CSE-4411M Test #2

Sur / Last Name:
Given / First Name:
Student ID:

- **Instructor:** Parke Godfrey
- **Exam Duration:** 75 minutes
- **Term:** Winter 2014

Answer the following questions to the best of your knowledge. Your answers may be brief, but be precise and be careful. The exam is closed-book and closed-notes. Calculators, etc., are fine to use. Write any assumptions you need to make along with your answers, whenever necessary.

There are four major questions, each with parts. Points for each question and sub-question are as indicated. In total, the test is out of 50 points.

If you need additional space for an answer, just indicate clearly where you are continuing.

Marking Box	
1.	/10
2.	/15
3.	/10
4.	/15
Total	/50

1. (10 points) **Access Paths.** *Just what I need.* [Exercise]

a. (4 points) The following information is available on tables **Sailors** and **Reserves**.

- **Reserves**: 50,000 records
  - **Reserves**.bid: 50 distinct values
- **Sailors**: 1000 records
  - **Sailors**.level: 10 distinct values (1..10)

The primary key of **Sailors** is sid; of **Reserves** is sid + bid + day. Table **Reserves** has a foreign key on sid referencing **Sailors** (on sid). All columns are *not null*.

Consider the query

```
select S.sid, R.bid, R.day
    from Sailor S, Reserves R
    where S.sid = R.sid and
          R.bid = 7 and S.level >= 3 and S.level <= 4;
```

Our System-R style optimizer estimated the *cardinality* of the query to be 320 records.

You think the estimation is wrong. Why did our optimizer get it wrong?

What *should* the cardinality estimation be?

$50K \cdot 1/50 \cdot 8/10 \cdot 4/10 = 320$. *It took the three predicates as independent.*
Of course, **S**.*level* >= 3 *and* **S**.*level* <= 4 *are not independent. This is a range with selectivity* 2/10. *So,* $50K \cdot 1/50 \cdot 1/5 = 200$.

b. (4 points) Consider table **R** with attributes A, B, and some others. It has two hash indexes: one with the index search key A; the other with B. Both indexes are unclustered. For both, 100 data entries are on a bucket page, on average. In the table file for the records, there are 10 records per page, on average. **R** contains one million records.

Consider the query

```
select * from R
    where A = 37 and B = 41;
```

The predicates A = 37 and B = 41 each match 1% of the records. Assume data independence of A and B.

You have a buffer pool allocation of 200 pages. Design an access path that evaluates the query in less than 1,000 I/O's.

We can use index intersection.
$A = 37$ *matches 10K data-entries, or 100 data-entry pages. Same for* $B = 41$. *So we find our data-entry pages for* $A = 37$ *and for* $B = 41$ *in 200 I/O's. Sort these by block: read in above; 200 I/O's to write out. Read in and merge: 200 I/O's. Fetch the 100 records matching* $A = 37$ and $B = 41$. *(We can see which to fetch by the data entries.) Total cost: 700 I/O's.*

c. (2 points) Consider a query that joins *four* tables. In level 3 of the join-enumeration algorithm in System R (so, which effectively produces plans that have joined three of the tables), at least how many join operations will the optimizer consider and cost? Explain your answer.

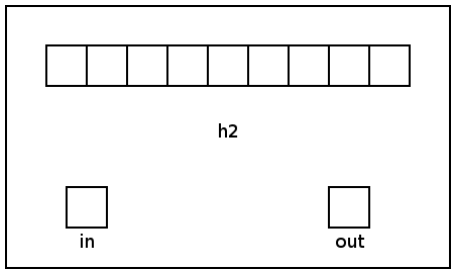
Each plan from level two (which is the join of two things) can be joined with one of the two remaining tables: there are 6 such plans from level 2, and two possible things to join for each. So this is 4 choose *2: 12.*
How many operations? However many algorithms we have available for each of the 12 to consider.

Extra space.

2. (15 points) **Algorithms.** *Er, a sort-index-hash-nested join!* [Analysis]

a. (5 points) Sketch out what the buffer-pool arrangement looks like during the the second pass of a hash join.

What provision is made so that the second pass of the hash join is not CPU-bound?



We read in a partition of the outer and build an internal hashtable for it with h_2. This means we only have to compare a tuple of the inner to one frame of the outer, thereby addressing the CPU-boundedness issue.
We need to reserve a frame to read in the matching partition of the inner page at a time, and a frame to construct the join results.

b. (3 points) Does the basic 2-pass hash join benefit from sequential reads and/or writes? If not, explain why it cannot. If so, describe what part of the algorithm benefits (e.g., writing the partitions for the outer in pass 0, etc.).

Yes. We can write partitions sequentially; no gain there, really. But we gain when we read them in on the second pass by sequential reads.

c. (4 points) Consider a merge join of **R** and **S** on the mutual attribute J. Let **R** be the outer stream and be sorted already on J, K.

Are there any conditions under which the output is *not* sorted on J, K? If so, what are they? And, if so, under which conditions *is* the output sorted on J, K?

We handle the outer as a stream: each tuple is matched to the group of the inner matching. The output of this will be the same J, K. So we then advance to the next outer and so forth.
Yes, it will always be sorted by J, K.

d. (3 points) Say that the outer stream of an *index-nested-loop join* is sorted in some way. That is, the stream has an *interesting order*.

Does the index-nested-loop join *preserve* the interesting order? That is, will the output stream have that same interesting order?

Explain why or why not.

It does. For the same reason as in the answer to Question 2c. We process the outer tuple at a time in its stream.

3. (10 points) **General.** *Jan, ken, pon.* [Multiple Choice]

Choose *one* best answer for each of the following. Each is worth one point. There is no negative penalty for a wrong answer.

In the rare case you feel a clarification to your answer is needed, write a brief clarification on the side.

For Questions 3a–3b, consider table **R** with attributes A and B and table **S** with attributes A and B. All joins below are natural joins.

---

a. Which of the following relational algebra expressions necessarily evaluate to the same result?

I. $\mathbf{R} \cap \mathbf{S}$
II. $\mathbf{R} \bowtie \mathbf{S}$
III. $(\mathbf{R} \bowtie \pi_{\mathsf{A}}(\mathbf{S})) \bowtie (\pi_{\mathsf{A}}(\mathbf{R}) \bowtie \mathbf{S})$

A. All three must evaluate to the same result.
B. They each may evaluate to a different result.
C. **I** & **II**
D. **I** & **III**
E. **II** & **III**
F. Not enough information is provided to determine this.

---

b. Which of the following relational algebra expressions necessarily evaluate to the same result?

I. $\mathbf{R} \cap \mathbf{S}$
II. $\mathbf{R} \bowtie \mathbf{S}$
III. $(\mathbf{R} \bowtie \pi_{\mathsf{A}}(\mathbf{S})) \bowtie (\mathbf{R} \bowtie \pi_{\mathsf{B}}(\mathbf{S}))$

A. All three must evaluate to the same result.
B. They each may evaluate to a different result.
C. **I** & **II**
D. **I** & **III**
E. **II** & **III**
F. Not enough information is provided to determine this.

---

c. Which of the following relational algebra expressions necessarily evaluate to the same result?

I. $\sigma_{\mathsf{A} \geq 5}(\mathbf{R} \bowtie \mathbf{S})$
II. $\sigma_{\mathsf{A} \geq 5}(\mathbf{R}) \bowtie \mathbf{S}$
III. $\sigma_{\mathsf{A} \geq 5}(\mathbf{R}) \bowtie \sigma_{\mathsf{A} \geq 5}(\mathbf{S})$

A. All three must evaluate to the same result.
B. They each may evaluate to a different result.
C. **I** & **II**
D. **I** & **III**
E. **II** & **III**
F. Not enough information is provided to determine this.

---

d. *Materializing* the temporary results of an operator in a query plan means that it
- **A.** works on blocks of records at a time.
- **B.** uses sequential reads or writes.
- **C.** is a blocking operator that cannot be pipelined.
- **D.** is a join operator.
- **E.** is the final operator in the query plan.

---

e. An iterator, pull-based architecture for the query processor
- **A.** eliminates the need for a buffer pool manager.
- **B.** eliminates the need for a query optimizer.
- **C.** is good for object-oriented database systems, but not for relational systems.
- **D.** does not allow for on-the-fly operations.
- **E.** implements pipelining.

---

f. Restricting focus to left linear join trees is beneficial for all but which of the following reasons?
- **A.** For any query plan based on a join tree that is not left linear, there is guaranteed to be a query plan based on left linear join tree that is less expensive.
- **B.** The inner relation for every join is a base table—or the evaluation of a single relation subquery—so a more accurate size estimation of the output can be obtained.
- **C.** The inner relation for every join is a base table—or the evaluation of a single relation subquery—so an index-nested-loops join remains possible.
- **D.** This helps prune the search search space of all possible join trees dramatically.
- **E.** Left linear join trees typically enable pipelining along the outer relations.

---

g. A System R style optimizer—the type of optimizer the textbook presents—preserves subplans that generate *interesting orders* while enumerating possible query plans because
- **A.** pipelining is impossible without them.
- **B.** this prunes the search space of possible query plans.
- **C.** such plans are *always* less expensive than plans that have no interesting order.
- **D.** such plans may allow certain subsequent operations to be done on-the-fly.
- **E.** hash joins are impossible without them.

---

h. The order of joins in a query plan matters because
- **A.** fewer joins are done in some join orders than others.
- **B.** intermediate result cardinalities can vary greatly between join orders.
- **C.** *selections* (σ's) are only possible with certain join orders.
- **D.** *order by* and *group by* are only possible with certain join orders.
- **E.** different join orders may result in different answers!

---

i. Which of the following is *true*?

A. Anywhere a (2-pass) hash join can be used, a (2-pass) sort-merge join could be used instead, at the same cost.

B. A block-nested-loops join is never better than a hash join or a sort-merge join. It is part of the repertoire simply for the cases when hash join and sort-merge join cannot be applied.

C. Select conditions can only be evaluated by use of an index.

D. Index-nested-loops join will rarely be the best choice unless there are other highly selective—returning few results—conditions in the query.

E. Hash joins and sort-merge joins cannot be used together in the same query plan.

---

j. Consider the queries

```
select distinct * ...
```

and

```
select all * ...
```

which are exactly the same, except for the distinct versus the all.

Consider the least expensive query plan **D** for the distinct query, and the least expensive query plan **A** for the all query.

A. **D** is guaranteed to be less expensive than, or the same expense as, **A**.

B. **A** is guaranteed to be less expensive than, or the same expense as, **D**.

C. **D** and **A** must be the same plan.

D. **D** and **A** may be different plans, but they must cost the same. (**D** just has an extra on-the-fly operation.)

E. There is not enough information to know whether **D** or **A** is less expensive.

4. (15 points) **Query Planning.** *Join up today!* [Analysis]

The table **Employee** has 100,000 records on 5,000 pages. It has the following indexes.

- An unclustered hash index on id, with 200 data entries per page.
  Employee's primary key is id.
- A clustered tree index on dept, with a depth of two index-page layers, and 80 data entries per leaf page.
  There are 1,000 distinct dept values.
- An unclustered tree index on manager, with a depth of two index-page layers, and 200 data entries per leaf page.
  There are 2,500 distinct manager values.

The indexes are of alternative #2.

You have a buffer pool allocation of 100 pages for each of the questions below.

---

a. (3 points) What would be the cost to evaluate

```
select * from Employee
    where E.dept = 'Sales';
```

using the index on dept?

Cardinality estimate: 100 records.
2 index page reads to find dept = 'Sales'. Read 2 data-entry pages to fetch the 100 matches. Read 5 data-record pages (100/20) to fetch the matching records. So around 9 I/O's, in total.

---

b. (3 points) What would be the cost to evaluate

```
select * from Employee
    where E.manager = 355;
```

using the index on manager?

So 40 employees per manager, on average. Estimate 40 matches, then. 2 index-page reads to find manager = 355. All 40 matching data entries on same data-entry page, likely: 1 I/O. 40 I/O's then to fetch the 40 records. Around 43 I/O's, in total.

c. (4 points) Describe an *efficient* way to evaluate the following query. (You will not earn full marks if System R would have found a much more efficient plan than your solution.)

```
select E.id, E.name, E.dept, M.name as boss
    from Employee E, Employee M
    where E.manager = M.id;
```

Provide an estimate of its cost and cardinality.

Cardinality estimation: 100,000 records. Join is over a foreign key: one manager matches per employee.
For the outer, sort **E***[1] on* manager*. This can be done in 2 passes: 20,000 I/O's. Now do a merge join on* **E***[2] as the inner by index-scan on* id*: 1,250 I/O's of data-entry pages and 5,000 I/O's of data-record pages. Total is 26,250 I/O's.*

d. (5 points) Consider the following catalog information.

- **Sale**: 1,000,000 records on 25,000 pages
  - **Sale**.p#: 1,000 distinct values
- **Product**: 10,000 records on 200 pages
  - **Product**.p#: 10,000 distinct values
  - **Product**.category: 100 distinct values

The primary key of **Sale** is p# + cust# + when and of **Product** is p#. Table **Sale** has a foreign key on p# referencing **Product**. All columns are *not null*.

Sale has a clustered tree index of type alternative #2 on p# + cust# + when that is three index-pages deep with 10,000 data-entry pages.

Assume that *tablet* is a category value in **Product**. Consider the query

```
select P.p#, P.name, S.cust#
    from Product P, Sale S
    where P.p# = S.p# and P.category = 'tablet';
```

Consider the query plan for this query that does an index-nested loop join with **Product** as the outer and **Sales** as the inner, and the selection "pushed" below the join.

What is the cost of this query plan?

For the outer, tablescan of **Product**, *200 I/O's, and apply the select of* category = *'tablet' on the fly. The reduction factor of* category = *'tablet' is* $10000/100 = 100$ *records.*
Now use an index-nested loop join. The inner is **Sales**. *There are 100 probes from the outer. Each probe costs 3 index-page reads, 100 matching sales per* p# *is estimated, costing one or 2 data-entry pages to collect, then 3 data-record pages (clustered) for the matches. So 8 I/O's per probe.*
Thus, the total is 1,000 I/O's.

Extra space.

Relax. Turn in your exam. Go home.