University of Illinois at Urbana-Champaign
Final Examination

CS411 Database Management Systems

Time Limit: 180 minutes
Exam Date: May. 8, 2008

- Closed notes; closed book; no sheet of formulas permitted.
- **Please write your NetID on the upper left hand side of each page.**
- Please write your answers directly on the exam sheet. The space we left for your answers is often more than what you actually need. Please use the back side of the exam as scratch paper.
- In case you find a question ambiguous, please write down your assumption and answer the question accordingly.
- **Answers to queries should not contain duplicates, for any query on this exam. Do not forget to remove duplicates!**
- You may use temporary relations, if you like, for any of the queries below. If you use the same temporary relation for a second exam question, you must redefine the relation in your answer to the second question. In other words, your answer to each question should be self-contained.

Your Name:

Good Luck!

I. [*True/False questions, 40 points*] For each of the following statements, indicate whether it is *TRUE* or *FALSE* by circling your choice. You will get *1 point* for each correct answer, *-1 point* for each incorrect answer, and *0 point* for each answer left blank.

1. *True* **False**
   According to the dependency theory, it is possible for a relation not to have a key.

2. **True** *False*
   Under set semantics, the final result size of a series of joins is the same no matter what order you do the joins in.

3. *True* **False**
   According to dependency theory, it is always bad if a relation has more than one key.

4. *True* **False**
   You should be always able to come up with a lossless, dependency preserving, BCNF version of your application's schema.

5. *True* **False**
   According to dependency theory, it is bad if the only key for a relation is all of its attributes together.

6. **True** *False*
   If your schema is in 3NF but not in 4NF, then you probably need to revise it.

7. **True** *False*
   If a decomposition is not lossless, then you should absolutely not use it for your application schema.

8. *True* **False**
   If a decomposition does not preserve dependencies, then you should absolutely not use it for your application schema.

9. *True* **False**
   One weakness of triggers is that they can only be activated *after* an update/insert/delete, not before.

10. **True** *False*
    Hybrid hash joins are a refinement of hash joins that makes them go faster, but depends on there being enough memory available.

11. **True** *False*
    One characteristic of nested-loop joins is that you can always use them, even if you have very little memory and the data is not sorted.

12. *True* **False**
    For a read of a database page, typically the longest component is the transfer time for the page.

13. **True** *False*
    You can find the best join order for a sequence of joins using a dynamic programming algorithm.

14. **True** *False*
    Generally speaking, a query plan that generates few intermediate result tuples is preferable to one that generates a lot of intermediate result tuples.

15. *True* **False**
    Given a choice between materializing the result of a join and pipelining it to the next operation, in general you should materialize it if you can.

16. **True** *False*
Generally speaking, it is a good idea to push a selection down past a join operation, if you can.

17. *True* **False**
To optimize a query whose selection clause mentions a view, you'll get the best result if you optimize the view definition and the query separately, and then paste the two of them together.

18. **True** *False*
If you have a join followed by a projection, you can always have the result of the join pipelined to the projection (i.e., the intermediate result does not have to be materialized).

19. **True** *False*
Query optimization is particularly important for queries with very long selection clauses (e.g., involving a lot of view definitions).

20. *True* **False**
If your database is not accessible from the Web, it cannot be successfully attacked by a SQL injection attack.

21. *True* **False**
As long as the database is behind a firewall and is not accessible from the Web, we don't need to worry about it being attacked by cross-site request forgery attacks.

22. *True* **False**
If we only allow serial executions of transactions, then the ACID properties are guaranteed.

23. *True* **False**
You can avoid the phantom problem by using a concurrency control and recovery paradigm that prevents cascading rollback.

24. *True* **False**
Transactions cannot deadlock due to contention for locks, if we use strict two phase locking.

25. **True** *False*
Every transaction that is strict two phase locked is also two phase locked.

26. *True* **False**
Lock mode "shared" conflicts with lock mode "shared".

27. **True** *False*
Typical commercial DBMSes have many more than two lock modes.

28. **True** *False*
One disadvantage of cascading rollback is that it wastes time and effort.

29. **True** *False*
Commercial DBMSes prefer to use a STEAL/NO FORCE paradigm for buffer management.

30. *True* **False**
The weakness of redo logging is that you have to force all the dirty pages of a transaction to disk before you can write out its "COMMIT" record to the log.

31. *True* **False**
Undo logging is not as good as undo/redo logging because you can't steal dirty buffer pages from uncommitted transactions if you use undo logging.

32. *True* **False**
The primary reason that commercial DBMSs tend to use undo/redo logging is that recovery after a crash is faster.

33. *True* **False**
The bad thing about a non-quiescent checkpoint is that it takes much longer to finish than a quiescent checkpoint does.

34. *True* **False**
The bad thing about a quiescent checkpoint is that recovery using it is much slower than it would be if we had used a non-quiescent checkpoint.

35. **True** *False*
If you complete a quiescent checkpoint and the system crashes afterward, you will never need to read the part of the log before the checkpoint wrote "START CHECKPOINT", no matter what kind of logging you use.

36. *True* **False**
To guard against the damage that can be caused by a disk crash, you can keep a log of transaction starts, writes, and commit operations.

37. **True** *False*
One disadvantage of allowing dirty reads is that you might get cascading rollback.

38. **True** *False*
If a transaction is about to commit and a schema-level integrity constraint is violated, then the transaction will be aborted.

39. **True** *False*
Under most logging and recovery paradigms, a transaction T is not committed until the "T COMMITS" log record is written out to nonvolatile storage.

40. **True** *False*
Once a transaction commits, it will not be undone, even if a crash occurs very soon.

II. [Short Answer Questions, 60 points, each 5 points].

1. What is the primary reason for taking checkpoints?
   Having checkpoints reduces the recovery time.

2. How does the DBMS typically get rid of duplicates when answering a SQL "SELECT DISTINCT" query?
   They do it through both sorting and hashing relations.

3. What does the D in ACID stand for?
   It stands for durability.

4. Suppose you are writing a new web database application. What is the best approach to take to protect yourself against SQL injection attacks?
   The best approach is to monitor and parse the web requests for suspicious patterns before passing it to the database.

5. What is the main reason that you (the DBA) would choose a B-tree index for an attribute rather than a hashed index (with linear or extensible hashing)? If you need to make an assumption to answer the question, then state your assumption.
   B-tree indices provide better response time for range queries. They also give better average and worst case performance when large number of tuples are inserted into the table periodically.

6. Give a specific example of when you might be willing not to enforce serializability for a particular transaction.
   There is no need to enforce seriazability for transactions that just collect some statistics from database.

7. Why do most DBMSs allow transactions to lock a page (as opposed to just individual tuples)?
   They lock a page in an attempt to prevent other transactions to re-arrange the records in the page.

Also, if most of the tuples that a transaction is dealing with reside in the same page, locking the entire page improves the performance.

8. What is the typical method used to point to a tuple on page (e.g., to point to it from a B-tree)? Hint: *not* the disk address of the tuple.
   The typical method is to use page number plus slot number.

9. Why don't we point to a tuple (e.g., from the leaf of a B-tree) by using its disk address?
   If the disk address is used, DBMS has to update the B-tree every time when records get relocated in disk. This degrades the performance considerably.

10. Suppose you are going to have just one index on relation Orders(OrderNumber, CustomerID, Date): a B-tree on OrderNumber. If order numbers are assigned sequentially in order of the arrival of orders (1, 2, 3, ...), is it better to use a dense or sparse index? Very briefly, why?
    Since the values of OrderNumber are arriving sorted, a sparse index is the best choice.

11. What if the only index on Orders is going to be a B-tree on CustomerID—is it better to use a dense or sparse index? Very briefly, why?
    As we are not sure if the values of CustomerID are stored sorted on the table, we have to use dense index.

12. Describe a situation where you would pick a two-pass merge join over a two-pass hash join.
    If the tables are already sorted, a two-pass merge join operation will be more efficient than a two-pass hash join operation.

III. Long Answer Questions, 170 points 1. Normalization, 20 points

Consider relation *Course(Number, Name, Area, Faculty)* where each course is taught by just one faculty member all the time, each course has a unique number, and no two different courses have the same combination of name and area. Provide a 3NF version of this relation and determine if your answer is in BCNF and why it is or it is not in BCNF.
Available functional dependencies are:

- $Number \rightarrow Name$; $Number \rightarrow Area$; $Number \rightarrow Faculty$
- $Name, Area \rightarrow Number$; $Name, Area \rightarrow Faculty$

Therefore, the table already is in 3NF. It is also in BCNF because there is not any other functional dependencies in the tables except the ones involving super keys in their left hand side. Any other solution based on reasonable assumptions is acceptable.

2. Security, 30 points, each part 15 points

(a) Show how to carry out a cross-site request forgery attack. You need to give enough detail in your example to convince us that you know what you are talking about.
An employee with the power to fire people from the company requests document *http://attacksite.org/cuteKittensScreensaver.html* and receives:

```
HTTP/1.1 200 OK Content?Length: 121
<html>
<imgsrc=http://cutecats.com/cat_of_the_week.gif>
<imgsrc="http://mycompany.org/admin/terminate_employee.php?employee_id=WilliamGates">
</html>
```

When the employee's browser requests the second image, the employee's browser will cause William Gates to be fired:

```
GET /admin/terminate_employee.php?employee_id=WilliamGates
HTTP/1.1
Host: mycompany.org
Cookie: PHPSESSID=123456789
```

Many people gave a CSS attack or some other legitimate non-CSRF attack as their solution, for which they received half credit.

(b) What is the best way to guard against such attacks, if you are writing a new database application?
A database can be successfully attacked by an external CSRF attack even if the database lives behind a firewall and the applications that access it are only available inside the company/organization – in other words, the database itself is not accessible from outside the company. For this reason, current guidelines for protecting against CSRF attacks recommend that (1) all DB applications that can be invoked with a GET/POST request (or indirectly as a side effect of such a request) be set up to require manual input from the user before any drastic action is carried out. For example, to counteract the example attack given above, a pop-up window might say, "Please click here to confirm that you want to fire employee William Gates." Current guidelines also recommend that (2) on receipt of a request, the DB application code should check to make sure that the DB request is made with POST rather than with GET. Also, the DB application itself should be set up to use POST rather than GET. A third recommendation is that (3) each DB application front-end form (e.g., the interface that is used to fire employees) sent to a user include a fresh token, and have this token be included when the form is submitted. If an appropriate token is not included with the request, or if the token is no longer fresh (i.e., has timed out), then the user should be sent an error message and the request should be ignored. Many people gave a remedy for some other legitimate non-CSRF attack as their solution (regardless of how they answered part (a)), for which they received half credit.

3. Query Optimization, 40 points
Consider the following relations:

```
Movie(Title, Year, Rating, StudioName)
Studio(Name, Country, Address)
```

Assume each movie is produced by just one studio, whose name is mentioned in the StudioName attribute of the Movie relation. Also, Title and Name are primary keys for Movie and Studio, respectively. The rating of a movie shows how good the movie is, and its range is [1,...,10]. The following statistics are available about the relations:

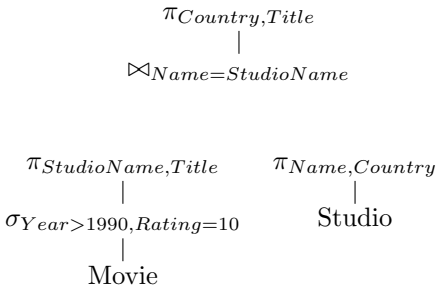
Movie	Studio
T(Movie) = 24000 V(Movie,StudioName)=800 V(Movie,Rating)=10	T (Studio)= 1000

The following query returns the movies with a rating of 10 produced in each country after 1990:

```
SELECT Country, Title
FROM Movie, Studio
WHERE Movie.StudioName = Studio.Name and Year > 1990 and Rating = 10
```

Suggest an optimized logical query plan for the above query. Then, estimate the size of each intermediate relation in your query plan. By intermediate relation we mean the relation created after each selection or join.

The optimized logical query plan for the above query is:



There are tow relations created after the selection and the join. For the relation created after the selection we have:
$N_1 = \frac{24000}{10*3} = 800$
and for the one after the join:
$N_2 = \frac{800*1000}{1000} = 800$

4. Concurrency control and recovery, 80 points each part 10 points
Consider the following schedule where, for clarity, we have included all the operations that will appear in the log, plus additional helpful detail.

Operation	———- Column 2 ———-
T1 STARTS	
T1 reads item B	
T1 writes item B with old value 11, new value 12	
T2 STARTS	
T2 reads item B	
T2 writes item B with old value 12, new value 13	
T3 STARTS	
T3 reads item A	
T3 writes item A with old value 29, new value 30	
T2 reads item A	
T2 writes item A with old value 30, new value 31	
T2 COMMITS	
T1 reads item D	
T1 writes item D with old value 44, new value 45 ***	
T3 COMMITS	
T1 COMMITS	

(a) What serial schedule is this equivalent to? If none, then explain why.
The serializability graph for the above schedule is: $T1 \longrightarrow T2 \longleftarrow T3$. Any order that complies with the topological order of the graph like $T1 \longrightarrow T3 \longrightarrow T2$ is an equivalent serial schedule for our schedule

(b) Is this schedule consistent with two phase locking? If your answer is yes, then in column 2, insert into the schedule a minimal set of additional operations that will make the schedule no longer consistent with two phase locking. Do not introduce any new transactions. Make sure to show exactly where your new operations should be inserted in the original schedule. If your answer is no, then in column 2, remove from the schedule a minimal set of operations, so that the revised schedule is consistent with two phase locking.

If we assume that all transactions get the locks exactly before the operation and release them afterwards, it is not consistent with two phase locking. This is because T1 releases its lock on B after its second operation while acquiring a lock on D at its last two operations. By removing the last two operations of T1 the schedule becomes 2PL.

If we assume that the transactions get all the locks they need at the beginning of the transaction, and release them after the finish the operation, this schedule will be 2PL. The minimum operations that could be added to the schedule will be "T1 reads item A". In this case, T1 has to acquire the lock on A again after releasing its lock on A after its first write.

(c) Consider the version of the schedule that is consistent with two phase locking (either the original one or your revised version, depending on your answer to the previous part). Is that version consistent with strict two phase locking? Why or why not?
It is not because transactions do not release their locks after they are committed.

Now consider an undo log corresponding to the original schedule. Suppose that a crash occurs right after the log record with asterisks next to it is written to disk, and the recovery procedure is then run.

(d) What transactions get rolled back?
Transactions T3 and T1 get rolled back.

(e) Does cascading rollback take place? Why or why not?
As transaction T2 reads a value from T1, it gets rolled back as well.

(f) Which operations in the log get undone?
All the write operations by transactions T1, T2, and T3 will be undone.

(g) At the time of the crash, which of the five data write operations are guaranteed to have taken place on disk?
All the write operations by T2 are guaranteed to have taken place on disk.

(h) At the time of the crash, which of the five data write operations are guaranteed to **not** have taken place on disk?
None. All of the write operation could have already taken place on the disk.