Department of Electrical Engineering and Computer Science

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

6.830 Database Systems: Fall 2015 Quiz I

There are 12 questions and 13 pages in this quiz booklet. To receive credit for a question, answer it according to the instructions given. *You can receive partial credit on questions.* You have **80 minutes** to answer the questions.

Write your name on this cover sheet AND at the bottom of each page of this booklet.

Some questions may be harder than others. Attack them in the order that allows you to make the most progress. If you find a question ambiguous, be sure to write down any assumptions you make. Be neat. If we can't understand your answer, we can't give you credit!

THIS IS AN OPEN BOOK, OPEN NOTES QUIZ.
LAPTOPS MAY BE USED; NO PHONES OR INTERNET ALLOWED.

Do not write in the boxes below

1-3 (xx/37)	**4-6 (xx/19)**	**7-9 (xx/26)**	**10-12 (xx/18)**	**Total (xx/100)**

Name:

I Column Stores

1. [12 points]: Using a column-oriented physical database design makes some types of queries faster. Suppose you have a database table with 100 columns. Which types of queries would you expect to go significantly faster when using a column-oriented representation, assuming compression is NOT used?
(Circle "Faster" or "Not Faster" for each choice below.)

(A) Faster Not Faster Queries that read 2 or 3 columns of one record.

"Not faster". Reads of single records are unlikely to be faster as they will have to perform multiple I/Os to retrieve a record. It's possible that the savings of not reading the 97 or 98 other columns will offset this, but given the difference in sequential vs random I/O it's very unlikely the columns store will be substantially faster.

(B) Faster Not Faster Queries that read 2 or 3 columns of many records.

"Faster". Column stores make reads of large numbers of few columns faster by saving large amounts of sequential I/O on unneeded columns.

(C) Faster Not Faster Queries that update many records of 2 or 3 columns.

The intended answer was "not faster", since updates in general are going to be slower in a column store (due to multiple writes), but because of the awkward phrasing we gave credit for either answer.

(D) Faster Not Faster Queries that read all columns of many records.

"Not faster". Column stores don't offer any significant benefits when reading all data.

II SQL

Consider the following pairs of SQL queries. Your job is to figure out which of them are semantically equivalent (i.e., compute the same answer for all possible inputs).

2. [15 points]: For each pair, indicate if they are equivalent. If not, briefly describe a situation or example in which they would produce different answers.

(Circle "Equivalent" or "Not Equivalent" for each pair.)

Query 1

```
SELECT x,y,z
FROM  A,
  (SELECT MAX(B.b) AS m
   FROM B)
as C
WHERE A.a = C.m
```

Query 2

```
SELECT x,y,z
FROM A
WHERE A.a =
  (SELECT MAX(B.b)
   FROM A,B
   WHERE A.a = B.b)
```

(A) Equivalent Not Equivalent

If "Not Equivalent", briefly describe a case in which their output differs:

Not equivalent. Consider the following tables:

A

a	x	y	z
1000	1	1	1

B

b
1000
10000

Query 1

```
SELECT x,y,z FROM
A,  (SELECT B.b FROM B
     WHERE B.c > 100) as X
WHERE A.a = X.b
```

Query 2

```
SELECT x,y,z
FROM A,B
WHERE A.a = B.b
AND B.c > 100
```

(B) Equivalent Not Equivalent

If "Not Equivalent", briefly describe a case in which their output differs:

Equivalent. We accepted the answer "not equivalent" when accompanied by an explanation that the x,y,z columns could be a part of table B, in which case query 1 would produce an error.

Name:

Query 1

```
SELECT COUNT(*)
FROM A
WHERE A.a IN
  (SELECT x FROM B
   WHERE B.c = A.b)
```

Query 2

```
SELECT COUNT(*)
FROM A,B
WHERE A.a = B.x
AND B.c = A.b
```

(C) Equivalent Not Equivalent

If "Not Equivalent", briefly describe a case in which their output differs:

Not Equivalent. Consider the following example:

A

a	b
1	1

B

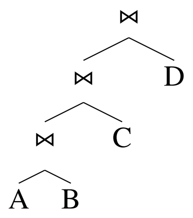
x	c
1	1
1	1

Here query 2 produces 2 results and query 1 produces 1 result. Note that query 1 is a correlated subquery, which means that the inner query is evaluated once per record in A, supplying the A.b value and then checking to see if the corresponding A.a value satisfies the 'IN' clause.

Name:

III Join Ordering

Consider the Selinger optimizer as described in the paper "Access Path Selection in a Relational Database Management System". It chooses to join four tables, A, B, C, and D in the order (((AB)C)D). Here the parentheses indicate the order of joins, as discussed in class: the bottom-most join is A-B, followed by a join with C, followed by a join with D. Graphically, this plan looks like this:



None of tables fit into memory and there are no indexes on any of the tables.

3. [10 points]: Based on what you know about how the Selinger algorithm works, its choice of optimal join ordering, and the fact that the optimizer only considers left-deep plans, which of the following statements must be true?

(Circle 'T' or 'F' for each choice.)

T F The cost estimate of the join ((AB)C) is less than or equal to the cost estimate of (A(BC))

False. (A(BC)) is non-left deep so we don't know what its cost estimate is because it wasn't considered by the optimizer.

T F The cost estimate of the join (AB) is less than or equal to the cost estimate of the join (BA)

True. As discussed in class, these plans are different (different inner and outer), and AB must be less than or equal in cost to BA if AB if BA was chosen in the plan.

T F The cost estimate of the join (((AB)C)D) is less than the cost estimate of the join (((AB)D)C)

False. D might be a cross product with (AB), so wasn't costed by the optimizer.

T F The cost estimate of the join (CD) is less than the cost estimate of the join (AB)

False. The cost of (CD) could be lower than that of (AB); it may not have been chosen as the first join because the cost of joining (CD) to (A) or (B) could be high. For example, (AB) might produce data in a beneficial sort order.

T F The optimizer will not choose nested loops for any of the joins

False. Even though none of the tables fit in memory, it's possible that some of the joins produce very few results, which could make them good candidates for nested loops joins, especially if indexes are present.

Name:

IV Join Algorithms

Consider a database system with the ability to perform both grace and in-memory hash joins, as well as nested loops and index-nested loops joins (but not sort-merge joins of any type).

Suppose you are joining three tables, A, B, and C:

```
SELECT x,y,z
FROM A,B,C
WHERE
A.a > B.b AND
B.d = C.e
AND B.f = 'x'
```

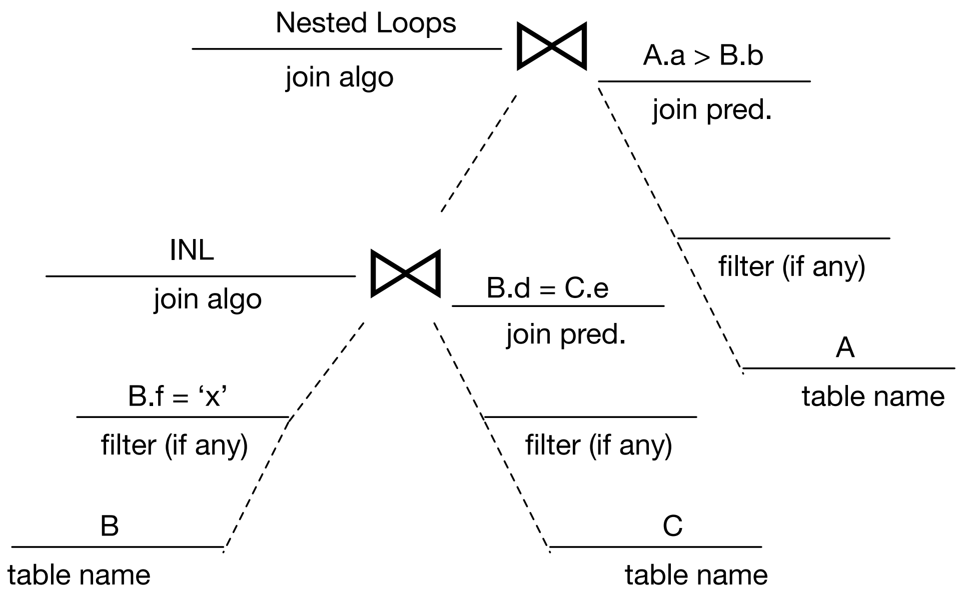
Here `C.e` and `B.f` are primary keys.

Suppose you have 10 MB of RAM, that each table is 100 MB and 1 million tuples, and that each attribute is uniformly distributed. Suppose disk seeks take 10 ms, the disk can read 100 MB/sec, and that each predicate evaluation, hash, or has lookup takes 1 microsecond. Suppose there are unclustered indexes on B.f and C.e, and no other indexes.

For simplicity, you can assume that all B+Trees are 4 levels deep, and that no pages are cached in the buffer pool (i.e., the memory is used only for buffering in the joins, and for intermediate tuples flowing through the query plan.)

Name:

4. [8 points]: In the query plan template below, indicate the join ordering you recommend as well as the type of join you would choose.



Performing the B-C join first is a good choice, because the predicate on B.f produces only one tuple, C.e is a key of C, so the join produces only one result. An index nested loops join is best because in both cases a sequential scan can be avoided, resulting in an single index lookup on both tables. The join with A needs to be a nested loops join because hashing doesn't work for range predicates. Since there is only one tuple from the B-C join this results in a single sequential pass over A.

5. [5 points]: Estimate the runtime of the plan you drew, to the nearest tenth of a second.

- Each index lookup takes 4 index seeks, plus on heap file seek = 50 ms x 2 = 100 ms
- The sequential scan of A takes 1 second (100 MB read at 100 MB/sec)
- The nested loops join does 1 million comparisons, at a cost of 1 million microseconds, or 1 second
- The other CPU costs (e.g., to do the B-C join) are much less than a tenth of a second
- Total cost: 2.1 seconds

6. [6 points]: Ben BitDiddle recommends clustering the B heap file on the B.f attribute. Will this improve the performance of your query? Why or why not?

No. Because B.f is a primary key, we will only ever fetch one record from the B table. Clustering only helps when multiple (consecutive in key space) records are being retrieved from an index.

Name:

V Entity and Schema Design

You are designing the database schema for an online e-commerce platform. Your database needs to store the following information:

- For each user, their name, id, date of birth and address.
- For each item, its id, description, name, price.
- For each seller, their name, id, and country. A seller may sell multiple items. Each item id is sold by exactly one seller.
- For every order that is placed by a user, store the items in the order, the user who placed the order, quantity of each item, and order priority. An order may have multiple items, and an item may be a part of multiple orders.

Name:

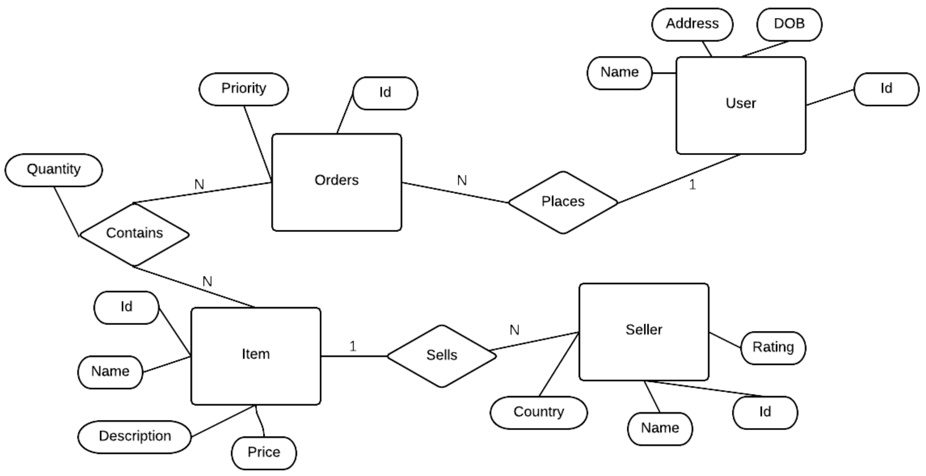
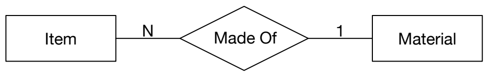


Figure 1: ER diagram for solution. Ignore the `rating` attribute.

7. [10 points]: Draw an entity relationship diagram for this database. Please draw entities as squares, attributes as ovals, and denote relationships as diamonds between pairs of entities. Label each edge with a "1" or an "N" to indicate whether the entity on the other side of the relationship connects to 1 or N entities on this side of the relationship. For example, the following would indicate that each item is made of a single material (e.g. wood, metal), and multiple items can be made of the same material:



Give each entity, relationship, and attribute a name.

(Draw your diagram in the space below)

.

Name:

8. [10 points]: Use your ER diagram to determine a relational schema for this database. For each table, use the form:

TablenameN (field1-name, ..., fieldn-name)

to denote its schema. If you wish, you can create an additional field for each table to serve as a unique identifier. Underline the primary keys and use the "... references ..." syntax for foreign keys in your schema.

(Write your answer in the space below.)

```
user(uid, country, name, dob)
seller(sid, name, country)
item(iid, description, name, price, sid references(seller_sid))
order(oid, priority, uid references(user_uid))
item_order(oid references(order_oid), iid references(item_iid), quantity)
```

We allowed the ER diagram to omit the id attributes, as long as the schema had them. We also awarded partial credit to schemas (and diagrams) that attempted to unify seller and user into one single table depending on how they did it.

9. [6 points]: Since every item is supplied by exactly one seller, Ben BitDiddle recommends that you store seller information along with item information; i.e., each item tuple, in addition to information about the item, should include information about its seller.

List three reasons why this schema is not a good choice.

1. There are multiple copies of the same seller information in the table and potentially, if there is still a seller table (the question was ambiguous about it), yet another copy.

2. Related to the first point, updating any seller information would involve more work and would involve updating every item sold by that seller.

3. If the seller table no longer exists, then there would be problems with seller data for sellers that currently have no items listed.

We also awarded partial credit to answers that mentioned BCNF with some explanation of why that is desirable.

Name:

VI Query Planning

The following table in Postgres fits in memory.

```
postgres@test# \d test.tab3
       Table "test.tab3"
 Column  |  Type   | Modifiers
---------+---------+-----------
 p_id    | bigint  |
 p_attr1 | integer |
 p_attr2 | integer |
Indexes:
    "idx1" btree (p_id, p_attr1)
```

Consider the following query and two different plans:

```
SELECT b.p_id, a.p_id
FROM test.tab3 AS b, test.tab3 AS a
ORDER BY by b.p_id, a.p_id
```

```
                                  QUERY PLAN 1
-----------------------------------------------------------------------------
 Sort  (cost=10014538047.38..10014788047.38 rows=100000000 width=16)
   Sort Key: b.p_id, a.p_id
   ->  Nested Loop  (cost=0.00..1250335.00 rows=100000000 width=16)
         ->  Seq Scan on tab3 b  (cost=0.00..155.00 rows=10000 width=8)
         ->  Materialize  (cost=0.00..205.00 rows=10000 width=8)
               ->  Seq Scan on tab3 a  (cost=0.00..155.00 rows=10000 width=8)
```

```
                                        QUERY PLAN 2
---------------------------------------------------------------------------------------------
 Sort  (cost=10014538623.70..10014788623.70 rows=100000000 width=16)
   Sort Key: b.p_id, a.p_id
   ->  Nested Loop  (cost=0.57..1250911.32 rows=100000000 width=16)
         ->  Index Only Scan using idx1 on tab3 b  (cost=0.29..443.16 rows=10000 width=8)
         ->  Materialize  (cost=0.29..493.16 rows=10000 width=8)
               ->  Index Only Scan using idx1 on tab3 a  (cost=0.29..443.16 rows=10000 width=8)
```

10. [6 points]: Which of the following statements about these plans are true? **(Circle 'T' or 'F' for each choice.)**

T F The final sorting step in the first plan is not necessary. False. It is necessary as the heap file cannot assumed sorted

T F The final sorting step in the second plan is not necessary. True. The indices are already sorted by `p_id` , and nested loops will naturally produce an output sorted by `(b.p_id, a.p_id)`

Name:

Ben Bitdiddle is experimenting with different indexes and index only scans, testing out different plans. The table is the same as before, but initially has no indexes on it. He runs the query:

```
SELECT p_id FROM test.tab3
```

He consistently gets the following results after running this query multiple times.

Sequential Scan Plan: First, he runs the query with sequential scans enabled:

```
postgres@test# explain analyze select p_id from test.tab3;

 Seq Scan on tab3
(cost=0.00..1541.00 rows=100000 width=8)
(actual time=0.005..14.462 rows=100000 loops=1)
 Planning time: 0.035 ms
 Execution time: 19.371 ms
```

Index Only Plan 1: Then, he creates an index on `p_id`, `p_attr1`, and `p_attr2`, and disables sequential scans:

```
postgres@test# explain analyze select p_id from test.tab3;

 Index Only Scan using tab3_p_id_p_attr1_p_attr2_idx on tab3
(cost=0.42..3052.42 rows=100000 width=8)
(actual time=0.011..19.501 rows=100000 loops=1)
   Heap Fetches: 0
 Planning time: 0.034 ms
 Execution time: 24.501 ms
(4 rows)
```

Index Only Plan 2: Finally, he creates an index on just `p_id`, and disables sequential scans:

```
postgres@test# explain analyze select p_id from test.tab3;

 Index Only Scan using tab3_p_id_idx on tab3
(cost=0.29..2604.29 rows=100000 width=8)
(actual time=0.013..15.033 rows=100000 loops=1)
   Heap Fetches: 0
 Planning time: 0.035 ms
 Execution time: 19.944 ms
```

11. [4 points]: Give one reason for the performance difference between the two index-only scans.

Name:

12. [8 points]: The second index-only scan is very close in time to the heap scan. Ben expected it to be much faster considering it only holds the values pf `p_id`. Give two reasons why B+Tree performance may have been less than expected.

1. There is more information than just the `p_id` in the index pages. There are at least the pointers to the heap pages and siblings mixed in with them. If these pointers are 8 bytes each, then index only scans must scan through as much data as there is in the heap file (which has 2 other columns).

2. Btree pages are between 50% and 100% full (so that they can take new inserts). This means there are more b-tree leaf pages to look at than there are heap pages to look at. Any per-page processing needed will cost more for the B-tree leaf pages. Additionally, there are jumps between pages. This is not as big a deal once you are in memory when compared to disk, but compact and sequential is always better either in memory or on disk.

Notice that the data was mentioned to be fully in memory at the very start of the problem. Answers that emphasized disk-seeks in the index leave nodes got some partial credit, but keep in mind there were no disks nor disk seeks.

Some answers mentioned the initial tree traversal. Notice that the data was 10k rows, with at least 16B per row, this yields at least 40 4KB heap pages (and more leaf pages), possibly only a 1 level tree depth, so the initial traversal is not a significan cost.