`constexpr` SQL

STUDENT: MICHAEL KITZAN

SUPERVISOR: BILL BIRD

Problems with Data Processing

SQL is the standard language for specifying data processing operations

Using SQL often requires a DBMS

- DBMS have run-time and integration overhead

Without a DBMS, relational algebra must be handwritten

- Handwritten relational algebra is inflexible and prone to suboptimal implementations

Light Weight Data Processing

Light weight data processing contexts

- Data processing is not the primary goal in the system
- Data may not require persistence
- Data processing query does not change at run-time

Addressing the problems of data processing in light weight contexts

- Library which transforms SQL queries into a usable relational algebra expression trees
- Library which provides representations of SQL entities like tables and rows

`constexpr` SQL Library

C++20 library targeting light weight data processing contexts

Enables the usage of SQL in C++ programs without

- Integrating a DBMS backend
- Hardcoding SQL queries and relational algebra expression trees

Library features

- Compile-time parsing and planning of SQL queries into relational algebra expression trees
- Supports modern C++ idioms like range loops, structured binding declarations, and iterators
- Includes helper functions to ease integration into systems

Example Part 1: Single Header

```
#include "sql.hpp"
```

Include the library's single header file

`sql.hpp` includes

- SQL entities
- Query functionality
- Helper functions

Example Part 2: Schemas

```
using books =
    sql::schema<
        sql::index<"year", "title">,
        sql::column<"title", std::string>,
        sql::column<"genre", std::string>,
        sql::column<"year", unsigned>
    >;
using authored =
    sql::schema<
        sql::index<>,
        sql::column<"title", std::string>,
        sql::column<"author", std::string>
    >;
```

`schema` defines a type representing a table

`index` specifies how to sort the data

`column` binds a name and a type

Example Part 3: Queries

```
using query =
    sql::query<
        "SELECT title AS book, author "
        "FROM T0 NATURAL JOIN T1 "
        "WHERE genre = \"science fiction\"",
        books,
        authored
    >;
```

query is parsed into a relational algebra tree

Table **Tn** will substitute for the **n**th schema listed after the query

Example Part 4: Loading Data

```
books t0
{
    sql::load<books, '\t'>("books.tsv")
};
authored t1
{
    sql::load<authored, '\t'>("authored.tsv")
};
```

`load` is templated on a **`schema`** and delimiter

`load` converts each element into the type specified in the **`schema`**

The file must not have a row of column names

Example Part 5: Iterating Output

```
for (query q{ t0, t1 }; auto const& [b, a] : q)
{
    std::cout << b << '\t' << a << '\n';
}
```

Construct **`query`** objects within a safe scope

Structured binding declarations explode rows into elements

Code Comparison

```
books_type query(books_type const& b, authored_type const& a)
{
    using std::get;
    books_type output{};
    std::unordered_map<std::string, authored_row> cache{};

    for (auto const& row : a)
    {
        cache[get<0>(row)] = row;
    }

    for (auto const& row : b)
    {
        auto it{ cache.find(get<0>(row)) };

        if (it != cache.end() && get<1>(get<1>(*it)) == "Barbara W. Tuchman")
        {
            output.push_back(row);
        }
    }

    return output;
}
```

```
using query =
    sql::query<
        "SELECT title, year FROM T0 NATURAL JOIN T1 "
        "WHERE author = \"Barbara W. Tuchman\"",
        books,
        authored
    >;
```

Future Work

Implement more SQL features

- INNER JOIN, LEFT JOIN, and RIGHT JOIN
- GROUP BY, HAVING, and ORDER BY
- IN operation within WHERE clause

Implement template argument error detection

Resources and References

https://github.com/mkitzan/constexpr-sql

https://github.com/hanickadot/compile-time-regular-expressions

University of Victoria: Department of Computer Science

University of Victoria