Sprint 3 (Invierno) – Nina Nguyen

August 9, 2019

Spring 2019

**For this sprint, I initially had some troubles with my partner communicating and contributing to the assignment, we agreed to split the implementation of both Milestones. I haven’t really heard from her throughout the sprint but on 8/10/2019, she suddenly submitted both completed Milestones on github (even implementations I agreed that I will do). I’m really perplexed with the whole situation. Since I already did my part and our helper functions are a little different, I will use her implementation and incorporated it into my code for both Milestone.**

**Milestone 5:**

Note: I implemented:

* static ValueDict \*get\_where\_conjunction(const hsql::Expr \*expr, const ColumnNames \*col\_names);
* static QueryResult \*insert(const hsql::InsertStatement \*statement);
* static QueryResult \*del(const hsql::DeleteStatement \*statement);

In this Milestone,

SQLExec.cpp class takes the SQL statement from the user and uses the parse tree to turn it into actual SQL language.

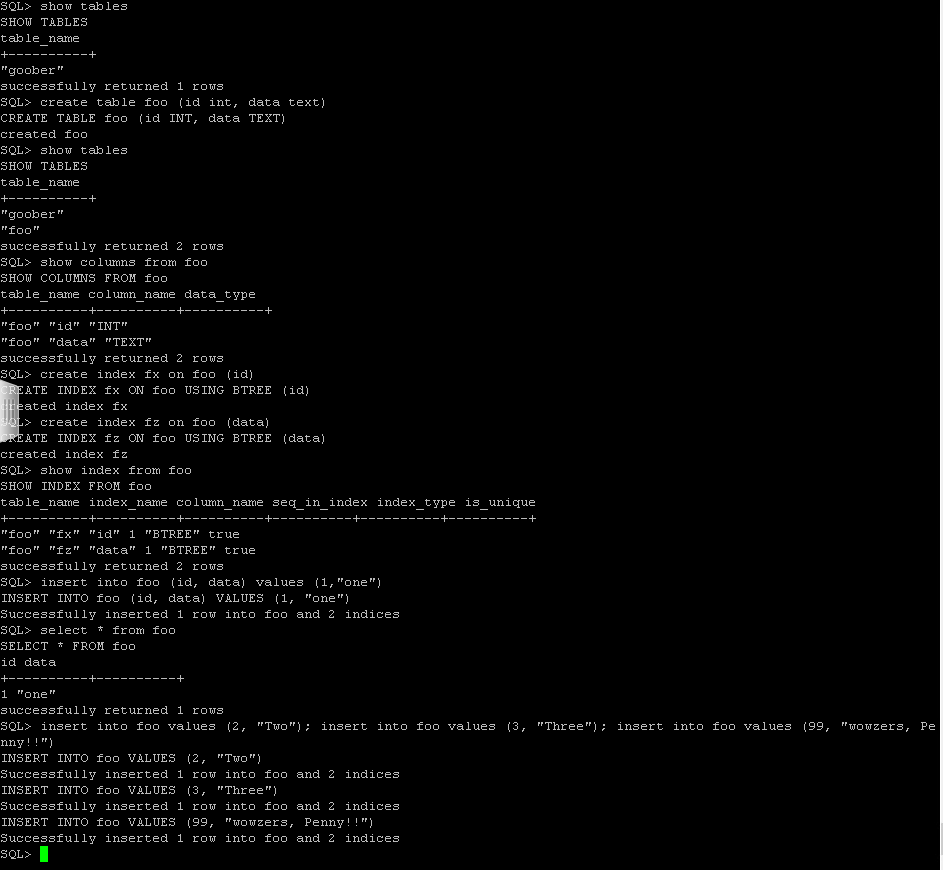
get\_where\_conjunction() is a helper methods that looks at the parse tree for the WHERE. If user did not specify a where, it will be null. Ultimately return a ValueDict object with the rows that matches the statement.

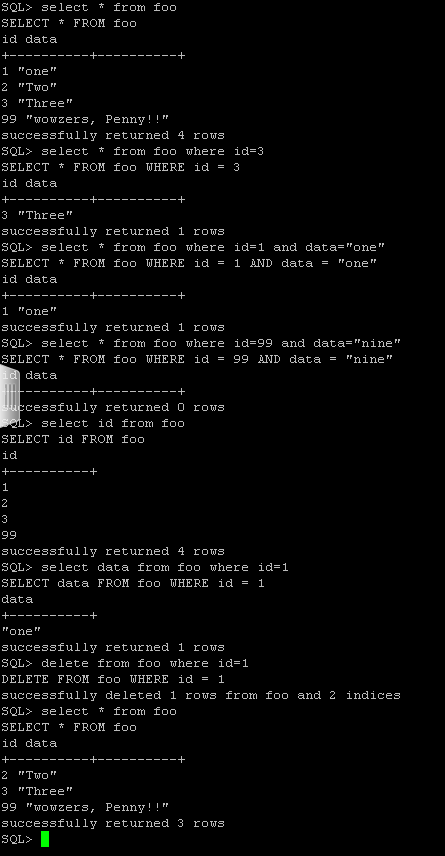
insert() get the table and column names from table. From the parse tree, get the columns and value from the insert statement and insert it into the table. Also does the same for the indices.

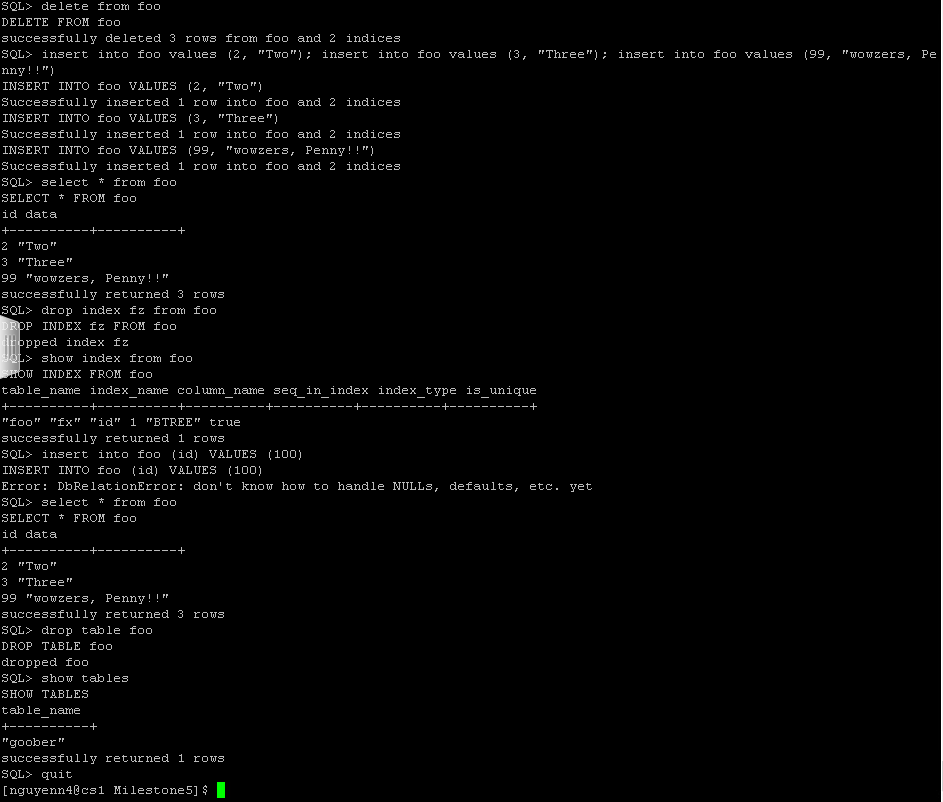
del() create an evaluation plan to get all the handles that need to be deleted. If there is no WHERE clause, the evaluation will select everything to be deleted. Get handles by calling pipeline() on it and once that’s done, it can be remove from the indices. Finally, the handles will be deleted from the table.

Select() accepts an equality condition but we’re only implementing AND operator at this time. This function translate the statement into an evaluation plan along with a project plan (ProjectAll or Project). Then the plan is optimized and evaluate to get the results.

The screenshot example after adding Maggie’s code:







**Milestone 6:**

This milestone is where we implement the B+ Tree Index (insert and lookup)

My responsibility for this Milestone was:

* BTreeIndex(DbRelation& relation, Identifier name, ColumnNames key\_columns, bool unique); - stat and root starts out as nullptr until create() is called
* virtual ~BTreeIndex();
* void build\_key\_profile(); - Get the datatype of key\_columns
* virtual void create(); - Gets the stat of the BTree, leaf/root, then insert the handle into itself from the relation
* virtual void drop(); - delete the file
* virtual void open(); - get the stat of BTree, if height is 1 then root will be a leaf or else it’ll be an interior node, then it closes the file.
* virtual void close(); - close underlying file, set stat and root to nullptr

Example Screenshot Below:

