LEVEL UP LIFE

"Track your everyday achievements in life."

By: Team DataDiggers
Krisha Shah
Mansi Pandya
Yash Oza
Pratham Patel
Vivek Dixit

Declaration by Author

This is to declare that this report has been written by us and no part of the report has been plagiarized from other sources. All the information included from other sources has been duly acknowledged. We affirm that if any part of the report is found to be plagiarized, we shall take full responsibility for it.

Acknowledgement

It is our pleasure to express thanks and heartiest gratitude to all those who have helped us during the period of the development of this project. We would like to thank our Professor Mr. Chaiyaporn Mutsalklisana for all his continuous guidance and support and his insights and suggestions towards the project. In spite of his tight schedule, he has always given us time for the difficulties we faced during the ongoing project and answered all our queries.

And also grateful to him for giving us this opportunity to work under him. We would also like to thank our Teaching Assistant Nilesh Nerkar who guided us how to go about the project whenever we faced issues and his support.

Abstract

All the citizens have a certain set of responsibilities towards the "*Better Citizen*" status quo. The application basically promotes the spirit that - "*The chance to help someone out in between our daily routine life and to do something good everyday is given to a few lucky ones*".

This application would allow users to keep track of their own responsibilities as an individual towards the society. The application would allow users to take up tasks for themselves in various different categories and then work towards those tasks to keep adding Experience points till they reach the status they desired from the start. Through this system, the points the user earns by performing these tasks of social responsibility can be used towards a "*Better Citizen*" program. These Experience points and higher the level the users are on later is beneficial to them in different ways as they get extra privileges from the Government as these points count in their Credit History while giving them loans on a lower interest rate and also help them to avail benefits on their taxes, insurance schemes etc.

Table of Contents

Chapter 1

Literature Survey

1.1 Introduction to Data management and Database design

A Database management system is a collection of a wide range of interrelated data and a set of programs to access those data. The collection of data is referred to as what is called the Database. The Database contains information relevant to an enterprise. The primary goal of the database system is to provide a way to store and retrieve data/information that is both convenient and efficient.

Database systems are designed to manage large number of information. Management of data involves both defining structures for storage of information. It also ensures the safety of the data if the system crashes or some unauthorized user try to access the database. As per the rules of the database system, the data is stored in such a way that it acquires less space as the redundant data i.e. the duplicate data is removed from the database. Besides storing data in an optimized way, it is also important that we retrieve data quickly when required. Database ensures that data is retrieved as quickly as possible.

- Types of Database management system:

1) Hierarchical:
   In Hierarchical database the data is organized in a tree like structure and the data is stored from top to down in a hierarchical manner. Data is represented using a parent-child relationship. In this format, parents may have many children but children have only one parent.

2) Network:
   Network database model allows each child to have multiple parents.

It supports many-to-many relationship. In this model, entities are organized in a graph structure, which can be accessed through several paths.

3) Relational:
   The most widely used database model is the Relational database model as it is one of the easiest. This model is based on normalizing data in the rows and columns of the tables. Relational model stored in fixed structures and is manipulated using SQL.

4) Object Oriented:
   In this model data is stored in the form of objects. The structure which is called classes which display data within it. It defines a database as a collection of objects that stores both data member's values and operations.

- Applications:
  Used in various sectors such as Airlines, Universities, Banking, and Telecommunication etc.

- Design:
  The database should be cost effective, esthetic that satisfies the specifications. Professions such as engineering and architecture are concerned with design. Design is an iterative process rarely in the real world is a problem specified such that there is a unique optimal solution. Thus the designer works iteratively.

- User Interface:
  A database management system interface is a user interface which allows for the ability to input queries to a database without using the query language itself. A DBMS interface could be a web client, a local client that runs on a desktop computer, or even a mobile app.

- What is MySQL?
  MySQL is an open source relational database management system. The data in MySQL is stored in a Database objects called tables. A table is a collection of entries and consists of rows and columns. And it stores data categorically.

Chapter 2

Analysis and Requirement Specification

2.1 What is Level Up?

This is a database application that celebrates the spirit of doing random acts of kindness in between our daily lives.
By putting a structured system in place, the application is capable of quantifying such acts and reward the deserving ones using a game like simulation.

- The User will start at Level 1 and work his/her way up through the tasks to avail rewards.
- The rewards will depend on the task performed and the Experience points received to the user depending on the tasks performed.
- The tasks vary from donating merchandise to helping the poor.
- Doing these tasks will ensure the well-being of a society as a whole and also be beneficial to the individual pertaining to the rewards criteria.
- The user will have a range of tasks available to them depending on the level they are at.
- As they complete each task, they are assigned a certain amount of Experience points.
- The task performed by them is validated by a government official that oversees their work and provides them with a token that the user can input in their system to get the points for that particular task.
- The points can be cashed in for benefits that are given to them by the government agencies.
- The higher the points, the wider the range of benefits that are available to them.
- The application will need a proper system in place that will have the coordination of both the Government Agencies and the Users of the application.

2.2 Purpose

The purpose of this application is to motivate the Citizens to help others when in need and showing their responsibility for the society. When in turn leads the

citizens to earn points for completing particular tasks and increase their points and gain benefits from the Government.

It's designed to track achievements and reward for completing them with experience points which later convert into gaining a higher level and upgrading your skills. You can just create an account and immediately start working your way towards a new level. Achievements are sorted in categories based on their content and context, they aren't all available at the beginning. *As you progress in the game (and life!), new and more challenging achievements will be unlocked.*

By completing achievements you also create statistics about your life, so Level Up Life can be used not only for motivation, but for self-tracking as well. Since skill points are divided into categories as well, you can see which aspects of your life are improving faster than the others.

2.3 Scope

The Scope of the project is managing the consistency of the data by the administrator. It provides most of the features that a Database Management System should have. It is developed by using MySQL database in MySQL Workbench.

2.4 Functional Requirements

- Admin: The role of an admin is plan, install, configure, design database, insert, update, troubleshoot as well as backup and data recovery.

- Users: The users can register, update or view their information. They can see their own information and rewards that they can avail in the system.

- Government & Government_Officials: An area will have one Government and many government officials under one government. The respective government officials will be responsible for officiating tasks done by citizens residing in their area.

2.5 Non-Functional Requirements

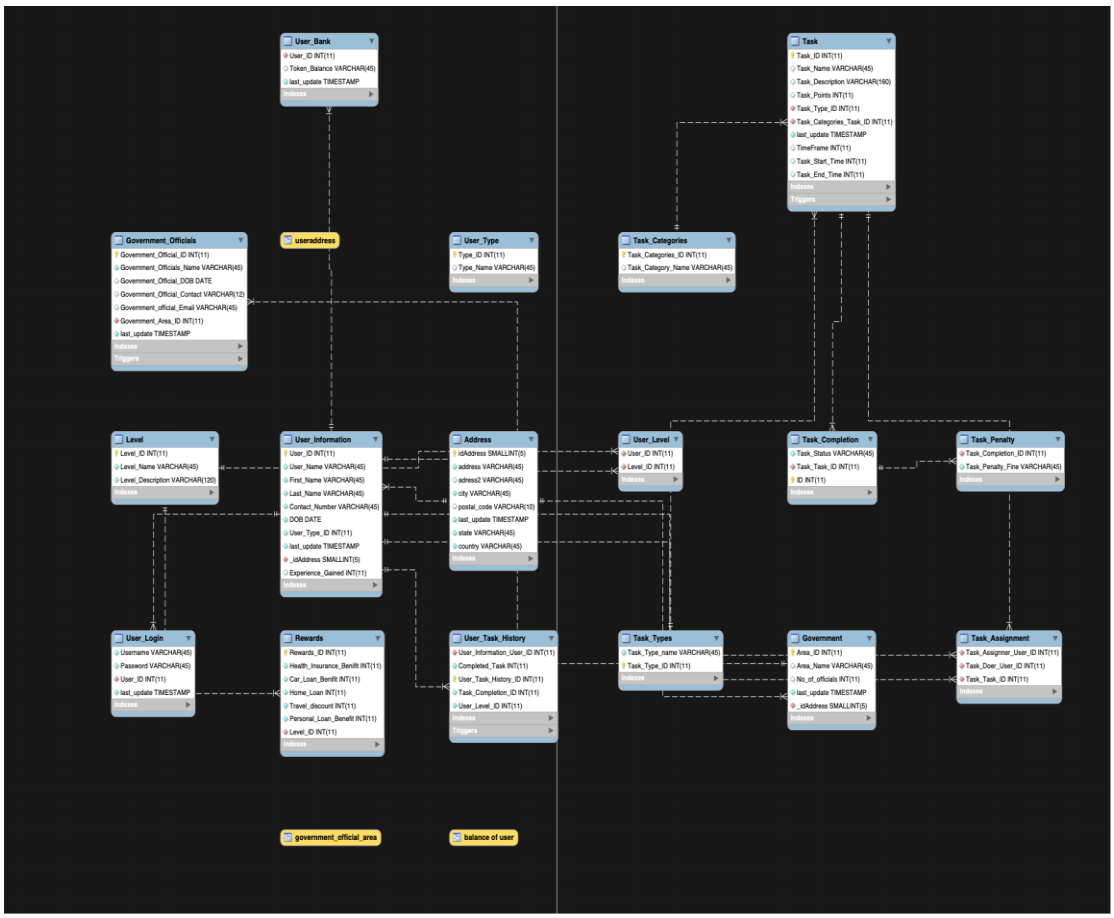
- Software Specifications :
  - MySQL Workbench
  - MySQL Libraries
  - Operating system: macOS, Windows

- Hardware Specifications :
  - Processor
  - Monitor
  - Keyboard
  - RAM

Chapter 3

System Design

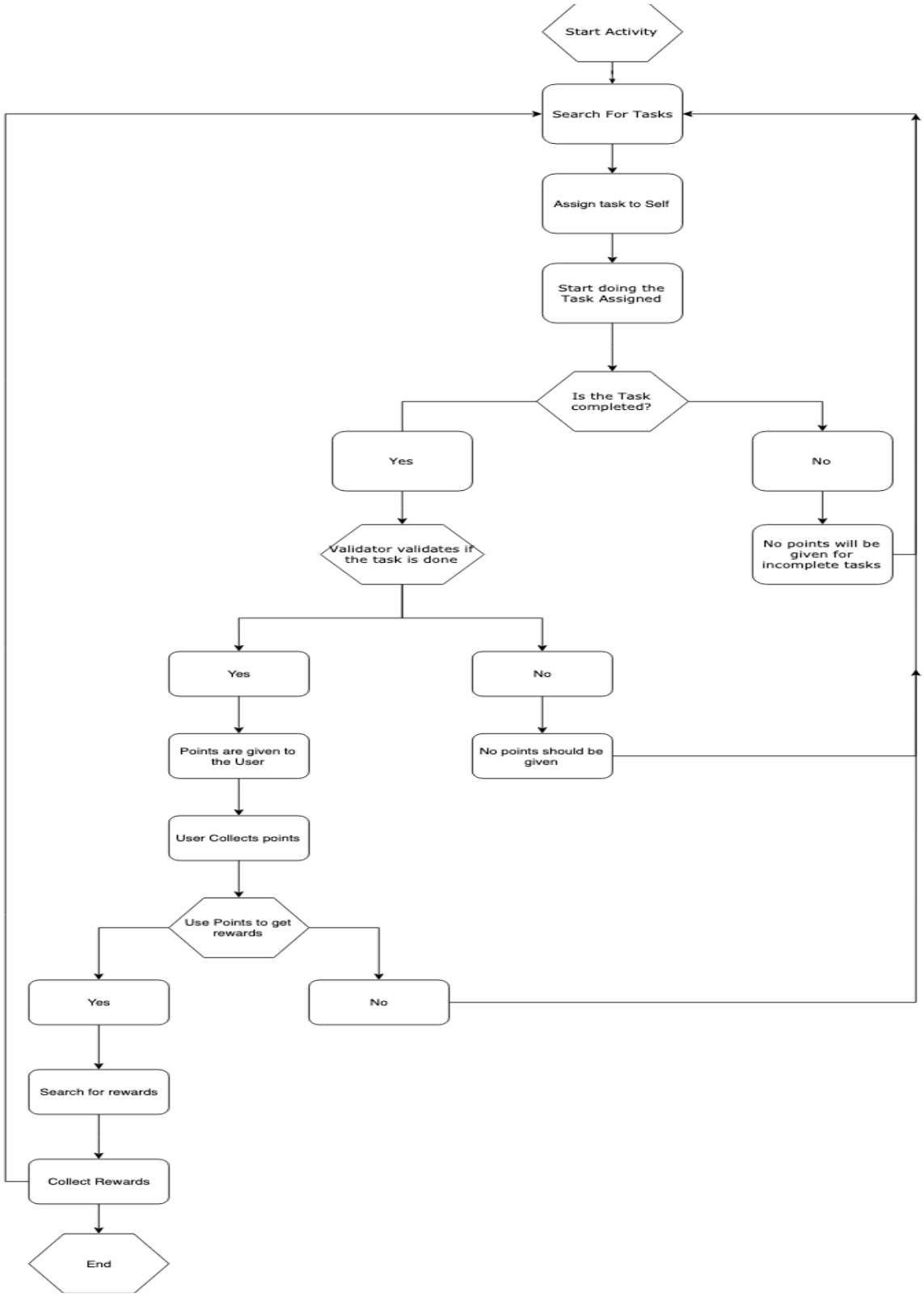
3.1 ER Diagram



3.2 Use Case Diagram

3.3 Activity Diagram

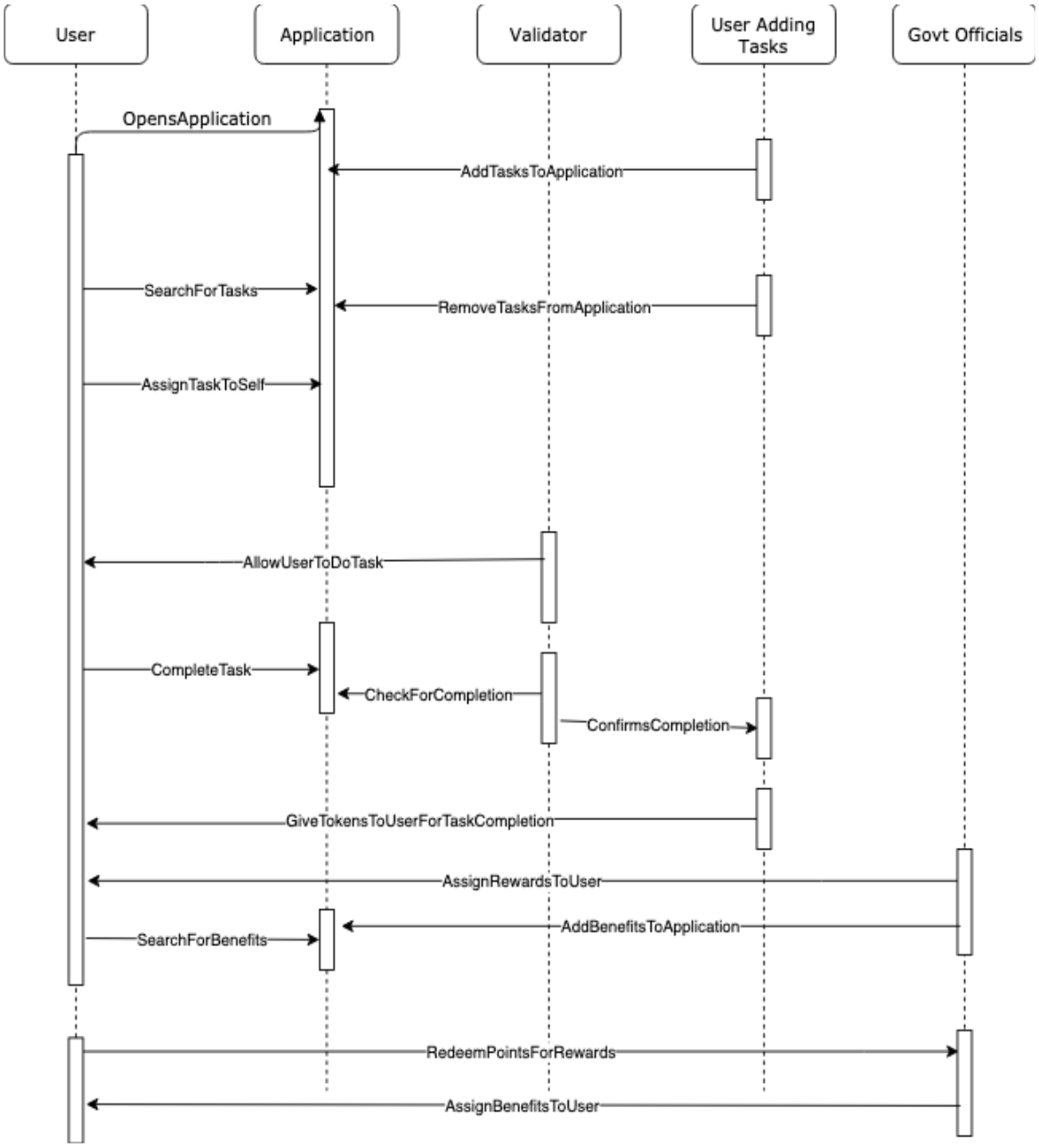
The user logins in and starts the game by searching for the tasks they are expert at and assign it to themselves, starts the task. If the task is completed within a given time frame then the task completion status is changed to complete or else incomplete. Once the user completes the task, gains points and experience count increase by 1. He/She unlocks the next level as their experience points increase and they can move further in the game. The user that earns maximum points gets awards from the Government.


Start Activity
Search For Tasks
Assign task to Self
Start doing the Task Assigned
Is the Task completed?
Yes
No
No points will be given for incomplete tasks
Validator validates if the task is done
Yes
No
Points are given to the User
No points should be given
User Collects points
Use Points to get rewards
Yes
No
Search for rewards
Collect Rewards
End

3.4 Sequence Diagram

There are Five actors in our sequence diagram which are:
1. User
2. Application
3. Validator
4. User Adding Task
5. Government Officials

The flow of the application is as follows:
The User first logs into the application using his/her application credentials and searches for tasks. Each users can add task according to the number of remaining tokens they have. The tasks are shown to users depending on their Area_ID. If the task has to be modified or removed from the application then, the Assigner can delete or modify it using the application. If a user is willing to do a task then, they assign the task to themselves. The application shows a
status of ongoing task once a user is assigned to a task. Once the task is completed the Government of that Area ID will have some government official of the same Area Id to verify the task completion. If the task is completed then the points will be transferred from the assigners account to the doers account. Using this earned points the user can search for rewards and benefits. The application will have a separate page for rewards and benefits, where each user can search for rewards as per their level in the game. Once the user finds a suitable benefit, they can claim it using their collected points.


User
Application
Validator
User Adding Tasks
Govt Officials
OpensApplication
AddTasksToApplication
SearchForTasks
RemoveTasksFromApplication
AssignTaskToSelf
AllowUserToDoTask
CompleteTask
CheckForCompletion
ConfirmsCompletion
GiveTokensToUserForTaskCompletion
AssignRewardsToUser
AddBenefitsToApplication
SearchForBenefits
RedeemPointsForRewards
AssignBenefitsToUser

Chapter 4

Implementation

4.1 Implementation Of Table Creation Page

Tables :

S.No.	Table Name	Primary Key	Foreign Key(s)
1.	Address	idAddress	-
2.	Government	Area_ID	_idAddress
3.	Government Official	Government_Official_ID	Government_Area_ID
4.	Levels	Level_ID	-
5.	Rewards	Rewards_ID	Level_ID
6.	Task	Task_ID	Task_Categories_Task_ID, Task_Type_ID
7.	Task Assignment	-	Task_Assigner_User_ID, Task_Doer_User_ID, Task_Task_ID
8.	Task Category	Task_Categories_ID	-
9.	Task Completion	ID	Task_Task_ID
10.	Task Penalty	-	Task_Completion_ID
11.	Task Types	Task_Type_ID	-
12.	User Bank	-	User_ID
13.	User Information	User_ID	_idAddress
14.	User Level	-	User_ID, Level_ID

15.	User Login	-	User_ID
16.	User Task History	User_Task_History_ID	User_Level_ID, User_Information_User_ID
17.	User Type	Type_ID	-

- **Code to create the Database :**

```
CREATE SCHEMA IF NOT EXISTS `mydb` DEFAULT CHARACTER SET
utf8 ;
USE `mydb` ;
```

1. **Address Table :** The Address Table stores the address of the users. A separate table was made for this purpose as there could be a possibility that one user can have many addresses associated to them and there's also the possibility that several users can stay at the same address.
   The primary key of this table is the idAddress which is the id and the table does not have a foreign key.

```
CREATE TABLE IF NOT EXISTS `mydb`.`Address` (
  `idAddress` SMALLINT(5) NOT NULL AUTO_INCREMENT,
  `address` VARCHAR(45) NOT NULL,
  `adress2` VARCHAR(45) NULL DEFAULT NULL,
  `city` VARCHAR(45) NOT NULL,
  `postal_code` VARCHAR(10) NULL DEFAULT NULL,
  `last_update` TIMESTAMP NOT NULL DEFAULT
CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
  `state` VARCHAR(45) NOT NULL,
  `country` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`idAddress`))
```

2. **Government Table :** The Government table will have the area ID and the name of the Area where the specific amount of the government officials work. It will tell us the number of Government Officials operating in a particular area that can validate and assign the tasks assigned to the users in that area.

The primary key of this table is the Area_ID and the foreign key is the idAddress.

```
CREATE TABLE IF NOT EXISTS `mydb`.`Government` (
  `Area_ID` INT(11) NOT NULL,
  `Area_Name` VARCHAR(45) NULL DEFAULT NULL,
  `No_of_officials` INT(11) NULL DEFAULT NULL,
  `last_update` TIMESTAMP NOT NULL DEFAULT
CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
  `_idAddress` SMALLINT(5) NOT NULL,
  PRIMARY KEY (`Area_ID`),
  INDEX `fk_Government_1_idx` (`_idAddress` ASC)
VISIBLE,
  CONSTRAINT `fk_Government_1`
    FOREIGN KEY (`_idAddress`)
    REFERENCES `mydb`.`Address` (`idAddress`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
```

3. **Government_officials Table :** The Government_Officials table will have the information of the Government Officials that are employed to work in the system. It will also have the Area_ID of the area that the Government Official is employed to work in.
   The primary key of this table is the Government_Official_ID and the foreign key is the Government_Area_ID.

```
CREATE TABLE IF NOT EXISTS
`mydb`.`Government_Officials` (
  `Government_Official_ID` INT(11) NOT NULL,
  `Government_Officials_Name` VARCHAR(45) NOT NULL,
  `Government_Official_DOB` DATE NULL DEFAULT NULL,
  `Government_Official_Contact` VARCHAR(12) NULL
DEFAULT NULL,
  `Government_official_Email` VARCHAR(45) NULL
DEFAULT NULL,
  `Government_Area_ID` INT(11) NOT NULL,
  `last_update` TIMESTAMP NOT NULL DEFAULT
CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
  PRIMARY KEY (`Government_Official_ID`),
```

```
  INDEX `fk_Government_Officials_Government1_idx`
(`Government_Area_ID` ASC) VISIBLE,
  CONSTRAINT `fk_Government_Officials_Government1`
    FOREIGN KEY (`Government_Area_ID`)
    REFERENCES `mydb`.`Government` (`Area_ID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
```

4. **Level Table :** The Level table will have the ID and the name of the level and the description of the level.
   The primary key of this table is the Level_ID and it doesn't contain any foreign key.

```
CREATE TABLE IF NOT EXISTS `mydb`.`Level` (
  `Level_ID` INT(11) NOT NULL,
  `Level_Name` VARCHAR(45) NOT NULL,
  `Level_Description` VARCHAR(120) NOT NULL,
  PRIMARY KEY (`Level_ID`))
```

5. **Rewards Table :** The Rewards table will have the rewards id and the level id. It will have the information of the rewards that will be available to any of the user at that particular level. These rewards include compensation in the user's Health Insurance, Home Loan, Travel Benefits etc.
   The primary key of this table is the Rewards_ID and the foreign key is the Level_ID.

```
CREATE TABLE IF NOT EXISTS `mydb`.`Rewards` (
  `Rewards_ID` INT(11) NOT NULL,
  `Health_Insurance_Benifit` INT(11) NOT NULL,
  `Car_Loan_Benifit` INT(11) NOT NULL,
  `Home_Loan` INT(11) NOT NULL,
  `Travel_discount` INT(11) NOT NULL,
  `Personal_Loan_Benefit` INT(11) NOT NULL,
  `Level_ID` INT(11) NOT NULL,
  PRIMARY KEY (`Rewards_ID`),
  INDEX `Level_ID_idx` (`Level_ID` ASC) VISIBLE,
  INDEX `Level_idx` (`Level_ID` ASC) VISIBLE,
  CONSTRAINT `Level`
    FOREIGN KEY (`Level_ID`)
```

```
    REFERENCES `mydb`.`Level` (`Level_ID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
```

6. **Task_Categories Table** : There are several categories that the tasks are divided in. A user can excel at a particular category of a task and gain more experience points in it to show that they are exceptionally good in it or have an inclination towards that particular category. The categories include Household chores, Construction Work, Charitable work and many more which are specified in the Task_Categories table.
   The primary key of this table is the Task_Categories_ID and this table doesn't consist of any foreign key.

```
CREATE TABLE IF NOT EXISTS `mydb`.`Task_Categories` (
  `Task_Categories_ID` INT(11) NOT NULL,
  `Task_Category_Name` VARCHAR(45) NULL DEFAULT NULL,
  PRIMARY KEY (`Task_Categories_ID`))
```

7. **Task_Type Table :** The tasks that can be assigned to the users are divided into two types which are Government task and Personal Task. The government tasks are the tasks that the user can complete to gain experience points and increase their level thereafter whereas the Personal tasks are the tasks that the user can complete to gain points/tokens from the other user and increase their token balance. The Task_Type table specifies the id of the type of task and the type of task.
   The Primary key of this table is the Task_Type_ID and this table doesn't consist of any foreign key.

```
CREATE TABLE IF NOT EXISTS `mydb`.`Task_Types` (
  `Task_Type_name` VARCHAR(45) NOT NULL,
  `Task_Type_ID` INT(11) NOT NULL,
  PRIMARY KEY (`Task_Type_ID`))
```

8. **Task Table :** The Task Table specifies the details of the task. It includes information of the Task such as the Task ID, Task Name, its description, the category it comes under and the time frame that it should be completed in.
   The primary key of this table is the Task_ID and this table consists of two foreign keys which are Task_Categories_Task_ID and Task_Type_ID.

```
CREATE TABLE IF NOT EXISTS `mydb`.`Task` (
  `Task_ID` INT(11) NOT NULL AUTO_INCREMENT,
  `Task_Name` VARCHAR(45) NULL DEFAULT NULL,
  `Task_Description` VARCHAR(160) NULL DEFAULT NULL,
  `Task_Points` INT(11) NULL DEFAULT NULL,
  `Task_Type_ID` INT(11) NOT NULL,
  `Task_Categories_Task_ID` INT(11) NOT NULL,
  `last_update` TIMESTAMP NOT NULL DEFAULT
CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
  `TimeFrame` INT(11) NULL DEFAULT NULL,
  `Task_Start_Time` INT(11) NULL DEFAULT NULL,
  `Task_End_Time` INT(11) NULL DEFAULT NULL,
  PRIMARY KEY (`Task_ID`),
  INDEX `fk_Task_Task_Types1_idx` (`Task_Type_ID`
ASC) VISIBLE,
  INDEX `fk_Task_Task_Categories1_idx`
(`Task_Categories_Task_ID` ASC) VISIBLE,
  CONSTRAINT `fk_Task_Task_Categories1`
    FOREIGN KEY (`Task_Categories_Task_ID`)
    REFERENCES `mydb`.`Task_Categories`
(`Task_Categories_ID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Task_Task_Types1`
    FOREIGN KEY (`Task_Type_ID`)
    REFERENCES `mydb`.`Task_Types` (`Task_Type_ID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
```

9. **User_Information Table :** As the name suggests, this table contains all the information of the user that the user can input and update. It also contains the last updated time that the user inserted/updated their own information.
   The primary key of this table is the User_ID and the foreign key is the _idAddress.

```
CREATE TABLE IF NOT EXISTS `mydb`.`User_Information`
(
```

```
  `User_ID` INT(11) NOT NULL AUTO_INCREMENT,
  `User_Name` VARCHAR(45) NOT NULL,
  `First_Name` VARCHAR(45) NOT NULL,
  `Last_Name` VARCHAR(45) NOT NULL,
  `Contact_Number` VARCHAR(45) NOT NULL,
  `DOB` DATE NOT NULL,
  `User_Type_ID` INT(11) NOT NULL DEFAULT '1',
  `last_update` TIMESTAMP NOT NULL DEFAULT
CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
  `_idAddress` SMALLINT(5) NOT NULL,
  `Experience_Gained` INT(11) NULL DEFAULT '0',
  PRIMARY KEY (`User_ID`),
  INDEX `fk_User_User_Type1_idx` (`User_Type_ID` ASC)
VISIBLE,
  INDEX `fk_User_Information_1_idx` (`_idAddress`
ASC) VISIBLE,
  CONSTRAINT `fk_User_Information_1`
    FOREIGN KEY (`_idAddress`)
    REFERENCES `mydb`.`Address` (`idAddress`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
```

10. **Task_Assignment Table :** This table consists the information of the personal tasks that the users assign to each other. It consists the id of the user that assigns the task (assigner id) and the id of the user that does the task (doer id) and the id of the task itself.
    There is no primary key in this table but there are 3 different foreign keys that are Task_Assigner_User_ID, Task_Doer_User_ID and Task_Task_ID

```
CREATE TABLE IF NOT EXISTS `mydb`.`Task_Assignment` (
  `Task_Assignner_User_ID` INT(11) NOT NULL,
  `Task_Doer_User_ID` INT(11) NOT NULL,
  `Task_Task_ID` INT(11) NOT NULL,
  INDEX `fk_Task_Assignment_User_Information1_idx`
(`Task_Assignner_User_ID` ASC) VISIBLE,
  INDEX `fk_Task_Assignment_User_Information2_idx`
(`Task_Doer_User_ID` ASC) VISIBLE,
  INDEX `fk_Task_Assignment_Task1_idx`
```

```
(`Task_Task_ID` ASC) VISIBLE,
  CONSTRAINT `fk_Task_Assignment_Task1`
    FOREIGN KEY (`Task_Task_ID`)
    REFERENCES `mydb`.`Task` (`Task_ID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Task_Assignment_User_Information1`
    FOREIGN KEY (`Task_Assignner_User_ID`)
    REFERENCES `mydb`.`User_Information` (`User_ID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Task_Assignment_User_Information2`
    FOREIGN KEY (`Task_Doer_User_ID`)
    REFERENCES `mydb`.`User_Information` (`User_ID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
```

11. **Task_Completion Table:** This table consists of information that tells us if the task assigned to the user has been completed or is still incomplete. The primary key of this table is ID and the foreign key is Task_Task_ID.

```
CREATE TABLE IF NOT EXISTS `mydb`.`Task_Completion` (
  `Task_Status` VARCHAR(45) NOT NULL,
  `Task_Task_ID` INT(11) NOT NULL,
  `ID` INT(11) NOT NULL AUTO_INCREMENT,
  PRIMARY KEY (`ID`),
  INDEX `fk_Task_Completion_Task1_idx`
(`Task_Task_ID` ASC) VISIBLE,
  CONSTRAINT `fk_Task_Completion_Task1`
    FOREIGN KEY (`Task_Task_ID`)
    REFERENCES `mydb`.`Task` (`Task_ID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
```

12. **Task_Penalty Table :** This table contains information on what the penalty will be fined to the user if a particular task has been left incomplete.
    The primary key is non existent in this table and the foreign key is Task_Completion_ID.

```
CREATE TABLE IF NOT EXISTS `mydb`.`Task_Penalty` (
  `Task_Completion_ID` INT(11) NOT NULL,
  `Task_Penalty_Fine` VARCHAR(45) NOT NULL,
  INDEX `fk_Task_Penalty_Task_Completion1_idx`
(`Task_Completion_ID` ASC) VISIBLE,
  CONSTRAINT `fk_Task_Penalty_Task_Completion1`
    FOREIGN KEY (`Task_Completion_ID`)
    REFERENCES `mydb`.`Task_Completion` (`ID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
```

13. **User_Bank Table :** This table consists of the information about the users and the token balance that each user has. It also notifies us of the last time the user completed a task and made a change in their token balance or the last time the user withdrew from their token balance to avail any benefits. There is no primary key in this table and the foreign key in this table is User_ID.

```
CREATE TABLE IF NOT EXISTS `mydb`.`User_Bank` (
  `User_ID` INT(11) NOT NULL,
  `Token_Balance` VARCHAR(45) NULL DEFAULT NULL,
  `last_update` TIMESTAMP NOT NULL DEFAULT
CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
  INDEX `fk_User_Bank_User_Information1_idx`
(`User_ID` ASC) VISIBLE,
  CONSTRAINT `fk_User_Bank_User_Information1`
    FOREIGN KEY (`User_ID`)
    REFERENCES `mydb`.`User_Information` (`User_ID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
```

14. **User_Level Table :** This table provides information on which level a user is at. It contains the id's of the user and the level id corresponding to the level they are at.
    This table doesn't have any primary key but it does have 2 foreign keys which are User_ID and Level_ID.

```
CREATE TABLE IF NOT EXISTS `mydb`.`User_Level` (
```

```
  `User_ID` INT(11) NOT NULL,
  `Level_ID` INT(11) NOT NULL,
  INDEX `fk_User_Information_has_Level_Level1_idx`
(`Level_ID` ASC) VISIBLE,
  INDEX
`fk_User_Information_has_Level_User_Information1_idx`
(`User_ID` ASC) VISIBLE,
  CONSTRAINT `fk_User_Information_has_Level_Level1`
    FOREIGN KEY (`Level_ID`)
    REFERENCES `mydb`.`Level` (`Level_ID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT
`fk_User_Information_has_Level_User_Information1`
    FOREIGN KEY (`User_ID`)
    REFERENCES `mydb`.`User_Information` (`User_ID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
```

15. **User_Login Table :** This table contains the username and the password of every user along with their user id and the last time that they logged into the system. The username and password will be used by the user to log into the system to insert their details or update them.
    There is no primary key in this table but the foreign key of this table is User_ID.

```
CREATE TABLE IF NOT EXISTS `mydb`.`User_Login` (
  `Username` VARCHAR(45) NOT NULL,
  `Password` VARCHAR(45) NOT NULL,
  `User_ID` INT(11) NOT NULL,
  `last_update` TIMESTAMP NOT NULL DEFAULT
CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
  INDEX `fk_User_Login_User_Information1_idx`
(`User_ID` ASC) VISIBLE,
  CONSTRAINT `fk_User_Login_User_Information1`
    FOREIGN KEY (`User_ID`)
    REFERENCES `mydb`.`User_Information` (`User_ID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
```

16. **User_Task_History Table :** This table contains the information of the user and the task that they have completed and the level that the user is at.
    The primary key of this table is User_Task_History_ID and the foreign key of this table is User_Level_ID and User_Information_User_ID.

```
CREATE TABLE IF NOT EXISTS `mydb`.`User_Task_History`
(
  `User_Information_User_ID` INT(11) NOT NULL,
  `Completed_Task` INT(11) NOT NULL,
  `User_Task_History_ID` INT(11) NOT NULL
AUTO_INCREMENT,
  `Task_Completion_ID` INT(11) NOT NULL,
  `User_Level_ID` INT(11) NOT NULL,
  PRIMARY KEY (`User_Task_History_ID`),
  INDEX `fk_User_Task_History_User_Information1_idx`
(`User_Information_User_ID` ASC) VISIBLE,
  INDEX `fk_Level_idx` (`User_Level_ID` ASC) VISIBLE,
  CONSTRAINT `fk_User_Task_History_User_Information1`
    FOREIGN KEY (`User_Information_User_ID`)
    REFERENCES `mydb`.`User_Information` (`User_ID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
```

17. **User_Type Table :** In our system, we have classified our users into 3 types which are Government user, Normal user or admin. This table contains the id and the name of the type of user that can help us to classify any user into one of these 3 types.
    The primary key in this table is Type_ID and there is no foreign key.

```
CREATE TABLE IF NOT EXISTS `mydb`.`User_Type` (
  `Type_ID` INT(11) NOT NULL,
  `Type_Name` VARCHAR(45) NULL DEFAULT NULL,
  PRIMARY KEY (`Type_ID`))
```

4.2 Implementation of Trigger

A trigger is a SQL procedure that initiates an action when an event such as INSERT, DELETE or UPDATE occurs. Triggers are event-driven specialized procedures. The trigger cannot be called, it is fired automatically as a result of a data modification to the associated table.

1. **Level Up :** This trigger checks whether the status of the task is completed or not. If the status is completed then the trigger increases the Experience points of the user by 1.

```
USE `mydb`$$
DROP TRIGGER IF EXISTS `mydb`.`level_up` $$
USE `mydb`$$
CREATE
DEFINER=`root`@`localhost`
TRIGGER `mydb`.`level_up`
AFTER INSERT ON `mydb`.`User_Task_History`
FOR EACH ROW
begin
update (User_Information as ui JOIN
User_Task_History as uh ON ui.User_ID =
uh.User_Information_User_ID)
   JOIN Task_Completion as tc on tc.ID =
uh.Task_Completion_ID
     set ui.Experience_Gained =
ui.Experience_Gained + 1
     where tc.Task_Status = "Complete" and tc.ID =
NEW.Task_Completion_ID;
    end$$
```

2. **Government Officials :** This trigger takes in the information that is inserted in the Government_Officials table and then matches the area_id of the official that we just inserted in the Government_Officials table with the area_id in the Government table and increases the number of the officials by 1.

Basically this trigger helps us keep count of the government officials that are assigned in a particular area.

```
CREATE
DEFINER=`root`@`localhost`
TRIGGER `mydb`.`Government_Officials_AFTER_INSERT`
AFTER INSERT ON `mydb`.`Government_Officials`
FOR EACH ROW
BEGIN
     update (Government as g JOIN Government_Officials
as go ON g.Area_ID = go.Government_Area_ID)
     set g.No_of_officials = g.No_of_officials + 1
     where g.Area_ID = NEW.Government_Area_ID;
END$$
```

3. **Task Penalty :** This trigger is invoked if the task is not completed within a given duration i.e. if expected time is greater than duration (end_time – start_time) then status of the task is set to incomplete and when the status of the task is set to incomplete, the user is penalized and a certain amount of token points/experience points.

```
CREATE
DEFINER=`root`@`localhost`
TRIGGER `mydb`.`task_penalty`
AFTER INSERT ON `mydb`.`Task`
FOR EACH ROW
begin
   declare duration int;
   declare expected int;
   set duration = (select (Task_End_time -
Task_Start_Time) from Task where Task_ID =
new.Task_ID);
   set expected = (select TimeFrame from Task  where
Task_ID = new.Task_ID);
   if (expected < duration) then
       insert into
Task_Completion(Task_Status,Task_Task_ID)
```

```
        values('Complete',new.Task_ID);
    else
          insert into
Task_Completion(Task_Status,Task_Task_ID)
          values('InComplete',new.Task_ID);
      end if;
end$$
```

4.3 Implementation Of Store Procedure

1. **Add Task :** This Procedure when called allows us to add the task into the database along with all the information associated with it.

```
USE `mydb`;
DROP procedure IF EXISTS `mydb`.`Add_Task`;
DELIMITER $$
USE `mydb`$$
CREATE DEFINER=`root`@`localhost` PROCEDURE
`Add_Task`(IN tname VARCHAR(45), IN ds VARCHAR(160),
IN tp INT(11), IN tf INT(11), IN tst INT(11), IN tet
INT(11))
BEGIN
INSERT into Task(Task_Name, Task_Description,
Task_Points, Task_Type_ID, Task_Categories_Task_ID,
TimeFrame, Task_Start_Time, Task_End_Time) values
(tname, ds,tp, 1, 1, tf, tst, tet);
END$$
DELIMITER ;
```

2. **Add User :** This Procedure when called allows us to add the user into the database along with all the information associated with it.

```
USE `mydb`;
DROP procedure IF EXISTS `mydb`.`Add_User`;
DELIMITER $$
USE `mydb`$$
CREATE DEFINER=`root`@`localhost` PROCEDURE
`Add_User`(IN UName varchar(45),IN FName varchar(45),
```

```
IN LName varchar(45), IN CNumber varchar(45), IN Dob
DATE)
BEGIN
    insert into
User_Information(User_Name,First_Name,Last_Name,
Contact_Number, DOB, _idAddress) values
(UName,FName,LName,CNumber, Dob, 1);
END$$
DELIMITER ;
```

3. **People at Level :** This procedure allows us to input a level number and then we can observe how many users are at that level along with their user id.

```
USE `mydb`;
DROP procedure IF EXISTS `mydb`.`People_At_Level`;
DELIMITER $$
USE `mydb`$$
CREATE DEFINER=`root`@`localhost` PROCEDURE
`People_At_Level`(IN ld int(11))
BEGIN
    SELECT User_Information_User_ID FROM
user_task_history where User_Level_ID = ld;
END$$
DELIMITER ;
```

4. **Task Assignment :** This procedure when invoked asks us for the user id of the assigner as well as the doer and gives the output of the interaction between them which is the number of tasks that they have between them and the task information.

```
USE `mydb`;
DROP procedure IF EXISTS `mydb`.`TaskAssignment`;
DELIMITER $$
USE `mydb`$$
CREATE DEFINER=`root`@`localhost` PROCEDURE
`TaskAssignment`(IN Assigner INT, IN Doer INT)
BEGIN
    select * from Task join Task_Assignment on
```

```
Task.Task_ID=Task_Assignment.Task_Task_ID where
Task_Assignment.Task_Assignner_User_ID=Assigner and
Task_Assignment.Task_Doer_User_ID=Doer;
END$$
DELIMITER ;
```

5. **User Rewards :** This procedure when invoked allows us to view the rewards that a user has had over the course of their life in the system.

```
USE `mydb`;
DROP procedure IF EXISTS `mydb`.`User_Rewards`;
DELIMITER $$
USE `mydb`$$
CREATE DEFINER=`root`@`localhost` PROCEDURE
`User_Rewards`(IN UserID int(11))
BEGIN
     SELECT * FROM Rewards where Level_ID = (Select
Level_ID from User_Level where User_ID = UserID);
END$$
DELIMITER ;
```

4.4 Implementation of Views

1. **Balance of user :** This view when invoked, displays the token balance of all the users in the system.

```
USE `mydb`;
CREATE  OR REPLACE ALGORITHM=UNDEFINED
DEFINER=`root`@`localhost` SQL SECURITY DEFINER VIEW
`mydb`.`balance of user` AS select
`mydb`.`user_information`.`User_ID` AS
`User_ID`,`mydb`.`user_bank`.`Token_Balance` AS
`Token_Balance` from (`mydb`.`user_information` join
`mydb`.`user_bank`) where
(`mydb`.`user_bank`.`User_ID` =
`mydb`.`user_information`.`User_ID`);
```

2. **Government official Area :** This view when invoked displays the official area under a particular government employee.

```
USE `mydb`;
CREATE  OR REPLACE ALGORITHM=UNDEFINED
DEFINER=`root`@`localhost` SQL SECURITY DEFINER VIEW
`mydb`.`government_official_area` AS select
`mydb`.`government`.`Area_ID` AS
`Area_ID`,`mydb`.`government_officials`.`Government_O
fficial_ID` AS
`Government_Official_ID`,`mydb`.`government_officials
`.`Government_Officials_Name` AS
`Government_Officials_Name` from (`mydb`.`government`
join `mydb`.`government_officials`
on((`mydb`.`government`.`Area_ID` =
`mydb`.`government_officials`.`Government_Area_ID`)))
;
```

3. **User Address :** This view when invoked displays the information of the user including their address.

```
USE `mydb`;
CREATE  OR REPLACE ALGORITHM=UNDEFINED
DEFINER=`root`@`localhost` SQL SECURITY DEFINER VIEW
`mydb`.`useraddress` AS select
`mydb`.`address`.`idAddress` AS
`idAddress`,`mydb`.`address`.`address` AS
`address`,`mydb`.`address`.`adress2` AS
`adress2`,`mydb`.`address`.`city` AS
`city`,`mydb`.`address`.`postal_code` AS
`postal_code`,`mydb`.`user_information`.`User_ID` AS
`User_ID` from (`mydb`.`address` join
`mydb`.`user_information`
on((`mydb`.`address`.`idAddress` =
`mydb`.`user_information`.`_idAddress`));
```

4.5 Implementation Of Grants

1. **Normal User :** A normal user has access to just insert and update the information related to themselves in the User_Information Table
2. **Admin :** The admin has access to the entire database and has the permission to modify everything in the database.

3. **Government :** The government user had access to insert, update and delete the information related to the Government and Government officials table.

4.6 Implementation Of Functions

1) **To calculate level :** When this function is called, it takes into account the Experience points of a user and then calculates the level of a user according to it.

```
DELIMITER $$
USE `mydb`$$
CREATE DEFINER=`root`@`localhost` FUNCTION `CalcLevel`(
starting_value INT ) RETURNS int(11)
BEGIN
   DECLARE Level INT;
   SET Level = 0;
label1: LOOP
     IF starting_value <= 300 AND starting_value > 201
THEN
       Set Level = 2;
    ELSEIF starting_value <= 100 AND starting_value > 0
THEN
       Set Level = 1;
     ELSEIF starting_value <= 200 AND starting_value >
101 THEN
       Set Level = 2;
    ELSEIF starting_value <= 400 AND starting_value >
301 THEN
       Set Level = 4;
    ELSEIF starting_value < 500 AND starting_value > 401
THEN
```

```
        Set Level = 5;
    ELSE
         SET LEVEL = 0;
     END IF;
     LEAVE label1;
     END LOOP label1;
    RETURN Level;
END$$
DELIMITER ;
```

Chapter 5

Future Enhancement

This initiative if taken under serious consideration could serve as an incentive to make society a better place through the collective efforts of all individuals in the society as a whole. This will help ensure a model society in which sincerity and trustworthiness become the conscious norms of action among all people.

Chapter 6

Assumptions and Limitations

- The authorities have given approval to share their data and update the data on a regular basis.
- The users are motivated to help each other and receive help when in need.
- The idea of cheating and exploiting the system is not exciting to them.
- The app is being utilized to build a new fabric of society, which is bound to grow in the digital age, when personal human interaction is limited.

Chapter 7

Conclusion

In this project we have created an application which is easy to access and user friendly. The application keeps a backup of the users performing the task and experience points earned if the level has been completed within a particular duration and move on to the next level to gain benefits from the government. This application gives people a chance to do something for the society or others in their daily lives and earns a good "*citizen ticket*".

Chapter 8

References

1. https://lvluplife.com
2. https://www.techopedia.com/definition/32169/dbms-interface
3. https://www.w3schools.com/sql/default.asp

Adventure is out there, and
the world needs more heroes.
Will you heed the call?