Part C

<u>Basic Part</u> (15 points)

1. [4 points] Consider relations $R(A, B)$ and $S(B, C)$. Are the following two SQL queries equivalent on all databases? If not, give a simple counterexample.

```
select R.A
from R, S
where R.B = S.B and S.C = 4

select R.A
from R
where R.B in (select S.B from S where S.C = 4)
```

2. [5 points] Consider relations $R(A, B)$ and $S(B, C)$, and the following query:

   $R \bowtie \sigma_{C=k}(S)$

   Assume that:

   - R contains r tuples.
   - S contains s tuples.
   - Column $S.B$ contains b distinct values uniformly distributed.
   - Column $S.C$ contain c distinct values uniformly distributed.
   - k is a constant.
   - S has an index on its B attribute.
   - S has an index on its C attribute.

   Consider using a nested-loops approach to evaluate this query with R as the outer relation. Suppose that the information given above is all that is available to the query optimizer. Further suppose that the query execution plan should exploit the index on $S.B$ or the index on $S.C$, but not both. Under what condition(s) is it preferred to use the index on $S.B$ rather than the index on $S.C$?

3. [6 points] Consider a database with our two favorite relations $R(A, B)$ and $S(B, C)$, and a materialized view $V = R \bowtie S$. (Note: A view is *materialized* when its contents are stored in a table in the database and kept consistent with the "base" tables over which the view is defined. This approach contrasts with a *virtual* view, whose contents are never stored, but whose definition is used to rewrite queries over the view into queries over the base tables. Currently most relational database management systems support virtual views only.)

   (a) Describe a simple query processing scenario in which it is beneficial to replace references to tables R and S in a given query Q with references to the materialized view (table) V.

   (b) Describe a simple query processing scenario in which it is beneficial to replace references to view V in a given query Q with references to base tables R and S.

Advanced Part (30 points)

4. [2 points] The *Exodus* and *Starburst* research prototypes both call themselves "extensible." However there is a very basic architectural difference between the two systems. What is it?

5. [1 point] The *POSTGRES* prototype also calls itself extensible. Based on the distinction you specified in problem #4, is POSTGRES architecturally more similar to Exodus or to Starburst?

6. [1 point] State how the TPC-D benchmark differs markedly from the TPC-A, TPC-B, and TPC-C benchmarks.

7. [3 points] Which join algorithm adapts most readily to full outerjoins: nested-loops, sort-merge, or hash-join? Why?

8. [5 points] Consider problem #2 in the Basic Part above. Is it possible to take advantage of both indexes in executing the query? If not, why not? If so, sketch an algorithm.

9. [6 points] An active rule, or *trigger*, may be executed in a *tuple-oriented* fashion (called "`FOR EACH ROW`" in SQL), in which case it executes once for each relevant modified tuple, or it may be executed in a *statement-oriented* fashion ("`FOR EACH STATEMENT`" in SQL), in which case it executes once at the end of a relevant update statement and operates over all modified tuples. Using any coherent "Event-Condition-Action" trigger language you like, write a simple trigger that exhibits different behavior if it is executed in a tuple-oriented fashion than if it is executed in a statement-oriented fashion. That is, there should be some database and triggering modification(s) for which the final state of the database after trigger execution is different if the trigger is executed in a tuple-oriented fashion than if it is executed in a statement-oriented fashion.

10. [12 points] Suppose you are writing the component of a query optimizer that establishes the ordering of the joins in the query execution plan. Rather than using an exhaustive *System-R* style approach, your task is to implement a set of heuristics that quickly chooses an ordering that is expected to perform reasonably well in most cases.

    (a) What essential properties of the query and/or the tables referenced in the query should your heuristics be based upon?

    (b) Extending your answer for part (a), sketch one possible algorithm for generating the join ordering.