School of Computing

IT5100A Final Report

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

1. Introduction

Slick is a modern library that allows scala users to query and access databases as if using scala collections. It provides users the flexibility to perform database CRUD operation not just with plain SQL language but also using scala directly, which benefits coders from the compositionality, static checking and compile-time safety features brought by scala.

In this project, the team has decided to mainly focus on the backend development utilizing slick through its functional relational mapping to implement a preliminary prototype for employee management system (EMS). The EMS contains three tables that are employee, department and performance to achieve a total of 12 features provided by the system. In the following section, the team would illustrate the philosophy of the database design followed by the details of program execution to highlight the main features of the system.

2.Database design

The database design follows the fourth normal form Boyce-Codd Normal Form (BCNF) to reduce possible redundancy. In addition, primary key and foreign keys are enforced to satisfy the business logic. The database schema is as shown in figure 1. Primary key of each table is highlighted in bold while foreign key constraints between each table are linked and highlighted in light blue. In the table employee, department_id is a foreign key that references the department_id in the table department to ensure that the department must exist before we record a new employee with the respective department_id into the system. Similarly, employee_id and department_id are foreign keys in the table performance restricts the fact that an employee must exist in the system before we could file a report for his/her performance evaluation.

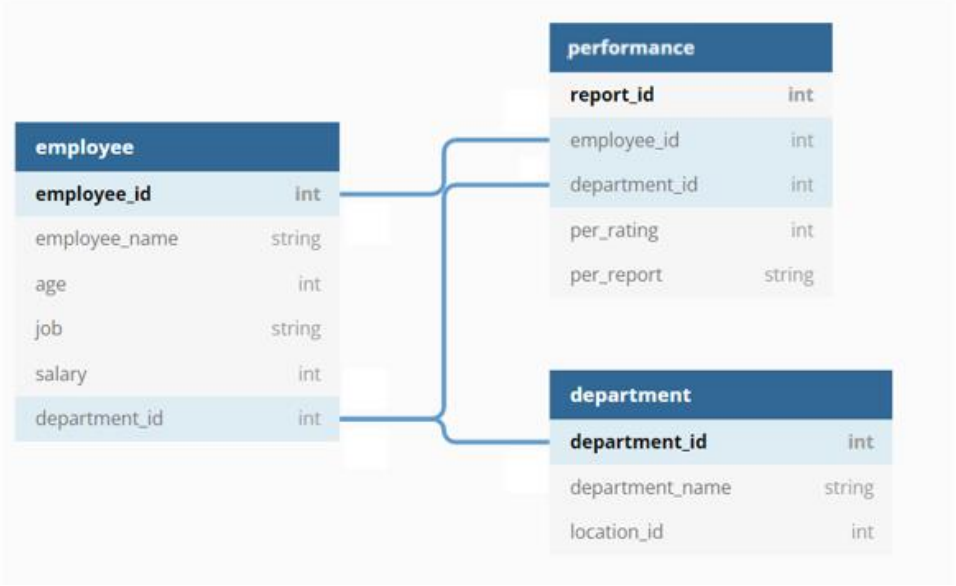


Figure 1

3.Program execution

3.1 Overview of the three tables

At the beginning, we created a database named “chinook”. By using “sbt run” command in the terminal to run our program, all three tables defined in the DB_define file will be auto generated in the “chinook” database, Figure 2 demonstrates the 3 tables “department”, “employee” and “performance” generated.

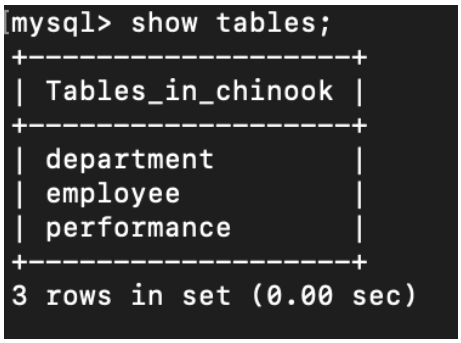


Figure 2

To demonstrate the functionality of the program, the team has created some data for initialization of the database. Data in “department”, “employee” and “performance” are shown below in Figure 3, Figure 4, and Figure 5 respectively.

The data in the department table:

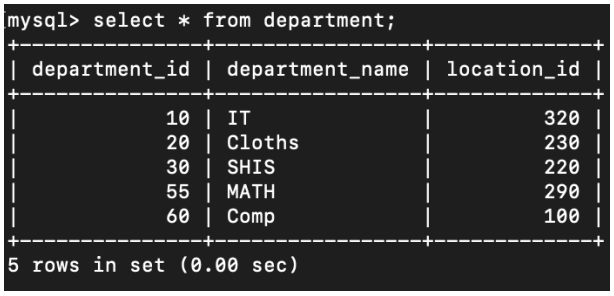


Figure 3

The data in the employee table:

```
mysql> select * from employee;
+---------------+-----+-----------+--------+---------------+-------------+
| employee_name | age | job       | salary | department_id | employee_id |
+---------------+-----+-----------+--------+---------------+-------------+
| Mary          |  18 | Calculate |   1800 |            10 |           1 |
| Michael       |  25 | Speak     |   1600 |            20 |           2 |
| Leo           |  27 | Jump      |   1800 |            30 |           3 |
+---------------+-----+-----------+--------+---------------+-------------+
3 rows in set (0.00 sec)
```

Figure 4

The data in the performance table:

```
mysql> select * from performance;
+-------------+---------------+------------+-----------------------------------------+-----------+
| employee_id | department_id | per_rating | per_report                              | report_id |
+-------------+---------------+------------+-----------------------------------------+-----------+
|           1 |            10 |         88 | Well done in the taskA                  |         1 |
|           1 |            10 |         60 | a lot of room for improvements in taksB |         2 |
|           2 |            20 |        100 | Performed perfectly in taksC            |         3 |
|           2 |            20 |        100 | Performed perfectly in taksD            |         4 |
+-------------+---------------+------------+-----------------------------------------+-----------+
4 rows in set (0.00 sec)
```

Figure 5

3.2 Features provided in the EMS

After executing the main_loop file, the system displays the welcome message and all the services that users can choose from. Overall, there are 13 services that users can select from as shown in Figure 6.In the event that the user wants to use a specific service, he/she can enter the number of that service in the terminal. Any unexpected input would prompt the user to re-enter.

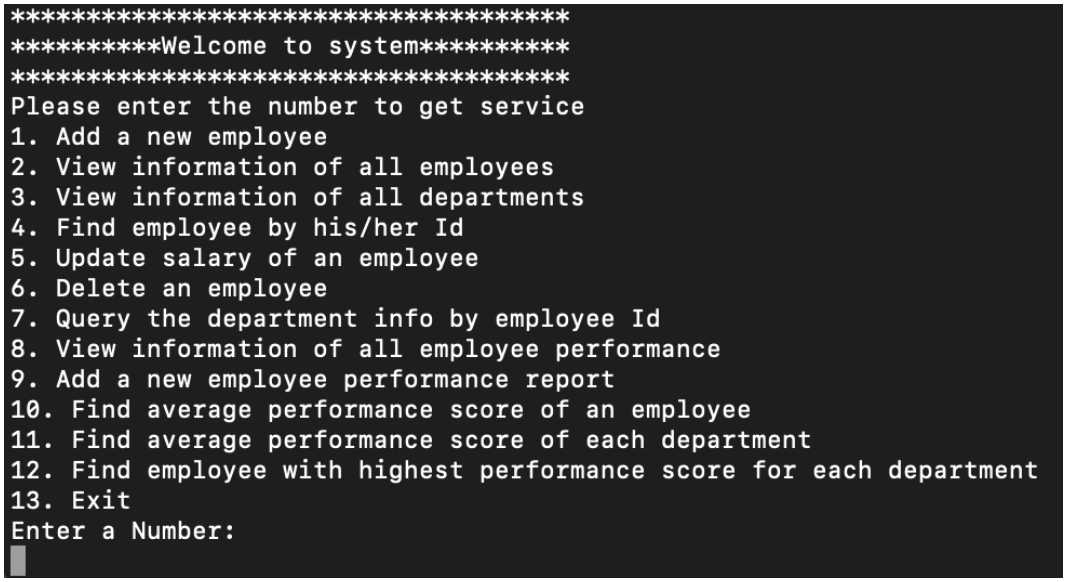


Figure 6

Feature 1: Add a new employee

To add a new employee, user is expected to provide the name, job, age, salary as well as the department_Id of the employee. Employee_id would be auto-assigned by the system to avoid key collision. Notice that the attribute department_Id is a foreign key referencing to the department_Id attribute in the department table, Hence, user must make sure that the provide department_Id exists in the database.

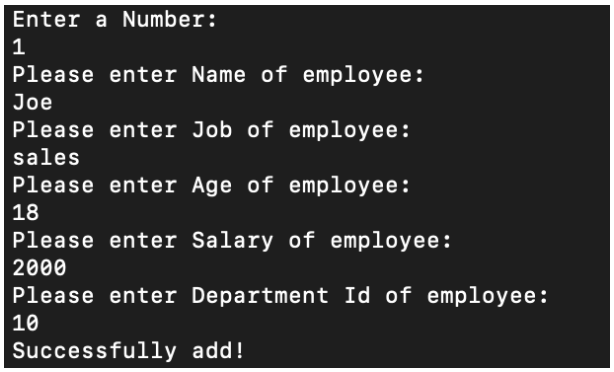


Figure 7

If the employee is added successfully, the system will print out “Successfully add!” as shown in Figure 7. In the figure 8, we can see that the employee “Joe” has been added into the database successfully.

```
mysql> select * from employee;
+---------------+-----+-----------+--------+---------------+-------------+
| employee_name | age | job       | salary | department_id | employee_id |
+---------------+-----+-----------+--------+---------------+-------------+
| Mary          |  18 | Calculate |   1800 |            10 |           1 |
| Michael       |  25 | Speak     |   1600 |            20 |           2 |
| Leo           |  27 | Jump      |   1800 |            30 |           3 |
| Joe           |  18 | sales     |   2000 |            10 |           5 |
+---------------+-----+-----------+--------+---------------+-------------+
4 rows in set (0.00 sec)
```

Figure 8

In the event a department_Id that does not exist is provided, the system will prompt the user to provide the correct department_id as shown in Figure 9.

```
Enter a Number:
1
Please enter Name of employee:
Joe
Please enter Job of employee:
sales
Please enter Age of employee:
18
Please enter Salary of employee:
2000
Please enter Department Id of employee:
1
Foreign key issues, the department id you key in is not in the department table! Please try again!
```

Figure 9

Feature 2: View information of all employees

This function aims at displaying all the information of the employees as shown Figure 10. Each EmpInfo contains information of employee_name, age, job, salary, department_id, employee_id. Such info can be customized based on business needs by simply altering the table schema.

```
Enter a Number:
2
EmpInfo(Mary,18,Calculate,1800,10,1)
EmpInfo(Michael,25,Speak,1600,20,2)
EmpInfo(Leo,27,Jump,1800,30,3)
EmpInfo(Joe,18,sales,2000,10,5)
```

Figure 10

Feature 3: View information of all departments

This function aims at showing all the information of the different departments. Each DepartmentInfo contains the information of department_id, department_name, location_id as shown below in Figure 11.

```
Enter a Number:
3
DepartmentInfo(10,IT,320)
DepartmentInfo(20,Cloths,230)
DepartmentInfo(30,SHIS,220)
DepartmentInfo(55,MATH,290)
DepartmentInfo(60,Comp,100)
```

Figure 11

Feature 4: Find employee by employee Id

In this function, we can find the information of an employee by entering his/her Id.

```
Enter a Number:
4
Please enter Id of employee:
1
EmpInfo(Mary,18,Calculate,1800,10,1)
```

Figure 12

If the Id does not exist, the system will alert the user to check the employee ID.

```
Enter a Number:
4
Please enter Id of employee:
100
the Id of employee does not exist, please check!
```

Figure 13

Feature 5: Update salary of an employee

This function aims at updating the employee salary with the provision of employee Id. Here is the employee table before the update as shown in Figure 14.

```
mysql> select * from employee;
+---------------+-----+-----------+--------+---------------+-------------+
| employee_name | age | job       | salary | department_id | employee_id |
+---------------+-----+-----------+--------+---------------+-------------+
| Mary          |  18 | Calculate |   1800 |            10 |           1 |
| Michael       |  25 | Speak     |   1600 |            20 |           2 |
| Leo           |  27 | Jump      |   1800 |            30 |           3 |
| Joe           |  18 | sales     |   2000 |            10 |           5 |
+---------------+-----+-----------+--------+---------------+-------------+
4 rows in set (0.00 sec)
```

Figure 14

We perform the update on the employee's salary whose employee_id = 1.

```
Enter a Number:
5
Please enter Id of employee:
1
Please enter salary of employee:
6000
successfully update!
```

Figure 15

It is observed that the salary of Mary (employee_id = 1) is updated to 6000 as expected.

```
mysql> select * from employee;
+---------------+-----+-----------+--------+---------------+-------------+
| employee_name | age | job       | salary | department_id | employee_id |
+---------------+-----+-----------+--------+---------------+-------------+
| Mary          |  18 | Calculate |   6000 |            10 |           1 |
| Michael       |  25 | Speak     |   1600 |            20 |           2 |
| Leo           |  27 | Jump      |   1800 |            30 |           3 |
| Joe           |  18 | sales     |   2000 |            10 |           5 |
+---------------+-----+-----------+--------+---------------+-------------+
4 rows in set (0.00 sec)
```

Figure 16

Similarly, this feature caters error handling. In the event that the employee Id provided does not exist in the database, the system will prompt the user of the issue.

```
Enter a Number:
5
Please enter Id of employee:
100
Please enter salary of employee:
3000
The employee Id you entered is wrong!
```

Figure 17

Feature 6: Delete an employee

If the employee Id does not exist in the database, the system will output a message to remind the user. Otherwise, the user will be deleted from the database. In addition, on Delete behavior of the foreign key in the performance table would be automatically triggered to delete the related entries in the table. This is because if an employee is fired from the company, it would be a waste of resources to continue store his/her performance evaluation in the database.

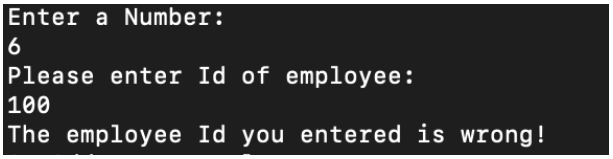


Figure 18

Feature 7: Query the department info by employee Id

This feature allows the user to retrieve department information of an employee in interest by providing the employee Id. In Figure 19, we retrieve the department information of the employee with Id =3.

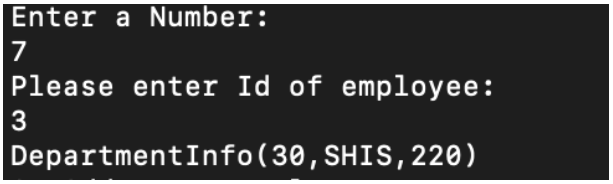


Figure 19

Similarly, if the employee Id does not exist, the system will prompt a message to alert the user.

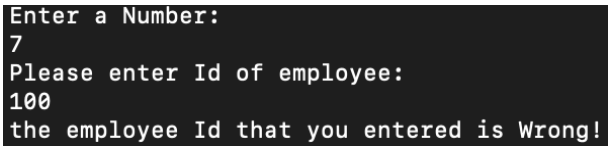


Figure 20

Feature 8: View information of all employee performance

In the event the user wants to have a summary of all the employees, he/she can utilize this feature.

Each EmpPer entry contains employee_id, department_id, per_rating, report, and report_id as shown in Figure 21.

```
Enter a Number:
8
EmpPer(1,10,88,Well done in the taskA,1)
EmpPer(1,10,60,a lot of room for improvements in taksB,2)
EmpPer(2,20,100,Performed perfectly in taksC,3)
EmpPer(2,20,100,Performed perfectly in taksD,4)
```

Figure 21

Feature 9: Add a new employee performance report

Each employee is most likely handling more than 1 tasks. Hence, it is possible that multiple evaluation reports are recorded for each employee.

```
Enter a Number:
9
Please enter Id of employee you want add report to:
1
Please enter Department Id of employee you want add report to:
30
Please enter the score for his/her performance:
6
Please provide comments for this evaluation:
well done
Successfully add!
```

Figure 22

It would only make sense to add an evaluation towards an existing employee, thus, the user must provide an existing employee Id and department Id to ensure the validity of the action.

It is observable that with the valid input the report has been successfully added in the database as shown in Figure 23.

```
mysql> select * from performance;
+-------------+---------------+------------+----------------------------------------+-----------+
| employee_id | department_id | per_rating | per_report                             | report_id |
+-------------+---------------+------------+----------------------------------------+-----------+
|           1 |            10 |         88 | Well done in the taskA                 |         1 |
|           1 |            10 |         60 | a lot of room for improvements in taksB |         2 |
|           2 |            20 |        100 | Performed perfectly in taksC           |         3 |
|           2 |            20 |        100 | Performed perfectly in taksD           |         4 |
|           1 |            30 |          6 | well done                              |         5 |
+-------------+---------------+------------+----------------------------------------+-----------+
5 rows in set (0.00 sec)
```

Figure 23

If the employee_Id or the department_Id the user entered is not in the table, it will print out a message as shown in Figure 24.

```
Enter a Number:
9
Please enter Id of employee you want add report to:
100
Please enter Department Id of employee you want add report to:
10
Please enter the score for his/her performance:
90
Please provide comments for this evaluation:
cool
Foreign key issues, the department id you key in is not in the department table! Please try again!
```

Figure 24

Feature 10: Find average performance score of an employee

To facilitate employee performance evaluation, users can get the average score of specific employee with the provision of employee_id. Figure 25 shows the average score of employee with employee_id =1.

```
Enter a Number:
10
Please enter Id of employee:
1
EmpPer(1,10,88,Well done in the taskA,1)
EmpPer(1,10,60,a lot of room for improvements in taksB,2)
The average score is 74
```

Figure 25

If the employee_Id does not exist, the user would be reminded that the employee does not have any ratings yet.

```
Enter a Number:
10
Please enter Id of employee:
100
The employee doesn't have any ratings yet!
```

Figure 26

Feature 11: Find average performance score of each department

There are cases when the user wants to evaluate the overall performance of a department. Feature 11 allows the user to obtain the average score over all employees for respective department.

The system will output the average score of all the departments as shown in Figure 27.

```
Enter a Number:
11
Department_ID   Department_average_score
      10          74.00
      20          100.00
```

Figure 27

Feature 12: Find employee with highest performance score for each department

The user could use this feature to find the best employee in each department. Considering each employee can have more than one evaluation, the system would first calculate the average score of each employee over all the evaluations they received, then group the employee based on the department they belong to and provide employee with the highest average score.
The system provides related information of the employee with the highest performance score for each department.

```
Enter a Number:
12
Department_ID  Employee_ID  Employee_Name  Score
       1           10            Mary       74
       2           20          Michael      100
```

Figure 28

Feature 13: Exit

Lastly, the user can exit the system using key 13.

```
Enter a Number:
13
Thank you for using, byebye~
```

Figure 29

4.Further improvement

Based on the previously designed program, we have made some improvements to the program, followed the characteristics of OOP and functional programming, and designed the second version. But time is tight, this version may not be perfect. The Github link is as follows: **https://github.com/HeWenbin-bobo/IT5100A-Project.git**.

4.1 OOP programming

Component 1: class DBUtil – for database connection and management

The DBUtil class is designed to modify the configuration parameters in the configuration file without operating in the related program. In addition, this class can be called directly to connect to the database, execute SQL statements, and close related variables, which reduces the risk of data leakage.

```
object DBUtil {
    // 对于全局变量,可以用 _ 来表示默认初始值
    var driver:String = _
    var url:String = _
    var user:String = _
    var password:String = _
    var connectionPool:String = "disabled"
    loadConfig()
    var db = this.getConnection()

    // 读取属性文件properties并获取内容,返回值为Unit
    def loadConfig () : Unit = {
        // 准备一个空的map，没有key-value(没有找到scala的代替方法)
        val prop:Properties = new Properties

        // 读取文件，并将文件键值对存入Properties对象，指向resource文件夹下的文件
        // getClass是获得传入的变量的class类型(可以改成this.getClass)
        // getClassLoader是获取其类加载器（相当于确定了起始路径）
        // getResourceAsStream查找传入的文件，并获取其内容返回成流
        // .properties文件中变量的值不用加""，而.conf需要加""，其他没区别
        val is = DBUtil.getClass.getClassLoader.getResourceAsStream("application.properties") //classpath
```

Figure 30

Component 2: object DB_define – for database structure and initialization

The format of the tables is set, the key value type and the relationship between the tables are defined. In addition, the initialization of three tables is also performed in this object.

```scala
package pojo

import slick.jdbc.MySQLProfile.api._ // related functions to support the connection with mysql
import scala.concurrent.ExecutionContext.Implicits.global // support andThen

// Define table structure
object DB_define
{
    case class DepartmentInfo(department_id: Int, department_name: String, location_id: Int)
    class Department_table(tag: Tag) extends Table[DepartmentInfo](tag, "department")
    {

        def department_id = column[Int]("department_id", O.PrimaryKey)
        def department_name = column[String]("department_name")
        def location_id = column[Int]("location_id")

        def * = (department_id, department_name, location_id).mapTo[DepartmentInfo]

    }
    lazy val department = TableQuery[Department_table]
```

Figure 31

Component 3: trait EmpDao – framework for pre-define methods

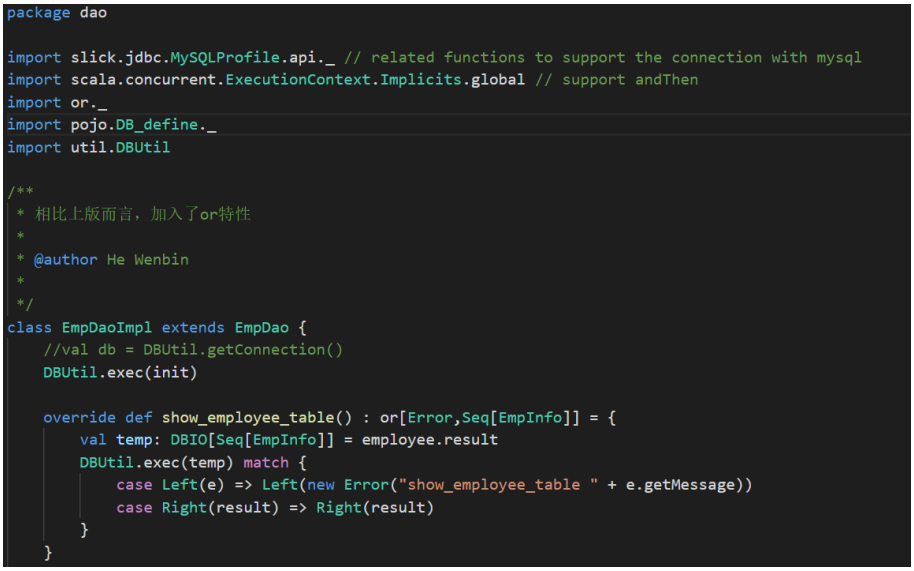
Create the trait EmpDao, pre-define the methods to be implemented, and provide the possibility of inheritance and rewriting.

```scala
package dao

import slick.jdbc.MySQLProfile.api._ // related functions to support the connection with mysql
import scala.concurrent.ExecutionContext.Implicits.global // support andThen
import pojo.DB_define._
import util.DBUtil
import or._

/**
 * 专门用来操作emp的接口
 * @author He Wenbin
 *
 */
trait EmpDao {

    /**
     * 查询所有员工信息(select all)
     * @return Seq[EmpInfo]
     */
    def show_employee_table() : or[Error,Seq[EmpInfo]]//通过Seq来存储EmpInfo类型的数据
```

Figure 32

Component 4: class EmpDaoImpl – provide actual implementation of methods

Provide the actual implementation of the methods defined by trait EmpDao.

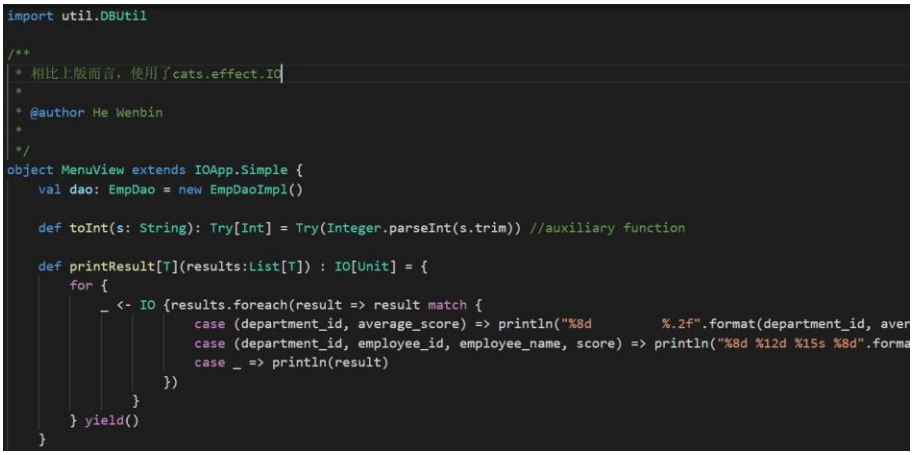


```scala
package dao

import slick.jdbc.MySQLProfile.api._ // related functions to support the connection with mysql
import scala.concurrent.ExecutionContext.Implicits.global // support andThen
import or._
import pojo.DB_define._
import util.DBUtil

/**
 * 相比上版而言，加入了or特性
 *
 * @author He Wenbin
 *
 */
class EmpDaoImpl extends EmpDao {
    //val db = DBUtil.getConnection()
    DBUtil.exec(init)

    override def show_employee_table() : or[Error,Seq[EmpInfo]] = {
        val temp: DBIO[Seq[EmpInfo]] = employee.result
        DBUtil.exec(temp) match {
            case Left(e) => Left(new Error("show_employee_table " + e.getMessage))
            case Right(result) => Right(result)
        }
    }
```

Figure 33

Component 5: object MenuView – main class

Receive and output text through the console, providing an interface for interacting with users.



```scala
import util.DBUtil

/**
 * 相比上版而言，使用了cats.effect.IO
 *
 * @author He Wenbin
 *
 */
object MenuView extends IOApp.Simple {
    val dao: EmpDao = new EmpDaoImpl()

    def toInt(s: String): Try[Int] = Try(Integer.parseInt(s.trim)) //auxiliary function

    def printResult[T](results:List[T]) : IO[Unit] = {
        for {
            _ <- IO {results.foreach(result => result match {
                        case (department_id, average_score) => println("%8d         %.2f".format(department_id, aver
                        case (department_id, employee_id, employee_name, score) => println("%8d %12d %15s %8d".forma
                        case _ => println(result)
                    })
                }
        } yield()
    }
```

Figure 34

4.2 Functional programming

In the second version, we mainly use two ideas from functional programming.

Feature 1: Either[Left,Right]

In the second version, we designed the or type, and its essence is the same as Either. Either is a container type that takes two type parameters: Either[A, B] contains either an instance of type A or an instance of type B.

By wrapping the results of errors or successful operations into or, effective exception handling can be performed, exception information can be obtained, and the danger of program stopping midway can be reduced.

In order to respect conventions, when Either is used to represent an error flag or an object value, the Left value is used to represent an error flag, such as an information string or an exception thrown by a lower-level library; and a Right object is used when returning normally.

```
// 执行传入的sql命令
def exec[T](program: DBIO[T]): or[Error, T] = {
    try {
        Right(Await.result(db.run(program), 2.seconds))
    } catch {
        case e : Throwable => Left(new Error("query failed " + e.getMessage)) // Don't print anything
    }
}
```

Figure 35

Feature 2: cats.effect.IO

cats.effect.IO is a monad that isolates side effects. It provides the side effect isolation function, and simply saves the code that executes the side effect in a parameterless function, so as to achieve the effect of delayed execution.

```
def find_best_perform_employee() : IO[Unit] = {
    for {
        _ <- dao.find_best_perform_employee() match {
            case Left(e) => IO.println(e.getMessage)
            case Right(result) => if (result.length > 0) printResult(result.toList) else IO.println("Empty database!")
        }
    } yield ()
}
```

Figure 36

5.Future works

There are still imperfections in the program. The first problem is that if the program reads in input that cannot be converted to a number, the program cannot handle exceptions efficiently. Second, if the user inputs all commands at once, the program should be able to differentiate intelligently, skip the unnecessary input and match the

desired one. This improvement can be achieved by using *java.util.Scanner*, such as *.nextInt()* and *.nextDouble()*.

Another problem is that the program is not flexible enough. As can be seen from the previous analysis, the program provides only 13 main features. If applicable, when the user wants to query additional information or other operations separately, the program will not be satisfied the needs. One of the possible solutions is using java.sql.PreparedStatement. It may be achieved by defining sql statements that do not fix query variables, so that possible queries are made during runtime.