W4111-002 Database Systems
Fall 2014
December 16, 2014

Final Exam Solutions. 40% of final grade.
(Closed Book Exam. No calculators or notes permitted.)

This exam consists of 170 points, one per minute of the exam. There are 7 questions, and this exam is 12 pages long. If the percentage grade on this exam is higher than the grade on the midterm, then the midterm grade will be replaced with the final exam grade. Show all work, since partial credit may be given. Questions must be answered in the spaces provided on this sheet; the spaces correspond to the expected length of answers. If you need additional space (due to erasures or large handwriting etc.) use the back of the page and indicate (in the answer slot) that your answer is continued on the back of the page. *We will not read excessively long answers.* Make sure that your name and UNI (or email address) appear below.

Name: _____ UNI/Columbia email: _____

1. (20 points) In at most two sentences each, explain the meaning [1 point] and relevance [1 point] of the following terms as they relate to database systems:

   (a) Checkpoint

   An entry written to the log containing information about the state of the transaction table and dirty page table. **[1 points]** *Used during the analysis phase of the Aries algorithm for crash recovery.* **[1 points]**

   (b) Object-Relational database

   A kind of database that allows the definition of objects using an object-oriented sublanguage and the use of objects within database tables. **[1 points]** *Any relevant fact about object-relational databases, such as the benefits of object orientation (e.g., encapsulation), the benefits of DB integration (query processing, sharing of code) etc.* **[1 points]**

   (c) Views

   A virtual table defined by a query. **[1 points]** *Any relevant information about views, e.g., that updates to views can be problematic, that views are commonly used for information hiding, that a query optimizer expands the view definition to translate a query on the view, etc.* **[1 points]**

   (d) Log

   A persistent data structure used to record information relevant to database crash recovery. **[1 points]** *Any relevant information about the log, e.g., its use in Aries, the order in which items are written, the fact that it contains both before-values and after-values for updates, etc.* **[1 points]**

   (e) Update anomaly

   A situation in a database where a record describing a relationship may be updated, but other records describing the same relationship may not, leading to an inconsistency. **[1 points]** *Any relevant fact about anomalies, e.g., that they are a consequence of redundancy, that they are minimized using normalization, etc.* **[1 points]**

   (f) Assertion

   An integrity constraint specified in SQL outside of a table definition. **[1 points]** *Assertions are needed to model complex constraints that involve multiple tables.* **[1 points]**

   (g) Isolation

   The principle that operations should be isolated from the effects of other uncommitted operations, i.e., that an operation's behavior is unaffected by concurrent work. **[1 points]** *Isolation is a fundamental property of transactions.* **[1 points]**

   (h) Disk block

   For efficiency, disks return a batch of data at a time rather than just a few bytes; this batch is called a disk block. **[1 points]** *Any fact abut disk blocks for databases, e.g., that data structures such as indexes use the disk block size as the node size, that we count I/O (in blocks) as a measure of cost, etc.* **[1 points]**

(i) Deadlock

A situation in a locking protocol where no transaction in a group can make progress because each is waiting on a lock held by another member of the group. [**1 points**] *Any fact about deadlocks, e.g., that one identify then as a cycle of dependencies, that one can break them by aborting a member of the cycle, etc.* [**1 points**]

(j) Atomicity

The principle that an operation should execute in its entirety, or not at all. [**1 points**] *Atomicity is a fundamental property of transactions.* [**1 points**]

2. (15 points) In at most two sentences each, answer the following short questions:

   (a) Suppose a user Mary grants a privilege to another user Joe with the grant option. Joe then grants the permissions to Alice and Bob with the grant option. Mary revokes the grant option (but not the permission itself) from Joe. What changes result from the revoke statement?

   Alice and Bob lose all permissions **[2 points]** *. Joe retains the permission, but not the grant option* **[1 point]** *.*

   (b) Is every scheme that is in Boyce-Codd Normal Form (BCNF) also in Third Normal Form (3NF)? Why or why not?

   Yes **[1 point]** *, because the conditions for BCNF are logically stricter than the conditions for 3NF* **[2 points]** *.*

   (c) Let R and S be tables with n rows and m rows respectively. What is the smallest possible number of rows in $R \bowtie S$? What is the largest possible number of rows in $R \bowtie S$?

   Smallest is 0, largest is mn. **[3 points]**

   (d) Suppose that a commonly-asked query is optimized once, and never re-optimized, to save optimization time. Identify three (or more) kinds of changes that, if they take place after optimization time but before execution time, would impact the choice of an optimal plan.

   One point each, up to 3. *Creation/deletion of an index; substantial changes in table size due to insertion/deletion of rows; changes in resources, such as more memory; changes in hardware, such as a new disk with different performance parameters; changes in selectivity due to frequent updates of rows.*

   (e) What is wrong with the following statement by a database administrator? "Whenever I create a new database, I create indexes on all columns of all tables. That way, I never have to respond to user requests to add new indexes, and the indexes are there to help the optimizer find good query plans."

   The database administrator is ignoring the costs of indexes. **[1 point]** *These include extra space consumption* **[1 point]** *and time to update the indexes when the data is updated.***[1 point]**

3. (20 points) Below are three ER-diagrams. Explain the differences between the three diagrams in terms of the information that can (and cannot) be represented. Explanations should be in plain English, and should not use any technical ER-diagram terminology.

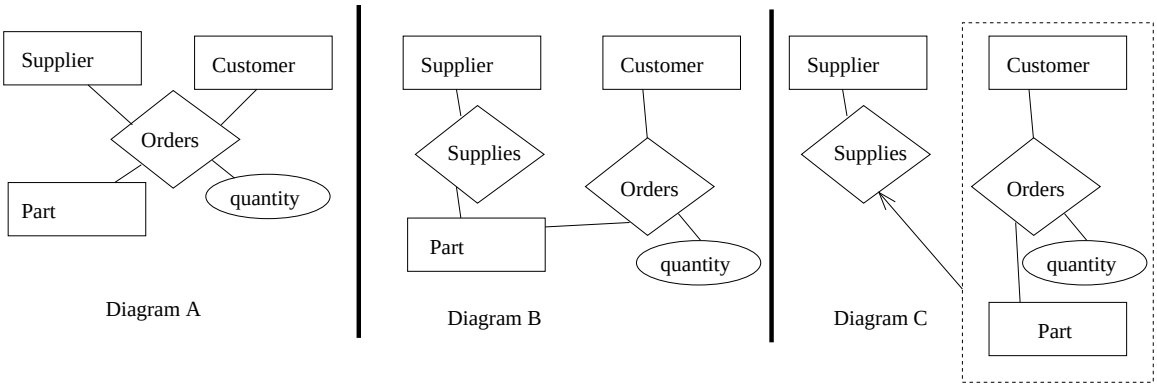

Diagram A associates a triple of supplier, part and customer, so we know which supplier supplies which item to which customer. [**2 points**] *Diagram A also allows multiple different suppliers to supply the same item to the same customer.* [**2 points**] *On the other hand, it is impossible in Diagram A to represent that a supplier supplies a part without a customer for the part, or to represent that a customer orders a part without a supplier for the part.* [**2 points**]

Diagram B cannot represent which supplier is associated with a particular order by a customer. [**2 points**] *It also does not allow a customer to make multiple orders of the same part.* [**2 points**] *On the other hand, it is possible to represent that a supplier supplies a part without a customer for the part, or to represent that a customer orders a part without a supplier for the part.* [**2 points**]

Diagram C associates a supplier to a particular order, i.e., to a combination of a customer and a part. [**2 points**] *In diagram C, it is possible to represent an order by a customer without knowing the supplier.* [**2 points**] *However, it is not possible to represent that a supplier supplies a part without there being a customer.* [**2 points**] *Diagram C requires that for a given customer and part, there is at most one order* [**2 points**] *and one supplier.* [**2 points**]

In diagram A, the quantity attribute represents the quantity per (supplier,customer,part) triple, while in diagrams B and C, quanitity is per (customer,part) pair. [**2 points**]

Note that there are actually 24 points available; if a students gets more than 20, trunctate to 20. If the student overuses technical ER-diagram terminology, subtract 2 points maximum altogether.

4. (20 points) In each of the following subquestions, a decomposition of a scheme $ABCD$ into a subscheme is described. In each case there is a different set of functional dependencies. For each part, identify the answer to each of the following yes/no questions. (For these parts, just a yes or a no is sufficient.)

   - Is the decomposition lossless?
   - Does the decomposition preserve dependencies?
   - Is the resulting scheme in 3NF?
   - Is the resulting scheme in BCNF?

   Finally, in one sentence explain when (if ever) the decomposition is a good choice.

   (a) BCD and AD where $ABC \to D$ and $D \to A$.

   Lossless? Yes
   Preserves dependencies? No
   3NF? Yes
   BCNF? Yes
   Good choice? When? *Most of the time: It achieves BCNF and reduces redundancy/anomalies, which may be more important than being able to check constraints on single tables.*

   (b) BCD and AD where $BC \to D$ and $A \to D$.

   Lossless? No
   Preserves dependencies? Yes
   3NF? Yes
   BCNF? Yes
   Good choice? When? *We never allow a lossy join because it means we can't reconstruct the original table.*

   (c) ABC and CD where $AB \to C$ and $C \to D$.

   Lossless? Yes
   Preserves dependencies? Yes
   3NF? Yes
   BCNF? Yes
   Good choice? When? *This decomposition achieves 3NF and BCNF (reducing redundancy) while also preserving dependencies and having a lossless join, so this decomposition is almost always a good choice.*

   (d) ABC and ABD where $AB \to C$ and $AB \to D$.

   Lossless? Yes
   Preserves dependencies? Yes
   3NF? Yes
   BCNF? Yes
   Good choice? When? *Even though the original scheme is already in BCNF, this overnormalization might be helpful in certain cases, such as when attribute C is queried often while attribute D is updated often (to reduce conflicting lock behavior).*

5. (12 points) Consider a relational database table Projects(ID,State,Budget) that records the identifier and budget of all government projects in each state. You are the DBA of this database system. You may assume that every state always has at least one active project. Several senators have told you that they want to keep track of the current total budget in each state. You are considering two options:

   - Creating a view StateProjectsView defined by the following SQL statement:
     `Select State, sum(Budget) as Total From Projects Group By State`
     Users would then be asked to reference the view if they want to see the total budget.
   - Creating a new table called StateProjectsTable with columns State and Total. You would also create triggers that, on every insertion, deletion, and update to the Projects table, made the corresponding change to that state's total in the StateProjectsTable table. Users would then be asked to reference the StateProjectsTable if they want to see the total budget.

   (a) (6 points) Explain the trade-offs you would need to consider when determining which of the two scenarios is better. For each solution, outline (in one sentence) a scenario when it is the preferred solution.

   If there are many updates of the Projects table relative to queries, then the view approach is better, because there is less overhead for each update. **[3 points]** *If there are few updates of the Projects table relative to queries, then the trigger approach is better because it requires less processing for queries, since the results are already computed.* **[3 points]**

   (b) (2 points) Are triggers executed within the same transaction as the triggering update, or in a separate transaction?

   Same transaction. **[2 points]**

   (c) (4 points) Suppose the database system did the wrong thing, i.e., did the opposite of your answer for part (b) above. Use the example above to explain what could go wrong.

   There are several possible answers. Two examples are given below. Any correct answer gets 4 points.

   - *If there's a database failure between the update transaction and the trigger transaction, there is no guarantee that the trigger transaction is restarted/resubmitted after the database comes back up.*
   - *A query that is scheduled between the two transactions, and reads data from both the Projects table and the StateProjectsTable will see inconsistent data.*

6. (36 points) Consider a hospital that wishes to record and keep up to date records of the patients within the hospital. Suppose that current patient/room information is kept in a table `PatRoom(patient-id,room-id)` where the two columns are unique identifiers for patients and rooms. Sometimes patients change rooms, which means that the room-id for a patient would be updated. Suppose that a patient can be assigned to just one room, and that a room can contain just one patient. There are separate `Patient(patient-id,...)` and `Room(room-id,...)` tables that keep additional information about patients and rooms, respectively. Some patients (e.g., those in the emergency room) may not be assigned to hospital rooms, and some rooms may be empty. The hospital periodically determines its current room occupancy level by running the query `Q`:
`Select count(*) from PatRoom`.

   (a) (6 points) What integrity constraints should be associated with the `PatRoom` table? Write them in SQL.

   Both patient-id and room-id should be unique in PatRoom, but only one can be the primary key. The choice of which to make the primary key is arbitrary. It's ok to have no primary key, and just two unique constraints. It's also ok to slightly misuse the SQL syntax (e.g., omitting parentheses) as long as the intention is clear.

   PRIMARY KEY (patient-id)
   UNIQUE (room-id)
   FOREIGN KEY (patient-id) REFERENCES Patient
   FOREIGN KEY (room-id) REFERENCES Room
   [1.5 points] *per constraint.*

   (b) (6 points) Suppose that a patient with patient-id 123 is moved from room R45 to room R67. Suppose that this change is recorded in the database via two transactions: one to delete (123,R45) from the `PatRoom` table, and another to subsequently insert (123,R67) into the table. Explain what is wrong with this approach.

   The problem here is that an operation that should be atomic in the database is performed non-atomically. **[2 points]** *Any example of why breaking atomicity is a problem in this scenario is worth* **[4 points]** *. Two examples:*

   - *If somebody was to concurrently execute query `Q` that was scheduled between the two transactions, it would report an incorrect number of patients in the hospital.*
   - *The first transaction might succeed while the second fails (e.g., because room R67 is occupied, or due to a crash). This would leave the database in an inconsistent state where patient 123 has no room.*

(c) (6 points) Suppose as before that a patient with patient-id 123 is moved from room R45 to room R67. Suppose that this change is recorded in the database via a single transaction that deletes (123,R45) from the `PatRoom` table, and inserts (123,R67) into the table. Is this sufficient to meet all of the hospital's needs? *Hint: What happens if two patients switch rooms?*

If two patients switch rooms, then doing one of the room-changes in its own transaction won't work, because the room that the patient is being moved to will appear occupied, violating a constraint from part (a). [**6 points**]

(d) (6 points) Suppose that to switch a pair of patients, we use the following single transaction.

```
 i. Delete (P1,R1)
ii. Delete (P2,R2)
iii. Insert (P1,R2)
iv. Insert (P2,R1)
```

Suppose also that at the same time that this transaction is submitted, the query `Q` above is concurrently run to determine the room occupancy levels. Explain what might go wrong (in the absence of any concurrency control) if the whole of query `Q` were executed between steps (ii) and (iii) above in a schedule.

In the absence of concurrency control, query `Q` *will see a state in which patients P1 and P2 are missing from the PatRoom table, and will thus return a number of patients that is off by 2.* [**4 points**] *This is incorrect because the schedule is not serializable, i.e., it is not equivalent to running* `Q` *either before or after the update transaction.* [**2 points**]

(e) (6 points) Suppose that strict two-phase locking were used by the hospital's transaction-processing system. Would the interleaved schedule in part (d) above be possible? If not explain why not. If so, explain why this observation does not violate correctness properties of strict 2-phase locking. Assume that to execute a query, a shared lock on each row needed by the query is obtained.

This schedule would be possible because after the deletion of the two rows, those rows are not needed by `Q` *and hence don't need to be locked.* `Q` *can lock all the remaining rows and complete.* [**3 points**] *The correctness (serializability) of schedules respecting strict 2PL depends on there being a fixed set of items. In this example, the set of items changes. This is an example of the phantom problem.* [**3 points**]

(f) (6 points) If the transaction were instead written as

```
 i. Update PatRoom set room-id=R2 where patient-id=P1
ii. Update PatRoom set room-id=R1 where patient-id=P2
```

and query `Q` was scheduled between the two update statements, then `Q` would appear to give the correct count of rooms occupied in the absence of concurrency control. Explain why such a schedule would not be allowed under strict two-phase locking, and why this is appropriate even though `Q` would give the right answer without locking.

Under strict 2PL, the update transaction will obtain an exclusive lock on the row with patient-id=P1. Query `Q` *cannot complete because it needs a shared lock on all rows of PatRoom, including the row with patient-id=P1.* [**3 points**] *This is appropriate because the database system's concurrency control module does not know how the data is being used, other than that it is being read or written. While this query* `Q` *gives a correct answer, even a slight modification of* `Q` *with the same set of lock requests would not (e.g., change count(*) to count(distinct room-id)).* [**3 points**]

7. (47 points) In a census database the schema has the following tables:

   - Person(person-id,date-of-birth,gender,zip-code,job-code)
   - Regions(zip-code,area)
   - Jobs(job-code,title,education-level)

   The person table records information about every person in the USA, about 300 million rows of data. Regions are determined by their zip-code, and have a total area within the zip code. Jobs are categorized based on job-codes, and have titles and an associated education level. The census database is gathered once every ten years and is not updated once it has been built.

   (a) (8 points) Write the following query in SQL: List all zip-codes in decreasing order of density, where density is the total population within that zip-code divided by the area of the region. You may assume that all zip-codes have a non-zero population.

   ```
   Select P.zip-code, count(*)/R.area as density
   From Person P, Region R
   Where P.zip-code=R.zipcode
   Group by P.zip-code
   Order by count(*)/R.area desc
   ```

   Minor variations are ok, e.g., using the "join ... on" syntax in the from clause. Strictly speaking, the question does not require the density in the select clause, only the zip-code. [**8 points**]

   (b) (8 points) Write the following query in SQL: List all zip-codes that have the following property: for each job-code in the Jobs table, the zip code has one or more people with that job-code. In other words, find zip-codes in which every job is represented.

   There are multiple ways to approach this question, all of which use some kind of nesting. Here is one solution, but any correct answer should get full credit.

   ```
   Select R.zip-code
   From Region R
   Where (Select count(distinct job-code)
         From Person P
         Where P.zip-code=R.zip-code)
         = (Select count(distinct job-code) From Jobs)
   ```

   [**8 points**]

(c) (4 points) Suppose that a B+-tree index is created on the date-of-birth attribute of the Person table. If a record-identifier takes 8 bytes, a date-of-birth takes 4 bytes, and a disk block is 1000 bytes, how many leaf nodes would you expect given that each node is 2/3 full?

There are $3 \times 10^8 \times 12$ *bytes worth of entries in the leaves. Each leaf is a disk block. Since disk blocks are 2/3 full, the answer is* $3 \times 10^8 \times 12 \times \frac{3}{2} \div 1000 = 5.4 \times 10^6$. **[4 points]**

(d) (3 points) If the branching factor in this B+-tree was about 50, how many levels would the B+-tree have? Make sure to count the root and leaf levels in your answer. Round to the nearest integer.

The root level has one node. The next level has 50 nodes. The next has 50^2*, etc.* $50^4 \approx 6 \times 10^6$ *is slightly larger than the number of leaves in part (c). So the total number of levels is 5, counting the root.* **[3 points]**

(e) (8 points) Suppose we are interested in answering a query of the form

```
Select gender, count(*)
From Person
Where date-of-birth < C
Group By gender
```

where `C` is some threshold supplied by the user. Under what circumstances would the index on date-of-birth be useful in answering this query? Explain how the database system would decide whether to use the index. (No other indexes are available.)

The query needs to access both gender and date-of-birth. There are two alternative plans: (a) scan the main file, or (b) use an index on date-of-birth to get RIDs of main file records that meet the date condition, then look up the main file for those records. **[2 points]** *A database system would decide whether to use the index based on evaluating a cost function for each plan and selecting the plan with lowest cost.* **[2 points]** *In this example, the cost depends on the selectivity of the condition. The database might keep track of a histogram of the domain in order to estimate selectivities.* **[2 points]** *If a particular* `C` *is very selective, then the index plan is good because it will touch just a few main file pages. On the other hand, if the condition is not selective, then the scan is probably better because the index plan will do a lot of (potentially repeted) random I/O to the main file compared with the scan.* **[2 points]**

(f) (6 points) Suppose we are interested in a slightly different query of the form

```
Select count(*)
From Person
Where date-of-birth < C
```

where `C` is some threshold supplied by the user. Under what circumstances would the index on date-of-birth be useful in answering this query?

Unlike the previous case, the index now contains enough information to answer the query without consulting the main file; we can therefore use an index-only plan that scans the leaves until a value greater than `C` *is reached, counting the entries along the way. This plan will always do less I/O then either of the plans mentioned in part (e), so the optimizer will always choose a plan for which the index is useful.* **[6 points]**

(g) (10 points) Consider the following relational algebra expression (attribute names are replaced by their first letter):

$$\pi_t \sigma_{e>12 \ AND \ d<01-01-1960 \ AND \ \text{Person.j=Jobs.j}} \left(\text{Person} \times \text{Jobs}\right)$$

Use equivalences of relational algebra to rewrite this expression into an equivalent expression that is more efficient: (a) intermediate results should contain as few rows and columns as possible, and (b) cross-products should be avoided.

First we push selections so that they are applied as early as possible, and so that cross-products can become joins.

$$\pi_t \left(\sigma_{d<01-01-1960} Person \bowtie \sigma_{e>12} Jobs\right)$$

Then we push projections so that the join operates on narrower records. We need to keep j for the join, and t for the query result.

$$\pi_t \left(\pi_j \sigma_{d<01-01-1960} Person \bowtie \pi_{t,j} \sigma_{e>12} Jobs\right)$$

[10 points]