

**<<HR MANAGER>>**

**Software Design Document**

**Instructor: Nguyễn Trung Kiên**

**Author: Nguyễn Ngọc Nam – HE131010**

**Vũ Hải Đăng - HE150264**

**Nguyễn Huy Tùng - HE150305**

**Lê Nguyên Hùng - HE141720**

**Ngô Quốc Bảo - HE150436**

**Lê Khả Chính - HE151217**

**Class: SE1514-NET**

– Hanoi, August 2019 –

**Table of Contents**

[I. Overview](#_gjdgxs) **3**

[1. Code Packages/Namespaces](#_30j0zll) 3

[2. Coding Conventions](#_1fob9te) 3

[II. Code Designs](#_3znysh7) **5**

[1. <Home page>](#_sw5tcz5xpeyq) 5

[2. <Login>](#_ici9mz76pa32) 5

[3. <Register>](#_ehzrsivcnd5i) 8

[a. Class Diagram](#_k2a55c2uzl0) 8

[b. Class Specification](#_tttzc12aczbq) 8

[c. Sequence Diagram(s)](#_et4m716ezior) 11

[d. Database queries](#_b2zsbj2k95sm) 11

[4. <Setting List>](#_pnc6v31wsz3g) 12

[Class Diagram](#_opo2f9776lui) 12

[Class Specifications](#_gvw5rjqsfsh3) 12

[EmployeeDAO Class](#_s47zqb7y7muv) 12

[Sequence Diagram(s)](#_19nowf2il2mz) 13

[Database queries](#_f3ox4xi7pysv) 13

[5. <Setting Detail>](#_6vzarx3nsn9z) 14

[Class Diagram](#_bxvxsju3n90j) 14

[Class Specifications](#_enagblyt2lhz) 14

[EmployeeDAO Class](#_ncqg9oj3u3j6) 14

[Sequence Diagram(s)](#_sxdrb4ydqjkb) 15

[Database queries](#_asys760rr4a) 15

[6. <User Profile>](#_1jozpzlnh8uu) 15

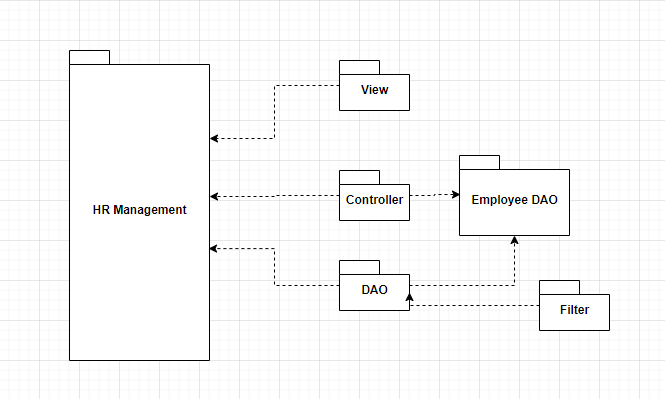
[III. Database Design](#_17dp8vu) **17**

[1. Database Schema](#_7rd1wrohltnt) 17

[2. Table Description](#_tgexln6yysqu) 18

# I. Overview

## 1. Code Packages/Namespaces



***Package descriptions & package class naming conventions***

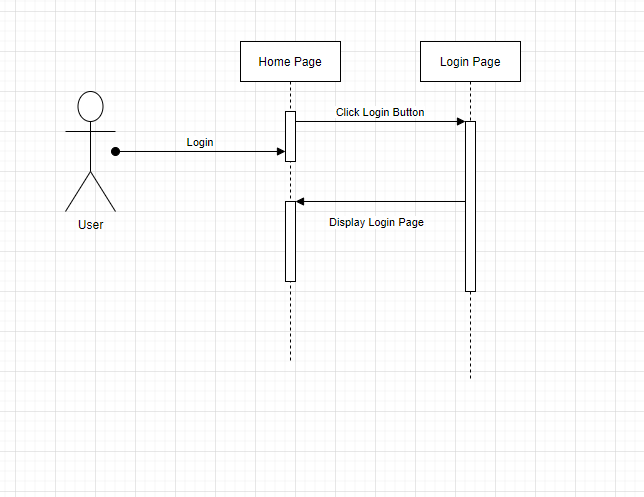
| **No** | **Package** | **Description** |
| --- | --- | --- |
| *01* | *View* | *Package view is package for jsp file display front-end from website* |
| *02* | *Controller* | *Package Controller is servlet file use to control way user interact with* |
| *03* | *DAO* | *Package DAO is package use to get connection from database* |
| *04* | *Filter* | *Package Filter is package for filtering search by name or type* |
| *05* | *Employee DAO* | *Package Employee DAO is package control data insert delete get value from database* |

## 2. Coding Conventions

* File Names
  + File Suffixes: .java
  + File Names: capitalize the first letter of each word
* File Organization
  + Java source files have the following ordering:
    - Comments
    - Package and Import Statements
    - Class and Interface Declarations
* Indentation
  + Line Length
    - Lines shorter than 70 characters
  + Wrapping Lines
    - When break an expression:
      * Break after a comma.
      * Break before an operator.
      * Prefer higher-level breaks to lower-level breaks.
      * Align the new line with the beginning of the expression at the same level on the previous line.
      * If the above rules lead to confusing code or to code that’s squished up against the right margin, just indent 8 spaces instead.
* Declarations
  + Number Per Line
    - One declaration per line
    - variables and functions be declared or different types not on the same line.
  + Placement
    - Put declarations only at the beginning of blocks
  + Initialization
    - initialize local variables where they’re declared
* Statements
  + Simple Statements: Each line contain at most one statement
  + Compound Statements
    - * The enclosed statements indented one more level than the compound statement
      * The opening brace at the end of the line that begins the compound statement; the closing brace begin a line and be indented to the beginning of the compound statement.
      * Braces are used around all statements
  + A return statement with a value not use parentheses unless they make the return value more obvious in some way
* Naming Conventions
  + Class and Interfaces names are nouns, in mixed case with the first letter of each internal word capitalized, keep class names simple and descriptive.
  + Methods are verbs, in mixed case with the first letter lowercase, with the first letter of each internal word capitalized.
  + Except for variables, all instance, class, and class constants are in mixed case with a lowercase first letter. Internal words start with capital letters, short yet meaningful.
  + The names of variables declared class constants and of ANSI constants are all uppercase with words separated by underscores (“\_”)

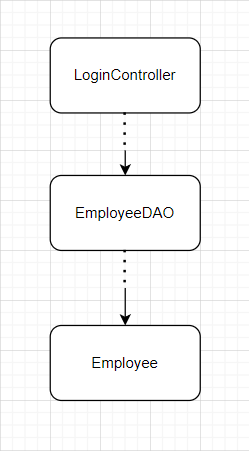
# II. Code Designs

## 1. <Home page>



## 2. <Login>

a. Class Diagram:

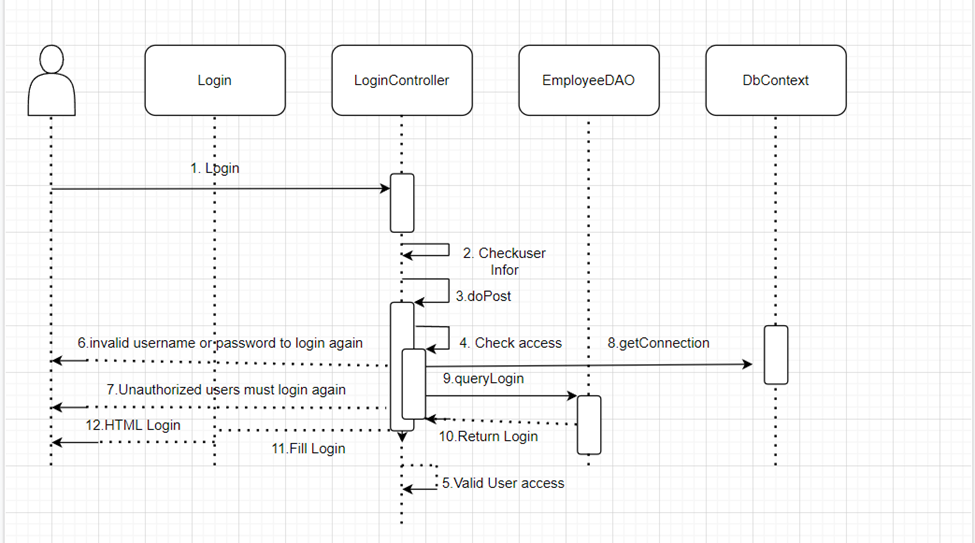


b. Class Specifications:

EmployeeDAO

| **No** | **Method** | **Description** |
| --- | --- | --- |
| *01* | *Login* | *login method queries sql statement to database get username or email and password to login* |
| *02* | *getStatus* | *the status method queries the sql statement to get the user's status* |

c. Class Sequence Diagram(s):



d. Database queries:

SELECT \* FROM hr\_system.employee where (username =? or email = ?) and password=?

SELECT \* FROM hr\_system.employee where status =?

## 

## 

## 

## 

## 

## 

## 

## 

## 

## 

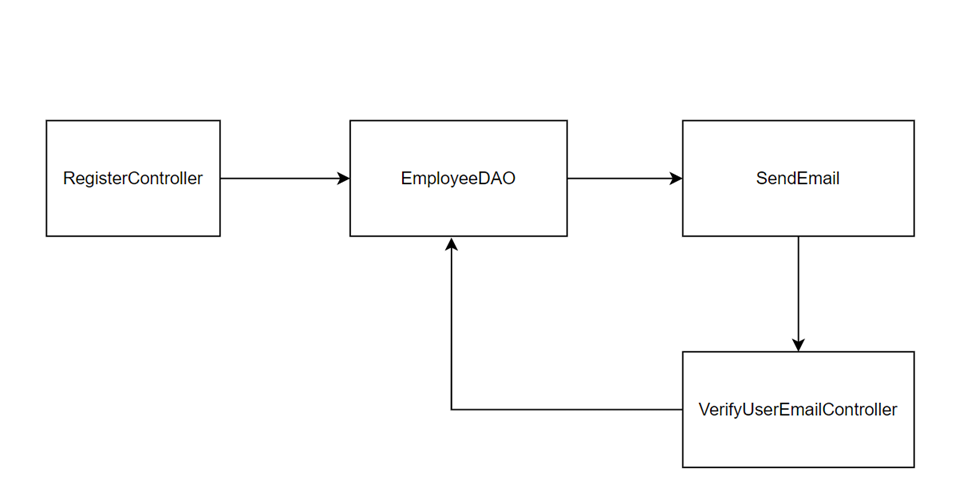
## 

## 

## 

## 3. <Register>

### a. Class Diagram

****

### b. Class Specification

**RegisterCOntroller**

| **No** | **Method** | **Description** |
| --- | --- | --- |
| *01* | *doGet* | Direct user to RegsiterView Page |
| *02* | *doPost* | - Get user information: full name, username, password, email.  - Use Employee DAO class to check username and email not exist in the database.  - Use SendEmail class to send verify code to user email |

**EmployeeDAO**

| **No** | **Method** | **Description** |
| --- | --- | --- |
| *01* | *checkUsernameExist* | Check user name exist in the database |
| *02* | *checkEmailExist* | Check email exist in the database |
| *03* | *addEmployee* | Add user’s information to the database |

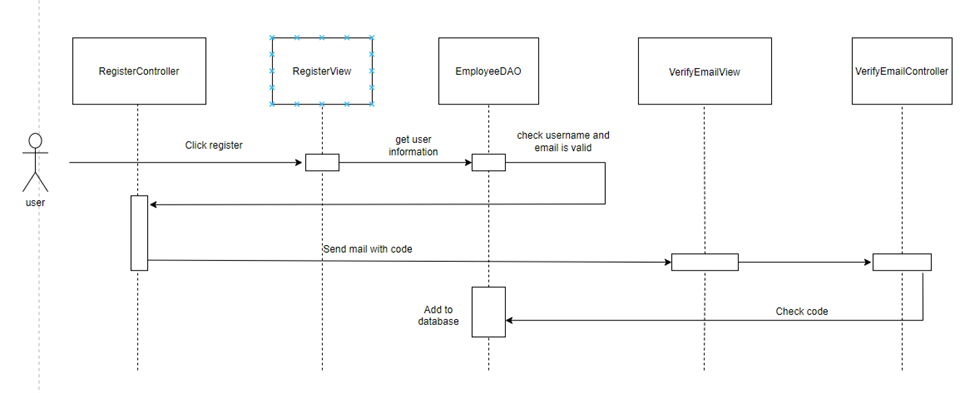
**SendMail**

| **No** | **Method** | **Description** |
| --- | --- | --- |
| *01* | *send* | Send email with message (in this case email send random code) to user email |
| *02* | *getRandom* | Random a number with 6 digit |

**VerifyUserEmailController**

| **No** | **Method** | **Description** |
| --- | --- | --- |
| *01* | *doPost* | Check user’s code. If user’s code is identical to system code, send  success message otherwise send error message. Then direct user to  login page |

### c. Sequence Diagram(s)



### d. Database queries

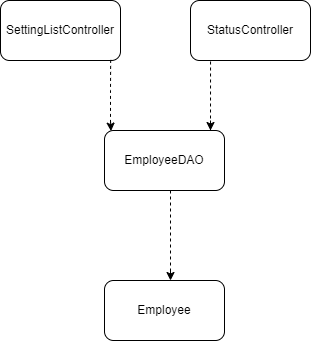
1. Check username exist: SELECT \* FROM hr\_system.employee WHERE username = ?

2. Check email exist: SELECT \* FROM hr\_system.employee WHERE email = ?

3. Add user to database: INSERT INTO `hr\_system`.`employee` (`fullname`,`username`,`password`,`email`) VALUES (?,?,?,?)

## 4. <Setting List>

### Class Diagram

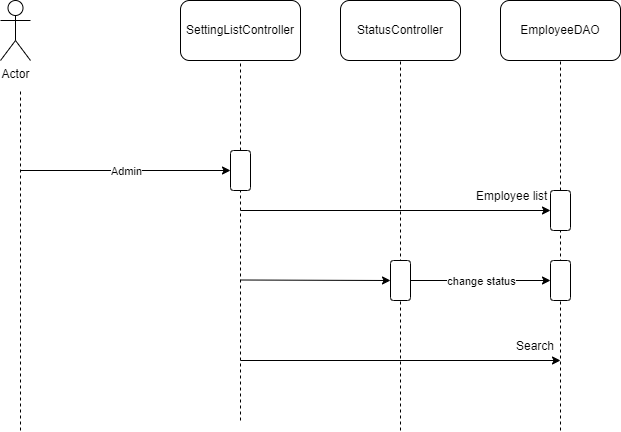


### Class Specifications

#### EmployeeDAO Class

| **No** | **Method** | **Description** |
| --- | --- | --- |
| *01* | *getEmployeeList* | *Input: current page number that user stay*  *Process: queries sql statement to database, list out employee*  *Output: list of employee in that page* |
| *02* | *editStatus* | *Input: employee\_id that user choose and its status*  *Process: queries sql statement to database to update*  *Output: update status* |
| *03* | *getEmployeeBySearch* | *Input: setting\_name and its value*  *Process: queries sql statement to database list out employee that satisfy input*  *Output: list of employee* |
| *04* | *getTotalEmployee* | *Input: (none)*  *Process: queries sql statement to database to count total number of employee*  *Output: number of total employee* |

### Sequence Diagram(s)



### Database queries

select \* FROM hr\_system.employee e, hr\_system.type t where e.type\_id = t.type\_id limit 5 offset ?

UPDATE `hr\_system`.`employee` SET `status` = ? WHERE (`employee\_id` = ?)

SELECT \* FROM hr\_system.employee e, hr\_system.type t where " + setting\_type + " = ? and e.type\_id = t.type\_id

select count(\*) FROM hr\_system.employee

## 

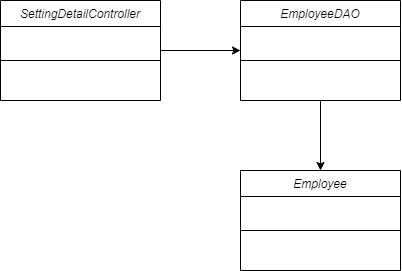
## 

## 

## 

## 5. <Setting Detail>

### Class Diagram

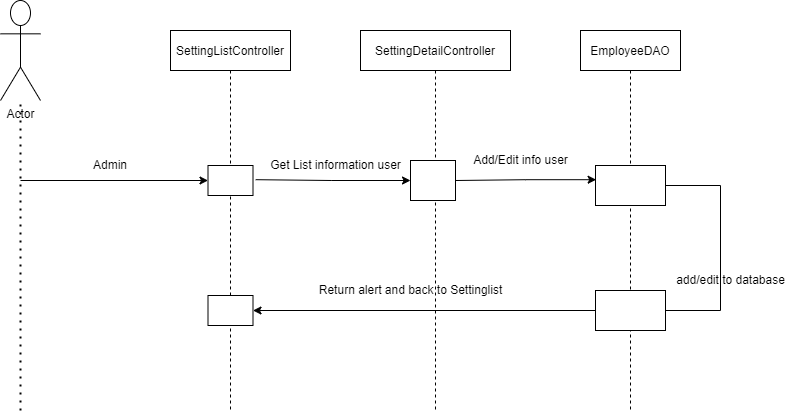


### Class Specifications

#### EmployeeDAO Class

| **No** | **Method** | **Description** |
| --- | --- | --- |
| *01* | *getEmployeeList* | *Input: current page number that user stay*  *Process: queries sql statement to database, list out employee*  *Output: list of employee in that page* |
| *02* | *addEmployee* | *Input: employee\_id that user choose and its status*  *Process: add user's information in database*  *Output: list of employee in that page* |
| *03* | *updateEmployee* | *Input: employee\_id that user choose*  *Process: Edit the user's information in the database through ID*  *Output: list of employee in that page* |

### Sequence Diagram(s)



### Database queries

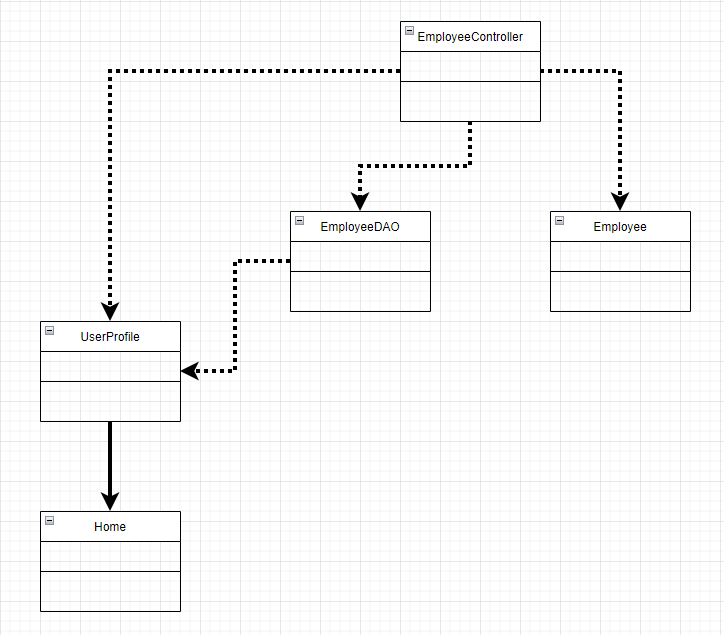
Select \* FROM hr\_system.employee e, hr\_system.type t where e.type\_id = t.type\_id limit 5 offset ?

INSERT INTO hr\_system.employee (employee\_id, fullname, username, password, email, avatar,status,type\_id) VALUES(?,?, ?, ?, ?, ?, ?, ?)

UPDATE hr\_system.employee SET fullname= ? ,username= ?, password= ?, email = ?, avatar= ?, status= ? ,type\_id= ? WHERE employee\_id= ?

## 6. <User Profile>

a. Class Diagram:

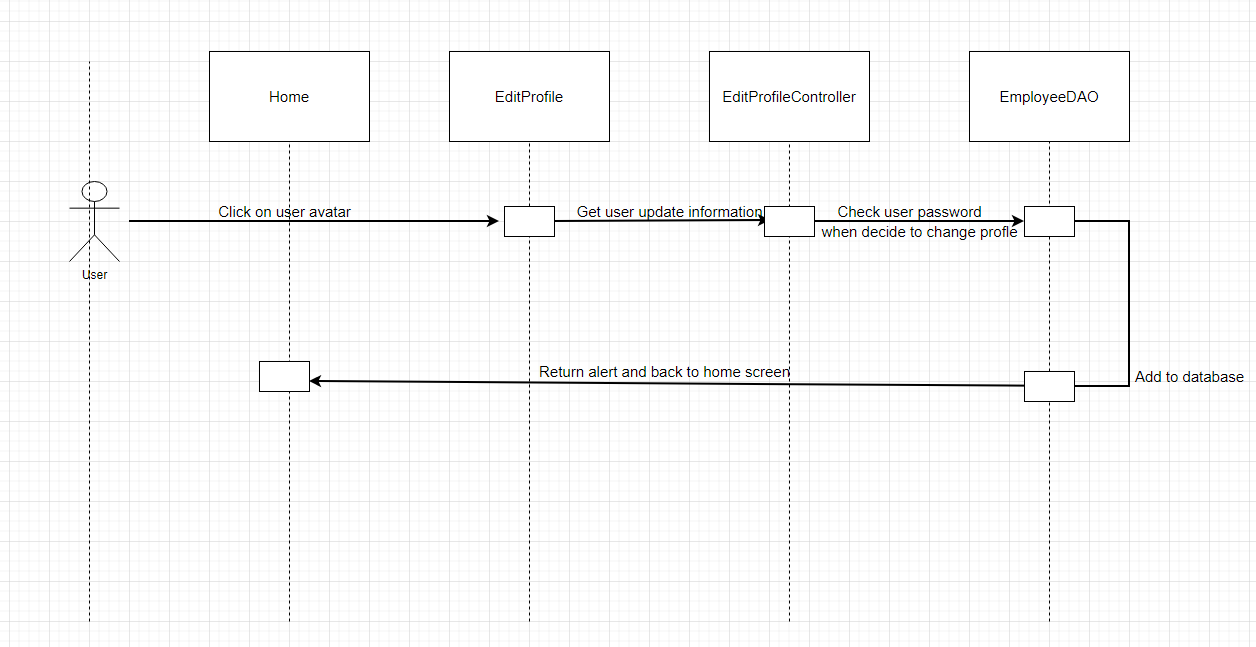


b. Class Specifications:

EmployeeDAO

| **No** | **Method** | **Description** |
| --- | --- | --- |
| *01* | *UpdateProfile* | *update method queries sql statement to database get user information: full name, avatar, username to change their information* |

c. Class Sequence Diagram(s):



d. Database queries:

UPDATE hr\_system.employee SET fullname = ?, avatar = ? WHERE username =?

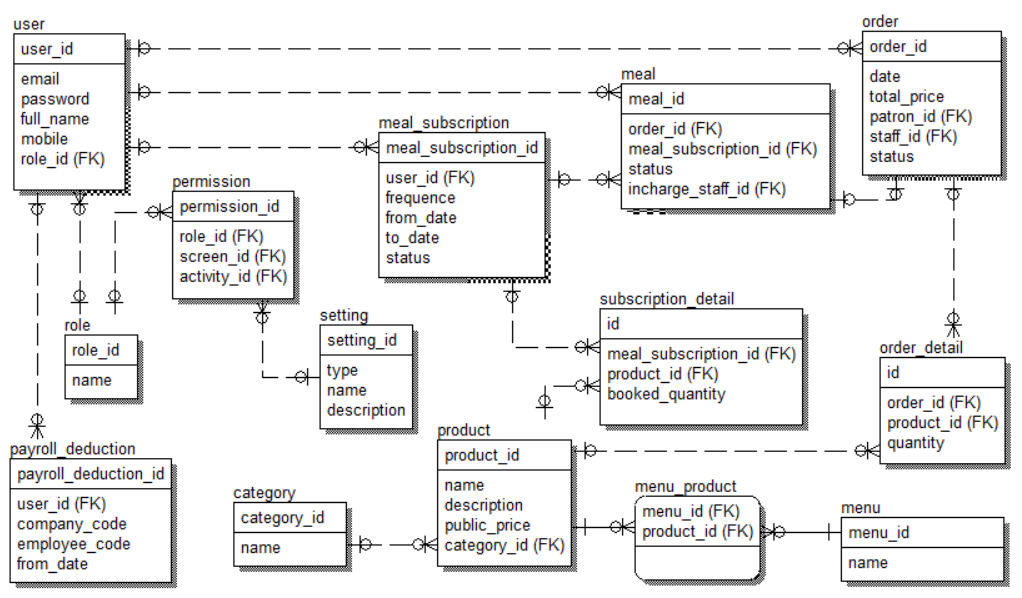
## 

## 

# III. Database Design

## 1. Database Schema

*[Provide the tables relationship like example below – following MySQL database naming convention]*



## 2. Table Description

| **No** | **Table** | **Description** |
| --- | --- | --- |
| *01* | *<Table name>* | *<Description of the table>*  *- Primary keys: <<list of primary key fields>>*  *- Foreign keys: <<list of foreign key fields>>* |
| *02* |  |  |