Database Mandatory Assignment

CRUD Application Using an ORM

April 2020

Martin Løseth Jensen

What is an ORM?

An ORM (object-relational mapping) is this concept of taking an object-oriented language and mapping it to a relational database. An ORM is, therefore, a framework that programming language can use to map objects into rows and columns.

There are many frameworks and libraries for many different programming languages that provide the mapping of objects into or from relational databases.
In Java, you might use Hibernate as an ORM. In most cases with the JVM, annotations are provided in with the library to make the work less painful for the developer.

I decided to use a library from JetBrains called Exposed. It embraces the language features of Kotlin with the ease of use with DSL. You will find a link to the GitHub repository in the link section at the end of this paper.

Code Snippets

To connect to a relational database (In my case I use MySQL):

```
fun main() {
   val database = Database.connect(
"jdbc:mysql://localhost:3306/mandatory_assignment?useUnicode=true&use
JDBCCompliantTimezoneShift=true&useLegacyDatetimeCode=false&serverTim
ezone=UTC",
       driver = "com.mysql.jdbc.Driver", user = "<user>", password =
"<password>"
   )
```

All SQL statements need to be inside a “transaction” scope:

```
transaction {
        // print sql to std-out
        addLogger(StdOutSqlLogger)
        ...
}
```

To create a object that correlates to a table (Note: we can easily use “IntIdTable” if there will be a unique integer id that is auto incremented):

```
object FavouriteAudioBooks : IntIdTable() {
   val title: Column<String> = varchar("title", 250)
   val author: Column<String> = varchar("author", 250)
   val narrator: Column<String> = varchar("narrator", 250)
   val genre: Column<String> = varchar("genre", 75)
}
```

Create a table inside the schema if table not already exists (Note: also need to be inside “transaction” scope):

```
SchemaUtils.create(FavouriteAudioBooks)
```

CREATE:

```
transaction {
        ...
        val favId = FavouriteAudioBooks.insertAndGetId {
            it[title] = "B"
            it[author] = "BB"
            it[narrator] = "BBB"
            it[genre] = "BBBB"
        }
        println(favId)
        ...
}
```

READ:

```
transaction {
        ...
        FavouriteAudioBooks
            .select { FavouriteAudioBooks.id eq 1 }
            .forEach {
                println(it[FavouriteAudioBooks.title])
            }

        // Distinct values
        val narrators = FavouriteAudioBooks
            .slice(FavouriteAudioBooks.title)
            .select { FavouriteAudioBooks.id lessEq 10 }
            .withDistinct()
            .map {
                it[FavouriteAudioBooks.title]
            }
        narrators.forEach { println(it) }
        ...
}
```

UPDATE:

```
transaction {
        ...
        FavouriteAudioBooks.update({ FavouriteAudioBooks.id eq 3 }) {
            it[title] = "C"
            it[author] = "CC"
            it[narrator] = "CCC"
            it[genre] = "CCCC"
        }
        ...
}
```

DELETE:

```
transaction {
    ...
    FavouriteAudioBooks.deleteWhere { FavouriteAudioBooks.id eq 4 }
    ...
}
```

The whole code:

```
import org.jetbrains.exposed.dao.id.IntIdTable
import org.jetbrains.exposed.sql.*
import org.jetbrains.exposed.sql.transactions.transaction

fun main() {
   val database = Database.connect(
"jdbc:mysql://localhost:3306/mandatory_assignment?useUnicode=true&use
JDBCCompliantTimezoneShift=true&useLegacyDatetimeCode=false&serverTim
ezone=UTC",
       driver = "com.mysql.jdbc.Driver", user = "<user>", password =
"<password>"
   )

   transaction {
       // print sql to std-out
       addLogger(StdOutSqlLogger)

       SchemaUtils.create(FavouriteAudioBooks)

       // CREATE
       val favId = FavouriteAudioBooks.insertAndGetId {
           it[title] = "B"
           it[author] = "BB"
           it[narrator] = "BBB"
           it[genre] = "BBBB"
       }
       println(favId)

       // READ
```

```
        FavouriteAudioBooks
            .select { FavouriteAudioBooks.id eq 1 }
            .forEach {
                println(it[FavouriteAudioBooks.title])
            }

        // Distinct values
        val narrators = FavouriteAudioBooks
            .slice(FavouriteAudioBooks.title)
            .select { FavouriteAudioBooks.id lessEq 10 }
            .withDistinct()
            .map {
                it[FavouriteAudioBooks.title]
            }
        narrators.forEach { println(it) }

        // UPDATE
        FavouriteAudioBooks.update(where = { FavouriteAudioBooks.id eq
3 }) {
            it[title] = "C"
            it[author] = "CC"
            it[narrator] = "CCC"
            it[genre] = "CCCC"
        }

        // DELETE
        FavouriteAudioBooks.deleteWhere { FavouriteAudioBooks.id eq 4
}
    }
}

object FavouriteAudioBooks : IntIdTable() {
    val title: Column<String> = varchar("title", 250)
    val author: Column<String> = varchar("author", 250)
    val narrator: Column<String> = varchar("narrator", 250)
    val genre: Column<String> = varchar("genre", 75)
}
```

Links

Public Repo on GitHub

JetBrains Exposed Library on GitHub