Project3- Team 24(Dexter)… Submitted by Ranga Raj and Pramit Choudhary

File Structure:

- Our implementation of Project 3: Implementing a Query Engine is all in the qe.cc, qe.h files. We have also submitted qetest.cc file which is our tester.
- We have submitted the PF, RM and IX layers along with this submission to complete the runtime.

Implementation Classes:

As indicated in the guidelines, the QE layer is the implementation of a Query Engine component and comprises of a number of iterator-based operators.

Filter Interface Class:

This implements the WHERE clause predicates in a typical statement. It provides an ability to specify a condition that filters the output of an iterator input.

Project Interface Class:

This implements the 'projection' in a SQL statement. The Project clause simply provides the ability to select a specified set of columns to display as output. The number of tuples returned by a Project Interface class matches the number of tuples sent to it by the iterator input.

Nested-Loop Join Interface Class:

This provides an ability to compute the cross-product between the output of an iterator and a TableScan iterator.

Index Nested-Loop Join Interface Class:

This provides an ability to compute the cross-product between the output of an iterator and a IndexScan Iterator.

HashJoin Interface Class:

This implements the HashJoin GRACE algorithm to join two iterators.

Implementation Details:

We have implemented all classes as per the guidelines provided. We support joins and comparisons on TypeInt, TypeReal or TypeVarChar columns.

Of particular mention is the HashJoin Interface.

We have implemented the following HashJoin algorithm:

In Phase I:

- For each relation (left and right)
  - Read a tuple one-by-one
  - Use a hash function to identify the partition to write the tuple into
  - The tuple is written straight into a Paged File. We have a separate PagedFile for each partition.
  - The hash function hashes the value of the join attribute
  - The number of partitions is specified by the constructor of the HashJoin interface

In Phase II:

- Read the first partition of the Right relation
- Construct a hash table in memory using a different hash function
- This hash function generates 10 times the number of buckets that the hash in Phase 1 would generate.
- For each tuple in the first partition of the Right relation

    - Hash the join attribute value on the tupe
    - Locate the key in the hash table
    - Scroll through the data entries in the hash table for that key
    - When the attribute keys match, join the attributes on the hash table entry with the attributes on the tuple from the right relation.
    - Save te joined result tuple into a Temporary Page File called a HashResult.
- Repeat the process for all the partitions that were created in Phase I

Clean up the Temporary paged files created during Phase I.

Retain the HashJoin Result Paged File to scroll through for the getNextTuple interface.