GROUP BY clause. If only t.id appears here then t.name cannot appear following SELECT.

We did allow answers that used TOP to find the "most popular" toy frequency. Although incredibly useful, TOP is not portable, although most of the major SQL implementations provide something similar.

Question 2. (20 points) Relational algebra. Consider the following query on the tables defined earlier:

```
SELECT c.city, count(*), average(weight)
FROM Toys t, Request r, Client c
WHERE c.id = r.client_id AND t.id = r.toy_id
GROUP BY c.city
```

(a) (10 points) Draw a relational algebra tree giving a relational algebra query that is equivalent to the SQL query shown above. You should assume that this query will be executed on a single machine.

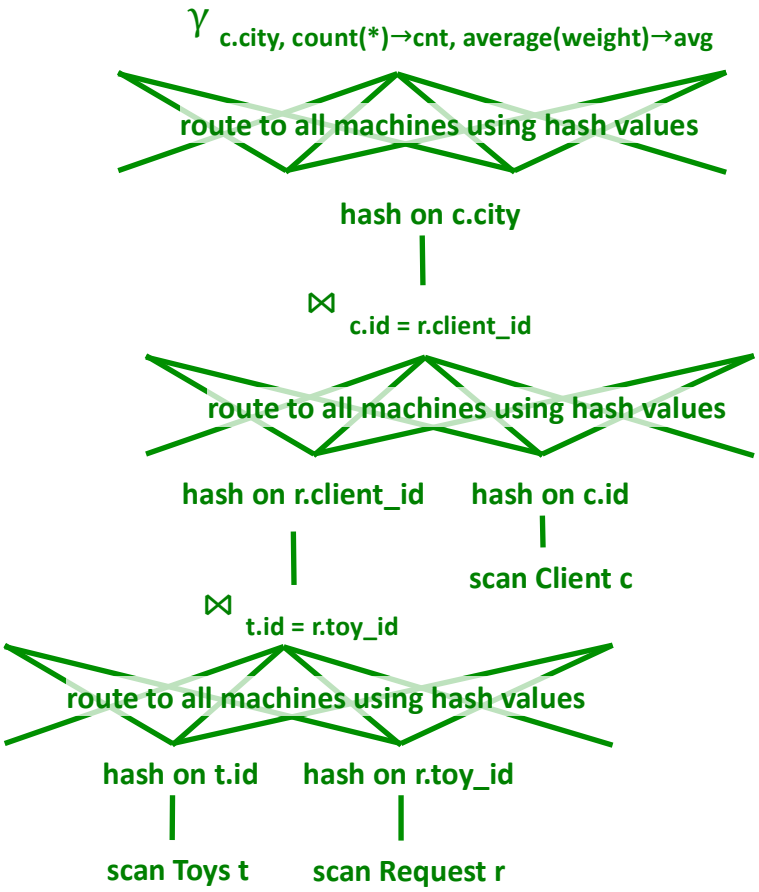


(continued next page)

Question 2. (cont.) (b) (10 points) Now, assume that we want to execute the same query on a 3-node, shared-nothing parallel SQL database engine. You should assume that the data for each of the three tables (Toys, Request, and Client) is *block partitioned* across the three nodes. That is, each node holds 1/3rd of the Toys table, 1/3rd of Request, and 1/3rd of Client, and the allocation of tuples to nodes was done arbitrarily without examining data values in the tuples. Draw a relational algebra tree giving a query plan for executing this query on this 3-node cluster. Note: this question is **not** about map-reduce or PIG; it is about distributed SQL. Query shown again here for reference:

```
SELECT c.city, count(*), average(weight)
FROM Toys t, Request r, Client c
WHERE c.id = r.client_id AND t.id = r.toy_id
GROUP BY c.city
```

To reduce clutter we just show the plan for only one machine, indicating where information needs to be routed to other machines. This is a straightforward adaption of the part (a) solution, routing information to machines as needed and using one routing pass for each operation.



Machine 1 1/3 of each table	Machine 2 1/3 of each table	Machine 3 1/3 of each table

Question 3. (18 points) Santa has some old programs that retrieve information from an earlier version of the database. In that earlier database there was a table that contained information about clients and toy requests with this schema:

ToyList (client_id, client_name, client_age, toy_id)

We want to define a SQL view to be used by the old programs to read data from the current tables.

(a) (8 points) Complete the following SQL view definition to define the ToyList view as an appropriate query on the existing Toys, Client, and Request tables:

```
CREATE VIEW ToyList AS

SELECT r.client_id, c.name as client_name, c.age as client_age, r.toy_id

FROM Client c, Request r

WHERE c.id = r.client_id;
```

Note that the "as client_name" and "as client_age" are needed so that the view schema matches the original schema. If this was missed it was a minor deduction.

(b) (10 points) Now suppose we have the following query that was written to use the old table, and now will be executed using the view defined in part (a):

```
SELECT client_name, client_age
FROM ToyList
WHERE client_age > 2
ORDER BY client_name
```

On the next page, show how this query would be expanded and processed by a SQL database engine. You should show enough detail so it is clear how the query is expanded to one that references the underlying database tables. Simplify the final query to get one that does not involve nested queries, if that is possible. Be sure to show enough steps so we can follow how you produced your answer.

Question 3 (cont.) Space for your answer. Here is the original query using the view.

```
SELECT client_name, client_age
FROM ToyList
WHERE client_age > 2
ORDER BY client_name
```

Write the expanded version of the query below.

Step 1 – expand the query using the view definition to get a nested query.

```
SELECT v.client_name, v.client_age
FROM (SELECT r.client_id, c.name as client_name, c.age as client_age, r.toy_id
      FROM Client c, Request r
      WHERE c.id = r.client_id) as v
WHERE v.client_age > 2
ORDER BY v.client_name
```

Step 2 – unnest the query

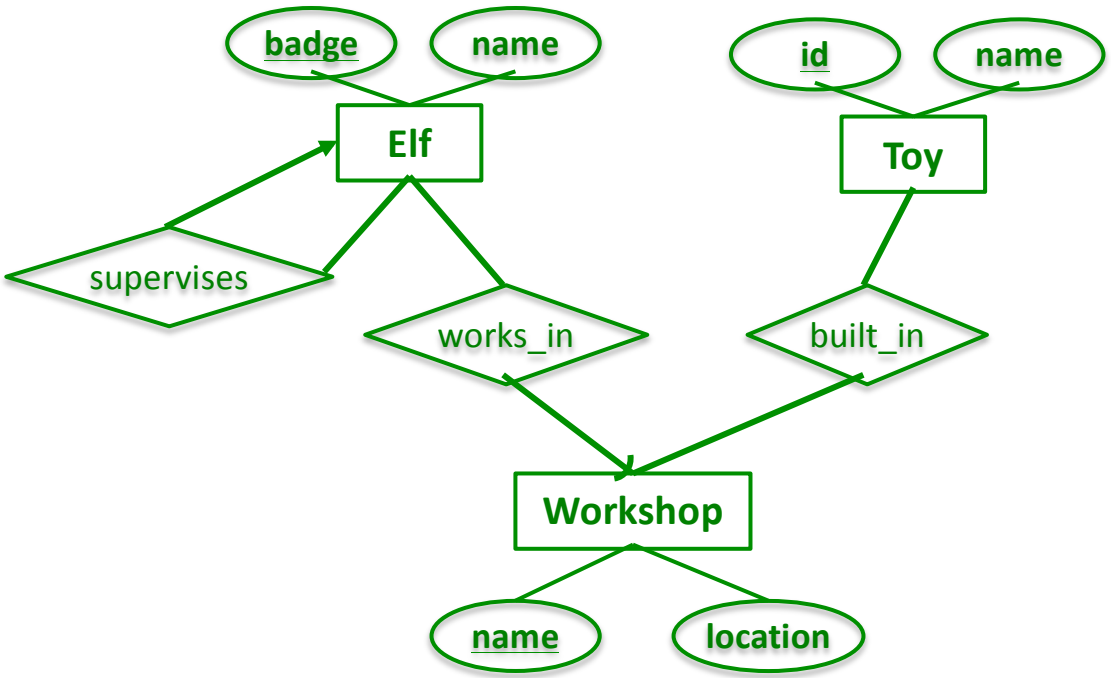
```
SELECT c.name as client_name, c.age as client_age
FROM Client c, Request r
WHERE c.id = r.client_id and c.age > 2
ORDER BY c.name;
```

Question 4. (20 points) Database design. To help him in his work, Santa has many Elves as helpers and he needs to design a database to keep track of them. The database needs to store information about the following:

- Elves: each elf has a unique badge number, a name, and is assigned to a specific workshop. Some elves are supervisors. A supervisor normally supervises several elves, but might only supervise one other elf or none at all if the supervisor has been recently promoted but has not yet been assigned elves to supervise.
- Workshops: Santa has many workshops. Each workshop has a unique name and also has a location. Each workshop builds one or more toys and has at least one elf working in it.
- Toys: Each toy has a unique id number and a name. Every toy is built in one or more workshops.

(a) (12 points) Draw an E/R diagram to model this information.

When grading we made allowances for different designs. This solution shows a simple version of the "supervises" relation as a many-to-one relation between elves. It might be more accurate to introduce a separate subtype ("is-a") entity for Supervisor Elves, but we did not require this.



(continued next page)

Question 4. (cont.) (b) (8 points) Write a sequence of SQL CREATE TABLE commands based on your design in part (a) to create suitable tables to hold the data. The statements should define appropriate SQL constraints that are consistent with the description of the data and your design from part (a). For full credit you should avoid creating unnecessary tables for data that can be easily represented using things like foreign keys in other tables.

```
CREATE TABLE Elf (
  badge_num INTEGER PRIMARY KEY,
  name VARCHAR(20),
  workshop_name VARCHAR(20) NOT NULL references Workshop(name),
)

CREATE TABLE Workshop (
  name VARCHAR(20) PRIMARY KEY,
  location VARCHAR(20)
)

CREATE TABLE Toy (
  id INTEGER PRIMARY KEY,
  name VARCHAR(20),
)

CREATE TABLE Built_in(
  toy_id INTEGER references Toy,
  workshop_name VARCHAR(20) references Workshop,
 PRIMARY KEY(toy_id, workshop_name)
)

CREATE TABLE Supervises(
  supervisor INTEGER references Elf,
  supervisee INTEGER references Elf,
  PRIMARY KEY(supervisor, supervisee)
)
```

Question 5. (20 points) Database normalization. Santa has many helpers in stores like Macy's to talk to boys and girls about what they would like. These helpers have been using spreadsheets as crude databases to keep track of these visits. The spreadsheet entries have the following schema:

Visitor(name, age, address, email, item, color, weight)

From looking at the data, we've discovered the following functional dependencies between columns (attributes):

name -> age	item -> weight
name -> address	email -> name

(a) (10 points) List all the superkeys of this relation and indicate which ones are also keys. Justify your answer in terms of functional dependencies and closures.

The only non-trivial closure that is a key as well as a superkey is {email, item, color} +, which gives the set of all attributes. Every superset of these attributes is also a superkey, but not a key.

(continued)

Question 5. (cont.) (b) (10 points) Decompose the Visitor(name, age, address, email, item, color, weight) relation into BCNF using the functional dependency information from part (a). If the relation is already in BCNF explain why. If one or more decomposition steps are needed, be sure to identify the functional dependency and closures used at each step so we can follow your work.

Looking at the original functional dependencies, we can quickly find these closures that are non-trivial and for which X ≠ X+ ≠ {all attributes}:

name+ = { name, age, address }
item+ = { item, weight }
email+ = { email, name, age, address }

We use the last one for the first split to get

T1 = { email, name, age, address }
T2 = { email, item, color, weight }

Table T1 is not in BCNF because name+ = {name, age, addresss} ≠ {email, name, age, address}. So we split to get

T11 = { name, age, address }
T12 = { email, name }

Both of these tables are in BCNF.

Table T2 is not in BCNF because item+ = {item, weight} ≠ {email, item, color, weight}. Split to get

T21 = { item, weight }
T22 = { item, email, color }

Both of these tables are in BCNF and we're done.

If different FDs are chosen at the various steps, the intermediate tables might be different. But the final BCNF tables will be the same. Answers that did the decomposition steps in different orders were fine as long as they were properly justified and produced the correct results.

Question 6. (20 points) XML. Consider the following XML data that lists some things that people are requesting from Santa for the Holidays. (Question continued on the next page.)

```
<lists>
  <list>
    <person>
      <name> Siena </name>
      <age> 4 </age>
    </person>
    <item>
      <name> Pointe Shoes </name>
      <color> Pink </color>
      <manufacturer location="London"> Freed of London </manufacturer>
    </item>
  </list>
  <list>
    <person>
      <name> Srini </name>
      <age> 9 </age>
    </person>
    <item>
      <name> Espresso Machine </name>
      <manufacturer location="Vermont"> Keurig </manufacturer>
    </item>
    <item>
      <name> PhD </name>
      <quantity> 1 </quantity>
    </item>
  </list>
  <list>
    <person>
      <name> Dan </name>
      <age> 5 </age>
    </person>
    <item>
      <name> Espresso Machine </name>
      <manufacturer location="Vermont"> Keurig </manufacturer>
    </item>
    <item>
      <name> Database Book </name>
      <quantity> 2 </quantity>
    </item>
  </list>
  <list>
    <person>
      <name> Sheng </name>
      <age> 10 </age>
    </person>
    <item>
      <name> Espresso Machine </name>
      <manufacturer location="Vermont"> Keurig </manufacturer>
      <quantity> 7 </quantity>
      <color> Red </color>
    </item>
  </list>
</lists>
```

Question 6 (cont.) The data is stored in a file named `lists.xml` in case that matters.

(a) (12 points) Write the output produced when each of these XPath expressions is executed using the above data.

/lists/list[item/quantity/text() > 2]/person/name

<name> Sheng> <\name>

/lists/list/person/age/text()

4 9 5 10

/lists/list/person[../item/manufacturer[@location = "London"]]

<person>
<name> Siena </name>
<age> 4 </age>
</person>

(b) (8 points) Write an XQuery expression that returns an XML document that gives the names of all items requested along with the names of everyone who requested it. For example, the answer would include:

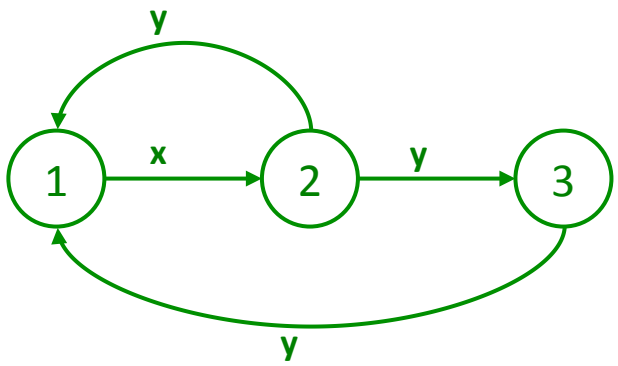
```
<item><name> Espresso Machine </name>
    <person> Dan </person> <person> Sheng </person> <person> Srini </person>
</item>
```

There should be a similar set of nested elements for the other items in the data.

FOR $i IN distinct-values(doc("lists.xml")/lists/list/item/name/text()),
RETURN <item>
<name> $i <\name>
{FOR $p IN doc("lists.xml")/lists/list/person[../item/name/text() = $i]
RETURN <person> $p/name/text() <\person>}
</item>

Question 7. (18 points) Serialization. For each of the following schedules, draw the precedence graph and decide if the schedule is conflict serializable. If it is, give an equivalent serial schedule of transactions (e.g., T2, T1, T3 – you don't need to list the individual read/write steps in the serial schedule). If the schedule is not conflict serializable, explain (very briefly) why not. Label the edges of the precedence graphs with the data that causes the conflict (e.g., X, Y, or Z).

(a) (9 points) r2(Y) w2(Y) r3(Y) r1(X) w1(X) w3(Y) r2(X) r1(Y) w1(Y)



This is not conflict-serializable because of the cycles in the precedence graph.

(b) (9 points) r3(Y) r3(Z) r1(X) w1(X) w3(Y) r2(Z) r1(Y) r2(X) w1(Y) w2(X)

Yes. The equivalent serial schedule is T3 T1 T2

Question 8. (16 points) Locks. Consider the following schedule:

r1(X) w1(X) r2(X) r1(Y) w1(Y) r2(Y) w2(Y) w2(X)

(a) (8 points) Could this schedule be produced by a scheduler that is using the two-phase locking protocol (2PL)? Explain briefly why or why not. (If it is possible, an easy way to demonstrate that is to rewrite (copy) the schedule with lock and unlock operations inserted in the appropriate places.)

Yes, this is possible. One equivalent schedule with 2PL inserted is

L1(X) L1(Y) r1(X) w1(X) U1(X) L2(X) r2(X) r1(Y) w1(Y) U1(Y) L2(Y) r2(Y) w2(Y) w2(X)

Note: several answers said that 2PL requires acquiring all locks at the beginning of a transaction. That is not so; the restriction in 2PL is that a transaction may not lock anything after it has released one or more of the locks it holds.

(b) (8 points) Could this schedule be produced by a scheduler following the strict two-phase locking protocol (strict 2PL)? Explain briefly why or why not. (If it is possible, an easy way to demonstrate that is to rewrite (copy) the schedule with lock and unlock operations inserted in the appropriate places.)

No. With strict 2PL, T1 cannot release either of its locks until it is finished. In the original schedule, T2 reads X before T1 finishes, but that cannot happen since T2 would not be able to acquire the lock on X while T1 is still active.

Question 9. (14 points) Map-Reduce. As Santa and his helpers deliver Toys, they record each delivery on a ticket. Each ticket has a unique number, the client id, the toy id, the toy weight, and the city where it was delivered.

All of this information has been stored in an unstructured sequential file of key-value pairs, where each record has the ticket number as a key and the rest of the data as the associated value:

(ticket#, (client_id, toy_id, weight, city))

Santa would like you to run a quick map-reduce job to produce a list with one entry for each city. Each entry should include the city name, the number of toys delivered to that city, and total weight of those toys. The list does not need to be in any particular order. All you need to do is specify the work that needs to be done by the map and reduce functions of the job. Fill in the pseudo-code below with those details. You do not need to worry about the precise notation – just be sure your pseudo-code is succinct and easy to follow. The input to the map function is written for you. You need to fill in the remaining details. Note: the question is about map-reduce code, **not** PIG.

function map(ticket#, (client_id, toy_id, weight, city))

```
EmitIntermediate ( city , weight )
```

function reduce(**city** , **list of weights**)

```
Emit ( city, length(weights), sum(weights) )
```

Question 10. (24 points, 2 each) For each of the following, write true if the statement is true and false if it is not. (You can also write just T or F, provided that your handwriting makes the difference clear. ☺)

True A *lossless* decomposition is one in which all of the data in the original table is recovered exactly if the decomposed tables are combined with natural joins. Every BCNF decomposition is lossless.

False Every superkey is a key.

False Data stored in XML requires a predefined schema.

True XML attributes are unordered.

False If a schedule is not conflict serializable, it is not possible for there to be an equivalent serial schedule.

False If a database system implements transactions using strict two-phase locking, no deadlock requiring a forced rollback of a transaction can occur.

True If we use the *cascade* referential integrity policy, changes in one table of a database can cause tuples in another table to be deleted.

False Materialized views are computed on-demand when they are referenced during a query and therefore transactions using them can be slow.

False Adding an index to a database system always makes all operations faster.

True Vertical partitioning speeds up queries that touch only some of the columns in the table.

True If a transaction is executed using the Read Committed isolation level, it is possible to get two different values if the same data is read twice.

False If data is written in a NoSQL database then all later read operations on the same database are guaranteed to read the newly written, updated value.

Best wishes for the holidays
See you in January

The CSE 344 Staff