Problem 1: Relational Algebra [25 points]

Recall from HW1 the following partial schema for the (real) Internet Movie Database (IMDB.com) where keys are underlined:

```
actors(id: int, first_name: string, last_name: string, gender: string)
directors(id: int, first_name: string, last_name: string)
director_genre(director_id: int, genre: string, prob: double)
movies(id: int, name: string, year: int, rank: double)
movies_directors(director_id: int, movie_id: int)
movies_genres(movie_id: int, genre: string)
roles(actor_id: int, movie_id: int, role: string)
```

Translate the following queries from SQL to (i) Tuple Relational Calculus and (ii) Relational Algebra:

a. `SELECT M.name, M.year
   FROM movies M, movies_directors MD, directors D
   WHERE M.id = MD.movie_id and MD.director_id = D.id and D.last_name
       like 'n';`

b. `SELECT a.id, a.first_name, a.last_name, a.gender
   FROM actors a, roles r, movies_genres mg
   WHERE r.actor_id = a.id and mg.movie_id = r.movie_id and mg.genre = 'Comedy'
   MINUS
   SELECT a.id, a.first_name, a.last_name, a.gender
   FROM actors a, roles r, movies_genres mg
   WHERE r.actor_id = a.id and mg.movie_id = r.movie_id and mg.genre = 'Drama'

   (Hint: For this query, use a single Tuple Relational Calculus expression with a NOT EXISTS predicate.)`

Next, launch your CIS 550 Virtual Machine and run Eclipse. (You may alternatively run Eclipse for Java installed on your local operating system.) Go to the Package Explorer pane, right-click, choose Import, then Git, Projects from Git. Hit Next, then Clone URI, then Next. Enter the URI `https://bitbucket.org/upenn-cis550/550-hw5.git`. Click Next, then be sure master is checked, and hit Next again. Choose a directory for your source code (in your workspace directory), then hit Next. Choose “Import existing Eclipse projects” when that option becomes available. Hit Finish.

This project contains a command-line implementation of a simple relational query engine, i.e., an evaluator of RA expressions. The main program is `edu.upenn.cis.cis550.hw5.RelationEngine.java`. 

1
If you right-click it, then choose Run As, Java Application. By default it will (1) read in the contents of the *imdb.sql* script (which can be used to load the same Internet Movie Database data into Oracle) and create several in-memory tables, (2) execute a set-difference query and (3) execute a join query.

You will need to familiarize yourself with the code in RelationEngine, as well as the query operators (....algebra), predicates (....operators), data fields (....data), and schema elements (....schema).

There are three kinds of data types, IntAttributes, StringAttributes, and DoubleAttributes; and three corresponding Fields. A Tuple consists of a Schema and a list of Fields. Tuples also contain information about keys and nulls.

There are three kinds of conditions/predicates: Equals, Like, and LessThan. Each is parameterized with the type of actual input (here always Tuple) and the type of field-value being compared (String, Integer, Double). The actual things being compared are either (1) ConstInt, StringLiteral, or ConstDouble, or (2) the contents of a de-referenced field, which you can get through Tuple.intGetter(), doubleGetter(), or stringGetter().

From the relational algebra, you should have join (NestedLoopsJoinOperator), project (ProjectOperator), select (SelectOperator), rename (RenameOperator), and set difference (MinusOperator). Additionally, there is an operator that reads the contents of a relation, the TableScanOperator. Observe that there is a pattern to executing a query: call initialize(), then iterate over tuple results from getNext(), then call shutdown().

d Update RelationEngine.java to define the schemas for the missing relations in the schema. Observe that the DataLoader “knows about” actors, movies, and roles. Augment it to load the remaining tables from the schema definition of Problem 1.

e Comment out the calls to executeMinusQuery and executeJoinQuery.

f Add (and call) two new functions: executeQueryA1 and executeQueryB1. Each of these should be an implementation of your relational algebra expression for Parts (a) and (b) above. Note that the Like operator in our implementation takes standard Unix-style regular expressions, so you will need to rewrite the SQL predicate accordingly.

g Now write new functions executeQueryA2 and executeQueryB2 that do the same as the above, but use a different relational algebra expression that is equivalent. Record the differences in numbers of tuples processed.

You will submit a file called hw5-1.pdf with answers to Parts a and b, as well as the size information from running the two versions of the relational algebra expression for Parts a and b. Include a paragraph describing why these are different and which is better, or an explanation of why the results are the same.

You should also submit your updated RelationEngine, in which you should expect that we will only call executeQueryA1, executeQueryA2, executeQueryB1, and executeQueryB2 with sample relations conformant to the schema. In other words, you should make sure the full RA expressions are computed in those two methods.
Problem 2: B+ Trees [25 points]

Consider the B+ tree index shown in Figure 1, which stores data entries at leaf nodes in the index. Each intermediate node can hold up to five pointers and four key values. Each leaf node can hold up to four records (indicated as a key value plus a “*”), and leaf nodes are doubly linked to their predecessor and successor nodes in the index (this is not shown in the figure). The order of the tree, i.e., the minimum number of keys allowed in an intermediate node, is two, and the minimum number of records allowed in a leaf node is also two. Underscores (“—”) indicate unallocated (free) entries.

1. Insert a record with search key 53 into the tree.

2. (Extra credit: 10 points): Delete from the original tree the records with keys 43, 69, 97.

Submit the solution to this problem as hw5-2.pdf (using a 2nd page for the extra credit problem if you do it).

Problem 3: Index Performance [50 points]

For this part of the assignment, you will need to run Oracle on eniac, using the sql command, as in previous assignments. The goal is to see the benefits of indexing in action.
The Oracle setup on eniac has a series of sample tables created for running database benchmarks — the so-called TPC-H benchmark. Additionally, each of these tables may have several alternative index structures. We will be comparing performance depending on which indices are used. To do this assignment, you will need to first enter the Oracle command `set timing on`, which will cause Oracle to report an “elapsed” time in seconds. You should run each query multiple times to ensure that small variations in system performance (e.g., due to your classmates’ queries) get averaged out. Note that we are using the “cold cache” hint, which asks Oracle to ignore any cached data from the table when it executes the query.

The `lineitem` table has nearly 300,000 rows and takes approximately 37MB of space. We have two indices in `lineitem`, each with a name: `SYS_C0011064` is an index over the attribute pair `(l_orderkey, l_linenumber)` and `lineno` is on the singleton attribute `(l_linenumber)`.

1. Start by running the query:

```sql
select /*+ cold cache */
  avg(l_linenumber)
from zives.lineitem
where l_linenumber <= 14;
```

which uses whatever indices Oracle prefers. Time the run and repeat it 5 times, averaging the total. What is that value?

2. Next, repeat the query with a special optimizer hint that says you want to scan the full table and avoid indices:

```sql
select /*+ full(l) cold cache */
  avg(l_linenumber)
from zives.lineitem l
where l_linenumber <= 14;
```

Average the running time for 5 runs and report the result. What can you say about the difference between the two runs? Do these results suggest that Oracle uses an index by default?

3. Try another hint, which makes use of the first index:

```sql
select /*+ index(l SYS_C0011064) cold cache */
  avg(l_linenumber)
from zives.lineitem l
where l_linenumber <= 14;
```

Run 5 times and report the result. How did this compare to the first two cases? Did that particular index offer any benefits?
4. Try forcing Oracle to use the `lineno` index as well. Which index, or indices, might have been used in the default, un-hinted Oracle query? Generalizing from the Oracle SQL hints shown above, what is the query hint that is equivalent to that default query?

5. For each of the two indices and for the non-indexed case, explain how what data and/or index structures will be examined in answering the query.

6. Run the two queries:

```sql
select /*+ index(l SYS_C0011064) cold cache */
    count(distinct l_suppkey)
from zives.lineitem l
where l_linenumber <= 14;

select /*+ full(l) cold cache */
    count(distinct l_suppkey)
from zives.lineitem l
where l_linenumber <= 14;
```

Report the averaged running times over 5 runs. Explain the relative differences (or lack thereof) in performance, based on your knowledge of index and data layout characteristics, as well as your knowledge of “covering” indices.

Your submission should be called `hw5-3.pdf`.

**Problem 4: Query Performance [25 points]**

Now we will have you specify query plans using the TPC-H benchmark data set, using Oracle. To do this assignment, you will need to repeat each query 5 times. Time it by using the Oracle command “set timing on” and reading the elapsed time. Remember to repeat the query 5 times and average the results, in order to get more consistent results.

Start with the query:

```sql
select /*+ cold cache */
    count(*)
from zives.lineitem, zives.orders
where l_orderkey = o_orderkey
and o_totalprice <= 1500;
```

1. What index or indices would be most useful on the lineitem and orders tables?

2. As with before, we are going to use Oracle’s hints to tell the optimizer how to run the query. Time the query:
select /*+ LEADING(l o ) USE_HASH(l o) cold cache */
   count(*)
from zives.lineitem l, zives.orders o
where l_orderkey = o_orderkey
and o_totalprice <= 1500;

select /*+ LEADING(o l) USE_HASH(l o) cold cache */
   count(*)
from zives.lineitem l, zives.orders o
where l_orderkey = o_orderkey
and o_totalprice <= 1500;

3. Replace “LEADING(o l)” with “LEADING(l o)” and repeat. This reverses the order of the join.

4. Replace “USE_HASH” with “USE_NL” and repeat, with both orderings of LEADING.

5. Was there any substantive difference? Which plan or plans seemed to perform best? If there was a difference, explain why. If not, explain why not.

Submit as hw5-4.pdf.