CIS64A Spring 2020

Homework 7

Joshua Saunders
6/10/2020

Purpose: In this home work, you will practice normalization of data to design a database.

In this assignment, the [MySQL C++ Connector 8.0] was used.

1. Connecting to a MySQL database with C++

In order to connect to a database, a `Session` object needs to be created with the host, port, user name, and password of the database as parameters to the constructor[1]. An object representing a schema (database) is created by passing the schema's (database's) name to the `Schema` constructor. A boolean of `true` is also passed to the constructor to make sure that the schema exists in the database[2]. This is wrapped in a `try`/`catch` clause so that exceptions that are raised can be dealt with.

There are two ways to access the data in the database: using `Table` objects and using `Collection` objects. `Table` objects were used because they were the most straightforward and the syntax is similar to the SQL syntax.

```
/* Example 1 - Connecting to the database */
#include <iostream>

#include <mysqlx/xdevapi.h>

int main()
{
    std::cout << "Connecting to the database...\n";

    try
    {
        mysqlx::Session session("localhost", 33060, "user", "password");
        mysqlx::Schema db = session.getSchema("cis64a", true);

        std::cout << "Done!\n";
    }
    catch (const mysqlx::Error& error)
    {
```

[1] https://dev.mysql.com/doc/x-devapi-userguide/en/database-connection-example.html

[2] https://dev.mysql.com/doc/dev/connector-cpp/8.0/classmysqlx_1_1abi2_1_1r0_1_1_session.html#a1d34ba08a59c5755537a1e6f85ef63f9

```
        std::cerr << "The following error occurred: " << error << std::endl;
    }
}
```

If everything worked, then the program should output the following to the terminal

```
Connecting to the database…
Done!
```

Otherwise, you will see an error printed to the terminal such as

```
Connecting to the database...
The following error occurred: CDK Error: Unknown database 'test'
```

if you attempted to connect to a nonexistent database (`'test'` in this instance).

2. **C**reating an entry in the database

In order to create an entry in a database, the database connection must first be established as shown in section 1. Once the database connection is established, then a `Table` object representing the desired table is created using the `Schema` object's `getTable` method with the name of the table and a flag to check if the table exists as arguments[3]. Now that a Table object has been instantiated, the desired data can be inserted into the table using methods that are similar to using SQL statements.

To insert a single entry, the pattern given in Fig. 1 below is followed. In English, the column name(s) is/are parameter(s) for the `insert` method. Next, chain the `values` method with the values of columns specified in the same order as the `insert` method. Finally, chain the `execute` method at the end. Multiple entries can be made by chaining successive calls to the `value` method.

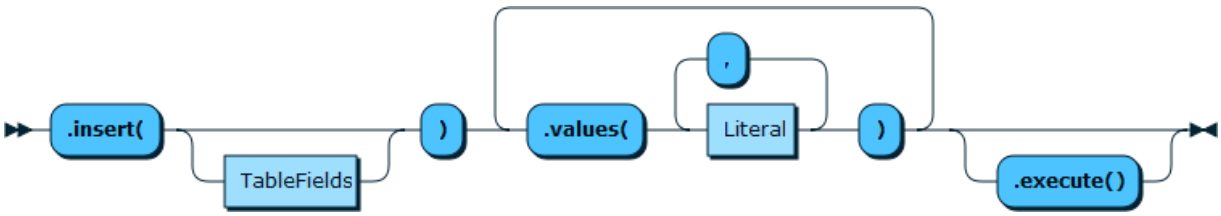

Fig. 1 Table.insert() Syntax Diagram

```
/* Example 2 - Creating an entry */
#include <iostream>

#include <mysqlx/xdevapi.h>

int main()
{
    std::cout << "Connecting to the database...\n";

    try
    {
        mysqlx::Session session("localhost", 33060, "user", "password");
        mysqlx::Schema db = session.getSchema("cis64a", true);

        mysqlx::Table table = db.getTable("CITY", true);
        table.insert("ID", "Name", "CountryCode", "District", "Population")
             .values(4080, "Twin Peaks", "USA", "Washington", 10500)
             .execute();

        std::cout << "Done!\n";
    }
    catch (const mysqlx::Error &error)
```

[3] https://dev.mysql.com/doc/x-devapi-userguide/en/devapi-users-working-with-relational-tables.html

```
    {
        std::cerr << "The following error occurred: " << error << "\n";
    }
}
```

If everything worked, then the program should output the following to the terminal

```
Connecting to the database…
Done!
```

Otherwise, you will see an error printed to the terminal such as

```
Connecting to the database...
The following error occurred: CDK Error: Duplicate entry '4080' for key 'city.PRIMARY'
```

if you attempted to insert an entry with a key that already exists.

3. Reading from the database

Before reading (selecting) an entry from the table, a `Table` object must be created as shown in section 2. Once the `Table` object has been created, the `select` method is with the desired column names as parameters, then the `where` method is chained with the desired criteria in a string parameter, the `bind` method is then chained, and finally the `execute` method is chained to execute the query. The pattern for the `select` method is given in Fig. 2 below.

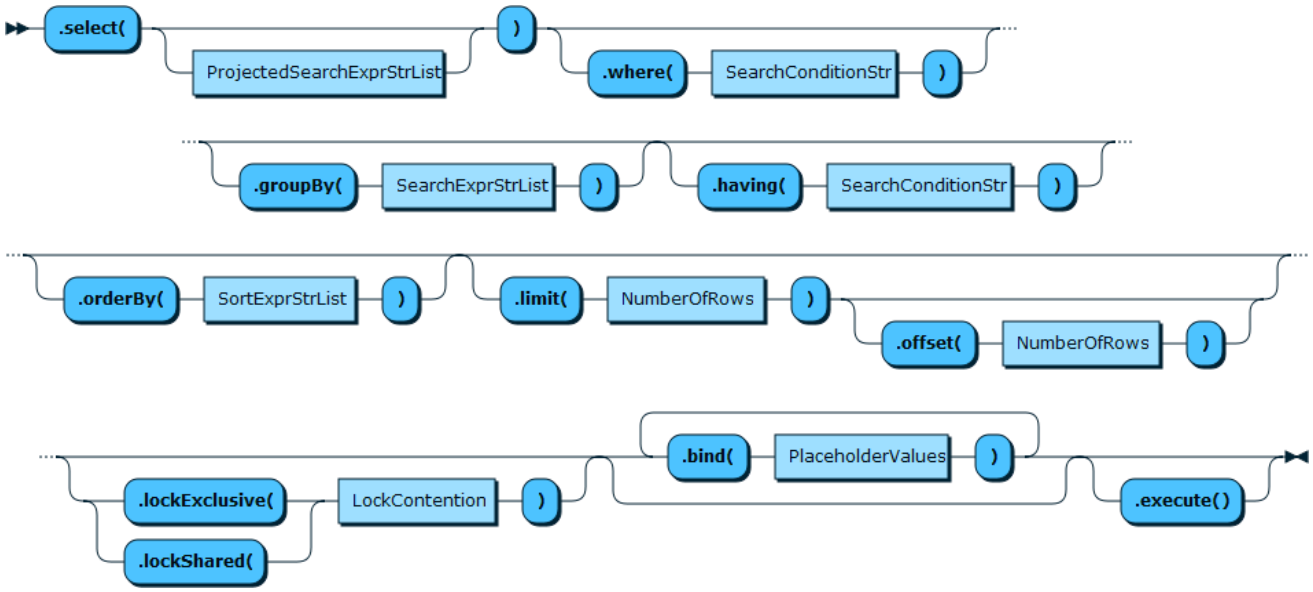

Fig. 2 Table.select() Syntax Diagram

Notice in the `where` method the value that is being checked for isn't directly used, but is given as an argument to the `bind` method. It is considered good practice to separate out the values from the expression string (the argument to the `where` method) into parameters (which have a `:` in front of them in the expression string)[4]. The `bind` method has two arguments, the first is the name of the parameter given in the `where` method and the second is its value.

The return value of this chain of method calls is saved into a `RowResult` object. Finally, a `Row` object is created by calling the `RowResult` object's `fetchOne` method. The data in the `Row` object can be accessed using indices just like an array or vector.

```
/* Example 3 - Reading from the database */
#include <iostream>

#include <mysqlx/xdevapi.h>

int main()
{
    std::cout << "Connecting to the database...\n";

    try
    {
```

[4] https://dev.mysql.com/doc/x-devapi-userguide/en/parameter-binding.html

```
        mysqlx::Session session("localhost", 33060, "user", "password");
        mysqlx::Schema db = session.getSchema("cis64a", true);

        mysqlx::Table table = db.getTable("CITY", true);
        mysqlx::RowResult result = table.select("*")
                                        .where("ID = :id")
                                        .bind("id", 4080)
                                        .execute();

        mysqlx::Row row = result.fetchOne();

        std::cout << "ID:          " << row[0] << "\n"
                  << "Name:        " << row[1] << "\n"
                  << "CountryCode: " << row[2] << "\n"
                  << "District:    " << row[3] << "\n"
                  << "Population:  " << row[4] << "\n\n";

        std::cout << "Done!\n";
    }
    catch (const mysqlx::Error &error)
    {
        std::cerr << "The following error occurred: " << error << "\n";
    }
}
```

If everything worked, then the program should output the following to the terminal

```
Connecting to the database...
ID:          4080
Name:        Twin Peaks
CountryCode: USA
District:    Washington
Population:  10500

Done!
```

Otherwise, you will see an error printed to the terminal such as

```
Connecting to the database...
ID:          <null>
Name:        <null>
CountryCode: <null>
District:    <null>
```

```
Population:  <null>


Done!
```

if you attempted to select an entry with a key that doesn’t exist.

4. **U**pdating an entry

Before updating an entry from the table, a `Table` object must be created as shown in section 2. Once the `Table` object has been created, the `update` method is called with no arguments and then the `set` method is called with the column name that is being updated along with its new value. Multiple columns can be updated with a succession of calls to the `set` method chained together. Finally, similar to section 3, the `where` and `bind` methods are used to select the desired row. The pattern for the `update` method is given in Fig. 3 below.

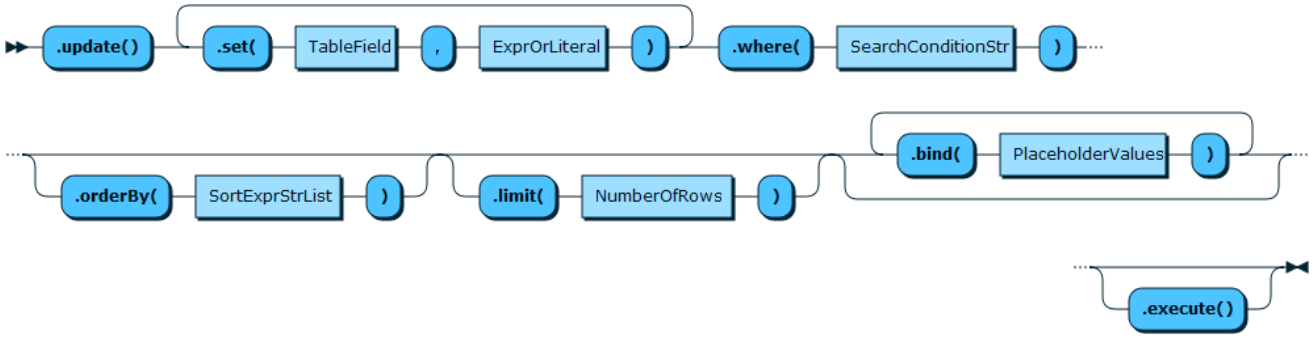

Fig. 3 Table.update() Syntax Diagram

```
/* Example 4 - Updating an entry */
#include <iostream>

#include <mysqlx/xdevapi.h>

int main()
{
    std::cout << "Connecting to the database...\n";

    try
    {
        mysqlx::Session session("localhost", 33060, "user", "password");
        mysqlx::Schema db = session.getSchema("cis64a", true);

        mysqlx::Table table = db.getTable("CITY", true);
        table.update()
             .set("population", 123456)
             .where("Name = :name AND CountryCode = :code AND District = :district")
             .bind("name", "Twin Peaks")
             .bind("code", "USA")
             .bind("district", "Washington")
             .execute();

        std::cout << "Done!\n";
    }
```

```
    catch (const mysqlx::Error &error)
    {
        std::cerr << "The following error occurred: " << error << "\n";
    }
}
```

If everything worked, then the program should output the following to the terminal

```
Connecting to the database…
Done!
```

5. **D**eleting an entry

Deleting an entry is similar to updating one, except that there's no need to make any calls to the set method. The pattern for the `remove` method is given in Fig. 4 below with one exception. The method shown in the syntax diagram shows a call to the `delete` method, but it's actually the `remove` method that's used. This was determined by using Visual Studio's "Peek Definition" functionality and inspecting the connector's source code.

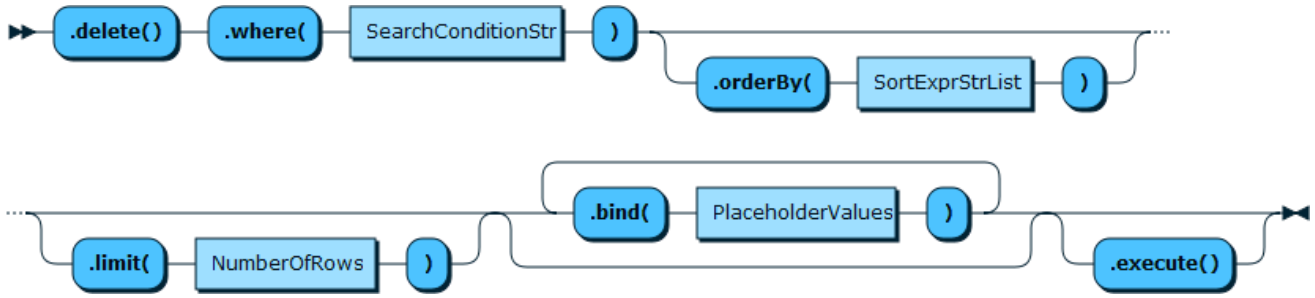

Fig. 4 Table.delete() Syntax Diagram (note that the Table.remove() method is actually used)

```
/* Example 5 - Deleting an entry */
#include <iostream>

#include <mysqlx/xdevapi.h>

int main()
{
    std::cout << "Connecting to the database...\n";

    try
    {
        mysqlx::Session session("localhost", 33060, "user", "password");
        mysqlx::Schema db = session.getSchema("cis64a", true);

        mysqlx::Table table = db.getTable("CITY", true);
        table.remove()
             .where("Name = :name AND CountryCode = :code AND District = :district")
             .bind("name", "Twin Peaks")
             .bind("code", "USA")
             .bind("district", "Washington")
             .execute();

        std::cout << "Done!\n";
    }
    catch (const mysqlx::Error &error)
```

```
    {
        std::cerr << "The following error occurred: " << error << "\n";
    }
}
```

If everything worked, then the program should output the following to the terminal

```
Connecting to the database…
Done!
```