CSE 344 Final Examination

December 16, 2015, 8:30am - 10:20am

Name: ______________________________

Question	Points	Score
1	20	
2	30	
3	20	
4	30	
Total:	100	

- This exam is CLOSED book and CLOSED devices.
- You are allowed TWO letter-size pages with notes (both sides).
- You have 1h:50 minutes; budget time carefully.
- Please read all questions carefully before answering them.
- Some questions are easier, others harder; if a question sounds hard, skip it and return later.
- Good luck!

1 XML, XPath, and XQuery

1. (20 points)

   (a) (10 points) Consider the following XML document stored in a file called data.xml:

```
<cluster>
  <machine id='M1'>
    <software>
      <dbms>MySQL</dbms>
      <dbms>PostgreSQL</dbms>
    </software>
    <hardware>
      <memory>16</memory>
      <cores>4</cores>
    </hardware>
  </machine>

  <machine id='M2'>
    <software>
      <dbms>MySQL</dbms>
    </software>
    <hardware>
      <memory>8</memory>
      <cores>2</cores>
    </hardware>
  </machine>

  <machine id='M3'>
    <software>
      <dbms>PostgreSQL</dbms>
    </software>
    <hardware>
      <memory>64</memory>
      <cores>16</cores>
    </hardware>
  </machine>

  <machine id='M4'>
    <software>
      <dbms>MongoDB</dbms>
    </software>
    <hardware>
      <memory>64</memory>
      <cores>16</cores>
    </hardware>
  </machine>
</cluster>
```

Write **XPath** expressions that compute the following:

1. The id of the machines that run MySQL (formatting of result is up to you):

2. The amount of memory available on machine M4 (formatting of result is up to you):

Solution:

1. Either one of the following:

```
/cluster/machine[.//dbms/text()="MySQL"]/@id
/cluster/machine[.//dbms="MySQL"]/@id
```

2. `/cluster/machine[@id="M4"]//memory/text()`

(b) (10 points) Write an XQuery expression that will transform data.xml into the following document:

```
<result>
  <dbms>
    <name>MySQL</name>
    <nb_servers>2</nb_servers>
    <cores>4</cores>
    <cores>2</cores>
  </dbms>
  <dbms>
    <name>PostgreSQL</name>
    <nb_servers>2</nb_servers>
    <cores>4</cores>
    <cores>16</cores>
  </dbms>
  <dbms>
    <name>MongoDB</name>
    <nb_servers>1</nb_servers>
    <cores>16</cores>
  </dbms>
</result>
```

Solution:

```
<result>
{
  for $d in doc("data.xml")/cluster
  for $x in distinct-values( $d//dbms/text())
  return
    <dbms>
      <name>{$x}</name>
      <nb_servers>
      {count($d/machine[software/dbms/text() = $x])}
      </nb_servers>
      {
      for $y in $d/machine[software/dbms/text() = $x]
      return $y/hardware/cores
      }
   </dbms>
}
</result>
```

2 E/R Diagrams, Constraints, Conceptual Design

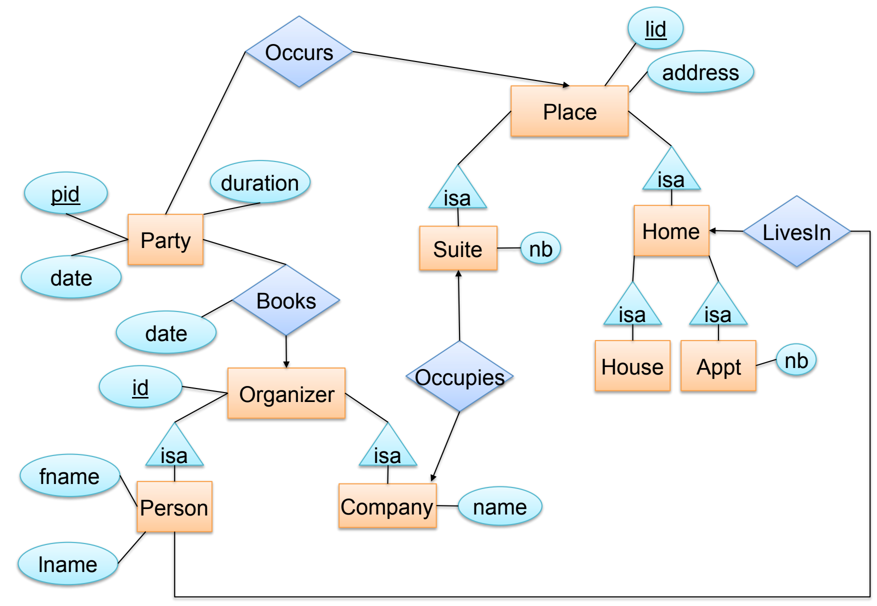
2. (30 points)

   (a) (10 points) Design an E/R diagram describing the following domain:

   - A **person** has attributes **id** (key), **fname**, and **lname**.
   - A **company** has attributes **id** (key) and **name**. Each company name is unique.
   - A **party** has attributes **pid**, **date**, and **duration**. The date attribute cannot be NULL. A party cannot last longer than 4 hours.
   - A party is **booked** by either a person or a company. We need to capture information about both **when the party takes place** and **when the party was booked**. The booking date must be earlier than the party date. A party is booked by zero or one organizers. A person or company can book many parties.
   - A party **occurs** in either a **house**, an **apartment**, or a **suite**. All these locations are uniquely identified with a **location id (lid)**. A house has an attribute **street address**. The address is an atomic string attribute. An apartment additionally has an **apartment number**. A suite has a street address and a **suite number**. A party occurs in zero or one place. A place can host multiple parties.
   - A company **occupies** zero or one suite and a suite hosts zero or one company (i.e., the application does not store historical data only current data).
   - A person **lives** in either a house or an apartment. A person lives in zero or one place. A given place can host many people.

   <u>**Answer**</u> (Draw an E/R Diagram on the next page):

Answer (Draw an E/R Diagram):



Solution:

(b) (10 points) Write the CREATE TABLE statements necessary to capture the **subset of the above ER diagram** that corresponds to **party, person, and company** entities as well as the **books** relationship. Include all primary key, foreign key, and other constraints. Avoid creating unnecessary tables when possible. Dates can be represented with the `datetime` type.

Answer (Write CREATE TABLE statements for **SUBSET** of ER diagram):

Solution:

```
 CREATE TABLE Party(
    pid int PRIMARY KEY,
    party_date datetime NOT NULL,
    duration float CHECK(duration > 0 and duration < 4),
    oid int REFERENCES Organizer,
    booking_date datetime,
    CHECK(booking_date < party_date))

CREATE TABLE Organizer (id int PRIMARY KEY)

CREATE TABLE Person(
    id int PRIMARY KEY REFERENCES Organizer,
    fname VARCHAR(100),
    lname VARCHAR(100))

CREATE TABLE Company(
    id int PRIMARY KEY REFERENCES Organizer,
    name VARCHAR(100) UNIQUE)
```

(c) (10 points) Consider the following relational schema and set of functional dependencies.

R(A,B,C,D,E,F,G) with functional dependencies:
E → C
G → AD
B → E
C → BF

- Give one example of non-trivial functional dependency implied by the ones above:

  Answer (Example FD):

  Solution: Many solutions are possible. One example is E → BF.

- Compute E^+, the closure of E.

  Answer (E^+):

  Solution: $\{E\}^+ = \{B, C, E, F\}$

- Why do we bother to decompose relations into BCNF? What does this normal form ensure?

  Answer (Explanation):

  Solution: BCNF ensures that, for each table, only "good" FDs exist: Whenever X → B is a non-trivial dependency, then X is a super key. Because only these good FDs exist, the relation will not suffer from data redundancy, update anomalies, nor deletion anomalies (see lecture 18 for details).

- Decompose R into BCNF. Show your work for partial credit. Your answer should consist of a list of table names and attributes and an indication of the keys in each table (underlined attributes).

  Answer (Decompose R into BCNF):

Solution: $\{E\}^+ \to \{B, C, E, F\}$ violates BCNF because it is neither trivial nor contains all attributes (i.e., E is not a superkey). So we need to decompose. We decompose into R1(B, C, $\underline{\text{E}}$, F) and R2(A, D, $\underline{\text{E, G}}$).

For R1(B, C, $\underline{\text{E}}$, F), there are no more FD that violate BCNF, so we continue with R2(A, D, $\underline{\text{E, G}}$). We find $\{G\}^+ = \{A, D\}$, so we decompose into R21($\underline{\text{G}}$, A, D) and R22($\underline{\text{G, E}}$). R22 has now two attributes, so it is necessarily in BCNF. For R21, there are no more FDs that violate BCNF.

3 Transactions

3. (20 points)

   (a) (10 points) Consider transactions executing concurrently on the same instance of SQL Server. For each of the execution traces below, circle the isolation levels (if any) that will allow the given interleaved execution to occur.

```
BEGIN TRANSACTION              |
                               |
                               | BEGIN TRANSACTION
                               |
SELECT * from Customers        |
                               |
                               | SELECT * from Customers
                               |
                               | SELECT * from Products
                               |
SELECT * from Products         |
                               |
COMMIT                         |
                               |
                               | COMMIT
```

Serialiazable   Repeatable Read   Read Committed

Solution: The above sequence of operations can occur at all levels of isolation since the transactions only read data without writing.

`BEGIN TRANSACTION`	
	`BEGIN TRANSACTION`

`UPDATE Customers SET id = 20`
`WHERE id = 10`	

	`UPDATE Products SET price = 0`
	`WHERE price is NULL`

	`SELECT * from Products`
`SELECT * from Customers`	
`COMMIT`	
	`COMMIT`

Serialiazable Repeatable Read Read Committed

Solution: The above sequence of operations can occur at all levels of isolation since the transactions operate on different tables.

`BEGIN TRANSACTION`	
	`BEGIN TRANSACTION`

`UPDATE Customers SET id = 10`
`WHERE id = 20`	

	`UPDATE Products SET price = 10`
	`WHERE price = 0`

	`SELECT * from Customers`
`SELECT * from Products`	
`COMMIT`	
	`COMMIT`

Serialiazable Repeatable Read Read Committed

Solution: The above sequence of operations cannot occur at any of the above levels of isolation. At all these levels, each transaction will acquire a write lock on the record that it updates. When the transactions try to read the tables, they will block trying to acquire a shared lock. This will cause a deadlock and one of the transactions will get aborted.

`BEGIN TRANSACTION`	
	`BEGIN TRANSACTION`
`SELECT * from Customers`	
	`INSERT INTO Customers VALUES(1)`
	`SELECT * from Customers`
`COMMIT`	
	`COMMIT`

Serialiazable Repeatable Read Read Committed

Solution: The above sequence of operations can occur only at the repeatable read and read committed levels. At those levels, locks are acquired on individual records. At the serializable level, predicate locking will block the insert until the first transaction either commits or aborts.

(b) (5 points) Consider the following transaction schedules. For each schedule, draw the precedence graph and indicate if it is **conflict-serializable** or not.

r1(A); w1(B); r2(B); w2(C); r3(C); w3(A);

Answer (YES/NO):

Solution: YES. The graph is $1 \rightarrow 2 \rightarrow 3$.

r1(A); r2(B); r3(B); w3(A); w2(C); r3(D); r3(C); w1(B);

Answer (YES/NO):

Solution: NO. There is a cycle between transactions 1 and 2. T1 precedes T3 on A but follows it on B.

(c) (5 points) What is strict 2PL?

Solution: With 2PL, once a transaction releases a lock, it cannot acquire any new locks. Hence, all lock operations precede all the unlock operations.

With strict locking, a transaction must hold all its locks until it either commits or aborts (and all changes are undone).

4 Parallel Data Processing

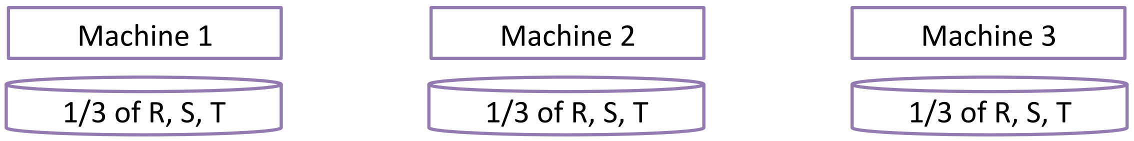
4. (30 points)

   (a) (10 points) Consider relations `R(a,b)`, `S(c,d)`, and `T(e,f)`. All three are horizontally partitioned across $N = 3$ machines as shown in the diagram below. Each machine locally stores approximately $\frac{1}{N}$ of the tuples in `R`, `S`, and `T`. The tuples are randomly organized across machines (i.e., `R` is block partitioned across machines).

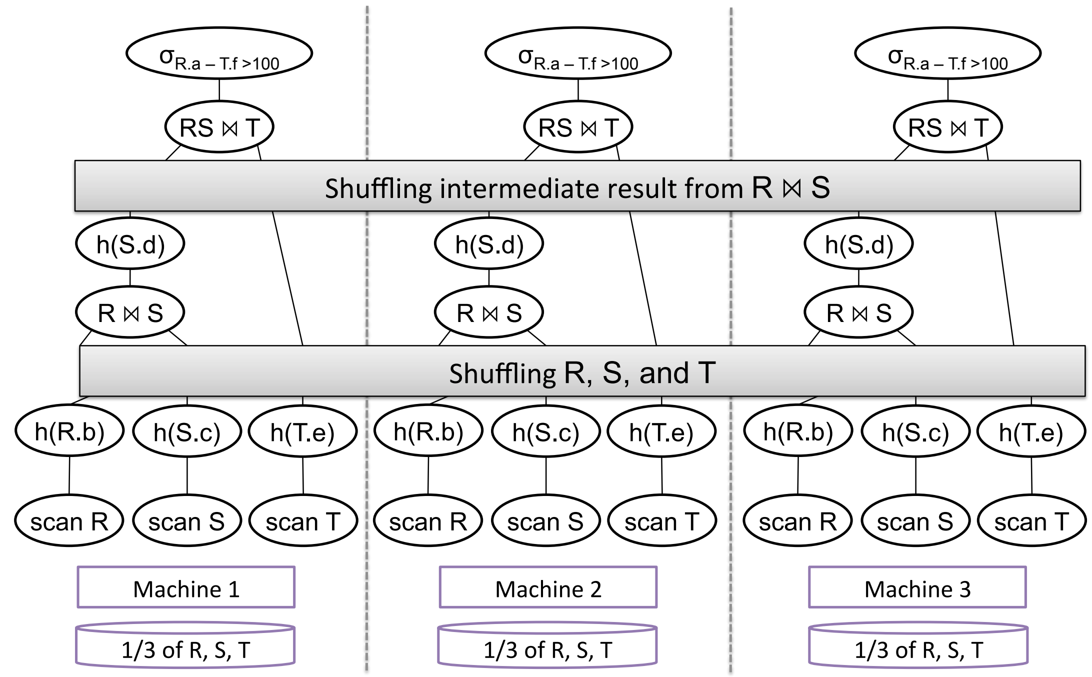
   Show a relational algebra plan for the following query and how it will be executed across the $N = 3$ machines. **Use hash-join (a.k.a shuffle-join) operators**. Include operators that need to re-shuffle data and add a note explaining how these operators will re-shuffle that data.

```
SELECT *
FROM R, S, T
WHERE R.b = S.c
AND S.d = T.e
AND (R.a - T.f) > 100
```

   <u>**Answer**</u> (Draw the parallel query plan):

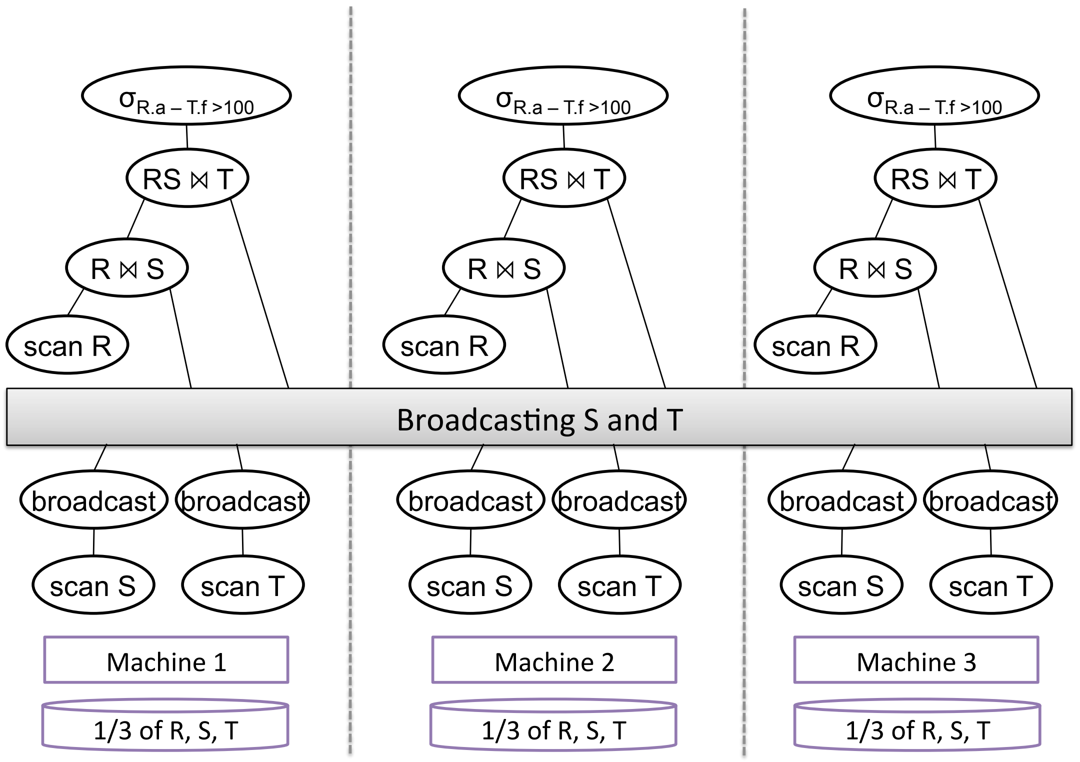


Solution:



(b) (10 points) Now consider the case where the R relation is very large and both S and T are very small. Show a plan that uses **broadcast joins** to compute the result of the query.

Solution:



(c) (10 points) Explain how the query would be executed in **MapReduce** (not Pig). Make sure to specify the computations performed in the map and the reduce functions. Use **hash-joins** (shuffle joins). No need to give the pseudocode. Just state what each function does and what it outputs in plain text format.

Answer (Describe the map and reduce functions of the executed MapReduce job(s)):

Solution: We will use two MapReduce jobs. The first job will compute the join of R and S.

In the first job, each map task scans either a block of R or a block of S and calls the map function for each tuple. For blocks of R, the map function outputs tuples of the form: key = $R.b$ and value = $('R', R.a, R.b)$. For blocks of S, the map function outputs tuples of the form: key = $S.c$ and value = $('S', S.c, S.d)$.

The MapReduce engine reshuffles the output of the map phase and groups it on the intermediate key, which is the join attribute.

Each reduce task computes the cartesian product of the tuples from the two relations for each group assigned to it (by calling the reduce function) and outputs the final result of the first join.

We now need a second MapReduce job that will perform the second join of RS and T.

In this second job, each map task scans either a block of RS or a block of T and calls the map function for each tuple. For blocks of RS, the map function outputs tuples of the form: key = $S.d$ and value = $('RS', R.a, R.b, S.c, S.d)$. For blocks of T, the map function outputs tuples of the form: key = $T.e$ and value = $('T', T.e, T.f)$.

The MapReduce engine reshuffles the output of the map phase and groups it on the intermediate key, which is the join attribute.

Each reduce task computes the cartesian product of the tuples from the two relations for each group assigned to it (by calling the reduce function). The reduce task selects tuples with R.a - T.f ¿ 100 and outputs the final result of the query.