Andrew Stange and Daniel Eigler-Harding

COSC 280 DBMS Project

General Usage:

* When more than one table is used in a query, all attributes used must have their table explicitly specified, including the use of “\*”.
* Supported data types for inputs are int, float, and string. Strings must be wrapped in quotations to avoid confusing them for attribute names.
* UNION, INTERSECT, DIFFERENCE between multiple queries is supported, but nested queries are not.
* Aggregate operators can only be used in the SELECT clause.
* Command “optimize\_on” to use faster printing method. “optimize\_off” to revert.
* “exit” command to terminate program.
* Each table only supports a single primary key and a single foreign key. A foreign key must reference the primary key of the table the foreign key is from, but does not have to be on the primary key of the table being created. A parent table can have an unlimited number of child tables.
* Referential integrity restricts insertions into a child table where the foreign key attribute does not match a value in the parent table, restricts updates to the referenced/referencing attributes in the parent and child relations, and implements a cascade policy for removals from the parent table.
* WHERE clauses permit limited complexity. Attributes from the same table cannot be compared to one another, but comparing attributes from different tables is supported (only equality is allowed). Arbitrarily complex comparisons using attributes from the same table is supported.
* NOW function has 2 uses in this implementation. First, “now()” can be specified in a query to output the current time appended to each outputted tuple. A “now” attribute can also be specified in a create table command (without specifying a data type) and the time of each insertion will be logged automatically. Insertions with tables that have “now” set must include “now” in the list of attributes of the table, but do not provide a value for “now” in the values section of the insert command. In order to get the value of the “now” attribute in a query, can specify <table name>.now in the select clause.
* The set clause of the DML update command accepts only simple, atomic operations. Can set an attribute to a value, or can apply a simple function to the current value of the specified attribute (ex. id = id \* 5). Addition, subtraction, multiplication, and division are supported.
* The where clauses for update and delete DML commands are parsed in the same way. Do not support comparisons of multiple attributes (ex. tableA.id == tableA.age is an invalid comparison, as is tableA.id = tableB.id).

Implementation Notes:

* Optimizer
  + Access by index
    - All conditions in the where clause of the query are examined to see if they are compatible with the use of an index. (ex. a.id = 30). The rest of the comparison tree is also examined to ensure that no OR clauses are violated by this. If a compatible statement such as (a.id = 30) is an operand to an OR operation, then the index is not used since the entire relation must be loaded anyways. If it is an operand to an AND operation, then the index is used since any tuples where the (a.id != 30) will fail on this condition anyway.
  + Selections
    - All selections are preformed immediately, before any projections, joins, or other operations.
  + Projections
    - Only certain projections can be performed early. Others are required for use in joins (ex. specified in the on clause or as a common attribute for a natural join). All projections that can be performed early are while others are delayed until after joins are complete.
  + Join policies
    - Joins are ordered according to aggregate size of the tables that they involve. Joins with the smallest aggregate size are performed first as they are expected to produce the smallest relation. Joins are also examined for the relative size of the relations that compose them. If the number of tuples in the relations differ by a factor of JOIN\_MULT, then the relations are considered significantly different in size and a nested join is used. JOIN\_MULT is set to a default of 1.5 so one relation must be 50% larger than the other in order for a nested join to be used. The order of the tables may be flipped when nested join is used so that the smaller relation is always the outer relation. A sort/merge join is used as a default for tables that are closer in size.
  + Aggregate operators
    - Performed last, just before returning the result of the query

Features to add if given more time:

* GROUPBY/HAVING
* Improved optimizer so a larger number of queries use an access by index (using index for small sets of tuples, ex. (a.id < 30)). This would involve writing a basic cost-based optimizer that stores information about the distribution of each attribute for each relation.
* Improving underlying memory structures to respect page sizes and to use a block-based access method
* Look into a multi-threaded approach to joins