CS 5300 Project #1

Jared Rainwater & Samuel K. Grush

October 30, 2017

1 The Compiler

In order to parse SQL commands, we are using a parsing library called **PEG.js**, which allows us to express a/n SQL syntax as a *Parsing Expression Grammar* (PEG), and build that grammar into a JavaScript parser. The grammar was initially structured after Phoenix's SQL grammar, but generally follows PostgreSQL's syntax and the corresponding ANSI SQL standard.

1.1 Grammar Rules

The grammar is defined in `src/parser/peg/sql.pegjs`.

Parsing starts out with the `Statements` rule, which is a semicolon delimited list of SQL `Statement`s. A `Statement` can be either a `Select` or `SelectPair`. `Select` is broken up into 6 clauses: `TargetClause`, `FromClause`, `WhereClause`, `GroupByClause`, `HavingClause` and `OrderByClause`. These correspond to all the possibilities of a valid SQL Select statement. A `SelectPair` is two seperate `Select` clauses paired together with a "UNION", "INTERSECT", or "EXCEPT" set operation. You can also apply the "ALL" or "DISTINCT" modifier to the pair.

The `TargetClause` can have the optional "DISTINCT" or "ALL" modifier followed by "*" (to allow everything) or a `TargetList`, a comma-delimited list of `TargetItem`s. A `TargetItem` is a column-like specifier; it can be a relation name with ".*" or an `Operand` with optional alias.

`FromClause` aliases `RelationList`, a list of comma-delimited relation-like fields, each of which may be a table name (with optional alias) or a `Join`. A `JOIN` is a pair of relation-like fields joined by a join-type ("CROSS", "INNER", "LEFT", etc) followed by an optional join-condition ("ON `Condition`" or "USING (`TargetList`)").

`WhereClause` and `HavingClause` are `Condition`s. The types of `Condition`s are: "OR" and "AND" (which join two `Condition`s); comparison, "LIKE", and "BETWEEN" (which join two `Operands`); and "IN" and "EXISTS" (which take `Select`-like arguments).

`GroupByClause` is simply a `TargetList` like the target clause. `OrderByClause` is a comma-delimited list of `Operand`s, each optionally with an ordering-condition ("ASC", "DESC" "USING ...").

An `Operand` is a `Term` optionally joined to other `Operand`s by value operations (e.g. arithmetic or concatenation). A `Term` is a `Literal`, aggregate function, or column reference. `Literal`s include numeric literals, booleans literals, and string literals (single-quoted).

A `Name`, which might refer to an operand or relation, is denoted by a bare-identifier (`/[a-z_][a-z0-9_]*/` and not a `ReservedWord`) or any string quoted with double-quotes (`"..."`) or backticks (`` `...` ``).

Both comment forms are supported: starting with `--` and consuming the rest of the line, and C-style starting with `/*` and ending at `*/`. Both are permitted anywhere whitespace is.

The `ReservedWord` rule contains 340 keywords that the ISO/ANSI SQL:2008 standard states are **never allowed as identifiers**. This set is almost certainly overkill, as most SQL implementations only reserve a *small* fraction of it. It is also excessively large, making up over $^1/_3$ of the grammar's sourcecode and **90%** of the uncompressed compiled grammar.

1.2 Interpretation

Classes and data structures discussed in this section defined in `src/parser/types.ts`.

While parsing the grammar, the PEG.js parser calls JavaScript classes that correspond to SQL concepts. These classes include `SqlSelect`, `SqlJoin`, `SqlConditional`, `SqlLiteral`, etc. This generates an object-oriented data structure—resembling a tree—that represents the "SQL Structure".

Once the SQL Structure is generated it can be converted into JavaScript classes that correspond to Relational Algebra concepts. These classes include `RelRestriction`, `RelProjection`, `RelJoin`, `RelConditional`, etc. This generates a data structure—more closely resembling a tree than before— that represents the "Relational Algebra Structure".

Top-level functions for parsing/conversion defined in `src/parser/parsing.ts`, *with conversion implementation functions defined in* `src/parser/sqlToRel.ts`.