CSE 344 Final Examination

December 14, 2011, 8:30am - 10:20am

Name: ____________________

Question	Points	Score
1	20	
2	15	
3	10	
4	20	
5	20	
6	10	
7	5	
Total:	100	

- This exam is an open 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 E/R Diagrams and Constraints

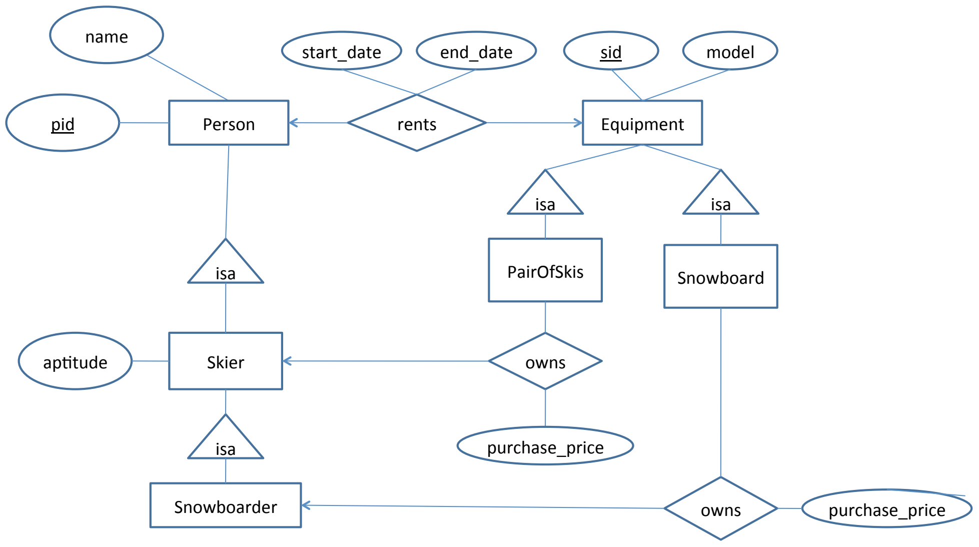
1. (20 points)

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

   - A **Person** has attributes **pid** (key) and **name**.
   - A **Skier** is a type of **Person** with attribute **aptitude**.
   - A **Snowboarder** is a type of **Skier**.
   - A **PairOfSkis** has attribute **sid** (key) and **model**.
   - A **Snowboard** has attribute **sid** (key) and **model**.
   - A Skier **owns** zero or more PairOfSkis. The ownership relation has a **purchase_price**. A PairOfSkis is owned by at most one Skier.
   - A Snowboarder **owns** zero or more Snowboards. The ownership relation has a **purchase_price**. A Snowboard is owned by at most one Snowboarder.
   - a Person can **rent** a PairOfSkis or a Snowboard. A person cannot rent more than one PairOfSkis or one Snowboard at the same time. A person cannot rent a PairOfSkis and a Snowboard at the same time either. A piece of equipment can be rented by at most one person at a time. The rental comes with a **start_date** and an **end_date**.

   Answer (Draw an E/R Diagram):

Solution:



Note: This solution assumes that when a person returns a piece of equipment, the corresponding tuple is deleted from the **rents** relation. If we wanted to keep the historical data, we would change the relationship multiplicity to many-many and we could enforce the constraint that dates cannot overlap using a check constraint or trigger. Additionally, we do not prevent a piece of equipment form being both a PairOfSkis and a Snowboard at the same time. Such a hybrid still counts as one piece of equipment to rent.

(b) (10 points) Write the SQL `CREATE TABLE` statement for the **owns** relation between Skier and PairOfSkis. Make sure that your statement specifies the PRIMARY KEY and any FOREIGN KEYS. Additionally, we would like to enforce the constraint that `purchase_price` be greater than zero.

Answer (Write a CREATE TABLE statement):

Solution:

```
CREATE TABLE owns(skier_id INT REFERENCES Skier,
                  ski_id   INT REFERENCES PairOfSkis,
                  purchase_price INT,
                  PRIMARY KEY(ski_id),
                  CHECK ( purchase_price > 0))
```

2 Conceptual Design

2. (15 points)

   (a) (5 points) Consider the following relational schema and set of functional dependencies. List all superkey(s) for this relation. Which of these superkeys form a key (i.e., a minimal superkey) for this relation? Justify your answer in terms of functional dependencies and closures.

   R(A,B,C,D,E) with functional dependencies AB → E and D → C.

   Answer (Find all the superkeys and keys):

Solution: A superkey is a set of attributes X s.t. X^+ = all attributes.

From the FDs above, we can derive:

$$\{A, B, D\}^+ = \{A, B, C, D\}^+ = \{A, B, D, E\}^+ = \{A, B, C, D, E\}^+ = \{A, B, C, D, E\}$$

Hence,
$\{A, B, D\}, \{A, B, C, D\}, \{A, B, D, E\}$, and $\{A, B, C, D, E\}$ are all superkeys.

A key is a set of attributes which form a superkey and for which no subset is a superkey. In our example, $\{A, B, D\}$ is the only key.

(b) (10 points) 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: Both functional dependencies violate BCNF.
Try $\{A, B\}^+ = \{A, B, E\}$. Decompose into R1($\underline{\text{A,B}}$,E) and R2($\underline{\text{A,B}}$,C,$\underline{\text{D}}$).

For R1, AB $\rightarrow$ E is the only FD and $\{A, B\}$ is a key, so R1 is in BCNF.

R2 is not in BCNF, since $\{D\}$ is not a key and we have D $\rightarrow$ C.

Try $\{D\}^+ = \{C, D\}$. Decompose into R3(C, $\underline{\text{D}}$) and R4($\underline{\text{A, B, D}}$)

End result: R1($\underline{\text{A,B}}$,E), R3(C, $\underline{\text{D}}$), and R4 ($\underline{\text{A, B, D}}$)

3 SQL and Views

3. (10 points)

   (a) (5 points) Discuss **two** scenarios where a view can be helpful in a database management system.

   Answer (Discuss two benefits of views):

   Solution: A view is a relation defined by a query and there are many benefits to views:

   - Views help provide logical data independence. That is, if applications execute queries against views, we can change the conceptual schema of the database without changing the applications. We only need to update the view definitions.
   - Views can help increase physical data independence: We can split base tables into smaller tables horizontally or vertically and hide this data organization behind a view.
   - Views can help secure the data by limiting access to that data: We can create views that only reveal the information users are allowed to see. We can then grant users access to the views instead of the base tables.
   - Views can help encapsulate complex queries.
   - Views can speed-up computations if we *materialize* the content of a view instead of re-computing it whenever the view is used.

(b) (5 points) Explain why it is not always possible to perform SQL UPDATE/DELETE/INSERT statements on top of a view.

Answer (Explain why updating through a view is not always possible):

Solution: Updates on views must be translated into one or more updates on base relations. For some views, the translation can be done in an unambiguous way. In that case, updates are allowed. For example, if a view selects tuples from a relation that satisfy a simple condition, then updates on the view can be translated into updates on the base table in a straightforward fashion.

In other cases, however, there may not be one unambiguous translation. For example, if a view joins two tables together and a user wants to delete a tuple from the view, that delete operation can be translated into deleting one or more tuples from one or both of the underlying tables. Depending on how we would implement the delete operation, additional tuples might also disappear from the view, which would be an undesirable side-effect. For such a view, the only reasonable solution is thus to disallow updates.

When it is possible to perform updates on a view, the view is called *updatable*.

4 Transactions

4. (20 points)

Consider a database consisting of a single relation R:

R:

A	B
1	10
2	0

(a) (10 points) The following two transactions run concurrently on this database:

Line	T1	T2
1	begin transaction;	begin transaction;
2	update R set B = B-10 where A=1;	select sum(B) from R;
3	update R set B = B+10 where A=2;	commit;
4	commit;	

Is it ever possible for T2 to see a value of zero in its output? Explain why or why not.

Answer (Discuss if a schedule where T2 sees a zero value is possible):

Solution: If transaction T2 runs using the READ UNCOMMITTED level of isolation, it may be allowed to read any intermediate value that T1 writes. If it happens that T2 executes its select statement in between the two update statements of T1, it may see the intermediate zero value.

Of course, whether the problem actually arises depends on the DBMS implementation. SQL Server is an example of a DBMS where T2 will read the intermediate, zero value, if it runs using the READ UNCOMMITTED isolation level and executes its select statement in between T1's update statements.

(b) (10 points) The following two transactions run concurrently on this database:

Line	T1	T2
1	begin transaction;	begin transaction;
2	insert into R values (3,150)	select sum(B) from R;
3	commit	select sum(B) from R;
4		commit;

Is it ever possible for T2 to see a different value as the output of the `select sum(B) from R` statements?

<u>**Answer**</u> (Discuss if a schedule where T2 sees different values is possible):

Solution: If transaction T2 runs using either the READ UNCOMMITTED, READ COMMITTED, or even REPEATABLE READ isolation levels, and if it executes the first select statement before T1 runs and executes the second select statement after T1 is done, then it may see a different value in the output.

Again, whether the problem actually arises depends on the DBMS implementation. SQL Server is an example DBMS where the problem will arise.

The reason why the problem arises even with REPEATABLE READ is that T1 is *inserting* a new tuple rather than updating an existing tuple. T2 does not have a lock on that tuple.

5 Parallel Data Processing

5. (20 points)

   (a) (10 points) We have two very large tables $R(A, B)$ and $S(A, C)$ where all attributes are integers. The data in each table is *randomly* distributed across three servers N_1, N_2, and N_3. Explain how a parallel DBMS can compute $R \bowtie_{R.A=S.A} S$ in parallel using all three servers.

   Answer (Explain parallel join):

   Solution: A common implementation of parallel join is through hashing. The algorithm works as follows:

   - On each server N_1 through N_3, the local DBMS instance will read the local data for relation R. It will apply a hash function to the join attribute $R.A$ and it will send the tuple to one of N_1 though N_3 based on the hash value. For example, we could compute $R.A\%3$ and send the tuple to N_1 if the result is 0, N_2 if the result is 1 and N_3 is the result is 2.
   - The local DBMS instance on each server will then perform the same operation on the local data for relation S.
   - Now, all tuples from R and S with the same value of the join attribute are located on the same server. The DBMS instance on each server can thus compute a local join between the two relations.

(b) (10 points) List **two features** that are different between a parallel relational DBMS and the Pig system that you used in HW6.

Solution: Many solutions were possible. We list a few here:

- A parallel DBMS uses the SQL query language while the Pig system uses Pig Latin. SQL is declarative. Pig Latin is a combination of declarative and procedural.
- Pig uses lazy evaluation. It does not evaluate any statements until the user requests to see the data (e.g. using the STORE statement). A DBMS evaluates statements as they are submitted.
- Pig is also implemented on top of MapReduce. It compiles Pig Latin scripts into directed acyclic graphs of MapReduce jobs. Any difference between a parallel DBMS and MapReduce is also a good answer to this question, including the fact that parallel DBMSs stream data from one operator to the next while MapReduce writes intermediate data to disk. When failures occur, MapReduce restarts only the failed operator while a parallel DBMS would restart the entire query. MapReduce also handles stragglers by running redundant tasks toward the end of the computation.
- Other solutions were also possible.

6 DBMS as a Service

6. (10 points)

   (a) (5 points) Today, instead or purchasing physical machines, an alternate approach to running a database management system is to run it in the cloud. One of the key promises of the cloud is the illusion of infinite resources and the ability for users to elastically grow and shrink the resource consumption of their DBMS.

   Why can it be difficult to scale a relational DBMS?

   Answer (Discuss the challenge of scaling a relational DBMS):

   Solution: In a relational DBMS, the most difficult feature to scale are transactions. If a database grows such that it no longer fits on one server but must instead be spread across multiple servers, then a single transaction may update data on multiple machines. In order to provide ACID properties, the transaction must be committed using an expensive two-phase commit protocol that requires multiple rounds of communication between the machines involved.

   If a database fits on one machine but must support a high transaction rate, scalability is also a challenge. One solution is to replicate the database such that the requests can be spread across the replicas. For queries that only read data, this approach works very well. In the case of updates, however, the challenge is in keeping the replicas up-to-date. If we need to propagate each update synchronously to all replicas, updates will be slow. If we return to the user without propagating the update to all replicas, some operations can end-up reading stale values and we also risk losing recent updates in case of failures.

   Finally, parallel queries that read and process a lot of data nicely scale but they also use a lot of resources.

(b) (5 points) Indicate one approach that the so-called "NoSQL" systems take to overcome the above challenge.

<u>**Answer**</u> (Indicate a feature of NoSQL systems designed to facilitate scalability):

Solution: Many solutions were possible including:

- Limiting support for transactions. A transaction can only touch one data item or a small group of data items.
- Providing eventual consistency instead of (or in addition to) strong consistency.
- Limiting query capabilities. Typically, these systems prevent or at least limit join capabilities.

7 Data Integration and Data Cleaning

7. (5 points)

   (a) (5 points) Company A recently bought company B. The two companies want to integrate their databases. Indicate **three** challenges that the integration will face. For each challenge give a concrete example.

   Answer (Indicate three challenges of data integration *with concrete examples*:)

   Solution: Multiple solutions were possible including:

   - DBMS heterogeneity: e.g., one company stores its data in a relational system while the other one uses a semi-structured system.
   - Schema heterogeneity: e.g., in one database the employee information can be stored in one table, while it can be spread across multiple tables in the other database.
   - Data type heterogeneity: e.g., unique employee identifiers can be strings in one database and integers in the other.
   - Value heterogeneity: e.g., the "cashier" position in one database could be called "associate" in the other database.
   - Semantic heterogeneity: e.g., the salary for an employee can be recorded as an hourly before tax value in one database and as a weekly, after-tax value with lunch allowance in the other database.

That's it! Happy Holidays!