Kenny Yu
30798260
CS165 - Project 3 Design Doc

Assignment: http://stratos.seas.harvard.edu/files/stratos/files/cs_165_project3.txt

Parser

Extend my parser with the following operations (adding new query types and sscanf lines to my parser code):

- add
- sub
- div
- mul
- min
- max
- avg
- hashjoin
- sortjoin
- loopjoin
- treejoin
- print

Also, allow all the aggregation operators to be ASSIGNable.

```
enum agg2_type {
      AGG2_ADD,
      AGG2_SUB,
      AGG2_MUL,
      AGG2_DIV,
};

// Aggregation on 2 columns
struct op_agg2 {
      enum agg2_type;
      bool agg2_assign;
      char op_agg2_left[COLUMNLEN];
      char op_agg2_right[COLUMNLEN];
      char op_agg2_var[COLUMNLEN];
};

enum agg1 {
      AGG1_MIN,
      AGG1_MAX,
      AGG1_AVG
};

struct op_agg1 {
      enum agg1_type;
      bool agg1_assign;
      char op_agg1_col[COLUMNLEN];
      char op_agg1_var[COLUMNLEN];
}

enum join_type {
      JOIN_HASH,
      JOIN_SORT,
      JOIN_LOOP,
      JOIN_TREE,
};

struct op_join {
      enum join_type op_join_type;
      char op_join_left[COLUMNLEN];
      char op_join_right[COLUMNLEN];
      char op_join_varleft[COLUMNLEN];
      char op_join_varright[COLUMNLEN];
};
```

Aggregations

To implement two-column math aggregations, add this new function:

```
// Takes two column values (returned from fetch) and a combination
// function, and returns a new column_val.
struct column_vals *column_combine(
      struct column_vals *left,
      struct column_vals *right,
      int (*f)(int, int));
```

To implement single column aggregations, add this new function:

```
// Takes a column and an aggregation function,
// and returns a column_vals with a singleton value.
struct column_vals *column_aggregate(
      struct column_vals *vals,
      int (*f)(struct column_vals *));
```

Join

We need to extend the struct column_vals to also keep track of the position for each result value (join returns positions):

```
struct column_vals {
    int *cval_vals;
    uint64_t *cval_pos; // keep track of positions as well
    unsigned cval_len;
};
```

Add this new function to perform LEFT JOIN RIGHT:

```
// Performs a join on the left and right vals using the given join
// type. On success, return 0, and set the ret pointers to point to
// the column_ids that pass the join. This will delegate to the
// appropriate join type handler.
int column_join(
      struct column_vals *left,
      struct column_vals *right,
      enum join_type,
      struct column_ids **retleft,
      struct column_ids **retright);
```

For sorted joins, I will use an external sort.

For hash joins, I will implement grace hash joins.

To create a hashtable, I will use static hashing (requires 2 passes).

Interactive Mode

I already have a primitive form of interactive mode (./client --interactive), but I will change this to use readline to be more robust.

Extra: Vectorized Plans

Use an iterator pattern on column_ids and column_vals to lazily fetch the ids in blocks (for the vectorized plans) and values only when forced, and then memoize the result.

```
struct iterator;

struct iterator *iter_create(void *data);
void iter_destroy(struct iterator *it);
bool iter_has_next(struct iterator *it);
void iter_get(struct iterator *it, void **retdata);
void iter_next(struct iterator *it);
```

Make type specializations of these for values/positions/value position tuples, depending on the type of object we are creating an iterator for.