Department of Electrical Engineering and Computer Science

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

6.830 Database Systems: Fall 2015 Quiz I

There are 12 questions and 11 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.
(B)	Faster	Not Faster	Queries that read 2 or 3 columns of many records.
(C)	Faster	Not Faster	Queries that update many records of 2 or 3 columns.
(D)	Faster	Not Faster	Queries that read all columns of many records.

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:

Name:

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:

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
```

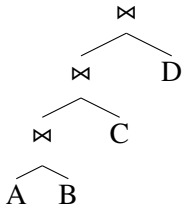
(C) Equivalent Not Equivalent

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

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))

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

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

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

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

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:

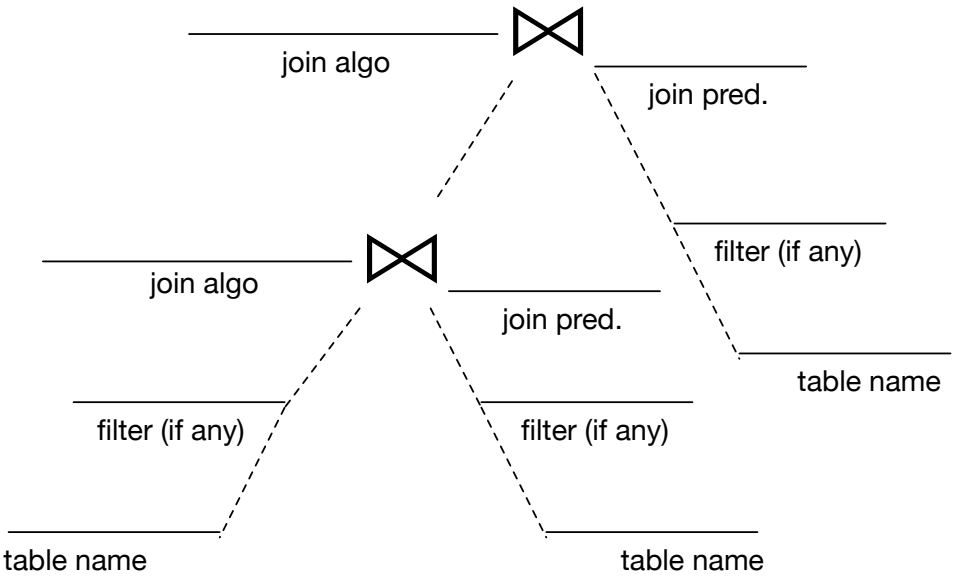
```
SELECT x,y,z
FROM A,B,C
WHERE
A.a > B.b AND
B.d = C.e
AND B.f = 'x'
```

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.)

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.



Name:

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

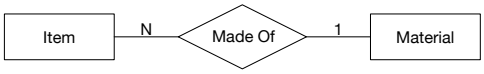
6. [6 points]: Ben BitDiddle recommends clustering the B heap file on the B.f attribute. Will this improve the performance of your query? Why or why not?

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.

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.)

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.

(Write your answer in the space below.)

1.

2.

3.

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.

T F The final sorting step in the second plan is not necessary.

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.

2.

Name: