Problem Set #2

1. Suppose an important customer says "It is essential that your DBMS processes join predicates of the form (A.x = B.x or A.y=B.y) quickly". Normally, you would say: tough beans. But the survival of your DBMS company is dependent on this customer using your product. Your DBMS supports only nested loops and hash join algorithms, and uses the System-R algorithm for generating physical access plans. You can't make major changes to your DBMS query optimization subsystem. Your manager has an insight – generalize the hash join algorithm. But how?

2. Another important customer says "it is essential that your DBMS processes join predicates of the form (A.x>B.x) quickly". As before, you'd rather say: tough beans. But your manager asked you again to perform another miracle. Using the same constraints as before, how would you save your company? Hint: think alternative main-memory data structures.

3. Consider the following nested SQL query, which says retrieve x values from each A tuple where q=40 and there are **no** tuples in C that could join with attribute w of that tuple in A:

```
select A.x
from A
where A.q=40 and not exists (
    select *
    from C
    where C.w = A.w )
```

   a. Use a Kim or magic set rewrite of this query as a sequence of 2 non-nested queries. Hint: SQL minus and select-into operations.
   b. Rewrite this query as a single unnested query. Hint: outerjoins.

4. What does this query mean? (This query could be executed on the database of Project 1).

```
select pname
from parts
where not exists (
        select cname
        from customers natural join zipcodes
        where city = 'Austin' and not exists (
                select ono
                from orders natural join odetails
                where cno = customers.cno
                        and parts.pno = pno )    )
```