CS 5300 Project #1

Jared Rainwater & Samuel K. Grush

October 31, 2017

Contents

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 `Conditions`. The types of `Conditions` are: "OR" and "AND" (which join two `Conditions`); 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 `Operands`, each optionally with an ordering-condition ("ASC", "DESC" "USING ...").

An `Operand` is a `Term` optionally joined to other `Operands` by value operations (e.g. arithmetic or concatenation). A `Term` is a `Literal`, aggregate function, or column reference. `Literals` 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`.

2 Source Code

All of this code is available at `https://github.com/SKGrush/sqlparse5300`

2.1 src/

2.1.1 src/index.ts

```
import * as React from "react"
import * as ReactDOM from "react-dom"

import './styles/tests.scss'

import Main from './Main'

ReactDOM.render(
  React.createElement(Main),
  document.getElementById("content")
)
```

2.1.2 src/Main.tsx

```
import * as React from "react"

import * as JSONPretty from 'react-json-pretty'

const Tracer = require('pegjs-backtrace')

import {Catalog} from 'parser/types'

import RelationsInput, {RelationsInputOutput} from './components/RelationsInput
    '
import QueryInput from './components/QueryInput'
import Tests from './components/Tests'
import TestCase from './components/TestCase'

export interface MainState {
  queryInputText: string
  status: string
  queryJSON: any
  relJSON: any
  catalog: Catalog | null

  debug: string
}

export default class Main extends React.Component<any, MainState> {

  constructor(props: any) {
    super(props)
    this.state = {
      queryInputText: "",
      status: "",
      catalog: null,
      queryJSON: null,
      relJSON: null,
      debug: ""
```

```
    }

    this.onRelationsInputUpdate = this.onRelationsInputUpdate.bind(this)
    this.onQueryInputUpdate = this.onQueryInputUpdate.bind(this)
  }

  onRelationsInputUpdate(output: RelationsInputOutput) {
    if (output.error) {
      this.setState({
        catalog: null,
        status: `Error Parsing Relations:  ${output.error}`,
        debug: output.traceback
      })
    } else {
      this.setState({
        catalog: output.catalog,
        status: "Successfully Parsed Relations",
        debug: ''
      })
    }
  }

  onQueryInputUpdate(text: string): void {
    this.setState({
      status: "Parsing Query...",
      queryInputText: text,
      queryJSON: null,
      relJSON: null,
      debug: ""
    })

  }

  render() {
    return (
      <main id="main">
        <RelationsInput onUpdate={this.onRelationsInputUpdate} />
        <QueryInput
          onUpdate={this.onQueryInputUpdate}
          disabled={!this.state.catalog}
        />
        <div id="parse-status">{this.state.status}</div>
        <div id="main-output">
          <TestCase
            catalog={this.state.catalog}
            queryInputText={this.state.queryInputText}
            doRun={true} // bad idea??
            anchor="main-test"
            name="Main Test"
          />
          <div id="debug-output" data-empty={!this.state.debug}>
            <pre><code>{this.state.debug}</code></pre>
          </div>
        </div>
        <hr />
        <hr />
        <Tests catalog={this.state.catalog} />
      </main>
    )
```

```
  }
}
```

2.2 src/components

2.2.1 src/components/QueryInput.tsx

```
import * as React from "react"

export interface QueryInputProps {
  onUpdate: (text: string) => void
  disabled: boolean
}

export default class QueryInput extends React.Component<QueryInputProps, any> {
  textInput: HTMLTextAreaElement

  constructor(props: QueryInputProps) {
    super(props)

    this.onSubmit = this.onSubmit.bind(this)
  }

  onSubmit(e?) {
    if (e) e.preventDefault()
    console.info("Submitting:", this.textInput.value)
    this.props.onUpdate(this.textInput.value)
  }
  render() {
    return (
      <div id="query-input-wrapper">
        <textarea
          id="query-input"
          placeholder="Query..."
          cols={80}
          rows={10}
          ref={(input: HTMLTextAreaElement) => {this.textInput = input}}
        />
        <button
          disabled={this.props.disabled}
          onClick={this.onSubmit}
        >Parse Query</button>
      </div>
    )
  }
}
```

2.2.2 src/components/RelationsInput.tsx

```
import * as React from "react"

const Tracer = require('pegjs-backtrace')

import {parseRelations} from '../parser/parsing'
import {Catalog} from '../parser/types'
```

```
7
8  const DEFAULT_INPUT = `
9  Sailors(sid:integer, sname:string, rating:integer, age:real)
10 Boats(bid:integer, bname:string, color:string)
11 Reserves(sid:integer, bid:integer, day:date)
12 `
13
14 export interface RelationsInputOutput {
15   catalog: Catalog | null
16   error: null | Error
17   traceback: '' | string
18 }
19
20 export interface RelationsInputProps {
21   onUpdate: (output: RelationsInputOutput) => void
22 }
23
24 interface RelationsInputState {
25   catalog: Catalog | null
26   text: string
27 }
28
29 export default class RelationsInput extends React.Component<RelationsInputProps
       , RelationsInputState> {
30
31   constructor(props) {
32     super(props)
33     this.state = {
34       catalog: null,
35       text: DEFAULT_INPUT
36     }
37
38     this.run = this.run.bind(this)
39     this.onChange = this.onChange.bind(this)
40   }
41
42   run(e?) {
43     const text = this.state.text
44     if (e) e.preventDefault()
45
46     const tracer = new Tracer(text, {
47       useColor: false,
48       showTrace: true
49     })
50
51     let catalog: Catalog|null = null
52     try {
53       catalog = parseRelations(text, {tracer})
54       this.props.onUpdate({ catalog, error: null, traceback: '' })
55     } catch (ex) {
56       this.props.onUpdate({
57         catalog,
58         error: ex,
59         traceback: tracer.getParseTreeString()
60       })
61     }
62     this.setState({catalog})
63   }
64
```

```
65    onChange(event) {
66      this.setState({text: event.target.value})
67    }
68
69    render() {
70      return (
71        <div id="relations-input-wrapper">
72          <textarea
73            id="relations-input"
74            value={this.state.text}
75            cols={80}
76            rows={10}
77            onChange={this.onChange}
78          />
79          <button onClick={this.run}>Parse Relations</button>
80        </div>
81      )
82    }
83  }
```

2.2.3 src/components/TestCase.tsx

```
1   import * as React from "react"
2   import * as JSONPretty from 'react-json-pretty'
3   const Tracer = require('pegjs-backtrace')
4
5   import {Catalog} from '../parser/types'
6   import {parseSql, SqlSyntaxError, sqlToRelationalAlgebra} from '../parser/
        parsing'
7   import {htmlHLR} from '../parser/relationalText'
8
9   import {Projection} from '../query_tree/operation'
10  import Node from '../query_tree/node'
11  import Tree from '../components/tree'
12
13  interface TestCaseProps {
14    catalog: Catalog | null
15    queryInputText: string
16    doRun: boolean
17    anchor: string
18    name?: string
19  }
20
21  interface TestCaseState {
22    status: string
23    treeStatus: string
24    queryJSON: any
25    relAlJSON: any
26    root: Node | null
27    relAlHTML: JSX.Element | null
28    color: string
29    tscolor: string
30    debug: any
31  }
32
33  export default class TestCase extends React.Component<TestCaseProps,
        TestCaseState> {
```

```
34    constructor(props) {
35      super(props)
36      this.state = this.initialState()
37      this.run = this.run.bind(this)
38    }
39
40    componentDidMount() {
41      this.propsReceived(this.props)
42    }
43
44    componentWillReceiveProps(newProps: TestCaseProps) {
45      this.propsReceived(newProps)
46    }
47
48    propsReceived(newProps: TestCaseProps) {
49      const {catalog, queryInputText, doRun} = this.props
50      if (newProps.catalog !== catalog ||
51          newProps.queryInputText !== queryInputText ||
52          newProps.doRun !== doRun
53         ) {
54        this.setState(this.initialState(), () => {
55          if (newProps.catalog && newProps.queryInputText && newProps.doRun)
56            this.run(newProps)
57        })
58      }
59    }
60
61    initialState(): TestCaseState {
62      return {
63        status: 'init',
64        treeStatus: '',
65        queryJSON: null,
66        relAlJSON: null,
67        relAlHTML: null,
68        root: null,
69        color: 'currentcolor',
70        tscolor: 'currentcolor',
71        debug: ''
72      }
73    }
74
75    run(props: TestCaseProps = this.props) {
76
77      const catalog = props.catalog as Catalog
78
79      const tracer = new Tracer(props.queryInputText, {
80        useColor: false,
81        showTrace: true
82      })
83
84      let status = ''
85      let treeStatus = ''
86      let queryJSON = null
87      let relAlJSON = null
88      let relAlHTML = null
89      let root: Node | null = null
90      let color = 'currentcolor'
91      let tscolor = 'currentcolor'
92      let debug = ''
```

```
93
94      try {
95        queryJSON = parseSql(props.queryInputText, {tracer})
96        status = "SQL Scanned and Tokenized"
97        color = "green"
98      } catch (ex) {
99        if (ex instanceof SqlSyntaxError)
100         status = `Parser Syntax Error: ${ex.message}`
101       else
102         status = `Other Parser ${ex}`
103       console.error(ex)
104       color = "red"
105       debug = tracer.getParseTreeString()
106     }
107
108     if (queryJSON) {
109       try {
110         relAlJSON = sqlToRelationalAlgebra(queryJSON, catalog) as any
111         status = "SQL Parsed and converted to Relational Algebra"
112         color = "green"
113       } catch (ex) {
114         status = `Relational Algebra ${ex}`
115         color = "red"
116         console.error(ex)
117       }
118     }
119     if (relAlJSON) {
120       try {
121         relAlHTML = htmlHLR(relAlJSON)
122         status = "Relational Algebra rendered to HTML"
123         color = "green"
124       } catch (ex) {
125         status = `HTML Conversion Error: ${ex}`
126         color = "red"
127         console.error(ex)
128       }
129       try {
130         root = new Node(relAlJSON)
131         status = "Tree Generated"
132         color = "green"
133       } catch (ex) {
134         treeStatus = `Tree Error: ${ex}`
135         tscolor = "red"
136         console.error(ex)
137       }
138     }
139
140     this.setState({
141       status,
142       treeStatus,
143       queryJSON,
144       relAlJSON,
145       relAlHTML,
146       root,
147       color,
148       tscolor,
149       debug
150     })
151   }
```

```
152
153   render() {
154     return (
155       <section id={this.props.anchor} className="testcase">
156         <hr />
157         <h3>{this.props.name || this.props.anchor}</h3>
158         <pre><code>{this.props.queryInputText}</code></pre>
159         <div className="testcase-status">
160           <span style={{color: this.state.color}}>
161             Status: {this.state.status || "OK"}
162           </span>
163           { this.state.treeStatus && (
164             <span style={{color: this.state.tscolor}}>
165               Tree Status: {this.state.treeStatus}
166             </span>
167           )}
168         </div>
169         <div className="testcase-inner">
170           <div className="relal-html" data-empty={!this.state.relAlHTML}>
171             <h4>Relational Algebra</h4>
172             {this.state.relAlHTML}
173           </div>
174           <div className="sql-json" data-empty={!this.state.queryJSON}>
175             <h4>SQL Structure</h4>
176             <JSONPretty json={this.state.queryJSON} />
177           </div>
178           <div className="relal-json" data-empty={!this.state.relAlJSON}>
179             <h4>Relational Algebra Structure</h4>
180             <JSONPretty json={this.state.relAlJSON} />
181           </div>
182           <div className="tree" data-empty={!this.state.root}>
183             <h4>Tree</h4>
184             { this.state.root &&
185                 <Tree root={this.state.root} margin={10} />
186             }
187           </div>
188           <div className="traceback" data-empty={!this.state.debug}>
189             <h4>Error Traceback</h4>
190             <pre><code>{this.state.debug}</code></pre>
191           </div>
192         </div>
193       </section>
194     )
195   }
196 }
```

2.2.4 src/components/Tests.tsx

```
1 import * as React from "react"
2
3 import {Catalog} from '../parser/types'
4 import TestCase from './TestCase'
5 import {selectTests} from "../parser/tests"
6
7 export function getTestName(testStr: string) {
8   if (testStr.startsWith('--'))
9     return testStr.split("\n", 1)[0].slice(2).trim()
```

```
  return ''
}

interface TestsProps {
  catalog: Catalog | null
}

interface TestsState {
  catalog: Catalog | null
  doRun: boolean
  queryNames: string[]
}

export default class Tests extends React.Component<TestsProps, TestsState> {
  constructor(props) {
    super(props)
    this.state = {
      catalog: props.Catalog,
      doRun: false,
      queryNames: selectTests.map(getTestName)
    }

    this.run = this.run.bind(this)
  }

  componentWillReceiveProps(nextProps: TestsProps) {
    const catalog = nextProps.catalog
    if (catalog !== this.props.catalog)
      this.setState({
        catalog,
        doRun: false
      })
  }

  run(e?) {
    if (e) e.preventDefault()
    if (this.state.catalog)
      this.setState({
        doRun: true
      })
  }

  render() {
    return (
      <div id="tests-div">
        <h2>Test Cases</h2>
        <button
          onClick={this.run}
          disabled={!this.state.catalog}
        >Run Tests</button>
        <nav id="tests-nav">
          <ol>
            {
              this.state.queryNames.map((qName, idx) => {
                const anchor = `#q${idx}`
                return (
                  <li key={anchor}>
                    <a href={anchor}>{qName || anchor}</a>
                  </li>
```

```
                )
              })
            }
          </ol>
        </nav>
        <div id="tests-list">
          {
            selectTests.map((testStr, idx) => (
              <TestCase
                queryInputText={testStr}
                catalog={this.state.catalog}
                doRun={this.state.doRun}
                key={idx}
                anchor={`q${idx}`}
                name={this.state.queryNames[idx] || undefined}
              />
            ))
          }
        </div>
      </div>
    )
  }
}
```

2.2.5 src/components/tree.tsx

```
import * as React from 'react'
import Node from 'query_tree/node'
import '../styles/tree.scss'

interface TreeProps {
  root: Node
  margin: number
}

export default
class Tree extends React.Component<TreeProps, any> {
  render() {
      const rows: JSX.Element[] = []
      let frontier: Node[] = [this.props.root]
      let key = 0
      while (frontier.length > 0) {
        const node: Node = frontier.shift() as Node
        const row = <TreeRow
                      node={node}
                      key={key}
                      offset={node.depth}/>
        rows.push(row)
        frontier = node.children.concat(frontier)
        key++
      }
      return (
      <div>
        {rows}
      </div>
      )
  }
```

```
}

interface TreeRowProps {
  offset: number
  node: Node
}

class TreeRow extends React.Component<TreeRowProps, any> {
  render() {
    return (
      <div className="tree-row">
        {"-".repeat(this.props.offset) + `${this.props.offset})`} {this.props.
            node.operation.html}
      </div>
    )
  }
}
```

2.3 src/parser

2.3.1 src/parser/parsing.ts

```

import { parse as RelationParse } from './peg/relations'
import { parse as SqlParse } from './peg/sql'
export { SyntaxError as SqlSyntaxError } from './peg/sql'
import * as types from './types'
import {fromSqlSelect, fromSelectPair} from './sqlToRel'

export function parseRelations(input: string, args?): types.Catalog {
  return types.Catalog.fromParse(RelationParse(input, args))
}

export function parseSql(input: string, args?) {
  return SqlParse(input, args)
}

export function sqlToRelationalAlgebra(sqlStatements, catalog: types.Catalog) {
  if (!Array.isArray(sqlStatements))
    throw new Error("Expected SQL statements")
  if (sqlStatements.length > 1)
    throw new Error("Multiple statements not supported")

  const TLStatement = sqlStatements[0]
  if (TLStatement instanceof types.SqlSelect)
    return fromSqlSelect(TLStatement, catalog)
  else if (TLStatement instanceof types.SqlSelectPair)
    return fromSelectPair(TLStatement, catalog)
  else
    throw new Error(`Unknown sqlToRelationalAlgebra arg ${TLStatement}`)
}
```

2.3.2 src/parser/relationalText.tsx

```
import * as React from 'react'
```

```
2
3  import * as types from './types'
4
5  export function getSymbol(input: string) {
6    switch (input) {
7      // passthroughs
8      case '||':
9      case '+':
10     case '-':
11     case '*':
12     case '/':
13     case '<':
14     case '>':
15       return input
16
17     case 'restriction':
18       return "ÏČ"
19     case 'projection':
20       return "Îă"
21     case 'rename':
22       return "ÏĄ"
23     case 'rename-divider':
24       return "âĹŢ"
25
26     case 'union':
27       return "âĹŧ"
28     case 'intersect':
29       return "âĹľ"
30     case 'except':
31       return "âĹŠ"
32
33     case 'join':
34       return "âŃĹ"
35     case 'left':
36     case 'ljoin':
37       return "âŃĽ"
38     case 'right':
39     case 'rjoin':
40       return "âŃŁ"
41     case 'cross':
42     case 'crossjoin':
43       return "âĺĽ"
44     case 'divide':
45       return "Ãů"
46
47     case 'eq':
48       return "="
49     case 'neq':
50       return "âľă"
51     case 'leq':
52       return "âľđ"
53     case 'geq':
54       return "âľě"
55     case 'and':
56       return "âĹğ"
57     case 'or':
58       return "âĹĺ"
59     case 'in':
60       return "âĹŁ"
```

```
    default:
      throw new Error(`Unknown symbol name "${input}"`)
  }
}

export function htmlARGS(args: types.HighLevelRelationish, noargs = false) {
  if (noargs) {
    return null
  } else {
    const ARGS = htmlHLR(args)
    return (
      <span className="args">
        (
          <span className="HLR">
            {ARGS}
          </span>
        )
      </span>
    )
  }
}

export function htmlRelRestriction(res: types.RelRestriction, noargs = false) {
  const SYM = getSymbol('restriction')
  const COND = htmlRelConditional(res.conditions)
  const ARGS = htmlARGS(res.args, noargs)
  return (
    <span className="RelRestriction">
      <span className="operator">{SYM}</span>
      <sub className="condition">
        {COND}
      </sub>
      {ARGS}
    </span>
  )
}

export function htmlRelProjection(res: types.RelProjection, noargs = false) {
  const SYM = getSymbol('projection')
  const COLUMNS: Array<string|HTMLSpanElement> = []
  res.columns.forEach((col, idx) => {
    if (idx > 0)
      COLUMNS.push(",")
    if (col instanceof types.RelColumn)
      COLUMNS.push(htmlRelColumn(col, idx))
    else if ((col as any) instanceof types.RelFunction)
      COLUMNS.push(htmlRelFunction(col, idx))
    else
      COLUMNS.push(col)
  })
  const ARGS = htmlARGS(res.args, noargs)
  return (
    <span className="RelProjection">
      <span className="operator">{SYM}</span>
      <sub className="columns">
        {COLUMNS}
      </sub>
      {ARGS}
    </span>
```

```
  )
}

export function htmlRelColumn(col: types.RelColumn, iter?: number) {

  if (col.as) {
    return (
      <span className="RelColumn" key={iter}>
        <span className="column-as">{col.as}</span>
      </span>
    )
  }

  if (!col.relation) {
    return (
      <span className="RelColumn" key={iter}>
        <span className="column-name">{getName(col.target)}</span>
      </span>
    )
  }

  return (
    <span className="RelColumn" key={iter}>
      <span className="relation-name">{getName(col.relation)}</span>
      .
      <span className="column-name">{getName(col.target)}</span>
    </span>
  )
}

export function htmlRelFunction(funct: types.RelFunction, idx?) {
  const NAME = funct.fname.toUpperCase()
  const EXPR = funct.expr === '*'
          ? '*'
          : htmlRelColumn(funct.expr)

  return (
    <span className="RelFunction" key={idx}>
      <span className="function-name">{NAME}</span>
      (
        {EXPR}
      )
    </span>
  )
}

export function getName(thing) {
  if (typeof(thing) === 'string')
    return thing
  if (thing instanceof types.RelRelation)
    return thing.name
  if (thing instanceof types.RelColumn)
    return thing.as || htmlRelColumn(thing)
  if (thing instanceof types.RelFunction)
    return htmlRelFunction(thing as types.RelFunction)
  if (thing instanceof types.Column)
    return thing.name
  console.info("getName", thing)
  throw new Error("unexpected thing to getName")
```

```
}

export function htmlRelRename(ren: types.RelRename, noargs = false) {
  const SYM = getSymbol('rename')
  const INPUT = getName(ren.input)
  const OUTPUT = ren.output
  const ARGS = htmlARGS(ren.args, noargs)

  return (
    <span className="RelRename">
      <span className="operator">{SYM}</span>
      <sub className="condition">
        {OUTPUT} {getSymbol('rename-divider')} {INPUT}
      </sub>
      {ARGS}
    </span>
  )
}

export function htmlRelRelation(rel: types.RelRelation) {
  const NAME = rel.name
  return (
    <span className="RelRelation">
      {NAME}
    </span>
  )
}

export function relJoinHelper(join: types.RelJoin): [string, JSX.Element | null
    ] {
  if (typeof(join.condition) === 'string') {
    return [getSymbol(join.condition), null]
  } else if (join.condition instanceof types.RelConditional) {
    let cond = htmlRelConditional(join.condition)
    if (cond) {
      cond = (
        <sub className="condition">
          {cond}
        </sub>
      )
    }
    return [getSymbol('join'), cond]
  } else {
    throw new Error(`unknown RelJoin condition ${join.condition}`)
  }
}

export function htmlRelJoin(join: types.RelJoin) {
  const [joinSymbol, cond] = relJoinHelper(join)
  const LHS = htmlHLR(join.lhs)
  const RHS = htmlHLR(join.rhs)

  return (
    <span className="RelJoin">
      {LHS}
      <span className="operator">{joinSymbol}</span>
      {cond}
      {RHS}
    </span>
```

```
237   )
238 }
239
240 export function htmlRelOperation(op: types.RelOperation) {
241   const OPSYM = getSymbol(op.op)
242   const LHS = htmlRelOperand(op.lhs as any)
243   const RHS = htmlRelOperand(op.rhs as any)
244
245   return (
246     <span className="RelOperation">
247       {LHS}
248       <span className="operator">{OPSYM}</span>
249       {RHS}
250     </span>
251   )
252 }
253
254 export function htmlRelOperand(operand: types.RelOperandType) {
255   if (typeof(operand) === 'string')
256     return operand
257   if (operand instanceof types.RelFunction)
258     return htmlRelFunction(operand)
259   if (operand instanceof types.RelOperation)
260     return htmlRelOperation(operand)
261   if (operand instanceof types.RelColumn)
262     return htmlRelColumn(operand)
263   // throw new Error("Unexpected operand type")
264   return htmlHLR(operand)
265 }
266
267 export function htmlRelConditional(cond: types.RelConditional) {
268   const OPSYM = getSymbol(cond.operation)
269   const LHS = cond.lhs instanceof types.RelConditional
270           ? htmlRelConditional(cond.lhs)
271           : htmlRelOperand(cond.lhs)
272   const RHS = cond.rhs instanceof types.RelConditional
273           ? htmlRelConditional(cond.rhs)
274           : ( cond.rhs instanceof Array
275               ? cond.rhs.map(htmlRelOperand)
276               : htmlRelOperand(cond.rhs)
277             )
278
279   return (
280     <span className="RelConditional">
281       <span className="lhs">
282         {LHS}
283       </span>
284       <span className="operator">{OPSYM}</span>
285       <span className="rhs">
286         {RHS}
287       </span>
288     </span>
289   )
290 }
291
292 export function htmlHLR(hlr: types.HighLevelRelationish) {
293   if (hlr instanceof types.RelRestriction)
294     return htmlRelRestriction(hlr)
295   if (hlr instanceof types.RelProjection)
```

```
296       return htmlRelProjection(hlr)
297     if (hlr instanceof types.RelRename)
298       return htmlRelRename(hlr)
299     if (hlr instanceof types.RelOperation)
300       return htmlRelOperation(hlr)
301     if (hlr instanceof types.RelRelation)
302       return htmlRelRelation(hlr)
303     if (hlr instanceof types.RelJoin)
304       return htmlRelJoin(hlr)
305     console.error("unknown HLR:", hlr)
306     throw new Error("Unknown type passed to htmlHLR")
307   }
```

2.3.3 src/parser/sqlToRel.ts

```
1
2   import * as types from './types'
3
4   type ColumnValueType = types.RelColumn | types.RelFunction | string
5
6   type RelationLookup = Map<string, types.RelRelation>
7
8   /* bubble a join/relation up to the calling function, also returning
9      the 'realOperation' that took place */
10  class BubbleUp<T> {
11    realOperation: T
12    relationish: types.HighLevelRelationish
13
14    constructor(realOp: T, relationish: types.HighLevelRelationish) {
15      this.realOperation = realOp
16      this.relationish = relationish
17    }
18  }
19
20  class RenameBubbleUp {
21    target: ColumnValueType
22    output: string
23
24    constructor(target: ColumnValueType, output: string) {
25      this.target = target
26      this.output = output
27    }
28  }
29
30  class ColumnLookup {
31    readonly map: Map<string, types.RelColumn[]>
32    readonly catalog: types.Catalog
33    readonly relations: RelationLookup
34
35    constructor(catalog: types.Catalog, relations: RelationLookup, init?) {
36      this.map = new Map(init)
37      this.catalog = catalog
38      this.relations = relations
39    }
40
41    addAlias(name: string, target: types.RelColumn) {
42      const cols = this.map.get(name)
```

```
43      if (!(target instanceof types.RelColumn)) {
44        target = new types.RelColumn(null, target, name)
45      }
46      if (!cols)
47        this.map.set(name, [target])
48      else
49        cols.push(target)
50      return target
51    }
52
53    lookup(columnName: string, relationName?: string, as?: string): types.
          RelColumn {
54      if (relationName) {
55        // column references a relation
56        if (!this.relations.has(relationName)) {
57          throw new Error(`Unknown relation "${relationName}"`)
58        }
59        const relation = this.relations.get(relationName) as types.RelRelation
60        const catRelation = this.catalog.relations.get(relation.name) as types.
              Relation
61        // if(!catRelation)
62        //   throw new Error(`${relationName} not in catalog`)
63        if (catRelation.columns.has(columnName))
64          return new types.RelColumn(relation,
65                                     catRelation.columns.get(columnName) as types
                                           .Column,
66                                     as)
67        else
68          throw new Error(`${catRelation.name} doesn't contain ${columnName}`)
69      } else {
70        // implicit relation reference
71        if (this.map.has(columnName)) {
72          // already in the map
73          const cols = this.map.get(columnName) as types.RelColumn[]
74          if (cols.length > 1)
75            throw new Error(`Ambiguous column name reference "${columnName}"`)
76
77          return cols[0].alias(as)
78
79        }
80        // not in map; search for columnName
81        console.group()
82        console.info(`Searching for ${columnName}`)
83        for (const val of this.relations.values()) {
84          // if (!this.catalog.relations.has(val.name)) {
85          //   throw new Error(`${val.name} not in catalog`)
86          // }
87          const catRel = this.catalog.relations.get(val.name) as types.Relation
88          console.info(`${val.name} in catalog, looking for ${columnName}`)
89          if (!catRel.columns.has(columnName))
90            continue
91          console.info(`found`)
92          console.groupEnd()
93          const col = catRel.columns.get(columnName) as types.Column
94          return new types.RelColumn(val, col, as)
95        }
96        console.info(`not found`)
97        console.groupEnd()
98        throw new Error(`Unknown column ${columnName}`)
```

```
99      }
100   }
101 }
102
103 function _joinArgHelper(hs: types.SqlJoin | types.SqlRelation,
104                         relations: RelationLookup,
105                         columns: ColumnLookup,
106                         catalog: types.Catalog,
107                         arg: types.SqlJoin,
108                         side): types.RelRelationish {
109   if (hs instanceof types.SqlJoin)
110     return fromJoin(hs, relations, columns, catalog)
111   else if (hs instanceof types.SqlRelation)
112     return fromRelation(hs, relations, columns, catalog) as types.RelRelation
113   console.error(`bad join arg ${side}`, arg, "lookup:", relations)
114   throw new Error("Bad join argument lhs")
115 }
116
117 function fromJoin(arg: types.SqlJoin,
118                   relations: RelationLookup,
119                   columns: ColumnLookup,
120                   catalog: types.Catalog): types.RelJoin {
121   const lhs = _joinArgHelper(arg.lhs, relations, columns, catalog, arg, 'left')
122   const rhs = _joinArgHelper(arg.rhs, relations, columns, catalog, arg, 'right'
      )
123   let cond: any = null
124   if (arg.condition) {
125     if (arg.condition instanceof types.SqlConditional)
126       cond = fromConditional(arg.condition, relations, columns, catalog)
127     else if (Array.isArray(arg.condition) && arg.condition.length === 2)
128       cond = fromTargetList(arg.condition[1], relations, columns, catalog)
129     else {
130       console.error("bad conditional", arg, "lookup:", relations)
131       throw new Error("bad conditional")
132     }
133   } else {
134     switch (arg.joinType) {
135       case "join":
136       case null:
137         cond = "cross"
138         break
139       case "leftouter":
140         cond = "left"
141         break
142       case "rightouter":
143         cond = "right"
144         break
145       case "fullouter":
146         throw new Error("full outer join not supported")
147       // case "natural" | "equi" | null:
148     }
149   }
150
151   const J = new types.RelJoin(lhs, rhs, cond)
152   return J
153 }
154
155 function fromColumn(arg: types.SqlColumn,
156                     relations: RelationLookup,
```

```
157                       columns: ColumnLookup,
158                       catalog: types.Catalog
159     ): RenameBubbleUp | ColumnValueType {
160     const alias = arg.alias
161     let target
162     if (arg.target instanceof types.SqlColumn) {
163       // column of column; either rename it or return target
164       target = fromColumn(arg.target, relations, columns, catalog)
165       if (!alias)
166         console.warn("Why double column?")
167       else if (target instanceof RenameBubbleUp) {
168         console.error("Double rename; arg,target =", arg, target)
169         throw new Error("Double rename not supported")
170       }
171     } else if (typeof(arg.target) === 'string') {
172       // column based on a name
173       target = columns.lookup(arg.target,
174                               arg.relation || undefined,
175                               arg.as || undefined)
176     } else if (arg.target instanceof types.SqlLiteral) {
177       target = fromLiteral(arg.target)
178     } else if (arg.target instanceof types.SqlAggFunction) {
179       target = fromAggFunction(arg.target, relations, columns, catalog)
180     } else {
181       throw new Error("Unexpected type in column")
182     }
183
184     if (alias) {
185       target = columns.addAlias(alias, target)
186       return new RenameBubbleUp(target, alias)
187     }
188     return target
189 }
190
191 function fromTargetList(targetColumns: types.SqlColumn[],
192                         relationLookup: RelationLookup,
193                         columnLookup: ColumnLookup,
194                         catalog: types.Catalog
195     ): [ColumnValueType[], RenameBubbleUp[]] {
196     console.info("fromTargetList:", targetColumns)
197     const renames: RenameBubbleUp[] = []
198     const cols = targetColumns.map((colarg) => {
199       const col = fromColumn(colarg,
200                              relationLookup,
201                              columnLookup,
202                              catalog)
203       if (col instanceof RenameBubbleUp) {
204         renames.push(col)
205         return col.target
206       }
207       return col
208     })
209     return [cols, renames]
210 }
211
212 function fromRelation(arg: types.SqlRelation,
213                       relations: RelationLookup,
214                       columns: ColumnLookup,
215                       catalog: types.Catalog): types.RelRename | types.
```

```
                            RelRelation | types.RelJoin {
  if (typeof(arg.target) === 'string') {
    let relat
    if (relations.has(arg.target))
      relat = relations.get(arg.target)
    else if (catalog.relations.has(arg.target)) {
      relat = new types.RelRelation(arg.target)
      relations.set(arg.target, relat)
    } else {
      console.error(`Unknown relation ${arg.target}`, arg, relations)
      throw new Error(`Unknown relation ${arg.target}`)
    }

    if (arg.alias) {
      const ren = new types.RelRename(relat, arg.alias, relat)
      relations.set(arg.alias, relat)
      return ren
    }
    return relat
  } else if (arg.target instanceof types.SqlRelation) {
    const relat = fromRelation(arg.target, relations, columns, catalog) as
        types.RelRelation
    if (!arg.alias)
      return relat
    const ren = new types.RelRename(relat, arg.alias, relat)
    relations.set(arg.alias, relat)
    return ren
  } else if (arg.target instanceof types.SqlJoin) {
    const J = fromJoin(arg.target, relations, columns, catalog)
    if (!arg.alias)
      return J
    else
      throw new Error("Renaming joins not supported ")
    // const ren = new types.RelRename()
  } else {
    console.error("bad arg.target type", arg, "lookup:", relations)
    throw new Error("bad arg.target type")
  }
}

function fromRelationList(arg: types.RelationList,
                          relations: RelationLookup,
                          columns: ColumnLookup,
                          catalog: types.Catalog) {
  if (arg instanceof types.SqlRelation)
    return fromRelation(arg, relations, columns, catalog)
  else
    return fromJoin(arg, relations, columns, catalog)
}

function fromLiteral(lit: types.SqlLiteral) {
  switch (lit.literalType) {
    case 'string':
      return `'${lit.value}'`
    case 'number':
    case 'boolean':
    case 'null':
      return String(lit.value)
    default:
```

```
273       throw new Error(`Unknown literal type ${lit.literalType} for ${lit.value}
              `)
274   }
275 }
276
277 function fromAggFunction(agg: types.SqlAggFunction,
278                          rels: RelationLookup,
279                          cols: ColumnLookup,
280                          cata: types.Catalog) {
281   switch (agg.fname) {
282     case 'count':
283       if (agg.expr === '*' || (agg.expr as types.TargetClause).targetlist === '
              *')
284         return new types.RelFunction('count', '*')
285       else
286         throw new Error("Counting columns not supported")
287     case 'avg':
288     case 'max':
289     case 'min':
290     case 'sum':
291       if (!(agg.expr instanceof types.SqlColumn))
292         throw new Error(`non-column arguments to aggregates not supported`)
293       const expr = fromColumn(agg.expr, rels, cols, cata) as types.RelColumn
294       return new types.RelFunction(agg.fname, expr)
295     default:
296       throw new Error(`Unknown aggregate function ${agg.fname}`)
297   }
298 }
299
300 function fromOperation(arg: types.SqlOperation,
301                        rels: RelationLookup,
302                        cols: ColumnLookup,
303                        cata: types.Catalog) {
304   const lhs = _condArgHelper(arg.lhs, rels, cols, cata)
305   const rhs = _condArgHelper(arg.rhs, rels, cols, cata)
306   return new types.RelOperation(arg.op, lhs, rhs)
307 }
308
309 /* takes an Operand argument */
310 function _condArgHelper(hs, rels, cols, cata) {
311   if (hs instanceof Array)
312     return fromTargetList(hs, rels, cols, cata)[0]
313   if (hs instanceof types.SqlConditional)
314     return fromConditional(hs, rels, cols, cata)
315   else if (hs instanceof types.SqlSelect)
316     return fromSqlSelect(hs, cata)
317   // Operand
318   else if (hs instanceof types.SqlLiteral)
319     return fromLiteral(hs)
320   else if (hs instanceof types.SqlAggFunction)
321     return fromAggFunction(hs, rels, cols, cata)
322   else if (hs instanceof types.SqlColumn)
323     return fromColumn(hs, rels, cols, cata)
324   else if (hs instanceof types.SqlOperation)
325     return fromOperation(hs, rels, cols, cata)
326   else
327     throw new Error(`Unknown conditional arg type ${hs}`)
328 }
329
```

```
330 function _handleSubquery(arg, lhs, op, relations, columns, catalog) {
331
332   const tmpRhs = (arg.rhs instanceof types.SqlSelectPair)
333                   ? fromSelectPair(arg.rhs, catalog)
334                   : fromSqlSelect(arg.rhs, catalog)
335
336   if (op === 'in')
337     op = 'eq'
338
339   // lhs = check-against
340   // rhs = Selectish
341   if (!(tmpRhs instanceof types.RelProjection))
342     throw new Error("'in' subqueries must select columns")
343
344   const rhsTarget = tmpRhs.columns
345
346   let conditional: types.RelConditional
347   if (rhsTarget.length > 1)
348     conditional = rhsTarget.reduce((L, R) =>
349                     new types.RelConditional(op, L, R), lhs)
350   else
351     conditional = new types.RelConditional(op, lhs, rhsTarget[0])
352
353   return new BubbleUp<types.RelConditional>(conditional, tmpRhs.args)
354 }
355
356 function fromConditional(arg: types.SqlConditional,
357                          relations: RelationLookup,
358                          columns: ColumnLookup,
359                          catalog: types.Catalog
360   ): types.RelConditional | BubbleUp<types.RelConditional> {
361   let binOp = true
362   let op: types.ThetaOp
363   switch (arg.operation) {
364     case 'not':
365     case 'isnull':
366     case 'exists':
367       binOp = false
368       // break
369     /* binary ops */
370     case 'like':
371     case 'between':
372       throw new Error(`"${arg.operation}" condition not yet supported`)
373
374     case 'or':
375     case 'and':
376     case 'in':
377     case '<':
378     case '>':
379       op = arg.operation
380       break
381     case '<>':
382     case '!=':
383       op = 'neq'
384       break
385     case '<=':
386       op = 'leq'
387       break
388     case '>=':
```

```
      op = 'geq'
      break
    case '=':
      op = 'eq'
      break
    default:
      throw new Error(`Unknown op "${arg.operation}"`)
  }
  let lhs = _condArgHelper(arg.lhs, relations, columns, catalog)
  if (lhs instanceof RenameBubbleUp) {
    lhs = lhs.target
  }

  if (op === 'in' && arg.rhs instanceof Array) {
    const rs = arg.rhs.map((R) => {
      const tcond = _condArgHelper(R, relations, columns, catalog)
      if (tcond instanceof RenameBubbleUp)
        return tcond.target
      return tcond
    })
    const cond = new types.RelConditional('in', lhs, rs)
    if (arg.not)
      throw new Error("'not' conditional is not supported")
    return cond
  }
  if (arg.rhs instanceof types.SqlSelect ||
      arg.rhs instanceof types.SqlSelectPair) {
    return _handleSubquery(arg, lhs, op, relations, columns, catalog)
  }
  if (op === 'in') {
    throw new Error("'in' argument should be array or subquery")
  }

  if (!binOp)
    throw new Error("unary operators not supported")
  let rhs = _condArgHelper(arg.rhs, relations, columns, catalog)
  if (rhs instanceof RenameBubbleUp)
    rhs = rhs.target

  const condit = new types.RelConditional(op, lhs, rhs)

  if (arg.not)
    throw new Error("'not' conditional is not supported")
  return condit
}

function fromOrderings(orderings, rels, cols, cata) {
  if (!orderings || !orderings.length)
    return null
  return orderings.map(([col, cond]) => {
    const column = fromColumn(col, rels, cols, cata)
    if (column instanceof RenameBubbleUp)
      return [column.target, cond]
    return [column, cond]
  })
}

export function fromSelectPair(selPair: types.SqlSelectPair,
                               catalog: types.Catalog) {
```

```
448   const lhs = fromSqlSelect(selPair.lhs, catalog)
449   let rhs
450   if (selPair.rhs instanceof types.SqlSelect)
451     rhs = fromSqlSelect(selPair.rhs, catalog)
452   else
453     rhs = fromSelectPair(selPair.rhs, catalog)
454
455   if (lhs instanceof types.RelProjection &&
456       rhs instanceof types.RelProjection) {
457     if (lhs.columns.length !== rhs.columns.length)
458       throw new Error(`Joining on unequal degrees: ` +
459                       `${lhs.columns.length} vs ${rhs.columns.length}`)
460     const newLhs = lhs.args
461     const newRhs = rhs.args
462     const newColumns = lhs.columns
463     const args = new types.RelOperation(selPair.pairing, newLhs, newRhs)
464     return new types.RelProjection(newColumns, args)
465   }
466
467   const operation = new types.RelOperation(selPair.pairing, lhs, rhs)
468   return operation
469 }
470
471 function _renameReducer(arg: types.HighLevelRelationish, ren: RenameBubbleUp) {
472   return new types.RelRename(ren.target, ren.output, arg)
473 }
474
475 function applyRenameBubbleUps(renames: RenameBubbleUp[],
476                               args: types.HighLevelRelationish) {
477     return renames.reduce(_renameReducer, args)
478   }
479
480 export function fromSqlSelect(select: types.SqlSelect, catalog: types.Catalog)
      {
481
482   // map names to the actual instances
483   const relations = new Map()
484   const columns = new ColumnLookup(catalog, relations)
485
486   let fromClause: types.HighLevelRelationish
487       = fromRelationList(select.from, relations, columns, catalog)
488
489   let targetColumns
490   let renames: RenameBubbleUp[] = []
491   if (select.what.targetlist === '*')
492     targetColumns = '*'
493   else {
494     [targetColumns, renames] = fromTargetList(select.what.targetlist,
495                                               relations,
496                                               columns,
497                                               catalog)
498   }
499
500   // const whereClause = select.where
501   //     ? fromConditional(select.where, relations, columns, catalog)
502   //     : null
503   let whereClause: any = null
504   if (select.where) {
505     whereClause = fromConditional(select.where, relations, columns, catalog)
```

```
506     if (whereClause instanceof BubbleUp) {
507       fromClause = new types.RelJoin(fromClause, whereClause.relationish, '
              cross')
508       whereClause = whereClause.realOperation as types.RelConditional
509     }
510   }
511
512   if (renames.length) {
513     fromClause = applyRenameBubbleUps(renames, fromClause)
514   }
515
516   const groupBy = select.groupBy
517       ? fromTargetList(select.groupBy, relations, columns, catalog)
518       : null
519
520   const having = select.having
521       ? fromConditional(select.having, relations, columns, catalog)
522       : null
523
524   const orderBy = fromOrderings(select.orderBy, relations, columns, catalog)
525
526   const Rest = whereClause
527       ? new types.RelRestriction(whereClause, fromClause)
528       : fromClause
529
530   const Proj = targetColumns === '*'
531       ? Rest
532       : new types.RelProjection(targetColumns, Rest)
533
534   return Proj
535 }
```

2.3.4 src/parser/tests.ts

```
1
2  export const selectTests = [
3
4  `-- Query 2a
5  SELECT    S.sname
6  FROM      Sailors AS S, Reserves AS R
7  WHERE     S.sid=R.sid AND R.bid=103`,
8
9  `-- Query 2b
10 SELECT    S.sname
11 FROM      Sailors AS S, Reserves AS R, Boats AS B
12 WHERE     S.sid=R.sid AND R.bid=B.bid AND B.color=âĂŹredâĂŹ`,
13
14 `-- Query 2c
15 SELECT    sname
16 FROM      Sailors, Boats, Reserves
17 WHERE     Sailors.sid=Reserves.sid AND Reserves.bid=Boats.bid AND
18 Boats.color=âĂŹredâĂŹ
19 UNION
20 SELECT    sname
21 FROM      Sailors, Boats, Reserves
22 WHERE     Sailors.sid=Reserves.sid AND Reserves.bid=Boats.bid AND
23 Boats.color='green'`,
```

```
24
25  `-- Query 2d (invalid)
26  -- unescaped reserve word 'day', invalid reference 'R.rating'
27  SELECT    S.sname
28  FROM      Sailors AS S, Reserves AS R
29  WHERE     R.sid = S.sid AND R.bid = 100 AND R.rating > 5 AND R.day =
30  âĂŸ8/9/09âĂŹ`,
31
32  `-- Modified Query2d (invalid)
33  -- still unknown reference 'R.rating'
34  SELECT    S.sname
35  FROM      Sailors AS S, Reserves AS R
36  WHERE     R.sid = S.sid AND R.bid = 100 AND R.rating > 5 AND R.\`day\` =
37  âĂŸ8/9/09âĂŹ`,
38
39  `-- Query 2e
40  SELECT    sname
41  FROM      Sailors, Boats, Reserves
42  WHERE     Sailors.sid=Reserves.sid AND Reserves.bid=Boats.bid AND
43  Boats.color=âĂŹredâĂŹ
44  INTERSECT
45  SELECT    sname
46  FROM      Sailors, Boats, Reserves
47  WHERE     Sailors.sid=Reserves.sid AND Reserves.bid=Boats.bid AND
48  Boats.color=âĂŹgreenâĂŹ`,
49
50  `-- Query 2f (invalid)
51  -- illegal identifier '2color' of B
52  SELECT    S.sid
53  FROM      Sailors AS S,  Reserves AS R, Boats AS B
54  WHERE     S.sid=R.sid AND R.bid=B.bid AND B.color=âĂŸredâĂŹ
55  EXCEPT
56  SELECT    S2.sid
57  FROM      Sailors AS S2,  Reserves AS R2, Boats AS B2
58  WHERE     S2.sid=R2.sid AND R2.bid=B2.bid AND B.2color=âĂŸgreenâĂŹ`,
59
60  `-- Modified Query 2f
61  SELECT    S.sid
62  FROM      Sailors AS S,  Reserves AS R, Boats AS B
63  WHERE     S.sid=R.sid AND R.bid=B.bid AND B.color=âĂŸredâĂŹ
64  EXCEPT
65  SELECT    S2.sid
66  FROM      Sailors AS S2,  Reserves AS R2, Boats AS B2
67  WHERE     S2.sid=R2.sid AND R2.bid=B2.bid AND B2.color=âĂŸgreenâĂŹ`,
68
69  `-- Query 2g (invalid)
70  -- unknown reference 'Reserve'
71  SELECT    S.sname
72  FROM      Sailors AS S
73  WHERE     S.sid IN ( SELECT   R.sid
74                       FROM     Reserve AS R
75                       WHERE    R.bid = 103)`,
76
77  `-- Modified Query 2g
78  SELECT    S.sname
79  FROM      Sailors AS S
80  WHERE     S.sid IN ( SELECT   R.sid
81                       FROM     Reserves AS R
82                       WHERE    R.bid = 103)`,
```

```
83
84  `-- Query 2h (invalid)
85  -- unknown reference 'Reserve'
86  SELECT    S.sname
87  FROM      Sailors AS S
88  WHERE     S.sid IN ((SELECT   R.sid
89                       FROM     Reserve AS R, Boats AS B
90                       WHERE    R.bid = B.bid AND B.color = âĂŸredâĂŹ)
91                     INTERSECT
92                     (SELECT   R2.sid
93                       FROM     Reserve AS R2, Boats AS B2
94                       WHERE    R2.bid = B2.bid AND B2.color = âĂŸgreenâĂŹ))`,
95
96  `-- Modified Query 2h
97  SELECT    S.sname
98  FROM      Sailors AS S
99  WHERE     S.sid IN ((SELECT   R.sid
100                      FROM     Reserves AS R, Boats AS B
101                      WHERE   R.bid = B.bid AND B.color = âĂŸredâĂŹ)
102                    INTERSECT
103                    (SELECT   R2.sid
104                      FROM     Reserves AS R2, Boats AS B2
105                      WHERE   R2.bid = B2.bid AND B2.color = âĂŸgreenâĂŹ))`,
106
107 `-- Query 2i (invalid)
108 -- bad inner condition string, also unknown reference 'R'
109 SELECT   S.sname
110 FROM      Sailors AS S
111 WHERE     S.age > ( SELECT    MAX (S2.age)
112                     FROM      Sailors S2
113                     WHERE    R.sid = S2.rating = 10)`,
114
115 `-- Modified Query 2i
116 SELECT   S.sname
117 FROM      Sailors AS S
118 WHERE     S.age > ( SELECT    MAX (S2.age)
119                     FROM      Sailors S2
120                     WHERE     S2.rating = 10)`,
121
122 `-- Query 2j
123 SELECT   B.bid, Count (*) AS reservationcount
124 FROM      Boats B, Reserves R
125 WHERE     R.bid=B.bid AND B.color = âĂŸredâĂŹ
126 GROUP BY  B.bid`,
127
128 `-- Query 2k
129 SELECT    B.bid, Count (*) AS reservationcount
130 FROM      Boats B,  Reserves R
131 WHERE     R.bid=B.bid AND B.color = âĂŸredâĂŹ
132 GROUP BY  B.bid
133 HAVING    B.color = âĂŸredâĂŹ`,
134
135 `-- Query 2l (invalid)
136 -- typo "SLECT", misuse of nonstandard 'contains' WHERE predicate, 'Sname'
137 SELECT    Sname
138 FROM      Sailors
139 WHERE     Sailor.sid IN (SELECT   Reserves.bid, Reserves.sid
140                          FROM     Reserves
141                          CONTAINS
```

```
                                    (SLECT Boats.bid
                                     FROM  Boats
                                     WHERE Boats.name  =  âĂŸinterlakeâĂŹ) )`,

`-- Modified Query 2l (invalid, system-specific)
-- unknown reference 'Sname'
SELECT    Sname
FROM      Sailors
WHERE     Sailor.sid IN (SELECT   Reserves.bid, Reserves.sid
                         FROM     Reserves
                         WHERE    EXISTS (
                                  SELECT Boats.bid
                                  FROM  Boats
                                  WHERE Boats.name  =  âĂŸinterlakeâĂŹ
                                        AND Boats.bid = Reserves.bid ) )`,

`-- Query 2m (invalid)
-- Bad TargetList
SELECT    S.rating, Ave (S.age) As average
FROM      Sailors S
WHERE     S.age > 18
GROUP BY  S.rating
HAVING    Count (*) > 1`,

`-- Modified Query 2m
SELECT    S.rating, Avg (S.age) As average
FROM      Sailors S
WHERE     S.age > 18
GROUP BY  S.rating
HAVING    Count (*) > 1`
]
```

2.3.5 src/parser/types.ts

```

export const LITERAL_TYPE       = "literal"
export const COLUMN_TYPE        = "column"
export const JOIN_TYPE          = "join"
export const RELATION_TYPE      = "relation"
export const CONDITIONAL_TYPE   = "conditional"
export const AGGFUNCTION_TYPE   = "aggfunction"
export const OPERATION_TYPE     = "operation"
export const SELECTCLAUSE_TYPE  = "selectclause"
export const TARGETCLAUSE_TYPE  = "targetclause"
export const SELECTPAIR_TYPE    = "selectpair"

export const REL_RESTRICTION_TYPE = "restriction"
export const REL_PROJECTION_TYPE  = "projection"
export const REL_RENAME_TYPE      = "rename"

export const REL_RELATION_TYPE    = "relrelation"
export const REL_COLUMN_TYPE      = "relcolumn"
export const REL_CONDITIONAL_TYPE = "relconditional"
export const REL_JOIN_TYPE        = "reljoin"
export const REL_FUNCTION_TYPE    = "relfunt"
export const REL_OPERATION_TYPE   = "relop"

```

```
24 /**
25  * IFF rhs is non-empty, run reduce using f on rhs initialized by lhs.
26  * Else return lhs
27  */
28 export function reduceIfRHS(lhs: any, rhs: any[], f: (L, R) => any) {
29   if (rhs.length)
30     return rhs.reduce(f, lhs)
31   return lhs
32 }
33
34 export class Catalog {
35
36   static fromParse(relations: Array<[string, Array<[string, string]>]>) {
37     const rels = new Map()
38     relations.forEach((ele) => {
39       const [tname, cols] = ele
40       const columnMap = new Map()
41       cols.forEach((col) => {
42         columnMap.set(col[0], new Column(col[0], col[1]))
43       })
44       rels.set(tname, new Relation(tname, columnMap))
45     })
46     return new Catalog(rels)
47   }
48
49   relations: Map<string, Relation>
50
51   constructor(relations: Map<string, Relation>) {
52     this.relations = relations
53   }
54 }
55
56 export class Relation {
57   name: string
58   columns: Map<string, Column>
59
60   constructor(name: string, columns: Map<string, Column>) {
61     this.name = name
62     this.columns = columns
63   }
64 }
65
66 export class Column {
67   name: string
68   typ: string
69
70   constructor(name: string, typ: string) {
71     this.name = name
72     this.typ = typ
73   }
74 }
75
76 export type JOINSTRING = "join"       // "," | "JOIN" | "CROSS JOIN"
77                        | "equi"       // "INNER JOIN" | "JOIN ... USING"
78                        | "natural"    // "NATURAL JOIN"
79                        | "leftouter"  // "LEFT [OUTER] JOIN"
80                        | "rightouter" // "RIGHT [OUTER] JOIN"
81                        | "fullouter"  // "FULL [OUTER] JOIN"
82
```

```
type OrderingCondition = "asc" | "desc" | "<" | ">"
type Ordering = [SqlColumn, OrderingCondition]

export type RelationList = SqlRelation | SqlJoin
type TargetList = SqlColumn[]

export interface TargetClause {
  type: "targetclause"
  spec: "distinct" | "all" | null
  targetlist: "*" | TargetList
}

export class SqlLiteral {
  readonly type = LITERAL_TYPE
  literalType: 'string' | 'number' | 'boolean' | 'null'
  value: string | number | boolean | null

  constructor(literalType: 'string' | 'number' | 'boolean' | 'null',
              value: string | number | boolean | null) {
    this.literalType = literalType
    this.value = value
  }
}

export type SqlSelectish = SqlSelect | SqlSelectPair
export type PairingString = 'union' | 'intersect' | 'except'
export type PairingCondition = 'all' | 'distinct' | null

export class SqlSelectPair {
  readonly type = SELECTPAIR_TYPE
  pairing: PairingString
  condition: PairingCondition
  lhs: SqlSelect
  rhs: SqlSelectish

  constructor(pairing: PairingString,
              condition: PairingCondition,
              lhs: SqlSelect,
              rhs: SqlSelectish) {
    this.pairing = pairing
    this.condition = condition || null
    this.lhs = lhs
    this.rhs = rhs
  }
}

export class SqlSelect {
  readonly SELECTCLAUSE_TYPE
  what: TargetClause
  from: RelationList
  where: SqlConditional | null
  groupBy: TargetList | null
  having: SqlConditional | null
  orderBy: Ordering[] | null

  constructor(what: TargetClause,
              from: RelationList,
              where: SqlConditional | null,
              groupBy: TargetList | null,
```

```
142               having: SqlConditional | null,
143               orderBy: Ordering[] | null) {
144     this.what = what
145     this.from = from
146     this.where = where
147     this.groupBy = groupBy
148     this.having = having
149     this.orderBy = orderBy
150   }
151 }
152
153 export type SqlOperandType = SqlLiteral | SqlAggFunction | SqlColumn |
154                       SqlOperation | string
155
156 export class SqlColumn {
157   readonly type = COLUMN_TYPE
158   relation: string | null
159   target: SqlOperandType
160   as: string | null
161   alias: string | null
162
163   constructor(relation: string | null,
164               target: SqlOperandType,
165               As: string | null = null,
166               alias: string | null = null) {
167     this.relation = relation
168     this.target = target
169     this.as = As || null
170     this.alias = alias || null
171   }
172 }
173
174 export class SqlJoin {
175   readonly type = JOIN_TYPE
176   joinType: JOINSTRING
177   condition: SqlConditional | ['using', TargetList] | null
178   lhs: SqlJoin | SqlRelation
179   rhs: SqlJoin | SqlRelation
180
181   constructor(lhs: SqlJoin | SqlRelation,
182               rhs: SqlJoin | SqlRelation,
183               joinType: JOINSTRING = 'join',
184               condition: SqlConditional | ['using', TargetList] | null = null
185   ) {
186     this.lhs = lhs
187     this.rhs = rhs
188     this.joinType = joinType || 'join'
189     this.condition = condition || null
190   }
191 }
192
193 export class SqlRelation {
194   readonly type = RELATION_TYPE
195   target: SqlRelation | SqlJoin | string
196   alias: string | null
197
198   constructor(target: SqlRelation | SqlJoin | string,
199               alias: string | null = null) {
200     this.target = target
```

```
201       this.alias = alias || null
202     }
203   }
204
205   export type SqlConditionalOp = 'or' | 'and' | 'not' | 'in' | 'exists' | 'like'
          |
206                                  'between' | 'isnull' | '<>' | 'contains' |
207                                  '<=' | '>=' | '=' | '<' | '>' | '!='
208
209   export class SqlConditional {
210     readonly type = CONDITIONAL_TYPE
211     operation: SqlConditionalOp
212     lhs: SqlConditional | SqlOperandType
213     rhs: SqlConditional | SqlOperandType | null
214     not: boolean
215
216     constructor(operation: SqlConditionalOp,
217                 lhs: SqlConditional | SqlOperandType,
218                 rhs: SqlConditional | SqlOperandType | null = null,
219                 not: boolean = false) {
220       if (operation === 'in' && lhs instanceof Array && lhs.length === 1)
221         lhs = lhs[0]
222       this.operation = operation
223       this.lhs = lhs
224       this.rhs = rhs || null
225       this.not = not
226     }
227   }
228
229   export type AggFuncName = 'avg' | 'count' | 'max' | 'min' | 'sum'
230
231   export class SqlAggFunction {
232     readonly type = AGGFUNCTION_TYPE
233     fname: AggFuncName
234     expr: SqlOperandType | TargetClause
235
236     constructor(fname: AggFuncName, expr: SqlOperandType | TargetClause) {
237       this.fname = fname
238       this.expr = expr
239     }
240   }
241
242   export type SqlOperationOps = '||' | '+' | '-' | '*' | '/'
243
244   export class SqlOperation {
245     readonly type = OPERATION_TYPE
246     op: SqlOperationOps
247     lhs: SqlOperandType
248     rhs: SqlOperandType
249
250     constructor(op: SqlOperationOps, lhs: SqlOperandType, rhs: SqlOperandType) {
251       this.op = op
252       this.lhs = lhs
253       this.rhs = rhs
254     }
255   }
256
257   /*** RELATIONAL ALGEBRA ***/
258   // literals are strings
```

```
259
260 export type RelRelationish = RelRelation | RelJoin
261 export type RelOperandType = RelOperation | string | RelColumn
262
263 export class RelOperation {
264   readonly type = REL_OPERATION_TYPE
265   op: SqlOperationOps | 'union' | 'intersect' | 'except'
266   lhs: RelOperandType | HighLevelRelationish
267   rhs: RelOperandType | HighLevelRelationish
268
269   constructor(op: SqlOperationOps | 'union' | 'intersect' | 'except',
270               lhs: RelOperandType | HighLevelRelationish,
271               rhs: RelOperandType | HighLevelRelationish) {
272     this.op = op
273     this.lhs = lhs
274     this.rhs = rhs
275   }
276 }
277
278 type ColumnValueType = Column | RelFunction | string
279
280 export class RelColumn {
281   readonly type = REL_COLUMN_TYPE
282   relation: RelRelation | null
283   target: ColumnValueType
284   as: string | null
285
286   constructor(relation: RelRelation | null,
287               target: ColumnValueType,
288               As: string | null = null) {
289     this.relation = relation
290     this.target = target
291     this.as = As || null
292   }
293
294   alias(alias?: string) {
295     if (!alias)
296       return this
297     return new RelColumn(this.relation, this.target, alias)
298   }
299 }
300
301 export class RelFunction {
302   readonly type = REL_FUNCTION_TYPE
303   fname: AggFuncName
304   expr: '*' | RelColumn // TODO: support correct args
305
306   constructor(fname: AggFuncName, expr: '*' | RelColumn) {
307     this.fname = fname
308     this.expr = expr
309   }
310 }
311
312 export type ThetaOp = 'eq' | 'neq' | 'leq' | 'geq' | '<' | '>' | 'and' | 'or' |
313                       'in'
314
315 export class RelConditional {
316   readonly type = REL_CONDITIONAL_TYPE
317   operation: ThetaOp
```

```
318   lhs: RelOperandType | RelConditional
319   rhs: RelOperandType | RelConditional | RelOperandType[]
320
321   constructor(op: ThetaOp, lhs: RelOperandType | RelConditional,
322               rhs: RelOperandType | RelConditional | RelOperandType[]) {
323     this.operation = op
324     this.lhs = lhs
325     this.rhs = rhs
326   }
327 }
328
329 export type HighLevelRelationish = RelRelationish | RelRestriction |
        RelProjection | RelRename | RelOperation
330
331 export class RelRestriction {
332   readonly type = REL_RESTRICTION_TYPE
333   conditions: RelConditional
334   args: HighLevelRelationish
335
336   constructor(conditions: RelConditional, args: HighLevelRelationish) {
337     this.conditions = conditions
338     this.args = args
339   }
340 }
341
342 export class RelProjection {
343   readonly type = REL_PROJECTION_TYPE
344   columns: RelColumn[]
345   args: HighLevelRelationish
346
347   constructor(columns: RelColumn[], args: HighLevelRelationish) {
348     this.columns = columns
349     this.args = args
350   }
351 }
352
353 type _RelRenameInputType = RelRelation | RelColumn | RelFunction |
354                            RelRename | string
355
356 export class RelRename {
357   readonly type = REL_RENAME_TYPE
358   input: _RelRenameInputType
359   output: string
360   args: HighLevelRelationish
361
362   constructor(input: _RelRenameInputType,
363               output: string,
364               args: HighLevelRelationish) {
365     this.input = input
366     this.output = output
367     this.args = args
368   }
369 }
370
371 export class RelRelation {
372   readonly type = REL_RELATION_TYPE
373   name: string
374
375   constructor(name: string) {
```

```
376       this.name = name
377     }
378   }
379
380   export type RelJoinCond = "cross" | "left" | "right" | RelConditional
381
382   // cross
383   // natural (no condition)
384   // theta join (with condition)
385   // semi (left and right)
386   export class RelJoin {
387     readonly type = REL_JOIN_TYPE
388     lhs: HighLevelRelationish
389     rhs: HighLevelRelationish
390     condition: RelJoinCond
391
392     constructor(lhs: HighLevelRelationish,
393                 rhs: HighLevelRelationish,
394                 cond: RelJoinCond) {
395       this.lhs = lhs
396       this.rhs = rhs
397       this.condition = cond
398     }
399   }
```

2.4 src/parser/peg

2.4.1 src/parser/peg/sql.pegjs

```
1   /*
2     Initially inspired by grammar of the "Phoenix" SQL layer
3       (https://forcedotcom.github.io/phoenix/index.html)
4
5     Primarily based on PostgreSql syntax:
6       https://www.postgresql.org/docs/9/static/sql-syntax.html
7       https://www.postgresql.org/docs/9/static/sql-select.html
8       https://github.com/postgres/postgres/blob/master/src/backend/parser/gram.y
9   */
10
11  start
12    = Statements
13
14  Statements
15    = _ lhs:Statement rhs:( _ ";" _ Statement )* _ ";"?
16    { return rhs.reduce((result, element) => result.concat(element[3]), [lhs]) }
17
18  Statement
19    = Selectish
20
21  Selectish
22    = SelectPair
23    / Select
24
25
26  SelectPair
27    = lhs:Select __
28      pairing:$( "UNION"i / "INTERSECT"i / "EXCEPT"i ) __
```

```
    spec:( "ALL"i __ / "DISTINCT"i __ )?
    rhs:( Selectish )
  {
    return new SqlSelectPair(pairing.toLowerCase(),
                             spec && spec[0].toLowerCase(),
                             lhs,
                             rhs)
  }

Select
  = "SELECT"i __ what:TargetClause __
    "FROM"i   __ from:FromClause
    where:(   __ "WHERE"i  __           WhereClause )?
    groupBy:( __ "GROUP"i  __ "BY"i __ GroupByClause )?
    having:(  __ "HAVING"i __           HavingClause )?
    orderBy:( __ "ORDER"i  __ "BY"i __ OrderByClause )?
  {
    return new SqlSelect(what, from, where && where[3],groupBy && groupBy[5],
                         having && having[3], orderBy && orderBy[5])
  }
  / "(" _ sel:Select _ ")" { return sel }

TargetClause
  = spec:$( "DISTINCT"i __ / "ALL"i __ )?
    target:(
      "*"
      / TargetList
    )
  { return {
      'type': TARGETCLAUSE_TYPE,
      'specifier': spec ? spec.toLowerCase() : null,
      'targetlist': target
    }
  }

FromClause
  = from:RelationList

WhereClause
  = where:Condition

GroupByClause
  = groupBy:TargetList

HavingClause
  = having:Condition

OrderByClause
  = lhs:Ordering rhs:( _ "," _ Ordering )*
  { return rhs.reduce((result, element) => result.concat(element[3]), [lhs]) }

Ordering
  = expr:Operand
    cond:(
        __ "ASC"i { return 'asc' }
      / __ "DESC"i { return 'desc' }
      / __ "USING"i _ op:$( "<" / ">" ) { return op }
    )?

```

```
88  RelationList
89    = item1:RelationItem _ "," _ items:RelationList
90      { return new SqlJoin(item1, items) }
91      / Join
92      / RelationItem
93
94  RelationItem "RelationItem"
95    = item:RelationThing __ ( "AS"i __ )? alias:Name
96    { return new SqlRelation(item, alias) }
97    / RelationThing
98
99  RelationThing
100   = "(" _ list:RelationList _ ")"
101   { return list }
102   / "(" _ join:Join _ ")"
103   { return join }
104   / tableName:Name
105   { return new SqlRelation(tableName) }
106
107 Join
108   = item1:RelationItem __
109     jtype:JoinType __
110     item2:RelationItem
111     jcond:(
112       __ "ON"i
113       __ expr:Condition
114       { return expr }
115       / __ "USING"i _
116         "(" _ list:TargetList _ ")"
117         { return ['using', list] }
118     )?
119   { return new SqlJoin(item1, item2, jtype, jcond) }
120
121 TargetList
122   = item1:TargetItem _ "," _ items:TargetList
123     { return [item1].concat(items) }
124     / item:TargetItem
125     { return [item] }
126
127 TargetItem "TargetItem"
128   = table:Name ".*"
129   { return new SqlColumn(table, '*', `${table}.*`, null) }
130   / op:Operand __ "AS"i __ alias:Name
131   { return new SqlColumn(null, op, alias, alias )}
132   / op:Operand __ alias:Name
133   { return new SqlColumn(null, op, alias, alias )}
134   / op:Operand _ "=" _ alias:Name
135   { return new SqlColumn(null, op, alias, alias) }
136   / op:Operand
137   { return (op instanceof SqlColumn) ? op : new SqlColumn(null, op) }
138
139 Condition "Condition"
140   = lhs:AndCondition rhs:( __ "OR"i __ Condition )?
141   { return rhs ? new SqlConditional('or', lhs, rhs[3]) : lhs }
142
143 AndCondition
144   = lhs:InnerCondition rhs:( __ "AND"i __ AndCondition )?
145   { return rhs ? new SqlConditional('and', lhs, rhs[3]) : lhs }
146
```

```
147 InnerCondition
148   = ( ConditionContains
149     / ConditionComp
150     / ConditionIn
151     / ConditionExists
152     / ConditionLike
153     / ConditionBetween
154     / ConditionNull
155 //    / Operand
156   )
157   / "NOT"i __ expr:Condition
158   { return new SqlConditional('not', expr) }
159   / "(" _ expr:Condition _ ")"
160   { return expr }
161
162 ConditionContains "Conditional-Contains"
163   // based on Transact-SQL
164   = "CONTAINS" _
165     "(" _
166       lhs:(
167         Operand
168         / "(" _ ops:OperandList _ ")"
169         { return ops }
170       )
171       rhs:SQStringLiteral
172     ")"
173   { return new SqlConditional('contains', lhs, rhs) }
174
175 ConditionComp "Conditional-Comparison"
176   = lhs:Operand _ cmp:Compare _ rhs:Operand
177   { return new SqlConditional(cmp, lhs, rhs) }
178
179 ConditionIn
180   = lhs_op:Operand __
181     not:( "NOT"i __ )?
182     "IN"i _
183     "(" _
184       rhs_ops:( Selectish / OperandList ) _
185     ")"
186   { return new SqlConditional('in', lhs_op, rhs_ops, not) }
187
188 ConditionExists
189   = "EXISTS"i _
190     "(" _ subquery:Selectish _ ")"
191   { return new SqlConditional('exists', subquery) }
192
193 ConditionLike
194   = lhs_op:Operand __
195     not:( "NOT"i __ )?
196     "LIKE"i __
197     rhs_op:Operand
198   { return new SqlConditional('like', lhs_op, rhs_op, not) }
199
200 ConditionBetween
201   = lhs_op:Operand __
202     not:( "NOT"i __ )?
203     "BETWEEN"i
204     rhs:(
205       __
```

```
206       rhs_op1:Operand __
207       "AND"i __
208       rhs_op2:Operand
209       { return [rhs_op1, rhs_op2] }
210       / _
211         "(" _
212           rhs_op1:Operand __
213           "AND"i __
214           rhs_op2:Operand
215         ")"
216         { return [rhs_op1, rhs_op2] }
217     )
218   { return new SqlConditional('between', lhs_op, rhs, not) }
219
220 ConditionNull
221   = lhs:Operand __ "IS"i __
222     not:( "NOT"i __ )?
223     NullLiteral
224   { return new SqlConditional('isnull', lhs, null, not) }
225
226 Term
227   = Literal
228     / AggFunction
229     / "(" _ op:Operand _ ")" { return op }
230     / ColumnRef
231
232 ColumnRef
233   = tbl:( table:Name "." )? column:Name
234     { return new SqlColumn(tbl && tbl[0],
235                            column,
236                            tbl ? `${tbl[0]}.${column}` : column
237                           ) }
238
239 AggFunction "aggregate function"
240   = AggFunctionAvg
241     / AggFunctionCount
242     / AggFunctionMax
243     / AggFunctionMin
244     / AggFunctionSum
245
246 AggFunctionAvg
247   = "AVG"i _
248     "(" _ term:Term _ ")"
249   { return new SqlAggFunction("avg", term) }
250
251 AggFunctionCount
252   = "COUNT"i _
253     "(" _
254       targ:TargetClause _
255     ")"
256   { return new SqlAggFunction("count", targ) }
257
258 AggFunctionMax
259   = "MAX"i _
260     "(" _
261       term:Term _
262     ")"
263   { return new SqlAggFunction("max", term) }
264
```

```
AggFunctionMin
  = "MIN"i _
    "(" _
      term:Term _
    ")"
  { return new SqlAggFunction("min", term) }

AggFunctionSum
  = "SUM"i _
    "(" _
      term:Term _
    ")"
  { return new SqlAggFunction("sum", term) }

/***** PRIMITIVES *****/

Name
  = DQStringLiteral
    / BTStringLiteral
    / !ReservedWord id:Ident {return id }

Ident "UnquotedIdent"
  = $( [A-Za-z_][A-Za-z0-9_]* )

OperandList
  = lhs:Operand
    rhs:( _ "," _ Operand )*
  {
    if (rhs.length)
      return rhs.reduce((result, element) => result.concat(element[3]), [lhs])
    else
      return lhs
  }

Operand // Summand | makeOperation
  = lhs:Summand
    rhs:( _ "||" _ Summand ) *
  { return reduceIfRHS(lhs, rhs, (lh, rh) => new SqlOperation("||",
                                                              lh, rh[3])) }
  / Selectish

Summand // Factor | makeOperation
  = lhs:Factor
    rhs:( _ ("+" / "-") _ Factor ) *
  { return reduceIfRHS(lhs, rhs, (lh, rh) => new SqlOperation(rh[1],
                                                              lh, rh[3])) }

Factor // literal | function | Operand | column | makeOperation
  = lhs:Term
    rhs:( _ ("*" / "/") _ Term ) *
  { return reduceIfRHS(lhs, rhs, (lh, rh) => new SqlOperation(rh[1],
                                                              lh, rh[3])) }

Compare
  = "<>"
    / "<="
    / ">="
    / "="
    / "<"
```

```
    / ">"
    / "!="

JoinType "JoinType"
  = ( "CROSS"i __ )? "JOIN"i
  { return "join" }
  / "INNER"i __ "JOIN"i
  { return "equi" }
  / "NATURAL"i __ "JOIN"i
  { return "natural" }
  / "LEFT"i __ ( "OUTER"i __ )? "JOIN"i
  { return "left" }
  / "RIGHT"i __ ( "OUTER"i __ )? "JOIN"i
  { return "right" }
  / "FULL"i __ ( "OUTER"i __ )? "JOIN"i
  { return "full" }

/***** LITERALS *****/

Literal "Literal"
  = SQStringLiteral
    / NumericLiteral
    / ExponentialLiteral
    / BooleanLiteral
    / NullLiteral

BTStringLiteral "backtick string"
  = $( '`' ( [^`] / '``' )+ '`' )

DQStringLiteral "double-quote string"
  = $( '"' ( [^"] / '""' )+ '"' )

SQStringLiteral "single-quote string"
  = lit:$( "'" ( [^'] / "''" )* "'" !SQStringLiteral )
  { return new SqlLiteral('string', lit.slice(1, -1)) }
  / lit:$( ("âĂŸ"/"âĂŹ") ( [^âĂŹ] )* "âĂŹ" ) // fancy single-quote
  { return new SqlLiteral('string', lit.slice(1, -1)) }

ExponentialLiteral "exponential"
  = val:$( NumericLiteral "e" IntegerLiteral )
  { return new SqlLiteral('number', parseFloat(val)) }

NumericLiteral "number"
  = IntegerLiteral
    / DecimalLiteral

IntegerLiteral "integer"
  = int:$( "-"? [0-9]+ )
  { return new SqlLiteral('number', parseInt(int)) }

DecimalLiteral "decimal"
  = value:$( "-"? [0-9]+ "." [0-9]+ )
  { return new SqlLiteral('number', parseFloat(value)) }

NullLiteral "null"
  = "NULL"i
  { return new SqlLiteral('null', null) }

BooleanLiteral "boolean"
```

```
383   = TruePrim
384     / FalsePrim
385
386 TruePrim
387   = "TRUE"i
388   { return new SqlLiteral('boolean', true) }
389
390 FalsePrim
391   = "FALSE"i
392   { return new SqlLiteral('boolean', false) }
393
394 _ "OptWhitespace"
395   = WS* (Comment WS*)* {}
396
397 __ "ReqWhitespace"
398   = WS+ (Comment WS*)* {}
399
400 WS
401   = [ \t\n]
402
403 Comment "Comment"
404   = "/*" ( !"*/" . )* "*/"   {}
405     / "--" ( !"\n" . )* "\n" {}
406
407 /** SQL2008 reserved words.
408     In alphabetical order but not always lexical order,
409       as there is no backtracking in PEG.js, e.g. for
410         "IN" / "INT" / "INTERSECT" / "INTERSECTION"
411       only "IN" is reachable.
412  **/
413 ReservedWord
414   = $("ABS"i / "ALL"i / "ALLOCATE"i / "ALTER"i / "AND"i / "ANY"i / "ARE"i /
415         "ARRAY_AGG"i / "ARRAY"i / "ASENSITIVE"i / "ASYMMETRIC"i / "AS"i /
416         "ATOMIC"i / "AT"i / "AUTHORIZATION"i / "AVG"i
417       / "BEGIN"i / "BETWEEN"i / "BIGINT"i / "BINARY"i / "BLOB"i / "BOOLEAN"i /
418         "BOTH"i / "BY"i
419       / "CALLED"i / "CALL"i / "CARDINALITY"i / "CASCADED"i / "CASE"i / "CAST"i
              /
420         "CEILING"i / "CEIL"i / "CHARACTER_LENGTH"i / "CHAR_LENGTH"i /
421         "CHARACTER"i / "CHAR"i / "CHECK"i / "CLOB"i / "CLOSE"i / "COALESCE"i /
422         "COLLATE"i / "COLLECT"i / "COLUMN"i / "COMMIT"i / "CONDITION"i /
423         "CONNECT"i / "CONSTRAINT"i / "CONVERT"i / "CORRESPONDING"i / "CORR"i /
424         "COUNT"i / "COVAR_POP"i / "COVAR_SAMP"i / "CREATE"i / "CROSS"i /
425         "CUBE"i / "CUME_DIST"i / "CURRENT_CATALOG"i / "CURRENT_DATE"i /
426         "CURRENT_DEFAULT_TRANSFORM_GROUP"i / "CURRENT_PATH"i / "CURRENT_ROLE"i
              /
427         "CURRENT_SCHEMA"i / "CURRENT_TIMESTAMP"i / "CURRENT_TIME"i /
428         "CURRENT_TRANSFORM_GROUP_FOR_TYPE"i / "CURRENT_USER"i / "CURRENT"i /
429         "CURSOR"i / "CYCLE"i
430       / "DATALINK"i / "DATE"i / "DAY"i / "DEALLOCATE"i / "DECIMAL"i /
431         "DECLARE"i / "DEC"i / "DEFAULT"i / "DELETE"i / "DENSE_RANK"i /
432         "DEREF"i / "DESCRIBE"i / "DETERMINISTIC"i / "DISCONNECT"i /
433         "DISTINCT"i / "DLNEWCOPY"i / "DLPREVIOUSCOPY"i / "DLURLCOMPLETE"i /
434         "DLURLCOMPLETEONLY"i / "DLURLCOMPLETEWRITE"i / "DLURLPATHONLY"i /
435         "DLURLPATHWRITE"i / "DLURLPATH"i / "DLURLSCHEME"i / "DLURLSERVER"i /
436         "DLVALUE"i / "DOUBLE"i / "DROP"i / "DYNAMIC"i
437       / "EACH"i / "ELEMENT"i / "ELSE"i / "END-EXEC"i / "END"i / "ESCAPE"i /
438         "EVERY"i / "EXCEPT"i / "EXECUTE"i / "EXEC"i / "EXISTS"i / "EXP"i /
439         "EXTERNAL"i / "EXTRACT"i
```

```
      / "FALSE"i / "FETCH"i / "FILTER"i / "FIRST_VALUE"i / "FLOAT"i / "FLOOR"i
        /
        "FOREIGN"i / "FOR"i / "FREE"i / "FROM"i / "FULL"i / "FUNCTION"i /
        "FUSION"i
      / "GET"i / "GLOBAL"i / "GRANT"i / "GROUPING"i / "GROUP"i
      / "HAVING"i / "HOLD"i / "HOUR"i
      / "IDENTITY"i / "IMPORT"i / "INDICATOR"i / "INNER"i / "INOUT"i /
        "INSENSITIVE"i / "INSERT"i / "INTEGER"i / "INTERSECTION"i /
        "INTERSECT"i / "INTERVAL"i / "INTO"i / "INT"i / "IN"i / "IS"i
      / "JOIN"i
      / "LAG"i / "LANGUAGE"i / "LARGE"i / "LAST_VALUE"i / "LATERAL"i /
        "LEADING"i / "LEAD"i / "LEFT"i / "LIKE_REGEX"i / "LIKE"i / "LN"i /
        "LOCALTIMESTAMP"i / "LOCAL"i / "LOCALTIME"i / "LOWER"i
      / "MATCH"i / "MAX_CARDINALITY"i / "MAX"i / "MEMBER"i / "MERGE"i /
        "METHOD"i / "MINUTE"i / "MIN"i / "MODIFIES"i / "MODULE"i / "MOD"i /
        "MONTH"i / "MULTISET"i
      / "NATIONAL"i / "NATURAL"i / "NCHAR"i / "NCLOB"i / "NEW"i / "NONE"i /
        "NORMALIZE"i / "NOT"i / "NO"i / "NTH_VALUE"i / "NTILE"i / "NULLIF"i /
        "NULL"i / "NUMERIC"i
      / "OCCURRENCES_REGEX"i / "OCTET_LENGTH"i / "OFFSET"i / "OF"i / "OLD"i /
        "ONLY"i / "ON"i / "OPEN"i / "ORDER"i / "OR"i / "OUTER"i / "OUT"i /
        "OVERLAPS"i / "OVERLAY"i / "OVER"i
      / "PARAMETER"i / "PARTITION"i / "PERCENTILE_CONT"i / "PERCENTILE_DISC"i /
        "PERCENT_RANK"i / "POSITION_REGEX"i / "POSITION"i / "POWER"i /
        "PRECISION"i / "PREPARE"i / "PRIMARY"i / "PROCEDURE"i
      / "RANGE"i / "RANK"i / "READS"i / "REAL"i / "RECURSIVE"i / "REFERENCES"i
        /
        "REFERENCING"i / "REF"i / "REGR_AVGX"i / "REGR_AVGY"i / "REGR_COUNT"i /
        "REGR_INTERCEPT"i / "REGR_R2"i / "REGR_SLOPE"i / "REGR_SXX"i /
        "REGR_SXY"i / "REGR_SYY"i / "RELEASE"i / "RESULT"i / "RETURNS"i /
        "RETURN"i / "REVOKE"i / "RIGHT"i / "ROLLBACK"i / "ROLLUP"i / "ROWS"i /
        "ROW_NUMBER"i / "ROW"i
      / "SAVEPOINT"i / "SCOPE"i / "SCROLL"i / "SEARCH"i / "SECOND"i /
        "SELECT"i / "SENSITIVE"i / "SESSION_USER"i / "SET"i / "SIMILAR"i /
        "SMALLINT"i / "SOME"i / "SPECIFICTYPE"i / "SPECIFIC"i /
        "SQLEXCEPTION"i / "SQLSTATE"i / "SQLWARNING"i / "SQL"i / "SQRT"i /
        "START"i / "STATIC"i / "STDDEV_POP"i / "STDDEV_SAMP"i / "SUBMULTISET"i
          /
        "SUBSTRING_REGEX"i / "SUBSTRING"i / "SUM"i / "SYMMETRIC"i /
        "SYSTEM_USER"i / "SYSTEM"i
      / "TABLESAMPLE"i / "TABLE"i / "THEN"i / "TIMESTAMP"i / "TIMEZONE_HOUR"i /
        "TIMEZONE_MINUTE"i / "TIME"i / "TO"i / "TRAILING"i /
        "TRANSLATE_REGEX"i / "TRANSLATE"i / "TRANSLATION"i / "TREAT"i /
        "TRIGGER"i / "TRIM_ARRAY"i / "TRIM"i / "TRUE"i / "TRUNCATE"i
      / "UESCAPE"i / "UNION"i / "UNIQUE"i / "UNKNOWN"i / "UNNEST"i / "UPDATE"i
        /
        "UPPER"i / "USER"i / "USING"i
      / "VALUES"i / "VALUE"i / "VARBINARY"i / "VARCHAR"i / "VARYING"i /
        "VAR_POP"i / "VAR_SAMP"i
      / "WHENEVER"i / "WHEN"i / "WHERE"i / "WIDTH_BUCKET"i / "WINDOW"i /
        "WITHIN"i / "WITHOUT"i / "WITH"i
      / "XMLAGG"i / "XMLATTRIBUTES"i / "XMLBINARY"i / "XMLCAST"i /
        "XMLCOMMENT"i / "XMLCONCAT"i / "XMLDOCUMENT"i / "XMLELEMENT"i /
        "XMLEXISTS"i / "XMLFOREST"i / "XMLITERATE"i / "XMLNAMESPACES"i /
        "XMLPARSE"i / "XMLPI"i / "XMLQUERY"i / "XMLSERIALIZE"i / "XMLTABLE"i /
        "XMLTEXT"i / "XMLVALIDATE"i / "XML"i
      / "YEAR"i
  ) !Ident
```

2.4.2 src/parser/peg/relations.pegjs

```
1
2  start
3    = _ rel:Relations _
4    { return rel }
5
6  Relations
7    = lhs:Relation
8      rhs:( _ Relations )*
9    { return rhs.reduce((l, r) => l.concat(r[1]), [lhs]) }
10
11 Relation
12   = table:Name
13     _ "(" _
14       cols:Columns
15     _ ")"
16   { return [table, cols] }
17
18 Columns
19   = lhs:Column rhs:( _ "," _ Column )*
20   { return rhs.reduce((l,r) => l.concat([r[3]]), [lhs]) }
21
22 Column
23   = name:Name _ ":" _ typ:Ident
24   { return [name, typ] }
25
26
27 /* sql primitives */
28
29 Name "Name"
30   = DQStringLiteral
31     / BTStringLiteral
32     / Ident
33
34 Ident "UnquotedIdent"
35   = $( [A-Za-z_][A-Za-z0-9_]* )
36
37 BTStringLiteral "backtick string"
38   = $( '`' ( [^`] / '``' )+ '`' )
39
40 DQStringLiteral "double-quote string"
41   = $( '"' ( [^"] / '""' )+ '"' )
42
43 _ "OptWhitespace"
44   = WS* Comment? WS* {}
45
46 __ "ReqWhitespace"
47   = WS+ Comment? WS* {}
48     / WS* Comment? WS+ {}
49
50 WS
51   = [ \t\n]
52
53 Comment "Comment"
54   = "/*" ( !"*/" . )* "*/"    {}
55     / "--" ( !"\n" . )* "\n" {}
```

2.5 src/query_tree

2.5.1 src/query_tree/node.ts

```
import {QTOperation, Relation, Join, Restriction, Projection, Rename,
        Operation} from './operation'

import * as types from '../parser/types'

// if RelRelation:    just name
// if RelJoin:        ....
// if RelRestriction: SYM _ (conditions)
// if RelProjection:  SYM _ (columns)
// if RelRename:      SYM _ (A / B)
// if RelOperation:   hlr SYM hlr

export default class Node {
  hlr: types.HighLevelRelationish
  operation: QTOperation
  children: Node[] = []
  depth: number = 0

  constructor(hlr: types.HighLevelRelationish, depth: number = 0) {
    this.hlr = hlr
    this.depth = depth
    this.generateOpAndKids()
  }

  generateOpAndKids() {
    if (this.hlr instanceof types.RelRelation) {
      this.operation = new Relation(this.hlr)
    } else if (this.hlr instanceof types.RelJoin) {
      this.operation = new Join(this.hlr)
      this.addNode(new Node(this.hlr.lhs, this.depth + 1))
      this.addNode(new Node(this.hlr.rhs, this.depth + 1))
    } else if (this.hlr instanceof types.RelRestriction) {
      this.operation = new Restriction(this.hlr)
      this.addNode(new Node(this.hlr.args, this.depth + 1))
    } else if (this.hlr instanceof types.RelProjection) {
      this.operation = new Projection(this.hlr)
      this.addNode(new Node(this.hlr.args, this.depth + 1))
    } else if (this.hlr instanceof types.RelRename) {
      this.operation = new Rename(this.hlr)
      this.addNode(new Node(this.hlr.args, this.depth + 1))
    } else if (this.hlr instanceof types.RelOperation) {
      this.operation = new Operation(this.hlr)
      this.addNode(new Node(this.hlr.lhs as types.HighLevelRelationish, this.
          depth + 1))
      this.addNode(new Node(this.hlr.rhs as types.HighLevelRelationish, this.
          depth + 1))
    } else {
      console.error("Unknown type", this.hlr)
      throw new Error("Unknown op type")
    }
  }

  addNode(node: Node) {
```

```
52      node.depth = this.depth + 1
53      this.children.push(node)
54    }
55  }
```

2.5.2 src/query_tree/operation.tsx

```
1  import * as React from 'react'
2  import * as types from '../parser/types'
3  import {htmlRelRelation, htmlRelProjection, relJoinHelper, htmlRelRestriction,
4          htmlRelRename, getSymbol } from '../parser/relationalText'
5
6  // if RelRelation:     just name
7  // if RelJoin:         ....
8  // if RelRestriction: SYM _ (conditions)
9  // if RelProjection:  SYM _ (columns)
10 // if RelRename:       SYM _ (A / B)
11 // if RelOperation:    hlr SYM hlr
12
13 export class QTOperation {
14   symbolName: string
15   hlr: types.HighLevelRelationish
16   html: JSX.Element
17
18   constructor(hlr: types.HighLevelRelationish) {
19     this.hlr = hlr
20   }
21 }
22
23 export class Relation extends QTOperation {
24   hlr: types.RelRelation
25   constructor(hlr: types.RelRelation) {
26     super(hlr)
27     this.html = htmlRelRelation(hlr)
28   }
29 }
30
31 export class Join extends QTOperation {
32   hlr: types.RelJoin
33   constructor(hlr: types.RelJoin) {
34     super(hlr)
35     this.html = this.generateHTML()
36   }
37
38   generateHTML() {
39     const [joinSymbol, cond] = relJoinHelper(this.hlr)
40     return (
41       <span className="RelJoin">
42         <span className="operator">{joinSymbol}</span>
43         {cond}
44       </span>
45     )
46   }
47 }
48
49 export class Restriction extends QTOperation {
50   hlr: types.RelRestriction
```

```
51    constructor(hlr: types.RelRestriction) {
52      super(hlr)
53      this.html = htmlRelRestriction(hlr, true)
54    }
55  }
56
57  export class Projection extends QTOperation {
58    hlr: types.RelProjection
59    constructor(hlr: types.RelProjection) {
60      super(hlr)
61      this.html = htmlRelProjection(hlr, true)
62    }
63  }
64
65  export class Rename extends QTOperation {
66    hlr: types.RelRename
67    constructor(hlr: types.RelRename) {
68      super(hlr)
69      this.html = htmlRelRename(hlr, true)
70    }
71  }
72
73  export class Operation extends QTOperation {
74    hlr: types.RelOperation
75    constructor(hlr: types.RelOperation) {
76      super(hlr)
77      this.html = this.generateHTML()
78    }
79
80    generateHTML() {
81      const SYM = getSymbol(this.hlr.op)
82      return (
83        <span className="operator">{SYM}</span>
84      )
85    }
86  }
87
88  /*export class From extends Operation {
89    constructor() {
90      super("From")
91    }
92
93    addTarget(data) {
94       if(data.lhs && data.rhs) {
95         this.addTarget(data.lhs)
96         this.addTarget(data.rhs)
97         return
98       }
99
100      else if(data.lhs || data.rhs) {
101        throw new Error('From without both lhs and rhs')
102      }
103
104     let arg = data.target.target
105     if(data.alias) arg += ` as ${data.alias}`
106     this.addArgument(arg)
107   }
108 }*/
109
```

```
/*export class Where extends Operation {
  constructor() {
    super("Where")
  }

  addTarget(data) {
    let lhs = this.getArgument(data.lhs)
    let rhs = this.getArgument(data.rhs)
    this.addArgument(lhs + ` ${data.operation} ` + rhs)
  }

  getArgument(data): string {
    if(data.lhs && data.rhs) {
      let lhs = this.getArgument(data.lhs)
      let rhs = this.getArgument(data.rhs)
      let arg = lhs + ` ${data.operation} ` + rhs
      return arg
    } else if(data.lhs || data.rhs) {
      throw new Error('lhs and rhs not both specified')
    }

    let arg
    if(data.relation) arg = `${data.relation}.${data.target}`
    if(data.relation && data.alias) arg += ` as ${data.alias}`
    if(data.literalType === "number" && data.value) arg = data.value
    if(data.literalType === "string" && data.value) arg = `\'${data.value}\'`

    return arg
  }
}*/
```