COSC-4411(M) Midterm #2

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

- **Instructor:** Parke Godfrey
- **Exam Duration:** 75 minutes
- **Term:** Winter 2004

Answer the following questions to the best of your knowledge. Be precise and be careful. The exam is open-book and open-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. Points for each question and sub-question are as indicated. In total, the exam is out of 50 points.

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

Regrade Policy

- Regrading should only be requested in writing. Write what you would like to be reconsidered. Note, however, that an exam accepted for regrading will be reviewed and regraded in entirety (all questions).

Grading Box	
1.	
2.	
3.	
4.	
Total	

1. (15 points) **Join Algorithm Selection.** *Would you join me in choosing a join?* [exercise / short answer]

   Consider tables **R** with attributes A and B and **S** with attributes B and C. Column B is unique in **S**. Values of B in **R** are the same as the values of B in **S**. Assume that there are no indexes available.

   Consider the sort-merge join to be the efficient *two-pass* version discussed in the textbook (in which the sort-merge's merge steps are integrated with the external sort's merge steps) and not the more general version. (In the more general version, the external sort is *entirely* done before the merge-join. Furthermore, assume that the sort-merge join algorithm produces runs of length $2B$ in "pass 0" of sorting.) Consider that there may be problems with skew.

   For each of the following, choose which of the following join algorithms is likely to be best. In each case, there are 250 buffer frames allocated for the join.

   A. A block nested loops join with **R** as the outer and with **S** as the inner.
   B. A block nested loops join with **S** as the outer and with **R** as the inner.
   C. A (2-pass) sort-merge join with **R** as the outer and with **S** as the inner.
   D. A (2-pass) sort-merge join with **S** as the outer and with **R** as the inner.
   E. A hash join with **R** as the outer and with **S** as the inner.
   F. A hash join with **S** as the outer and with **R** as the inner.

   Which would be the most cost effective method to evaluate $\mathbf{R} \bowtie_{\mathsf{B}} \mathbf{S}$? Briefly justify your answer in each case.

---

   a. (5 points) Table **R** is 200 pages with 80 records a page, and table **S** is 40,000 pages with 100 records a page.

b. (5 points) Table **R** is 50,000 pages with 80 records a page, and table **S** is 40,000 pages with 100 records a page.

c. (5 points) Consider the case of $\mathbf{R} \bowtie_{\mathsf{J}} \mathbf{S}$ when join column J has very few distinct values in **R** *and* in **S**. Would a hash join likely work well for $\mathbf{R} \bowtie_{\mathsf{J}} \mathbf{S}$?

Why or why not?

2. (10 points) **Indexes.** *To index or not to index. That is the question...* [analysis]

a. (5 points) Is it possible to use *index intersection* as an access path that is also an *index-only* plan?

Explain briefly why or why not.

b. (5 points) Consider table **R** with attributes A, B, and C (A is its primary key) and table **S** with attributes C, D, and E (C is its primary key)

There is a foreign key declared for **R** on C referencing **S** (on C). We know that queries which join **R** and **S** over C (that is, queries which involve $\mathbf{R} \bowtie_C \mathbf{S}$) will be very common. Is it worthwhile to build an index on **R**.C? Under what conditions would having an index on **R**.C help? Provide an example scenario, or explain why it probably does not help under any reasonable circumstances.

By "condition", we mean properties of **R** and **S** and of the types of queries asked that involve $\mathbf{R} \bowtie_C \mathbf{S}$. For example,

- *When* **R** *is small...*
- *When* **S** *is large...*
- *When the queries involve many conditions additional to the join condition (*$\mathbf{R}.C = \mathbf{S}.C$*)...*
- *When the query involves few conditions beyond the join condition (*$\mathbf{R}.C = \mathbf{S}.C$*)...*

3. (10 points) **General.** *It costs how much?!* [multiple choice]

a. (2 points) *Pipelining* saves what?

A. Space in the buffer pool because only one page of each file being used in the current operation needs to be present at any given time.

B. The I/O's of writing out a temporary result table and reading it in again when those results can be immediately used by the next operation.

C. Disk space by packing variable-length records on a page more efficiently.

D. Time by prefetching pages that might be needed into the buffer pool ahead of time.

E. I/O cost by reading and writing sequences of pages arranged sequentially on disk much faster than reading and writing them one at a time.

The following information is available on tables **Sailors** and **Reserves**.

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

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

b. (2 points)

```
select distinct S.sid, R.day
    from Sailor S, Reserves R
    where S.sid = R.sid and
          R.bid = 7 and S.level = 6;
```

Estimate the selectivity of the above query as the number of tuples it likely returns.

A. 2

B. 5

C. 20

D. 50

E. 100

F. 10,000

c. (2 points)

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

Estimate the selectivity of the above query as the number of tuples it likely returns.

A. 1

B. 5

C. 20

D. 80

E. 100

F. 10,000

d. (2 points) An efficient way intersect, $\mathbf{R} \cap \mathbf{S}$, might be implemented is

A. via union: $\overline{\overline{\mathbf{R}} \cup \overline{\mathbf{S}}}$.

B. via except: **R** except (**R** except **S**).

C. as a special case of project: project just those columns in **R** that also appear in **S**.

D. as a special case of join: join **R** and **S** using all the attributes as the join attributes.

E. via aggregation.

---

e. (2 points) Which following heuristic tends to result often in a less expensive query tree?

A. Pushing selects and projects below joins.

B. Pushing joins below selects and projects.

C. Preferring table scans to using an unclustered index.

D. Making the smaller table of the two in a join the outer table.

E. Making the larger table of the two in a join the outer table.

---

4. (15 points) **Query Plans.** *We've got the cheapest query plans in town!* [exercise]

The following information is available on tables **Reserves** and **Boats**.

- **Reserves**: on 2,500 pages
  - 100,000 records, primary key is sid + bid + day.
  - **Reserves**.bid: 1,000 values
  - **Reserves**.day: 5,000 values
- **Boat**: on 100 pages
  - **Boat**.bid: 1,000 values (primary key)
  - **Boat**.size: 10 values (3..12 meters)

Table **Reserves** has a foreign key on bid referencing **Boats** (on bid). All columns are *not null*.

Available indexes are as follows. Each follows alternative two (having data-entry pages).

- A unique, clustered hash index on bid for **Boat** (20 data-entry pages / buckets).
- A unique, clustered B+ tree index on sid + bid + day (in that order) for **Reserves** (750 data-entry pages / leaves).
- An unclustered hash index on day for **Reserves** (250 data-entry pages / buckets).

Consider the following query.

```
select R.sid, B.bid, B.bname
    from Reserves R, Boat B
    where R.bid = B.bid and
          B.size > 6 and
          R.day = '14 November 2003';
```

---

a. (5 points) What is the expected number of records returned by this query?

Make reasonable assumptions about reduction factors, fudge factors for accesses, etc., along the lines of Chapter 12 / System R.

b. (10 points) You have a buffer pool allocation of 15 frames for the query. Design the best query evaluation plan for the above SQL query that you can. Show it as a relational algebra tree. Annotate the tree with the chosen join methods, *on-the-fly*, and so forth.

   You should be able to design a query plan with an estimated cost better than 100 I/O's (but do the best you can).

   Estimate the cost of your query plan.

(Scratch space.)

Relax. Turn in your exam. Go home.