IT5100A

Team member:

Li Guoshen A0237348X He Wenbin(A0237325H) Huang Weifeng (A0237343H)

In our backend based database system project, we mainly use SLICK as the main library to build our system.

We begin by manipulating this table within the simplest database.

Database construction:

The database has three columns with employeeId, age and departmentId. For the sake of simplicity, we define all of them as integers.

Database initalization:

Using the employeeId as the primary key, we need to have the following code in scala to initialize the table in the database.

```
object SlickDB
{
    // table name: runoob_tbl
    case class UserInfo(id: Int, name: String, age: Int)

    class SlickModelTable(tag: Tag) extends Table[UserInfo](tag, "runoob_tbl")
    {
        // define column attribute
        def employeeId: Column[Int] = column[Int]("employeeId", O.PrimaryKey, O.AutoInc) // make sure here is prim
        def age: Column[String] = column[String]("age")
        def departmentId: Column[Int] = column[Int]("departmentId")
        def *: ProvenShape[UserInfo] = (employeeId, age, departmentId) <> (UserInfo.tupled, UserInfo.unapply)
    }
    def slick_table: TableQuery[SlickModelTable] = TableQuery[SlickModelTable]
}
```

In the code above, we can see that there are three lines of code defining the three columns of the table in the database. Finally, we define a slick_table as a way to do the query of the table in the database.

Database connection:

In order to access the database, we need to connect the database, with the correct url, driver, user and password.

```
val db = Database.forURL(
    url = "jdbc:mysql://localhost:3306/Chinook",
    driver = "com.mysql.cj.jdbc.Driver",
    user = "root",
    password = "12345678")
```

Database query:

1. If we want to query all the rows and columns in the table, we can use the following code.

```
val q1 = for (c <- slick_table) yield c


val result = db.withSession {
    session =>
        q1.list()( session )
}

println(result)
```

The query q1 aims at getting all the rows in the table. Then we convert to a list and then display the result.

```
[info] compiling 1 Scala source to /Users/liguoshen/IdeaProjects/Slick/target/sca
[info] running cn.scala.xtwy.Example
List(UserInfo(1,13,1), UserInfo(2,26,1), UserInfo(3,30,2), UserInfo(4,25,2))
```

The resulted List shown that it contains 4 rows, each one is a UserInfo() with all the data in it. It is shown that we have already successfully retrieve all the rows and columns in the table.

This works like **select * from runoob_tbl** in Mysql.

2. If we want to query with specific requirements just like using the “where” key word in Mysql.

   For example, If we want to query the rows with departmentId = 1.
   We can do the following:

```
val q2 = slick_table.filter(_.departmentId === 1)

val result = db.withSession {
    session =>
        q2.list()( session )
}

println(result)
```

   The result is shown as follow:

```
[info] compiling 1 Scala source to /Users/liguoshen/IdeaProjects/Slick/tar
[info] running cn.scala.xtwy.Example
List(UserInfo(1,13,1), UserInfo(2,26,1))
```

   We can see that the two rows that are satisfied the requirement are listed as above.

   This works like **select * from runoob_tbl where departmentId = 1** in Mysql.

3. We can also do a query like the limit and offset in SQL.

```
val q3 = slick_table.drop(1).take(1)

val result = db.withSession {
    session =>
        q3.list()( session )
}

println(result)
```

   The result is shown as follow:

```
[info] loading settings for project slick from build.sbt ...
[info] set current project to Scala Slick Examples (in build file:/Users/liguosh
[info] compiling 1 Scala source to /Users/liguoshen/IdeaProjects/Slick/target/sc
[info] running cn.scala.xtwy.Example
List(UserInfo(2,26,1))
```

This works like **select * from runoob_tbl limit 1 offset 1** in Mysql.

4. We can also sort the rows by a specific orders.

```
val q4 = slick_table.sortBy(_.age)

val result = db.withSession {
    session =>
        q4.list()( session )
}

println(result)
```

This means that we sort the row by the value of age (which is in an ascending order.)

The result:

```
[info] compiling 1 Scala source to /Users/liguoshen/IdeaProjects/Slick/target/scala-2.10/
[info] running cn.scala.xtwy.Example
List(UserInfo(1,13,1), UserInfo(4,25,2), UserInfo(2,26,1), UserInfo(3,30,2))
```

This works like **select * from runoob_tbl order by age asc** in Mysql.

5. We can also combine the query together. We sort first, and then do the filter.

```
val q4 = slick_table.filter(_.departmentId === 1)
val q5 = q4.sortBy(_.age)

val result = db.withSession {
    session =>
        q5.list()( session )
}

println(result)
```

The result:

```
[info] compiling 1 Scala source to /Users/liguoshen/IdeaProjects/Slick/target/scala-
[info] running cn.scala.xtwy.Example
List(UserInfo(1,13,1), UserInfo(2,26,1))
```

This works like **select * from runoob_tbl where departmentId = 1 order by age asc** in Mysql.

6. We can also query a specific column.

```
val q6 = slick_table.map(_.age)

val result = db.withSession {
    session =>
        q6.list()( session )
}

println(result)
```

The result is as follow:

```
[info] compiling 1 Scala source to /Users/liguoshen/IdeaProjects/Slick/tar
[info] running cn.scala.xtwy.Example
List(13, 26, 30, 25)
```

This works like **select age from runoob_tbl** in Mysql.

7. We can also combine the result by doing the following operation.

```
val q7 = slick_table.filter(_.departmentId === 1)
val q8 = slick_table.filter(_.departmentId === 2)
val q9 = q7 ++ q8
val result = db.withSession {
    session =>
        q9.list()( session )
}
println(result)
```

The result is as follows:

```
[info] compiling 1 Scala source to /Users/liguoshen/IdeaProjects/Slick/target/s
[info] running cn.scala.xtwy.Example
List(UserInfo(1,13,1), UserInfo(2,26,1), UserInfo(3,30,2), UserInfo(4,25,2))
```

8. We can also do the max, min, avg, sum aggregation which is as follow.

```
val q6 = slick_table.map(_.age)
val q7 = q6.max
val q8 = q6.min
val q9 = q6.avg
val q10 = q6.sum
```

This works like **select max(age), min(age), avg(age), sum(age) from runoob_tbl** in Mysql.

9. We can also do some complex query with group by.

```
var q10 = slick_table.groupBy(c => c.departmentId).map({
    case (departmentId, group) => (departmentId, group.map(_.age).avg)
})

val result = db.withSession {
    session =>
        q10.list()( session )
}
println(result)
```

These a few lines of code is to calculate the mean age of each department.
Select departmentId, avg(age) from runoob_tbl group by departmentId

The result:

```
[info] compiling 1 Scala source to /Users/liguoshen/IdeaProjects/Slick/targe
[info] running cn.scala.xtwy.Example
List((1,Some(19.5000)), (2,Some(27.5000)))
```

Then, we test the insertion, update and delete operation.
We modified age into a String variable for better visualization.

The original data：

```
mysql> select * from runoob_tbl;
+------------+-------+--------------+
| employeeId | age   | departmentId |
+------------+-------+--------------+
|          1 | 123   |            1 |
|          2 | test1 |            1 |
|          4 | test3 |            4 |
|          5 | test4 |            4 |
+------------+-------+--------------+
4 rows in set (0.00 sec)
```

10. insertion：

```
// Example2: Inserting data
// Define the new data
def freshTestData = Seq(
    UserInfo("test1", 1),
    UserInfo("test2", 2),
    UserInfo("test3", 3),
    UserInfo("test4", 4)
)

val insert: DBIO[Option[Int]] = slick_table ++= freshTestData
val insertAction = db.run(insert)
val rowCount = Await.result(insertAction, 2.seconds)
rowCount.map(x => println(x))
```

The output of Terminal:

```
4
[success] Total time: 6 s, completed
```

The database:

```
mysql> select * from runoob_tbl;
+------------+-------+--------------+
| employeeId | age   | departmentId |
+------------+-------+--------------+
|          1 | 123   |            1 |
|          2 | test1 |            1 |
|          4 | test3 |            4 |
|          5 | test4 |            4 |
|       1001 | test1 |            1 |
|       1002 | test2 |            2 |
|       1003 | test3 |            3 |
|       1004 | test4 |            4 |
+------------+-------+--------------+
8 rows in set (0.00 sec)
```

11. Deletion

```
val removeHal: DBIO[Int] = slick_table.filter(_.departmentId === 2).delete
val removeHalFuture = db.run(removeHal)
val removeHalResults = Await.result(removeHalFuture, 2.seconds)
println(removeHalResults)
```

The output of terminal:

```
1
[success] Total time: 6 s, completed
```

The output of database:

```
mysql> select * from runoob_tbl;
+------------+-------+--------------+
| employeeId | age   | departmentId |
+------------+-------+--------------+
|          1 | 123   |            1 |
|          2 | test1 |            1 |
|          4 | test3 |            4 |
|          5 | test4 |            4 |
|       1001 | test1 |            1 |
|       1003 | test3 |            3 |
|       1004 | test4 |            4 |
+------------+-------+--------------+
7 rows in set (0.00 sec)
```

12. Update the data.

```
val updateQuery: DBIO[Int] = slick_table.filter(_.departmentId === 3).map(_.departmentId).update(4)
val updateFuture = db.run(updateQuery)
val updateResults = Await.result(updateFuture, 2.seconds)
println(updateResults)
```

The output of terminal:

```
1
[success] Total time: 6 s, completed
```

The output of database.

```
mysql> select * from runoob_tbl;
+------------+-------+--------------+
| employeeId | age   | departmentId |
+------------+-------+--------------+
|          1 | 123   |            1 |
|          2 | test1 |            1 |
|          4 | test3 |            4 |
|          5 | test4 |            4 |
|       1001 | test1 |            1 |
|       1003 | test3 |            4 |
|       1004 | test4 |            4 |
+------------+-------+--------------+
7 rows in set (0.00 sec)
```