CDBC

A common approach to databases in C++

Mathieu Ropert
mathieu.ropert@murex.com

Jeremy Demeule
jeremy.demeule@murex.com

Background

SQL is a very popular but weak standard

Most applications need to store and access persistent data. Among the possible solutions, relational databases paired with SQL has been one of the most popular choices for decades.

While many languages like C#, Oracle® Java or Ruby offer a common API to interact with RDBMS, C++ does not. Today each program has to deal with the specificities of each vendor drivers or rely on 3rd party software.

Moreover, despite being a standard, SQL is not entirely portable across different database systems. Off all the major vendors in the industry, only PostgreSQL database has made a commitment to stick to the standard. Others have preferred to keep backward compatibility with their prior implementation.

Finally, some implementation details fall outside of the standard and hinder portability. One of the most notorious example is the syntax of placeholders in prepared statements for which no universal solution exists.

Constraints

What should an API provide?

When we decided to decommission our old in-house C API, we required the replacement to be C++ friendly, cross-platform, database agnostic and provide access to raw statements.

Our developers write code that accesses the database without knowing which vendor will be used in production, which implies some abstraction of the statement building.

Since many languages available in the industry already provide similar features (ADO.NET, JDBC), a C++ API should bear some similarities for someone who is familiar with them.

Finally, in a true "pay only for what you use" C++ fashion, the API should have a low overhead compared to native drivers.

Today, there is no standard C++ API that fills all those criteria.

MUREX™

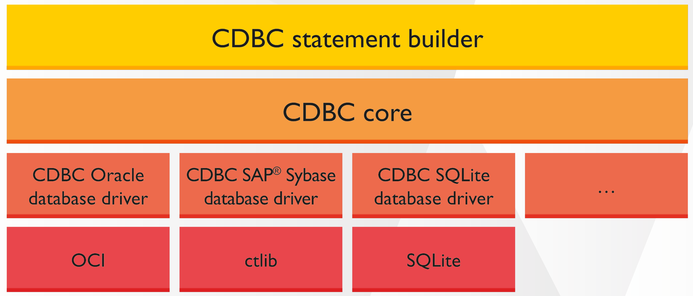
www.murex.com

CDBC philosophy

Build from the lowest, up to the top

To succeed as a standard, a database API would have to start low and build up from there. Once developers have a way to write and execute portable SQL queries without having to worry about drivers or non-standard extensions to the language, everything becomes possible.

Following this philosophy, we divided CDBC in two parts, each answering one of the primary concerns: abstracting vendors drivers and writing portable SQL queries.



CDBC core

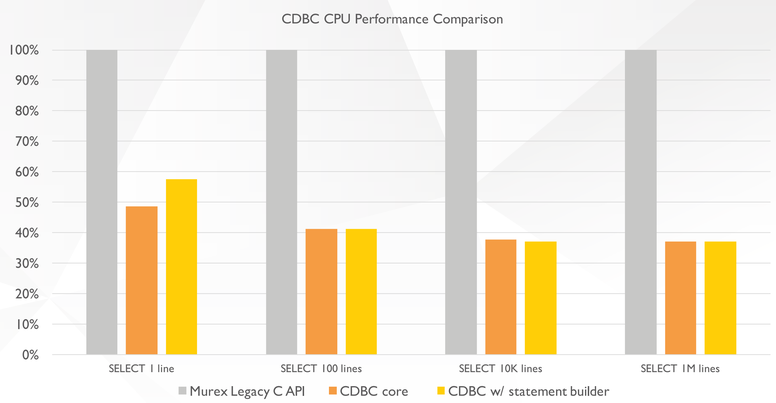
The lower tier of CDBC is meant to abstract the various vendor drivers into a single consistent API that each vendor can in turn implement.

It features a simple API for basic RDBMS operations (connection, statement execution, result set fetching and transactions) that abstracts the underlying vendor driver, as well as more advanced features such as connection pooling.

It is designed to provide performance that is very close to native drivers.

CDBC statement builder

Since most vendors deviate from the standard a little bit, CDBC provides a way to generate portable SQL. By using a builder pattern, it enables the developers to write universal queries that can be transformed to raw statements by the selected driver.



Implementation

Something new, yet familiar

The core of CDBC is inspired by N3886, the latest proposal to the ISO C++ Study Group 11 (database), written in 2014.

The syntax should be easy to understand to anyone with minimal background in SQL:

```
db::connection cnx("oracle");
cnx.open("db.murex.com");
db::statement stm("select * from EMPLOYEES", cnx);

for (db::result res(stm.execute()); !res.is_eof(); res.move_next()) {
   std::cout
         << "- " << res.get<std::string>("NAME")
         << ", age: " << res.get<int>("AGE")
         << ", salary: $" << res.get<double>("SALARY")
         << std::endl;
}
```

For CDBC's statement builder, we took inspiration from the LINQ syntax that can be found in .NET:

```
db::statement stm(
   db::statement_builder(cnx)
      .from("EMPLOYEES")
      .where(db::column("AGE") > db::var("max_age")
            && db::column("SALARY") < db::var("min_salary"))
      .select("NAME")
      .to_statement());

stm.bind(db::var("max_age"), 50);
stm.bind(db::var("max_salary"), 42000.0);
db::result res(stm.execute());
```

Conclusion

From a former draft to a new proposal

CDBC is currently offered in "early-access" internally to our developers. We plan to decommission our legacy C database API by rewriting our in-house ORM and DSL on top of it.

We plan to publish CDBC as open source as soon as it is ready and submit a draft proposal to the C++ normalization committee afterwards.

Today we have drivers for Oracle database, SAP Sybase database and SQLite database. We also have prototypes working with Oracle MySQL database and Apache Cassandra database. Since handling Cassandra Query Language (CQL) was not an issue, we think CDBC can be extended to NoSQL databases in the future

References

Anhofer, J. 2014, *A proposal to add a Database Access Layer to the Standard Library (N3886).*

Credits