CSE 344 Final Examination

March 14, 2012, 8:30am - 10:20am

Name: _____

Question	Points	Score
1	60	
2	20	
3	40	
4	50	
5	30	
Total:	200	

- This exam is a closed book exam.
- 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 SQL, Relational Calculus, Relational Algebra

1. (60 points)

   The following database constains information about email messages in a large organization:

   ```
   Person(pid,name)
   Email(eid, pidFrom, tid, body, length)
   EmailTo(eid,pidTo)
   ```

   Where:

   - `Person.pid` and `Email.eid` are keys.
   - `Email.pidFrom` and `EmailTo.pidTo` are foreign keys to `Person`.
   - `EmaiTo.eid` is a foreign key to `Email`.
   - Every email may be sent to several recipients, stored in the `EmailTo` table.
   - `tid` represents the thread to which the email belongs.

   (a) (10 points) Write the SQL statements that define the relational schema for this database. Assume that `pid`'s and `eid` s are integers, and `name` and `body` are character strings (choose an appropriate length).

   Answer (write a SQL statement):

Solution:

```
create table Person(
  pid int primary key,
  name varchar(20));
create table Email(
  eid int primary key,
  pidFrom int references Person,
  tid int,
  body varchar(1024),
  length int);
create table EmailTo(
  eid int references Email,
  pidTo int references Person);
```

```
Person(pid,name)
Email(eid, pidFrom, tid, body, length)
EmailTo(eid,pidTo)
```

(b) (10 points) For each email thread, compute the total number of distinct recipients of emails in that thread. For example, if the thread contains 100 emails, and all emails went to only one recipient, then your query should answer 1 for that thread id.

Answer (write a SQL statement):

Solution:

```
select e.tid, count(distinct e2.pidTo)
from Email e, EmailTo e2
where e.eid = e2.eid
group by e.tid
```

```
Person(pid,name)
Email(eid, pidFrom, tid, body, length)
EmailTo(eid,pidTo)
```

(c) (10 points) A *circle* is a set of three users a, b, c such that a sent an email to b, b sent an email to c, c sent an email to a, and all these three emails are in the same thread. Write a SQL query that computes the total number of circles in the database.

Answer (write a SQL query):

Solution:

```
select count(*)
from email e1, emailTo t1, email e2, emailTo t2, email e3, emailTo t3
where e1.eid = t1.eid and t1.pidTo = e2.pidFrom
  and e2.eid = t2.eid and t2.pidTo = e3.pidFrom
  and e3.eid = t3.eid and t3.pidTo = e1.pidFrom
  and e1.tid = e2.tid and e2.tid = e3.tid;
```

```
Person(pid,name)
Email(eid, pidFrom, tid, body, length)
EmailTo(eid,pidTo)
```

(d) (10 points) A *spammer* is a person who has sent at least one email in every thread. Write a SQL query to find all spammers. Your query should return the spammer's pid and their name.

Answer (write a SQL query):

Solution: I first write it in the Relational Calculus, then remove universal quantifiers:

$$Q(pid, n) = \texttt{Person}(pid, n) \wedge (\forall eid_1, pidf_1, tid, b_1, l_1.\texttt{Email}(eid_1, pidf_1, tid, b_1, l_1) \Rightarrow \exists eid_2, b_2, l_2.\texttt{Email}(eid_2, pid, tid, b_2, l_2))$$
$$= \texttt{Person}(pid, n) \wedge \not(\exists eid_1, pidf_1, tid, b_1, l_1.\texttt{Email}(eid_1, pidf_1, tid, b_1, l_1) \wedge \not\exists eid_2, b_2, l_2.\texttt{Email}(eid_2, pid, tid, b_2, l_2))$$

Next, datalog:

```
T(pid, tid) :- Email(eid2,pid,tid,b2,l2)
K(pid)      :- Person(pid,-), Email(eid1,pidf1,tid,b1,l1), not T(pid,tid)
Q(pid)      :- Person(pid,n), not K(pid)
```

The predicate `Person(pid,-)` in the definition of `K` is introduced in order to make the rule safe. We now translate to SQL, and may remove that predicate:

```
select p.pid
from Person p
where not exists (select *
                  from Email e1
                  where not exists (select *
                                    from Email e2
                                    where e2.pidFrom = p.pid and e2.tid=e1.tid));
```

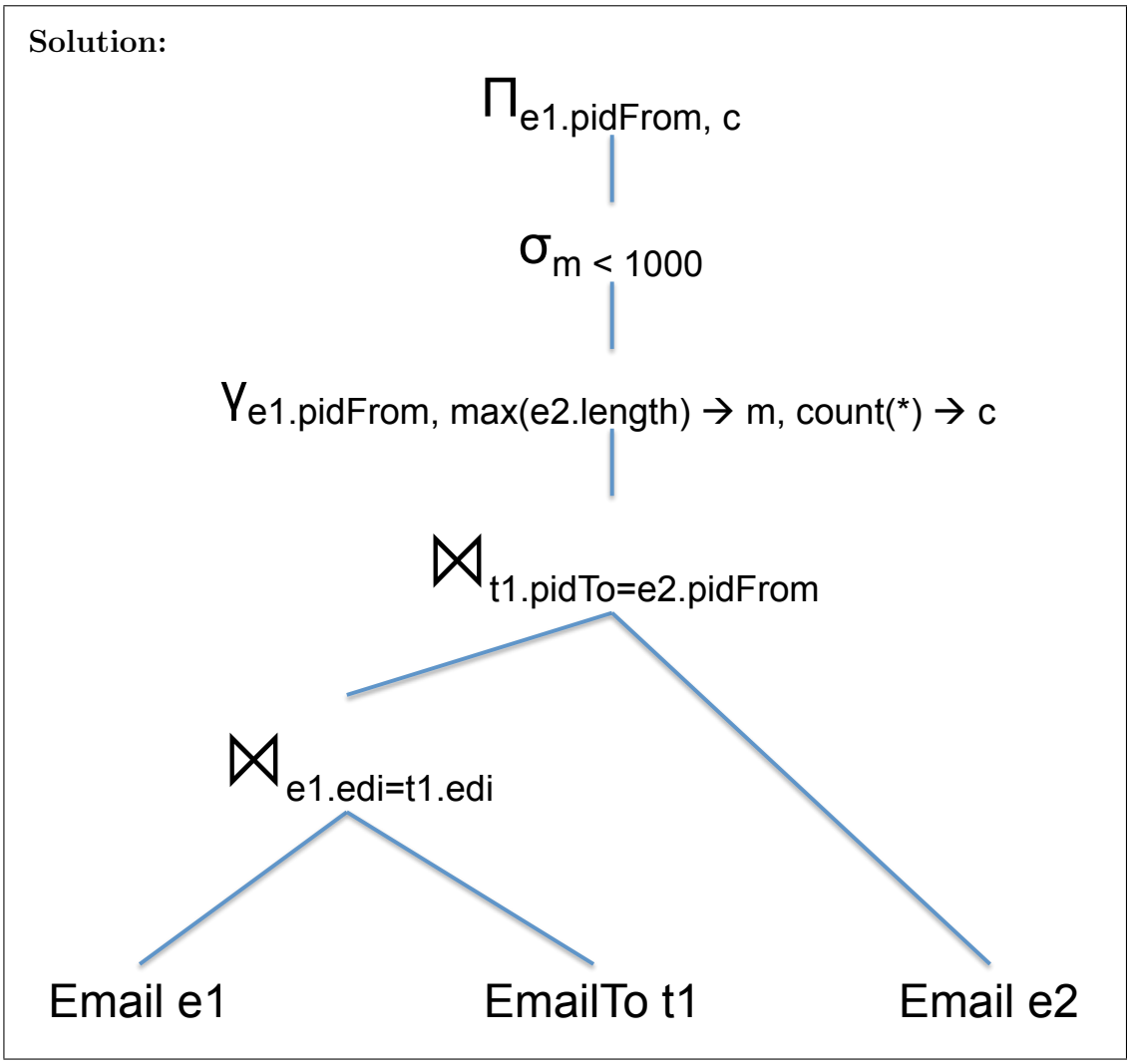
```
Person(pid,name)
Email(eid, pidFrom, tid, body, length)
EmailTo(eid,pidTo)
```

(e) (10 points) Write a query plan in the extended relational algebra that computes the following query:

```
select e1.pidFrom, count(*)
from Email e1, EmailTo t1, Email e2
where e1.eid = t1.eid and t1.pidTo = e2.pidFrom
group by e1.pidFrom
having max(e2.length) < 1000;
```

Answer (write a query plan; you may draw it as a tree):

Solution:



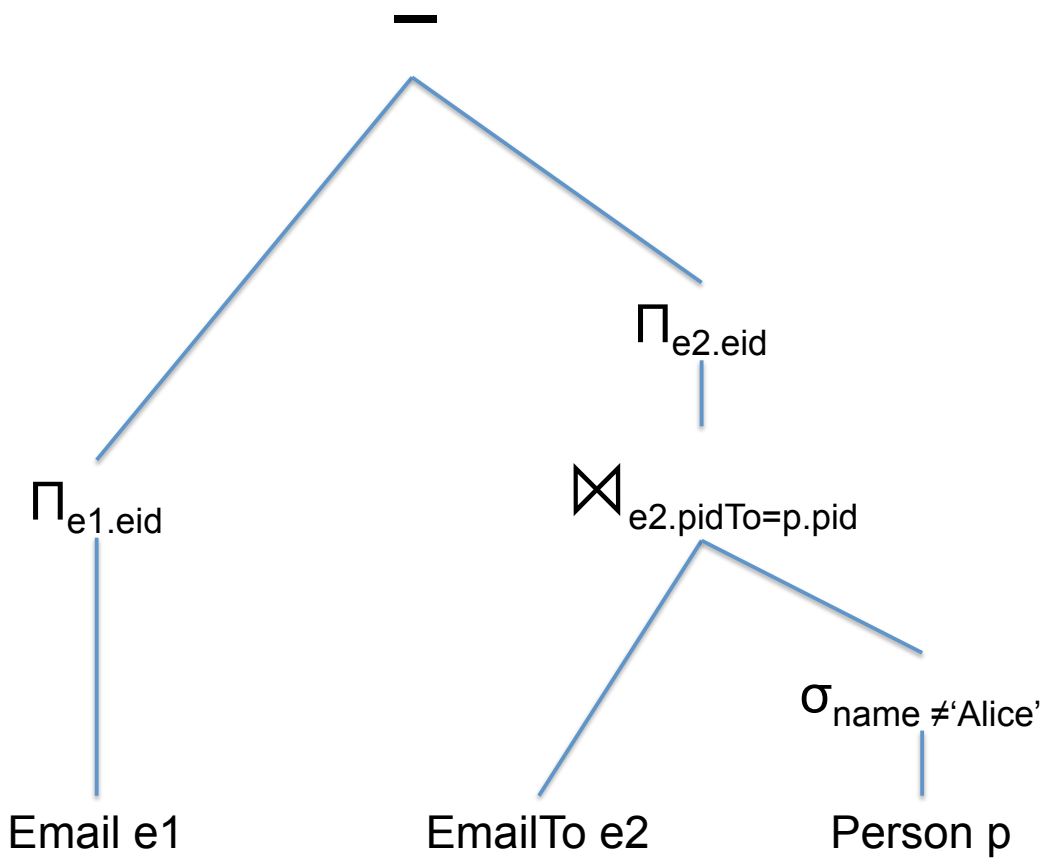
```
Person(pid,name)
Email(eid, pidFrom, tid, body, length)
EmailTo(eid,pidTo)
```

(f) (10 points) The query below retrieves all emails where every recipient was named Alice. Write a query plan in the relational algebra for this query.

```
select e1.eid
from Email e1
where not exists (select *
                  from EmailTo e2, Person p
                  where e1.eid = e2.eid
                    and e2.pidTo = p.pid
                    and p.name != 'Alice');
```

Answer (write a query plan; you may draw it as a tree):

Solution:



2 XML/XPath/XQuery

2. (20 points)

The following DTD describes an XML document about students and the courses they take:

```
<!DOCTYPE Enrollment [
<!ELEMENT Enrollment  (student* )>
<!ELEMENT student  (name,  address, course*)>
<!ELEMENT course  (title, instructor, grade?)>
<!ELEMENT name  (#PCDATA )>
<!ELEMENT address (#PCDATA )>
<!ELEMENT title  (#PCDATA )>
<!ELEMENT instructor  (#PCDATA )>
<!ELEMENT grade  (#PCDATA )>
]>
```

(a) (10 points) Write an XPath expression that retrieves the names of all students whose address is 'Seattle' and who received a grade < 3.0. You should assume that the query processor performs the correct type conversions: that is, is suffices for you to write `grade` < 3.0.

Answer (write a XPath expression):

Solution:

```
doc("problem2.xml")/Enrollment/student[address='Seattle'][course/grade<3.0]/name
```

(b) (10 points) The data is not normalized: the same course is listed multiple times, once for each student who took that course. You normalize the data and represent it as follows:

```
<!DOCTYPE Normalized [
<!ELEMENT Normalized  (students,courses,takes)>
<!ELEMENT students  (student* )>
<!ELEMENT courses   (course* )>
<!ELEMENT takes   (take* )>
<!ELEMENT student   (name, address)>
        <!ELEMENT course    (title, instructor)>
        <!ELEMENT take    (name, title, grade?)>
]>
```

Your Boss does not understand data anomalies and normalization theory, so he asks you to convert it back to the orignal format. Write an XQuery that transforms the `Normalized` data to the `Enrollment` data.

Answer (write a XPath expression):

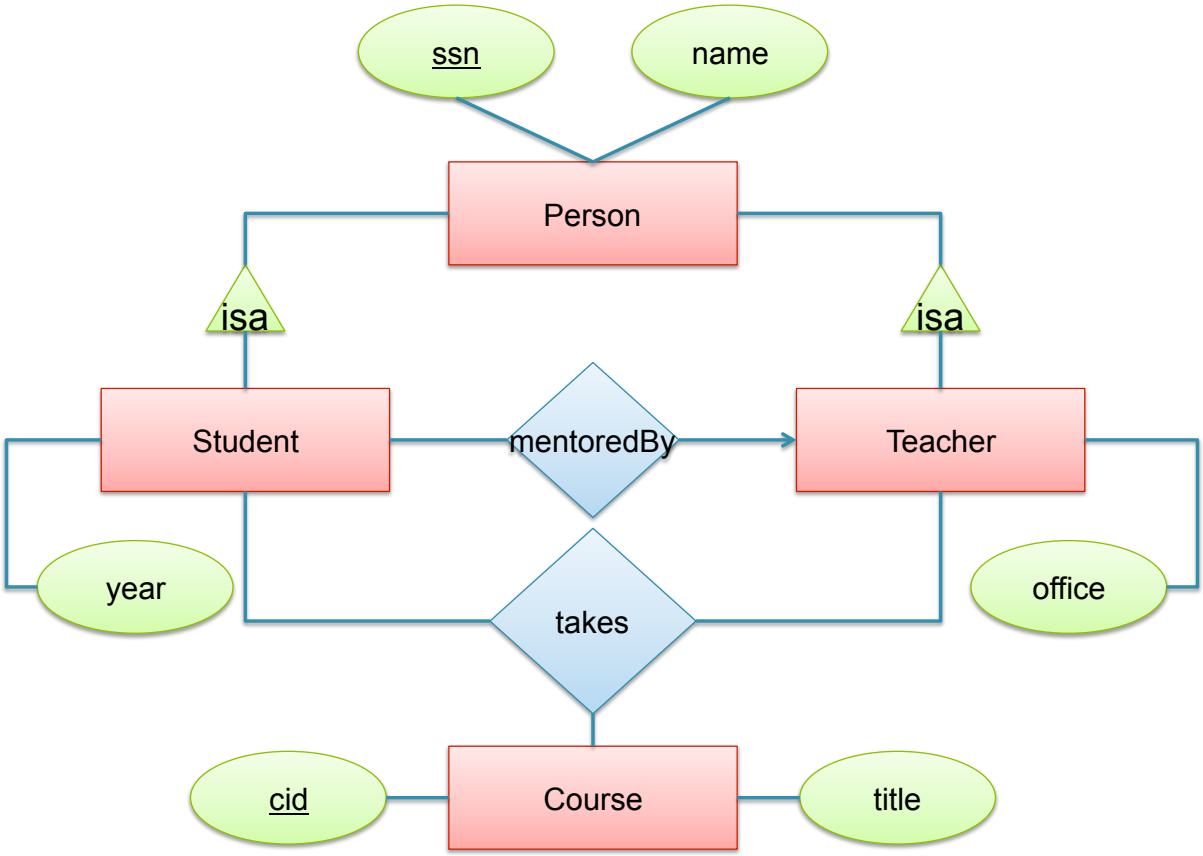
Solution:

```
<Enrollment>
  { for $n in doc("problem2b.xml")/Normalized,
    $s in $n/students/student
    return <student>
             { $s/name,
               $s/address,
               for $e in $n/takes/take[name=$s/name],
                   $c in $n/courses/course[title=$e/title]
               return <course>
                        { $c/title,
                          $c/instructor,
                          $e/grade
                        }
                      </course>
             }
           </student>
   }
</Enrollment>
```

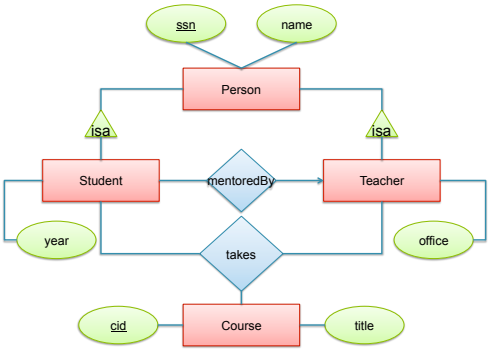
3 E/R Diagrams, Constraaints, Conceptual Design

3. (40 points)

   (a) (10 points) The following E/R diagram describes a database about students and teachers:



Every student and every teacher is a person. Some students have a teacher mentor. Every student is in some year (1, 2, 3, . . .) and every teacher has an office. Students take courses from teachers. Convert the E/R diagram to SQL. You should choose appropriate data types for all attributes, and should write all key and foreign key constraints.



Answer (Write CREATE TABLE statements):

Solution:

```
create table Person(ssn int primary key, name varchar(30));
create table Teacher(
        ssn int primary key references Person,
        office varchar(20));
create table Student(
        ssn int primary key references Person,
        mentoredby int references Teacher,
        year int);
create table Course(cid int primary key, title varchar(30));
create table Takes(
        sid int references Student,
        tid int references Teacher,
        cid int references Course);
```

(b) (10 points) Answer the following questions.

i. What is an update anomaly? Choose one of the following:

(a) One transaction reads an element that was updated by an earlier, uncommitted transaction.

(b) The application wants to update a foreign key to a new value that does not exists in the referenced relation.

(c) The same information is stored redundantly in the database, and only some, but not all copies are updated.

i. **(c)**

Answer (a), (b), or (c).

ii. Every relational schema in SQL is in 1st normal form.

ii. **true**

True or false?

iii. Every XML data is in 1st normal form.

iii. **false**

True or false?

iv. Which of the following statements best describes the main reason for representing a relational database in 1st normal form?

(a) To achieve physical data independence.

(b) To remove data anomalies (insertion, update, deletion anomalies).

(c) To save space on disk.

iv. **(a)**

Answer (a), (b), or (c).

v. Which of the following statements best describes the main reason for representing a relational database in BCNF?

(a) To achieve physical data independence.

(b) To remove data anomalies (insertion, update, deletion anomalies).

(c) To save space on disk.

v. **(b)**

Answer (a), (b), or (c).

(c) (10 points) Consider the following instance of a relation $R(A, B, C, D)$:

A	B	C	D
a	b	c	d
a'	b	c'	d
a'	b'	c	d'

For each of the following statements indicate whether it is true or false:

i. A is a key.

i. **false**

True or false?

ii. B is a key.

ii. **false**

True or false?

iii. AB is a key.

iii. **true**

True or false?

iv. BD is a key.

iv. **false**

True or false?

v. The functional dependency $A \rightarrow B$ holds.

v. **false**

True or false?

vi. The functional dependency $B \rightarrow D$ holds.

vi. **true**

True or false?

vii. $D+ = BD$ holds.

vii. **true**

True or false?

(d) (10 points) Consider two relations $R(A, B, C, D)$ and $S(A, B, C, D)$, with the following functional dependencies:

$$R: \quad A \to BCD$$
$$B \to ACD$$

$$S: \quad BC \to AD$$
$$D \to B$$

i. Find all keys in R.

i. A **and** B

The keys are:

ii. Find all keys in S.

ii. BC **and** CD

The keys are:

iii. Find all keys in $R \cap S$.

iii. A**,** B**, and** CD

The keys are:

Solution: A set of attribute is a superkey in $R \cap S$ iff it is a superkey in R or in S. The superkeys in R are: $A, AC, AD, ACD, B, BC, BD, BCD, ABC, ABD, ABCD$. The superkeys in S are: $BC, ABC, CD, ACD, BCD, ABCD$. The superkeys in $R \cap S$ are all them, and therefore the keys are A, B, and CD.

iv. Assume that the relations $R(A, B, C, D)$ and $S(A, B, C, D)$ do not have any common values. That is, the values of the attribute $R.A$ are distinct from those of the attribute $S.A$, and the same for the attributes B, C, D. Find all keys in $R \cup S$

iv. BC **and** ACD

The keys are:

Solution: A set of attributes is a superkey in $R \cup S$ iff it is a superkey in both R and S. Thus, the superkeys in $R \cup S$ are $BC, ABC, ACD, BCD, ABCD$. The keys are: BC and ACD.

4 Transactions

4. (50 points)

   (a) (30 points) Consider a database consisting of a single relation R:

R:

A	B
1	10
2	20

Three transactions run concurrently on this database, issuing commands at the following time stamps:

Time	T1	T2	T3
1	begin transaction;		
2	select * from R;		
3		begin transaction;	
4		select * from R where $A = 2$;	
5	update R set $B = 30$ where $A = 2$;		
6		select * from R where $A = 2$;	
7	commit;		
8			begin transaction;
9			select * from R where $A = 2$;
10		commit;	
11			
12			
13			commit;

Transaction T1 first reads the data, then updates the second record ($A = 2$). It attempts to commit at time stamp 7.

Transaction T2 attempts to read the second record ($A = 2$) at time stamp 4.

Transaction T3 attempts to read the second record ($A = 2$) at time stamp 9.

i. Find out what happened. You will consider two scenarios: when the database is managed with SQL Lite, and when the database is managed with SQL Server; in both cases the isolation level is set to SERIALIZABLE. For each command issued by a transaction, indicate one of the following outcome:

SUCCESS The request returns immediately, with success. In this case you should write SUCCESS, and also write the value that the transaction has read, if applicable.

ERROR The request returns immediately, with error. In this case you should write ERROR and indicate at which later time stamp the transaction needs to retry that command; if the command was a read, write down the value read by the transaction, when the command is reissued successfully.

WAIT The request causes the transaction to be placed on wait. In that case you should write WAIT, and also write at which later time stamp the transaction will be allowed to resume; if the command was a read, write down the value read by the transaction, when the command is resumed.

For example, you may answer as follows (not a real answer):

Time	T1	T2	T3
1	begin transaction;		
2	select * from R; – WAIT UNTIL $T = 4$		
3		begin transaction; – SUCCESS	
4	 – RESUMED: values$10, 30$	select * from R where $A = 2$; – SUCCESS: value= 30	
	...	...	...

This schedule is excecuted on **SQL Lite**			
	T1	T2	T3
1	begin transaction;		
2	select * from R;		
3		begin transaction;	
4		select * from R where $A = 2$;	
5	update R set $B = 30$ where $A = 2$;		
6		select * from R where $A = 2$;	
7	commit;		
8			begin transaction;
9			select * from R where $A = 2$;
10		commit;	
11			
12			
13			commit;

Solution:

Time	T1	T2	T3
1	begin transaction;		
2	select * from R; – SUCCESS: values 10,20		
3		begin transaction;	
4		select * from R where $A = 2$; – SUCCESS: value 20	
5	update R set $B = 30$ where $A = 2$; – SUCCESS		
6		select * from R where $A = 2$; – SUCCESS: value 20	
7	commit; ERROR: retry on 11		
8			begin transaction; SUCCESS
9			select * from R where $A = 2$; ERROR: retry on 12
10		commit; – SUCCESS:	
11	– REISSUE: commit; – SUCCESS;		
12			– REISSUE: select * from R where $A = 2$; – SUCCESS: value 30
13			commit; – SUCCESS

This schedule is excecuted on **SQL Server**			
	T1	T2	T3
1	begin transaction;		
2	select * from R;		
3		begin transaction;	
4		select * from R where $A = 2$;	
5	update R set $B = 30$ where $A = 2$;		
6		select * from R where $A = 2$;	
7	commit;		
8			begin transaction;
9			select * from R where $A = 2$;
10		commit;	
11			
12			
13			commit;

Solution:

Time	T1	T2	T3
1	begin transaction;		
2	select * from R; – SUCCESS: values 10,20		
3		begin transaction;	
4		select * from R where $A = 2$; – SUCCESS: value 20	
5	update R set $B = 30$ where $A = 2$; – SUCCESS		
6		select * from R where $A = 2$; – WAIT: until 7	
7	commit; – SUCCESS	 – SUCCESS: value 30	
8			begin transaction; – SUCCESS
9			select * from R where $A = 2$; – SUCCESS: value 30
10		commit; – SUCCESS:	
11			
12			
13			commit; – SUCCESS

ii. What is the serialization order of the three transactions on SQL Lite?

ii. **T2, T1, T3**

Give an order or the transactions T1, T2, T3:

iii. What is the serialization order of the three transactions on SQL Server?

iii. **T1, T2, T3**

Give an order or the transactions T1, T2, T3:

(b) (10 points) Recall that a *shared lock* (or *read lock*) may be held by several transactions, while an *exclusive lock* (or *write lock*) can be held by only one transaction.

- System A uses shared and exclusive locks.
- System B uses only exclusive locks.

Indicate which statements below are true.

i. System A may be able to execute more transactions per second than system B.

True or false?

i. **true**

ii. If all transactions are read-only, then on system A no transaction ever waits.

True or false?

ii. **true**

iii. If all transactions are write-only, then on system B no transaction ever waits.

True or false?

iii. **false**

iv. There exists schedules that result in a deadlock on system A but not in a deadlock on system B.

True or false?

iv. **true**

Solution: True. For example the following schedule: Start1, Start2, Read1(X), Read2(X), Write1(X), Write2(X), Commit1, Commit2. On system A, both transactions start with a read lock on X, then attempt to escalate it to a write lock, and result in deadlock. On system B, the second transaction is placed on wait during the operation Read2(X).

(c) (10 points) A *static database* is a database where no insertions or deletions are performed. A *dynamic database* is a database that allows insertions and deletions. We consider a scheduler that has one lock for each record in the database (like SQL Server). Answer the questions below:

i. In a static database, strict two phase locking guarantees that the schedule is serializable while two phase locking does not.

i. **false**

True or false

ii. Strict two phase locking guarantees that the schedule is recoverable, while two phase locking does not.

ii. **true**

True or false

iii. In a dynamic database, strict two phase locking can prevent phantoms, while two phase locking cannot.

iii. **false: needs table lock**

True or false

iv. Strict two phase locking is more difficult to implement and most database system do not support it.

iv. **false**

True or false

v. Strict two phase locking holds all the locks until the end of a transaction, while two phase locking may release the locks earlier.

v. **true**

True or false

vi. In both two phase locking and strict two phase locking all locks must precede all unlocks.

vi. **true**

True or false

vii. In strict two phase locking deadlocks are not possible, while in two phase locking deadlocks are possible.

vii. **false**

True or false

viii. If the database uses shared locks for read operations, then if all transactions are read-only then no deadlocks are possible.

viii. **true**

True or false

ix. SQL Server checks for deadlocks at regular intervals, and if it detects a deadlock then aborts a transaction

ix. **true**

True or false

x. Suppose that the table R has 1000 records. Then the transaction below needs to acquire 1000 locks:

```
begin transaction;
select * from R;
commit;
```

x. **true**

True or false

5 Parallel Data Processing

5. (30 points)

(a) (10 points) We have a large database, on which we need to run repeatedly SQL queries. Each SQL query has up to 5 joins, a group-by, and some selections. We use a parallel database system, and we consider the following alternative evaluation strategies: (a) inter-query parallelism, (b) inter-operator parallelism, (c) intra-operator parallelism. In each case we deploy only one strategy, i.e. we do not combine them. Consider a job J consisting of several SQL queries, and assume it has the following running time:

- Job J runs in $T = 100$ minutes on 10 nodes.

Estimate the running time of that job if we increase the number of nodes from 10 to 100, in each of the six cases below. In each case, assume that the database is capable of delivering linear speedup, when the execution strategy is parallelizable.

Write the running time T on 100 nodes			
Job J consists of:	Type of Parallelism		
	Inter-query	Inter-operator	Intra-operator
1 SQL query			
1000 SQL queries			

Solution:

Job J consists of:	Type of Parallelism		
	Inter-query	Inter-operator	Intra-operator
1 SQL query	100	100	10
1000 SQL queries	10	100	10

(b) (10 points) The query below computes the total number of customers with any given date of birth:

```
select birthdate, count(*)
from Customer x
group by x.birthdate
```

The attributes `birthdate` represents the date of birth of the customer: it contains the day and month only (not the year!). We evaluate the query using Map-Reduce. Assume:

- The relation `Customer` has 16MB

We consider choosing the block size to be one of the following three values: 128KB, 64KB, 32KB. (Recall that $1MB = 1024KB$; thus, if the block size is 128KB, then the `Customer` file has $16 \cdot 1024/128 = 128$ blocks.) Indicate the maximum number of instances that you can use and still achieve linear speedup, if the data is uniformly distributed. Assume that the number of map tasks is equal to the number of blocks, and that the number of reduce tasks is set to the number of instances.

i. The block size is 128KB.

Maximum number of instances:

i. $\underline{16MB/128KB} = 128$

ii. The block size is 64KB.

Maximum number of instances:

ii. $\underline{16MB/64KB} = 256$

iii. The block size is 32KB.

Maximum number of instances:

iii. **365: # days/year**

(c) (10 points) A Map/Reduce Job runs on 10 instances, and uses 100 Map Tasks and 50 Reduce Tasks. The input file has 5GB, and the block size is 50K. We assume that the map function produces an output whose size is approximatively equal to that of the input: in other words, the size of the intermediate result is also 5GB.

i. What is the total number of intermediate files to which the mappers write their outputs?

i. $\underline{50 \cdot 100 = 5000}$

Write the number of files:

ii. After a reducer copies, it needs to sort. How large is the file that needs to be sorted? Answer with the expected value.

ii. $\underline{5GB/50 = 100MB}$

Write the size of the file:

iii. At any time, one instance runs only one map task, or only one reduce task. Suppose that a map task takes 1 minute to finish, and a reduce task also takes 1 minute to finish. What is the total running time for the Map/Reduce job?

iii. $\underline{100\frac{1}{10} + 50\frac{1}{10}} = 15$**minutes**

Write the time in minutes:

iv. Continue to assume that each map task and each reduce task takes 1 minute. However, one single map task exhibits a skew, and take 10 minutes to complete instead of 1 minute; all other 99 map tasks still take 1 minute. What is the total running time for the Map/Reduce job?

iv. $\underline{15 + 9 = 24}$ **minutes**

Write the time in minutes: