wvlet-sql: Integrating SQL and Scala for Data Analytics

Taro L. Saito

Initial Design Doc: Feb 17th, 2017

GOALS

Provide a standard of SQL value objects for Scala in order to:

- **Reuse sbt-sql generated model classes**
- **Allow nesting SQL queries in sbt-sql**

BACKGROUND

sbt-sql https://github.com/xerial/sbt-sql helps creating model classes from SQL, but adding conditions to SQL is not so easy. For example, if we need to refine the original query:

```
select * from users
```

by adding a condition id = 1, we need to add a template parameter like

```
select * from users where id = ${user_id:Long}
```

Even though the result schema shape is the same, we need to prepare two SQL files and two SQL model classes in sbt-sql.

On the other hand, if we fully resort to Scala collection, we cannot push down the condition to SQL side:

```
Users
  .select             // Seq[user]    collect 10,000 user objects from JDBC query result
  .filter(_.id == 1)     // Seq[user]    1 record
```

If we run the whole steps in a single SQL, it will finish quickly. So instead of using Seq[user], we should use operator objects, which are monad to wrap the actual computations:

```
Users.select            // SQLOp[user]
        .filter(_.id == 1)  // FilterOp(SQLOp[user], _.id == 1)
```

SQLOp and FilterOp are just operator objects and does not hold the results, so we can optimize this sequence of operators as a single SQLOp:

```
SQLOp("select * from (select * from users where id = 1)")
```

DESIGN

- Define WvSeq[A] and SQLOp[A] operator
- Convert SQLOp[A] to an SQL statement
- Translate WvSeq[A].filter(_.(column name) (pred) (value)) to SQLOp[A]
    - Use Scala Macros to analyze the predicate expression
    - SQLOp[A] should not change the output type. This is for embedding it to another SQL that requests type A input.
- Add WvSeq output type support to sbt-sql
- Nesting SQL
    - SQL: select * from ${a:A} a join ${b:B} b on a.id = b.id
    - code: SQL.select(a = A.select.filter(_.id == 1), b = B.select.filter(_.country=="US"))
    - Result:
        - Return SQLOp("select * from (select * from a where id = 1) a join (select * from b where country = 'US') b on a.id = b.id") , which can be executed through JDBC.
        - If the filtering operation is too complex, we should detect it as compilation error (cannot generate SQL for this expression, etc.)

ALTERNATIVES AND WHY NOT?

In Scala, there are many libraries for accessing DBs. Slick, Quill, Doobie and ScalikeJDBC are these examples. An important question to decide `which to use?` is whether we are building **database applications** or **analyzing data with SQL?** wvlet-sql is targeting the latter; data analysis code in which we need to explore large data sets. The schema of the database might be quite huge (e.g., hundreds of columns) and in order to build correct SQL queries, we need check the actual data set and schemas.

Slick

Maybe too complex to use. And found several pain points in generated SQL:
http://fr.slideshare.net/normation/doobie-feedbacks-from-the-trenches-scalaio-2016#15

Quill

Quill https://github.com/getquill/quill is Scala macro based SQL generator, which is similar to the idea of wvlet-sql.

- This Scala expression parsing code would be useful for wvlet-sql https://github.com/getquill/quill/blob/master/quill-core/src/main/scala/io/getquill/quotation/Parsing.scala

Quill's design aims to **compile-time SQL generation**, which means we cannot automate creating model classes before the compilation. And as long as we know the result schema type in compile-time, generating the SQL syntax at compile-time is not necessary.

And also use of lift for making SQL as a function is less natural:

```
def users(id: Long) = quote {
  query[User].filter(u => u.id == lift(id))
}
```

Doobie

Doobie https://github.com/tpolecat/doobie and its design
http://tpolecat.github.io/presentations/doobie1.html#8
If we need to build SQL in a functional way, it would be useful, but this is not the goal of wvlet-sql.

Spark SQL and Dataset API

SQL syntax is Spark SQL specific. We cannot use Presto UDFs.

CONTACT

- Taro L. Saito (GitHub @xerial, e-mail: leo@xerial.org)