2004 Systems Qual, Database Area: Part B

B.1 **[10 points; 5 for each part]** Consider a relational database of airline passenger and reservation information:

```
Passenger(passID, name)       // passID is a key
Resvn(passID, flight#, date) // (passID,flight#,date) is a key
```

(a) Write a relational algebra expression to find the names of all passengers who have reservations on exactly two different dates.

(b) Write the same query in SQL. *Your solution will be graded on simplicity as well as correctness.*

B.2 **[4 points]** Consider a table $T(A, B)$ containing about 10,000 tuples, stored on disk. Suppose on average a disk page (block) contains about 50 tuples of T, tuples are distributed randomly among pages, and pages of T are contiguous on disk. A user issues the following query:

```
select * from T where A=25
```

Suppose there is an index on $T.A$, so the query optimizer must decide whether to use the index or scan the entire table. To keep things simple, assume the index is hash-based and stored entirely in memory. Consider four different scenarios: when the selectivity of the $A = 25$ predicate is estimated at 10%, at 1%, at 0.1%, and at 0.01%. For which of these selectivities should the index be used instead of a relation scan? Show any computation and briefly justify your answer.

B.3 **[6 points; 2 for each part]** Provide a *very brief* answer for each of the following questions about join processing.

(a) You are building a database system and you've just finished implementing three different methods for executing conventional theta-joins: *nested-loops join*, *sort-merge join*, and *hash-based join*. Now you've been told you need to support full outerjoins also. In terms of coding, which of your three join methods would be easiest to extend to perform full outerjoins and why?

(b) Continue to consider your system with three join methods: nested-loops join, sort-merge join, and hash-based join. Suppose you support as a query option "`Fetch` N `Rows Only`," which a user specifies in order to get just N rows from the result instead of the entire result. When executing a simple join query over unsorted relations that includes this clause, which of the three join methods is most appropriate and why? Assume N is expected to be much smaller than the full result size.

(c) A technique called *rocking* is sometimes used in the implementation of nested-loops or nested-block joins. With rocking, the first time the inner relation is scanned it is scanned forwards, the second time it is scanned backwards, the third time forwards, etc. Under what specific circumstances is this technique is expected to speed up join processing?