

VIET NAM NATIONAL UNIVERSITY HO CHI MINH CITY
HO CHI MINH UNIVERSITY OF TECHNOLOGY



DATABASE
ASSIGNMENT

Author:

Name	Student ID	Class
Nguyễn Nam Kha	2052515	CC03
Nguyễn Ngọc Hòa	2052485	CC03
Châu Nguyễn An Khang	1952757	CC03

Ho Chi Minh city, 12/2022

CONTRIBUTIONS

Name	Student ID	Task
Nguyễn Nam Kha	2052515	Part 1.II, Part 2.I, Part 3
Nguyễn Ngọc Hòa	2052485	Part 1.I, Part 2.II, Part 3
Châu Nguyễn An Khang	1952757	Part 1.I, Part 2.I, Part 3

Contents

PART 1: CREATE DATABASE	1
I. Create relational table.....	1
II. Insert data.....	18
PART 2: STORE PROCEDURE, FUNCTION, TRIGGER	31
I. Stored Procedure/Function	31
1. Write procedure/function GoiDichVu to show valid service packets of a customer.	31
2. Write procedure/function ThongKeLuotKhach to show statistics of a branch in a year.	32
II. Trigger	34
1. Trigger that update value	34
2. Check constraint: Customer can not buy the same service packet if not expired	40
PART 3: BUILD WEB APPLICATION	41
1. Create user.....	42
2. Main functionality	44
a. View Customer's information	44
b. Search Customer and View Booking's information	44
c. Add a new room type	45
d. Customer's Statistics.....	47



PART 1: CREATE DATABASE

To implement this assignment, we use MySQL – a free and open-source relational database management system.

All of our source code can be found here:

<https://github.com/namkha1032/databaseAssignment>

The code to create database, create stored procedures, functions, triggers and insert data are all stored in `script.sql` file.

I. Create relational table

1. branch (BranchID, Province, Address, PhoneNum, Email)

This table stores the data of each branch. In this table we set *BranchID* as the PRIMARY KEY. We also set the attributes *Address*, *PhoneNum*, *Email* as UNIQUE KEYS since those attributes cannot be duplicated.

```
CREATE TABLE branch_seq(
    id INT NOT NULL PRIMARY KEY AUTO_INCREMENT);
CREATE TABLE `branch` (
    `BranchID` varchar(6) NOT NULL,
    `Province` varchar(50) NOT NULL,
    `Address` varchar(100) NOT NULL,
    `PhoneNum` varchar(12) NOT NULL,
    `Email` varchar(64) NOT NULL,
    PRIMARY KEY (`BranchID`),
    UNIQUE KEY `Address_UNIQUE` (`Address`),
    UNIQUE KEY `PhoneNum_UNIQUE` (`PhoneNum`),
    UNIQUE KEY `Email_UNIQUE` (`Email`));
```

We also set a TRIGGER that defines a regex to ensure that the incoming *BranchIDs* always start with CN followed by an auto-increment integer.

```
DELIMITER $$ 
CREATE TRIGGER generate_branch_id
BEFORE INSERT ON `branch`
FOR EACH ROW
BEGIN
```



```
INSERT INTO branch_seq VALUES (NULL);
SET NEW.BranchID = CONCAT('CN', LAST_INSERT_ID());
END$$
DELIMITER ;
```

2. image_branch (BranchID, Image)

This table stores the data of the hotel's image. In this table we set both *BranchID* and *Image* as the PRIMARY KEY. *BranchID* is the FOREIGN KEY reference *branch(branchID)*

```
CREATE TABLE `image_branch` (
  `BranchID` varchar(6) NOT NULL,
  `Image` varchar(255) NOT NULL,
  PRIMARY KEY (`BranchID`, `Image`),
  CONSTRAINT `FK_branch_image` FOREIGN KEY (`BranchID`) REFERENCES
  `branch` (`BranchID`) ON UPDATE CASCADE ON DELETE CASCADE);
```

3. zone (BranchID, ZoneName)

This table stores the data of each zone. In this table we set both *BranchID* and *ZoneName* as the PRIMARY KEY. *BranchID* is the FOREIGN KEY reference *branch(branchID)*.

```
CREATE TABLE `zone` (
  `BranchID` varchar(6) NOT NULL,
  `ZoneName` varchar(20) NOT NULL,
  PRIMARY KEY (`BranchID`, `ZoneName`),
  CONSTRAINT `FK_branch_zone` FOREIGN KEY (`BranchID`) REFERENCES
  `branch` (`BranchID`) ON UPDATE CASCADE ON DELETE CASCADE
);
```

4. room_type (RoomID, RoomName, Area, NumGuest, Description)

This table stores the data of each room type. In this table we set *RoomID* as the PRIMARY KEY. *BranchID* is the FOREIGN KEY reference *branch(branchID)*. We also set constraints ‘LimitGuest’ ensuring that the number of guest will be between 1 and 10

```
CREATE TABLE `room_type` (
  `RoomID` int unsigned NOT NULL AUTO_INCREMENT,
```



```
`RoomName` varchar(45) NOT NULL,  
 `Area` int unsigned NOT NULL,  
 `NumGuest` int unsigned NOT NULL,  
 `Description` TEXT DEFAULT NULL,  
 PRIMARY KEY (`RoomID`),  
 CONSTRAINT `LimitGuest` CHECK(`NumGuest` BETWEEN 1 AND 10) ) ;
```

5. bed_info (RoomID, Size, Quantity)

This table stores the information about the bed. in this table we set both *RoomID* and *Size* as the PRIMARY KEY. *RoomID* is the FOREIGN KEY reference *room_type(roomID)*. We also set constraints ‘*LimitBed*’ ensuring that Quantity will be between 1 and 10

```
CREATE TABLE `bed_info` (  
 `RoomID` int unsigned NOT NULL,  
 `Size` DECIMAL(2,1) NOT NULL,  
 `Quantity` int NOT NULL DEFAULT '1',  
 PRIMARY KEY (`RoomID`, `Size`),  
 CONSTRAINT `FK_roomtype_bed` FOREIGN KEY (`RoomID`) REFERENCES `room_type`(`RoomID`) ON UPDATE CASCADE ON DELETE CASCADE,  
 CONSTRAINT `LimitBed` CHECK(`Quantity` BETWEEN 1 AND 10)) ;
```

6. roomtype_branch (RoomID, BranchID, RentalPrice)

This table stores the data of each special room type of each branch. In this table we set both *RoomID* and *BranchID* as the PRIMARY KEY. While *BranchID* is the FOREIGN KEY reference *branch(branchID)*, *RoomID* is the FOREIGN KEY reference *room_type(RoomID)*

```
CREATE TABLE `roomtype_branch` (  
 `RoomID` int unsigned NOT NULL,  
 `BranchID` varchar(6) NOT NULL,  
 `RentalPrice` int unsigned NOT NULL,  
 PRIMARY KEY (`RoomID`, `BranchID`),  
 KEY `BranchID2_idx` (`BranchID`),  
 CONSTRAINT `BranchID2` FOREIGN KEY (`BranchID`) REFERENCES `branch`(`BranchID`) ON UPDATE CASCADE ON DELETE CASCADE,  
 CONSTRAINT `RoomID2` FOREIGN KEY (`RoomID`) REFERENCES `room_type`(`RoomID`) ON UPDATE CASCADE ON DELETE CASCADE) ;
```



7. room(BranchID, RoomNumber, RoomID, ZoneName)

This table stores the data of each room. In this table we set both *RoomNumber* and *BranchID* as the PRIMARY KEY. While *BranchID* is the FOREIGN KEY reference *zone(branchID)*, *RoomID* is the FOREIGN KEY reference *room_type(RoomID)* and ('*BranchID*', '*ZoneName*') is the FOREIGN KEY reference *zone('BranchID', 'ZoneName')*

```
CREATE TABLE `room` (
  `BranchID` varchar(6) NOT NULL,
  `RoomNumber` varchar(3) NOT NULL,
  `RoomID` int unsigned NOT NULL,
  `ZoneName` varchar(20) NOT NULL,
  PRIMARY KEY (`BranchID`, `RoomNumber`),
  KEY `ZoneName_idx` (`ZoneName`),
  KEY `RoomID3_idx` (`RoomID`),
  CONSTRAINT `FK_branch_room` FOREIGN KEY (`BranchID`) REFERENCES `zone`(`BranchID`) ON UPDATE CASCADE ON DELETE CASCADE,
  CONSTRAINT `FK_roomtype_room` FOREIGN KEY (`RoomID`) REFERENCES `room_type`(`RoomID`) ON UPDATE CASCADE ON DELETE CASCADE,
  CONSTRAINT `FK_zone_room` FOREIGN KEY (`BranchID`, `ZoneName`)
  REFERENCES `zone`(`BranchID`, `ZoneName`) ON UPDATE CASCADE ON DELETE CASCADE
);
```

8. supply_type (SupplyID, SppName)

This table stores the data of each supply type. In this table we set *SppID* as the PRIMARY KEY.

```
CREATE TABLE `supply_type` (
  `SppID` varchar(6) NOT NULL PRIMARY KEY ,
  `SppName` varchar(50) NOT NULL UNIQUE
);
```

We also set a TRIGGER that defines a regex to ensure that the incoming *sppID* is always started with VT followed by a chain of six numbers [0-9][0-9][0-9][0-9][0-9][0-9]. If the incoming *SppID* does not follow that pattern, we will get an error 'Wrong Format!!!'

```
DELIMITER $$  
CREATE TRIGGER SppId_check BEFORE INSERT ON `supply_type`  
FOR EACH ROW
```



```
BEGIN
IF (NEW.SppID REGEXP '^VT[0-9]{4}$' ) = 0 THEN
    SIGNAL SQLSTATE '12345'
    SET MESSAGE_TEXT = 'Wrong Format!!!';
END IF;
END$$
DELIMITER ;
```

9. room_type_supply_type (SppID, RoomID, Quantity)

This table stores the data of unique supply type(s) in each room type. In this table we set (`*SppID*`, `*RoomID*`) as the PRIMARY KEY. *RoomID* is the FOREIGN KEY reference *room_type(RoomID)*. Likewise, *SppID* is the FOREIGN KEY reference *supply_type(SppID)*

```
CREATE TABLE `room_type_supply_type` (
`SppID` varchar(8) NOT NULL,
`RoomID` int unsigned NOT NULL,
`Quantity` int unsigned NOT NULL DEFAULT '1',
PRIMARY KEY (`SppID`, `RoomID`),
KEY `RoomID4_idx` (`RoomID`),
CONSTRAINT `FK_room` FOREIGN KEY (`RoomID`) REFERENCES `room_type`(`RoomID`)
ON UPDATE CASCADE ON DELETE CASCADE,
CONSTRAINT `FK_supply_type` FOREIGN KEY (`SppID`) REFERENCES
`supply_type` (`SppID`) ON UPDATE CASCADE ON DELETE CASCADE) ;
```

10. supplies (BranchID, SppID, SupplyIndex, Condition, RoomNumber)

This table stores the data of each supply. In this table we set (`*BranchID*`, `*SupplyIndex*`, `*SppID*`) as the PRIMARY KEY. *BranchID* is the FOREIGN KEY reference for both *branch(branchID)* and *room(BranchID)*. While (`*BranchID*`, `*RoomNumber*`) is the FOREIGN KEY reference *room(`BranchID`, `RoomNumber`)*, *SppID* is the FOREIGN KEY reference *SupplyID('SppID')*

```
CREATE TABLE `supplies` (
`BranchID` varchar(6) NOT NULL,
`SppID` varchar(6) NOT NULL,
`SupplyIndex` int unsigned NOT NULL,
`Condition` varchar(45) DEFAULT 'Good',
```



```
`RoomNumber` varchar(3) DEFAULT NULL,  
PRIMARY KEY (`BranchID`, `SupplyIndex`, `SppID`),  
KEY `SppID2_idx` (`SppID`),  
CONSTRAINT `FK_branch_supplies` FOREIGN KEY (`BranchID`) REFERENCES  
branch` (`BranchID`) ON UPDATE CASCADE ON DELETE CASCADE,  
CONSTRAINT `BranchID51` FOREIGN KEY (`BranchID`) REFERENCES `room`  
(`BranchID`) ON UPDATE CASCADE ON DELETE CASCADE,  
CONSTRAINT `FK_room_supplies` FOREIGN KEY (`BranchID`, `RoomNumber`)  
REFERENCES `room` (`BranchID`, `RoomNumber`) ON UPDATE CASCADE ON DELETE  
CASCADE,  
CONSTRAINT `FK_supply_type_supplies` FOREIGN KEY (`SppID`) REFERENCES  
`supply_type` (`SppID`) ON UPDATE CASCADE ON DELETE CASCADE) ;
```

11. supplier (SupplierID, SupplierName, Mail, Address)

This table stores the data of each supplier. In this table we set *SupplierID* as the PRIMARY KEY.

```
CREATE TABLE `supplier` (  
`SupplierID` varchar(8) NOT NULL,  
`SupplierName` varchar(50) NOT NULL,  
`Mail` varchar(45) NOT NULL,  
`Address` varchar(50) NOT NULL,  
PRIMARY KEY (`SupplierID`)) ;
```

We also set a TRIGGER that defines a regex to ensure that the incoming SupplierIDs always start with NCC followed by a chain of six numbers [0-9][0-9][0-9][0-9][0-9][0-9]. If the incoming SppID does not follow that pattern, we will get an error ‘Wrong Format!!!’

```
DELIMITER $$  
CREATE TRIGGER SupplierID_check BEFORE INSERT ON `supplier`  
FOR EACH ROW  
BEGIN  
IF (NEW.SupplierID REGEXP '^NCC[0-9]{4}$' ) = 0 THEN  
    SIGNAL SQLSTATE '12345'  
    SET MESSAGE_TEXT = 'Wrong Format!!!';  
END IF;  
END$$  
DELIMITER ;
```



12. supplying (SppID, BranchID, SupplierID)

This table stores the data of each zone. In this table we set ('*SppID*', '*BranchID*') as the PRIMARY KEY. While *BranchID* is the FOREIGN KEY reference *branch(branchID)*, *SppID* is the FOREIGN KEY reference *supply_type(SppID)* and *SupplierID* is the FOREIGN KEY reference *supplier(SupplierID)*.

```
CREATE TABLE `supplying` (
  `SppID` varchar(8) NOT NULL,
  `BranchID` varchar(6) NOT NULL,
  `SupplierID` varchar(8) NOT NULL,
  PRIMARY KEY (`SppID`, `BranchID`),
  KEY `SupplierID_idx` (`SupplierID`),
  KEY `BranchID6_idx` (`BranchID`),
  CONSTRAINT `FK_branch_supplying` FOREIGN KEY (`BranchID`) REFERENCES `branch` (`BranchID`) ON UPDATE CASCADE ON DELETE CASCADE,
  CONSTRAINT `FK_supply_type_supplying` FOREIGN KEY (`SppID`) REFERENCES `supply_type` (`SppID`) ON UPDATE CASCADE ON DELETE CASCADE,
  CONSTRAINT `FK_supplier_supplying` FOREIGN KEY (`SupplierID`) REFERENCES `supplier` (`SupplierID`) ON UPDATE CASCADE ON DELETE CASCADE
);
```

13. customer (CustomerID, CitizenID, FullName, Phone, Email, Username, Password, Point, CustomerType)

This table stores the data of each customer. In this table we set *CustomerID* as the PRIMARY KEY. We also set the attributes *CitizenID*, *Phone*, *Email*, *Username* as UNIQUE KEYS since those attributes cannot be duplicated.

```
CREATE TABLE `customer` (
  `CustomerID` varchar(10) NOT NULL,
  `CitizenID` varchar(12) NOT NULL,
  `FullName` varchar(45) NOT NULL,
  `Phone` varchar(12) NOT NULL,
  `Email` varchar(45) DEFAULT NULL,
  `Username` varchar(45) DEFAULT NULL,
  `Password` varchar(45) DEFAULT NULL,
  `Point` int unsigned DEFAULT '0',
  `CustomerType` int unsigned DEFAULT '1',
```



```
PRIMARY KEY (`CustomerID`),
UNIQUE KEY `CitizenID_UNIQUE` (`CitizenID`),
UNIQUE KEY `Phone_UNIQUE` (`Phone`),
UNIQUE KEY `Email_UNIQUE` (`Email`),
UNIQUE KEY `UserName_UNIQUE` (`Username`)) ;
```

We also set a TRIGGER that defines a regex to ensure that the incoming *CustomerIDs* always start with KH followed by a chain of six numbers [0-9][0-9][0-9][0-9][0-9][0-9]. If the incoming *CustomerID* does not follow that pattern, we will get an error ‘Wrong Format!!!’

```
DELIMITER $$  
CREATE TRIGGER CustomerID_check BEFORE INSERT ON `customer`  
FOR EACH ROW  
BEGIN  
IF (NEW.CustomerID REGEXP '^KH[0-9]{6}$' ) = 0 THEN  
    SIGNAL SQLSTATE '12345'  
    SET MESSAGE_TEXT = 'Wrong Format!!!';  
END IF;  
END$$  
DELIMITER ;
```

14. service_packet (PackageName, DayNum, GuestNum, Price)

This table stores the data of the packets that the hotel offers. We set *PackageName* as the PRIMARY KEY. We also set constraints specifying that *DayNum* must be between 1 and 100; and *GuestNum* must be between 1 and 10.

```
CREATE TABLE `service_packet` (  
`PackageName` varchar(50) NOT NULL,  
`DayNum` int unsigned NOT NULL,  
`GuestNum` int unsigned NOT NULL,  
`Price` int NOT NULL,  
PRIMARY KEY (`PackageName`),  
CONSTRAINT `LimitDay` CHECK(`DayNum` BETWEEN 1 AND 100),  
CONSTRAINT `LimitGuestPacket` CHECK(`GuestNum` BETWEEN 1 AND 10)) ;
```



15. packet_bill (CustomerID, PackageName, PurchaseDate, StartDate, TotalPay, RemainDay)

This table stores the data of the packets that are purchased by customers. We set *(CustomerID, PackageName, PurchaseDate)* as the PRIMARY KEY of the table. *CustomerID* is the FOREIGN KEY reference *customer(CustomerID)*. Likewise, *PackageName* is the FOREIGN KEY reference to *service_packet(PackageName)*. CONSTRAINT *CheckDate* is set to ensure that *StartDate > PurchaseDate*.

```
CREATE TABLE `packet_bill` (
  `CustomerID` varchar(10) NOT NULL,
  `PackageName` varchar(50) NOT NULL,
  `PurchaseDate` datetime NOT NULL,
  `StartDate` datetime NOT NULL,
  `TotalPay` int DEFAULT NULL,
  `RemainDay` int unsigned DEFAULT NULL,
  PRIMARY KEY (`CustomerID`, `PackageName`, `PurchaseDate`),
  KEY `PName1_idx` (`PackageName`),
  CONSTRAINT `FK_customer_bill` FOREIGN KEY (`CustomerID`) REFERENCES `customer` (`CustomerID`) ON UPDATE CASCADE ON DELETE CASCADE,
  CONSTRAINT `FK_packet_bill` FOREIGN KEY (`PackageName`) REFERENCES `service_packet` (`PackageName`) ON UPDATE CASCADE ON DELETE CASCADE,
  CONSTRAINT `CheckDate` CHECK(StartDate > PurchaseDate)) ;
```

16. booking (BookingID, BookingDate, GuestNum, CheckIn, CheckOut, Status, TotalPay, CustomerID, PackageName)

This table stores the data of each booking. *BookingID* is the PRIMARY KEY of the table. *CustomerID* and *PackageName* are the FOREIGN KEYS reference *customer(CustomerID)* and *service_package(PackageName)* respectively. Constraints are set to ensure that *CheckIn > BookingDate* and *CheckOut > CheckIn*.

```
CREATE TABLE `booking` (
  `BookingID` varchar(20) NOT NULL,
  `BookingDate` datetime NOT NULL,
  `GuestNum` int unsigned NOT NULL,
```



```
`CheckIn` datetime NOT NULL,  
 `CheckOut` datetime NOT NULL,  
 `Status` int unsigned NOT NULL DEFAULT '0',  
 `TotalPay` int unsigned NOT NULL DEFAULT '0',  
 `CustomerID` varchar(10) NOT NULL,  
 `PackageName` varchar(45) DEFAULT NULL,  
 PRIMARY KEY (`BookingID`),  
 KEY `CusID3_idx` (`CustomerID`),  
 KEY `PName_idx` (`PackageName`),  
 CONSTRAINT `FK_customer_booking` FOREIGN KEY (`CustomerID`) REFERENCES  
 `customer` (`CustomerID`) ON UPDATE CASCADE ON DELETE CASCADE,  
 CONSTRAINT `FK_package_booking` FOREIGN KEY (`PackageName`) REFERENCES  
 `service_packet` (`PackageName`) ON UPDATE CASCADE ON DELETE CASCADE,  
 CONSTRAINT `CheckDate2` CHECK(CheckIn > BookingDate),  
 CONSTRAINT `CheckDate3` CHECK(CheckOut > CheckIn)) ;
```

We also used TRIGGER to auto generate *BookingID* for each booking being added to the table. In the TRIGGER, the *BookingID* is defined by the pattern DP[DDMMYYYY][id] with DDMMYYYY is the date of the booking and *id* is an auto increment value created by table *booking_seq*.

```
CREATE TABLE booking_seq(  
    id INT NOT NULL PRIMARY KEY AUTO_INCREMENT  
);  
DELIMITER $$  
CREATE TRIGGER generate_booking_id  
BEFORE INSERT ON `booking`  
FOR EACH ROW  
BEGIN  
    INSERT INTO booking_seq VALUES (NULL);  
    SET NEW.BookingID = CONCAT('DP', DATE_FORMAT(CAST(NEW.BookingDate as  
DATE), '%d%m%Y'), LPAD(LAST_INSERT_ID(), 6, '0'));  
END$$  
DELIMITER ;
```

17. booking_room (BookingID, BranchID, RoomNumber)

This table stores data of the booking room in each booking. All three attributes (*BookingID*, *BranchID*, *RoomNumber*) are set as PRIMARY KEY. *BookingID* is the FOREIGN KEY reference *booking* (*BookingID*). (*BranchID*, *RoomNumber*) is the FOREIGN KEY reference *room* (*BranchID*, *RoomNumber*).



```
CREATE TABLE `booking_room` (
    `BookingID` varchar(20) NOT NULL,
    `BranchID` varchar(6) NOT NULL,
    `RoomNumber` varchar(3) NOT NULL,
    PRIMARY KEY (`BookingID`, `BranchID`, `RoomNumber`),
    KEY `BranchID_idx` (`BranchID`),
    CONSTRAINT `BookingID` FOREIGN KEY (`BookingID`) REFERENCES `booking`(`BookingID`) ON UPDATE CASCADE ON DELETE CASCADE,
    CONSTRAINT `BranchID7` FOREIGN KEY (`BranchID`, `RoomNumber`) REFERENCES `room` (`BranchID`, `RoomNumber`) ON UPDATE CASCADE ON DELETE CASCADE) ;
```

18. booking_bill (BillID, CheckIn, CheckOut, BookingID)

This table stores data about the Bill of each booking. *BillID* is the PRIMARY KEY of the table. *BookingID* is the FOREIGN KEY reference *booking(BookingID)*.

```
CREATE TABLE `booking_bill` (
    `BillID` varchar(20) NOT NULL,
    `CheckIn` TIME NOT NULL,
    `CheckOut` TIME NOT NULL,
    `BookingID` varchar(20) NOT NULL,
    PRIMARY KEY (`BillID`),
    KEY `BookingID_idx` (`BookingID`),
    CONSTRAINT `BookingID2` FOREIGN KEY (`BookingID`) REFERENCES `booking`(`BookingID`) ON UPDATE CASCADE ON DELETE CASCADE) ;
```

TRIGGER is used to auto generate *BillID* for each bill being added to the table. In the TRIGGER, the *BillID* is defined by the pattern HD[DDMMYYYY][id] with DDMMYYYY is the current date and id is an auto increment value created by table *booking_bill_seq*.

```
CREATE TABLE booking_bill_seq(
    id INT NOT NULL PRIMARY KEY AUTO_INCREMENT
);
DELIMITER $$;
CREATE TRIGGER generate_booking_bill_id
BEFORE INSERT ON `booking_bill`
FOR EACH ROW
BEGIN
    INSERT INTO booking_bill_seq VALUES (NULL);
```



```
SET NEW.BillID = CONCAT('HD', DATE_FORMAT(CAST(NOW() as
DATE),'%d%m%Y'), LPAD(LAST_INSERT_ID(), 6, '0'));
END$$
DELIMITER ;
```

19. enterprise (EnterpriseID, EnterpriseName)

This table stores data about the enterprises that collaborate with the hotel.
EnterpriseID is the PRIMARY KEY of the table.

```
CREATE TABLE `enterprise` (
`EnterpriseID` varchar(8) NOT NULL,
`EnterpriseName` varchar(45) NOT NULL,
PRIMARY KEY (`EnterpriseID`)) ;
```

We also use a TRIGGER that defines a regex to ensure that the incoming *EnterpriseIDs* always start with DN followed by a chain of four numbers [0-9][0-9][0-9][0-9]. If the incoming *EnterpriseID* does not follow that pattern, we will get an error ‘*Wrong Format!!!*’

```
DELIMITER $$

CREATE TRIGGER EnterpriseID_check BEFORE INSERT ON `enterprise`
FOR EACH ROW
BEGIN
IF (NEW.EnterpriseID REGEXP '^DN[0-9]{4}$' ) = 0 THEN
    SIGNAL SQLSTATE '12345'
        SET MESSAGE_TEXT = 'Wrong Format!!!';
END IF;
END$$
DELIMITER ;
```

20. service (ServiceID, ServiceType, GuestNum, Style, EnterpriseID)

This table stores data about services that the enterprises can offer. *ServiceID* is the PRIMARY KEY of the table. *EnterpriseID* is the FOREIGN KEY reference *enterprise(EnterpriseID)*.

```
CREATE TABLE `service` (
`ServiceID` varchar(8) NOT NULL,
`ServiceType` varchar(1) NOT NULL,
`GuestNum` int unsigned DEFAULT '0',
`Style` varchar(45) DEFAULT NULL,
```



```
`EnterpriseID` varchar(10) NOT NULL,  
PRIMARY KEY (`ServiceID`),  
CONSTRAINT `FK_enterprise_service` FOREIGN KEY (`EnterpriseID`)  
REFERENCES `enterprise` (`EnterpriseID`) ON UPDATE CASCADE ON DELETE  
CASCADE) ;
```

TRIGGER is also used to ensure that the incoming *ServiceID* follows a pattern of *DV[R/S/C/M/B][0-9][0-9][0-9]* with *R* stands for *Restaurant*, *S* stands for *Spa*, *C* stands for *Convenience store*, *M* stands for *Souvenir store* and *B* stands for *Bar*. If the incoming *ServiceID* does not follow that pattern, we will get an error ‘Wrong Format!!!’

```
DELIMITER $$  
CREATE TRIGGER ServiceID_check BEFORE INSERT ON `service`  
FOR EACH ROW  
BEGIN  
IF (NEW.ServiceID REGEXP '^DV[RSCMB][0-9]{3}$' ) = 0 THEN  
    SIGNAL SQLSTATE '12345'  
    SET MESSAGE_TEXT = 'Wrong Format!!!';  
END IF;  
IF (SUBSTR(NEW.ServiceID,3,1) <> NEW.ServiceType) THEN  
    SIGNAL SQLSTATE '12345'  
    SET MESSAGE_TEXT = 'Wrong ServiceType!!!';  
END IF;  
END$$  
DELIMITER ;
```

21. spa_service (ServiceID, ProvidedService)

This table stores data about the services offered by the spa. (*ServiceID*, *ProvidedService*) is the PRIMARY KEY of the table. *ServiceID* is the FOREIGN KEY reference *service* (*ServiceID*).

```
CREATE TABLE `spa_service` (  
`ServiceID` varchar(10) NOT NULL,  
`ProvidedService` varchar(50) NOT NULL,  
PRIMARY KEY (`ServiceID`, `ProvidedService`),  
CONSTRAINT `FK_service_spa` FOREIGN KEY (`ServiceID`) REFERENCES  
`service` (`ServiceID`) ON UPDATE CASCADE ON DELETE CASCADE) ;
```

TRIGGER is also used to ensure that the incoming *ServiceID* follows a pattern of *DV[R/S/C/M/B][0-9][0-9][0-9]*. If the incoming *ServiceID* does not



follow that pattern, we will get an error ‘*Wrong Format!!!*’. Then it will check if the incoming *ServiceID* follows the pattern of *DV[S][0-9][0-9][0-9]*. If it does not, we will get an error ‘*Wrong ServiceType!!!*’.

```
DELIMITER $$  
CREATE TRIGGER Spa_ServiceID_check BEFORE INSERT ON `spa_service`  
FOR EACH ROW  
BEGIN  
IF (NEW.ServiceID REGEXP '^DV[RSCMB][0-9]{3}$' ) = 0 THEN  
    SIGNAL SQLSTATE '12345'  
    SET MESSAGE_TEXT = 'Wrong Format!!!';  
END IF;  
IF (SUBSTR(NEW.ServiceID,3,1) <> 'S') THEN  
    SIGNAL SQLSTATE '12345'  
    SET MESSAGE_TEXT = 'Wrong ServiceType!!!';  
END IF;  
END$$  
DELIMITER ;
```

22. souvenir_category (*ServiceID, Category*)

This data stores data about the Category of products that souvenir stores offer. (*ServiceID, Category*) is the PRIMARY KEY of the table.

```
CREATE TABLE `souvenir_category` (  
    `ServiceID` varchar(10) NOT NULL,  
    `Category` varchar(20) NOT NULL,  
    PRIMARY KEY (`ServiceID`, `Category`),  
    CONSTRAINT `svID2` FOREIGN KEY (`ServiceID`) REFERENCES `service`(`ServiceID`) ON UPDATE CASCADE ON DELETE CASCADE  
) ;
```

TRIGGER is used to ensure that the incoming *ServiceID* follows a pattern of *DV[R/S/C/M/B][0-9][0-9][0-9]*. If the incoming *ServiceID* does not follow that pattern, we will get an error ‘*Wrong Format!!!*’. Then it will check if the incoming *ServiceID* follows the pattern of *DV[M][0-9][0-9][0-9]*. If it does not, we will get an error ‘*Wrong ServiceType!!!*’.

```
DELIMITER $$  
CREATE TRIGGER Souvenir_ServiceID_check1 BEFORE INSERT ON  
`souvenir_category`  
FOR EACH ROW
```



```
BEGIN
IF (NEW.ServiceID REGEXP '^DV[RSCMB][0-9]{3}$' ) = 0 THEN
    SIGNAL SQLSTATE '12345'
    SET MESSAGE_TEXT = 'Wrong Format!!!';
END IF;
IF (SUBSTR(NEW.ServiceID,3,1) <> 'M') THEN
    SIGNAL SQLSTATE '12345'
    SET MESSAGE_TEXT = 'Wrong ServiceType!!!';
END IF;
END$$
DELIMITER ;
```

23. souvenir_brand (ServiceID, Brand)

This data stores data about the Brand of products that souvenir stores offer.
(*ServiceID, Brand*) is the PRIMARY KEY of the table.

```
CREATE TABLE `souvenir_brand` (
    `ServiceID` varchar(10) NOT NULL,
    `Brand` varchar(45) NOT NULL,
    PRIMARY KEY (`ServiceID`, `Brand`),
    CONSTRAINT `svID3` FOREIGN KEY (`ServiceID`) REFERENCES `service`(`ServiceID`)
    ON UPDATE CASCADE ON DELETE CASCADE
) ;
```

TRIGGER is used to ensure that the incoming *ServiceID* follows a pattern of *DV/R/S/C/M/B][0-9][0-9][0-9]*. If the incoming *ServiceID* does not follow that pattern, we will get an error ‘Wrong Format!!!’. Then it will check if the incoming *ServiceID* follows the pattern of *DV[M][0-9][0-9][0-9]*. If it does not, we will get an error ‘Wrong ServiceType!!!’.

```
DELIMITER $$
CREATE TRIGGER Souvenir_ServiceID_check2 BEFORE INSERT ON
`souvenir_brand`
FOR EACH ROW
BEGIN
IF (NEW.ServiceID REGEXP '^DV[RSCMB][0-9]{3}$' ) = 0 THEN
    SIGNAL SQLSTATE '12345'
    SET MESSAGE_TEXT = 'Wrong Format!!!';
END IF;
IF (SUBSTR(NEW.ServiceID,3,1) <> 'M') THEN
    SIGNAL SQLSTATE '12345'
```



```
SET MESSAGE_TEXT = 'Wrong ServiceType!!!!';
END IF;
END$$
DELIMITER ;
```

24. block (BranchID, BlockID, Length, Width, RentalPrice, Description, ServiceID, StoreName, Logo)

This table stores data about blocks of each branch. (*BranchID*, *BlockID*) is the PRIMARY KEY of the table. *BranchID* is a FOREIGN KEY reference *branch* (*BranchID*). *ServiceID* is a FOREIGN KEY reference *service* (*ServiceID*). CONSTRAINT *CheckBlockID* is set to ensure that values of incoming *BlockID* must be between 1 and 50.

```
CREATE TABLE `block` (
  `BranchID` varchar(6) NOT NULL,
  `BlockID` int unsigned NOT NULL DEFAULT '1',
  `Length` float NOT NULL,
  `Width` float NOT NULL,
  `RentalPrice` int unsigned NOT NULL,
  `Description` text,
  `ServiceID` varchar(10) DEFAULT NULL,
  `StoreName` varchar(50) DEFAULT NULL,
  `Logo` varchar(45) DEFAULT NULL,
  PRIMARY KEY (`BranchID`, `BlockID`),
  KEY `svID4_idx` (`ServiceID`),
  CONSTRAINT `FK_branch_block` FOREIGN KEY (`BranchID`) REFERENCES `branch` (`BranchID`) ON UPDATE CASCADE ON DELETE CASCADE,
  CONSTRAINT `svID4` FOREIGN KEY (`ServiceID`) REFERENCES `service` (`ServiceID`) ON UPDATE CASCADE ON DELETE CASCADE,
  CONSTRAINT `CheckBlockID` CHECK(`BlockID` BETWEEN 1 AND 50)
) ;
```

25. store_image(BranchID, BlockID, Image)

This table stores the data about the image of each store. (*BranchID*, *BlockID*, *Image*) is the PRIMARY KEY of the table. (*BranchID*, *BlockID*) is the FOREIGN KEY reference *block* (*BranchID*, *BlockID*). CONSTRAINT



LimitBlockID is set to ensure that values of incoming *BlockID* must be between 1 and 50.

```
CREATE TABLE `store_image` (
    `BranchID` varchar(6) NOT NULL,
    `BlockID` int unsigned NOT NULL DEFAULT '1',
    `Image` varchar(60) NOT NULL,
    PRIMARY KEY (`BranchID`, `BlockID`, `Image`),
    KEY `BlockID_idx` (`BlockID`),
    CONSTRAINT `BlockID2` FOREIGN KEY (`BranchID`, `BlockID`) REFERENCES
    `block` (`BranchID`, `BlockID`) ON UPDATE CASCADE ON DELETE CASCADE
);
```

26. active_time (BranchID, BlockID, openTime, closeTime)

This table stores data about the active time of each block at each branch. (*BranchID*, *BlockID*, *openTime*) is the PRIMARY KEY of the table. (*BranchID*, *BlockID*) is the FOREIGN KEY reference *block* (*BranchID*, *BlockID*).

```
CREATE TABLE `active_time` (
    `BranchID` varchar(6) NOT NULL,
    `BlockID` int unsigned NOT NULL DEFAULT '1',
    `openTime` TIME NOT NULL,
    `closeTime` TIME NOT NULL,
    PRIMARY KEY (`BranchID`, `BlockID`, `openTime`),
    KEY `BlockID_idx` (`BlockID`),
    CONSTRAINT `BlockID3` FOREIGN KEY (`BranchID`, `BlockID`) REFERENCES
    `block` (`BranchID`, `BlockID`) ON UPDATE CASCADE ON DELETE CASCADE
);
```

II. Insert data

1. branch (BranchID, Province, Address, PhoneNum, Email)

In this table, we add 4 entries.

Showing rows 0 - 3 (4 total, Query took 0.0007 seconds.)

```
SELECT * FROM `branch`
```

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

Show all | Number of rows: 25 Filter rows: Search this table Sort by key: None

Extra options

	BranchID	Province	Address	PhoneNum	Email
<input type="checkbox"/>	Edit Copy Delete CN1	Phan Thiet	504/58C Kinh Duong Vuong, Phu Nhuan ward, district...	0010000001	phanthiet@gmail.com
<input type="checkbox"/>	Edit Copy Delete CN2	Vung Tau	94 bis Ly Chieu Hoang, ward 10, district 6	0010000002	vungtau@gmail.com
<input type="checkbox"/>	Edit Copy Delete CN3	Nha Trang	235 Nguyen Van Cu, ward 4, district 5	0010000003	nhatrang@gmail.com
<input type="checkbox"/>	Edit Copy Delete CN4	Da Nang	268 Ly Thuong Kiet, ward 14, district 10	0010000004	danang@gmail.com

2. image_branch (BranchID, Image)

In this table, we add 4 entries.

Showing rows 0 - 3 (4 total, Query took 0.0002 seconds.)

```
SELECT * FROM `image_branch`
```

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

Show all | Number of rows: 25 Filter rows: Search this table Sort by key: None

Extra options

	BranchID	Image
<input type="checkbox"/>	Edit Copy Delete CN1	https://elitetour.com.vn/files/images/Blogs/victor...
<input type="checkbox"/>	Edit Copy Delete CN2	https://cdn.vntrip.vn/cam-nang/wp-content/uploads/...
<input type="checkbox"/>	Edit Copy Delete CN3	https://statics.vinpearl.com/styles/images_1600_x_...
<input type="checkbox"/>	Edit Copy Delete CN4	https://cdn3.ivivu.com/2017/10/InterContinental-Da...



3. zone (BranchID, ZoneName)

In this table, we add 4 entries.

Showing rows 0 - 15 (16 total, Query took 0.0003 seconds.)

```
SELECT * FROM `zone`
```

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

Show all | Number of rows: 25 Filter rows: Search this table Sort by key: None

Extra options

	BranchID	ZoneName
<input type="checkbox"/>	Edit Copy Delete	CN1 Zone A
<input type="checkbox"/>	Edit Copy Delete	CN1 Zone B
<input type="checkbox"/>	Edit Copy Delete	CN1 Zone C
<input type="checkbox"/>	Edit Copy Delete	CN1 Zone D

4. room_type (RoomID, RoomName, Area, NumGuest, Description)

In this table, we add 4 entries.

Showing rows 0 - 3 (4 total, Query took 0.0002 seconds.)

```
SELECT * FROM `room_type`
```

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

Show all | Number of rows: 25 Filter rows: Search this table Sort by key: None

Extra options

	RoomID	RoomName	Area	NumGuest	Description
<input type="checkbox"/>	Edit Copy Delete	1 Single	30	1	room for 1 guest
<input type="checkbox"/>	Edit Copy Delete	2 Double	40	2	room for 2 guests
<input type="checkbox"/>	Edit Copy Delete	3 Triple	50	3	room for 3 guests
<input type="checkbox"/>	Edit Copy Delete	4 Quad	60	4	room for 4 guests

5. bed_info (RoomID, Size, Quantity)

In this table, we add 5 entries.

Showing rows 0 - 4 (5 total, Query took 0.0002 seconds.)

```
SELECT * FROM `bed_info`
```

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

Show all | Number of rows: 25 Filter rows: Search this table Sort by key: None

Extra options

		RoomID	Size	Quantity
<input type="checkbox"/>		1	1.5	1
<input type="checkbox"/>		2	2.0	1
<input type="checkbox"/>		3	1.5	1
<input type="checkbox"/>		3	2.0	1
<input type="checkbox"/>		4	2.0	2

6. roomtype_branch (RoomID, BranchID, RentalPrice)

In this table, we add 16 entries

Showing rows 0 - 15 (16 total, Query took 0.0002 seconds.)

```
SELECT * FROM `roomtype_branch`
```

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

Show all | Number of rows: 25 Filter rows: Search this table Sort by key: None

Extra options

		RoomID	BranchID	RentalPrice
<input type="checkbox"/>		1	CN1	1000
<input type="checkbox"/>		1	CN2	1000
<input type="checkbox"/>		1	CN3	1000
<input type="checkbox"/>		1	CN4	1000
<input type="checkbox"/>		2	CN1	1000
<input type="checkbox"/>		2	CN2	1000
<input type="checkbox"/>		2	CN3	1000
<input type="checkbox"/>		2	CN4	1000
<input type="checkbox"/>		3	CN1	1000
<input type="checkbox"/>		3	CN2	1000
<input type="checkbox"/>		3	CN3	1000
<input type="checkbox"/>		3	CN4	1000
<input type="checkbox"/>		4	CN1	1000
<input type="checkbox"/>		4	CN2	1000
<input type="checkbox"/>		4	CN3	1000
<input type="checkbox"/>		4	CN4	1000

7. room(BranchID, RoomNumber, RoomID, ZoneName)

In this table, we add 64 entries

Showing rows 0 - 24 (64 total, Query took 0.0007 seconds.)

SELECT * FROM `room`

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

1 > >> | Show all | Number of rows: 25 | Filter rows: Search this table | Sort by key: None

Extra options

	BranchID	RoomNumber	RoomID	ZoneName
<input type="checkbox"/>	Edit Copy Delete	CN1	101	1 Zone A
<input type="checkbox"/>	Edit Copy Delete	CN1	102	2 Zone A
<input type="checkbox"/>	Edit Copy Delete	CN1	103	3 Zone A
<input type="checkbox"/>	Edit Copy Delete	CN1	104	4 Zone A

8. supply_type (SupplyID, SppName)

In this table, we add 8 entries

Showing rows 0 - 7 (8 total, Query took 0.0252 seconds.)

```
SELECT * FROM `supply_type`
```

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

Show all | Number of rows: 25 ▾ Filter rows: Search this table Sort by key: None ▾

Extra options

	← ↑ →	▼	SppID	SppName
<input type="checkbox"/>				VT0006 Cabinet
<input type="checkbox"/>				VT0001 Chair
<input type="checkbox"/>				VT0008 Clock
<input type="checkbox"/>				VT0002 Desk



9. room_type_supply_type (SppID, RoomID, Quantity)

In this table, we add 64 entries.

Showing rows 0 - 24 (32 total, Query took 0.0002 seconds.)

```
SELECT * FROM `room_type_supply_type`
```

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

1 > >> | Show all | Number of rows: 25 Filter rows: Search this table Sort by key: None

Extra options

		SppID	RoomID	Quantity
<input type="checkbox"/>	VT0001	1	1	1
<input type="checkbox"/>	VT0001	2	1	1
<input type="checkbox"/>	VT0001	3	1	1
<input type="checkbox"/>	VT0001	4	1	1

10. supplies (BranchID, SppID, SupplyIndex, Condition, RoomNumber)

In this table, we add 512 entries

Showing rows 0 - 24 (512 total, Query took 0.0256 seconds.)

```
SELECT * FROM `supplies`
```

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

1 > >> | Number of rows: 25 Filter rows: Search this table Sort by key: None

Extra options

		BranchID	SppID	SupplyIndex	Condition	RoomNumber
<input type="checkbox"/>	CN1		VT0001	1	new	101
<input type="checkbox"/>	CN1		VT0002	1	new	101
<input type="checkbox"/>	CN1		VT0003	1	new	101
<input type="checkbox"/>	CN1		VT0004	1	new	101



11. supplier (SupplierID, SupplierName, Mail, Address)

In this table, we add 4 entries

Showing rows 0 - 3 (4 total, Query took 0.0002 seconds.)

```
SELECT * FROM `supplier`
```

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

Show all | Number of rows: 25 | Filter rows: Search this table | Sort by key: None

Extra options

	SupplierID	SupplierName	Mail	Address
<input type="checkbox"/>	Edit Copy Delete NCC0001	FPT Shop	fptshop@gmail.com	644/4/23 3/2 street, ward 14, district 10
<input type="checkbox"/>	Edit Copy Delete NCC0002	The Gioi Di Dong	thegioididong@gmail.com	82/4 Lien Khu 16-18, ward 14, Binh Tan district
<input type="checkbox"/>	Edit Copy Delete NCC0003	Phong Vu	phongvu@gmail.com	123 Tran Hung Dao street, ward 10, district 1
<input type="checkbox"/>	Edit Copy Delete NCC0004	Gear VN	gearvn@gmail.com	1 Dong Khoi street, ward 1, district 1

12. supplying (SppID, BranchID, SupplierID)

In this table, we add 32 entries

Showing rows 0 - 24 (32 total, Query took 0.0304 seconds.)

```
SELECT * FROM `supplying`
```

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

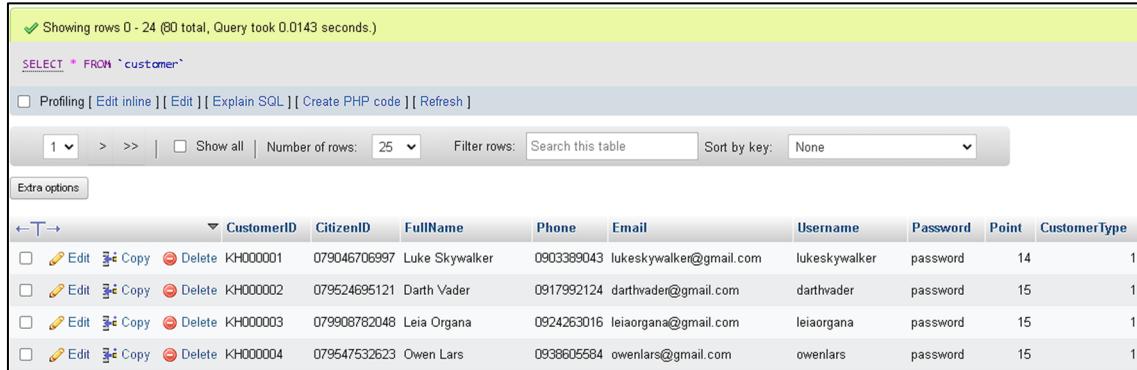
1 > >> | Show all | Number of rows: 25 | Filter rows: Search this table | Sort by key: None

Extra options

	SppID	BranchID	SupplierID
<input type="checkbox"/>	Edit Copy Delete VT0001	CN1	NCC0001
<input type="checkbox"/>	Edit Copy Delete VT0001	CN2	NCC0001
<input type="checkbox"/>	Edit Copy Delete VT0001	CN3	NCC0001
<input type="checkbox"/>	Edit Copy Delete VT0001	CN4	NCC0001

13. customer (CustomerID, CitizenID, FullName, Phone, Email, Username, Password, Point, CustomerType)

In this table, we add 80 entries.



Showing rows 0 - 24 (80 total, Query took 0.0143 seconds.)

SELECT * FROM `customer`

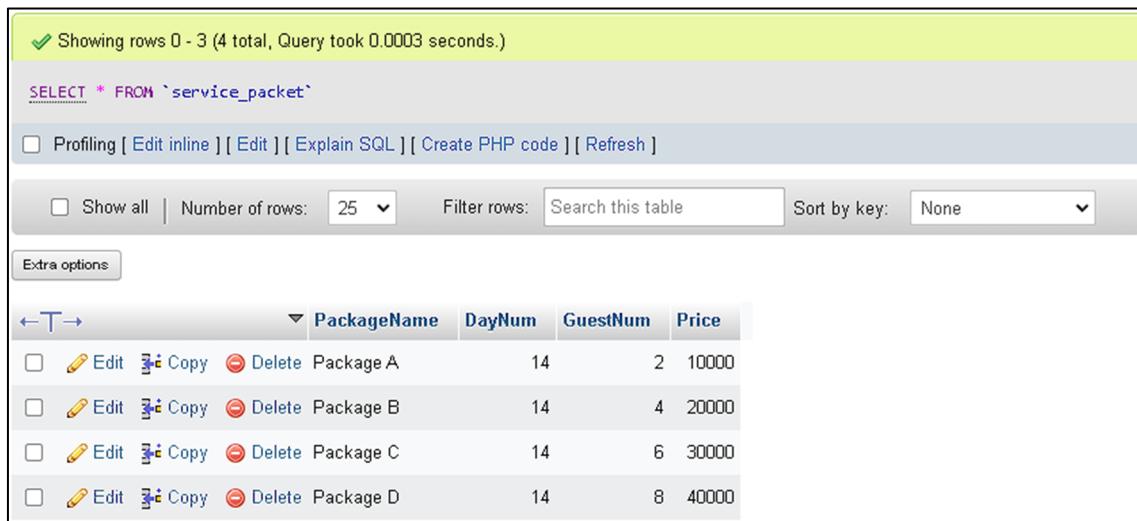
Profile [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

Show all | Number of rows: 25 | Filter rows: Search this table | Sort by key: None

	CustomerID	CitizenID	FullName	Phone	Email	Username	Password	Point	CustomerType
<input type="checkbox"/>	Edit Copy Delete KH000001	079046706997	Luke Skywalker	0903389043	lukeskywalker@gmail.com	lukeskywalker	password	14	1
<input type="checkbox"/>	Edit Copy Delete KH000002	079524695121	Darth Vader	0917992124	darthvader@gmail.com	darthvader	password	15	1
<input type="checkbox"/>	Edit Copy Delete KH000003	079908782048	Leia Organa	0924263016	leiaorgana@gmail.com	leiaorgana	password	15	1
<input type="checkbox"/>	Edit Copy Delete KH000004	079547532623	Owen Lars	0938605584	owenlars@gmail.com	owenlars	password	15	1

14. service_packet (PackageName, DayNum, GuestNum, Price)

In this table, we add 4 entries.



Showing rows 0 - 3 (4 total, Query took 0.0003 seconds.)

SELECT * FROM `service_packet`

Profile [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

Show all | Number of rows: 25 | Filter rows: Search this table | Sort by key: None

	PackageName	DayNum	GuestNum	Price
<input type="checkbox"/>	Edit Copy Delete Package A	14	2	10000
<input type="checkbox"/>	Edit Copy Delete Package B	14	4	20000
<input type="checkbox"/>	Edit Copy Delete Package C	14	6	30000
<input type="checkbox"/>	Edit Copy Delete Package D	14	8	40000



15. **packet_bill** (CustomerID, PackageName, PurchaseDate,
StartDate, TotalPay, RemainDay)

In this table, we add 8 entries

Showing rows 0 - 7 (8 total, Query took 0.0002 seconds.)

```
SELECT * FROM `packet_bill`
```

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

Show all | Number of rows: 25 ▾ Filter rows: Search this table Sort by key: None ▾

Extra options

	CustomerID	PackageName	PurchaseDate	StartDate	TotalPay	RemainDay
<input type="checkbox"/>	Edit Copy Delete KH000010	Package A	2021-01-01 00:00:00	2021-01-02 00:00:00	10000	14
<input type="checkbox"/>	Edit Copy Delete KH000020	Package A	2022-01-01 00:00:00	2022-01-02 00:00:00	10000	9
<input type="checkbox"/>	Edit Copy Delete KH000030	Package B	2021-01-01 00:00:00	2021-01-02 00:00:00	20000	14
<input type="checkbox"/>	Edit Copy Delete KH000040	Package B	2022-01-01 00:00:00	2022-01-02 00:00:00	20000	9

16. **booking** (BookingID, BookingDate, GuestNum, CheckIn,
CheckOut, Status, TotalPay, CustomerID, PackageName)

In this table, we add 1620 entries

Showing rows 0 - 24 (1620 total, Query took 0.0132 seconds.)

```
SELECT * FROM `booking`
```

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

1 > >> | Number of rows: 25 ▾ Filter rows: Search this table Sort by key: None ▾

Extra options

	BookingID	BookingDate	GuestNum	CheckIn	CheckOut	Status	TotalPay	CustomerID	PackageName
<input type="checkbox"/>	DP01012021000907	2021-01-01 16:51:24	8	2021-01-02 00:00:00	2021-01-03 00:00:00	1	1300	KH000061	NULL
<input type="checkbox"/>	DP01012021001502	2021-01-01 14:31:54	5	2021-01-02 00:00:00	2021-01-03 00:00:00	0	0	KH000057	NULL
<input type="checkbox"/>	DP01012022000310	2022-01-01 09:53:53	2	2022-01-02 00:00:00	2022-01-03 00:00:00	1	1300	KH000021	NULL
<input type="checkbox"/>	DP01012023000395	2023-01-01 19:43:07	5	2023-01-02 00:00:00	2023-01-03 00:00:00	1	1300	KH000027	NULL

17. booking_room (BookingID, BranchID, RoomNumber)

In this table, we add 1220 entries.

Showing rows 0 - 24 (1220 total, Query took 0.0002 seconds.)

```
SELECT * FROM `booking_room`
```

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

1 > >> | Number of rows: 25 Filter rows: Search this table Sort by key: None

Extra options

	BookingID	BranchID	RoomNumber
<input type="checkbox"/>	DP01012021000907	CN2	111
<input type="checkbox"/>	DP01012022000310	CN3	115
<input type="checkbox"/>	DP01012023000395	CN1	107
<input type="checkbox"/>	DP01012023000646	CN2	116

18. booking_bill (BillID, CheckIn, CheckOut, BookingID)

In this table, we add 1220 entries.

Showing rows 0 - 24 (1220 total, Query took 0.0285 seconds.)

```
SELECT * FROM `booking_bill`
```

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

1 > >> | Number of rows: 25 Filter rows: Search this table Sort by key: None

Extra options

	BillID	CheckIn	CheckOut	BookingID
<input type="checkbox"/>	HD12122022000001	05:40:44	10:37:06	DP23112022000001
<input type="checkbox"/>	HD12122022000002	22:38:49	20:50:56	DP15052023000002
<input type="checkbox"/>	HD12122022000003	12:19:39	21:03:56	DP11032021000003
<input type="checkbox"/>	HD12122022000004	06:55:34	06:19:19	DP24122023000004

19. enterprise (EnterpriseID, EnterpriseName)

In this table, we add 5 entries

Showing rows 0 - 4 (5 total, Query took 0.0001 seconds.)

```
SELECT * FROM `enterprise`
```

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

Show all | Number of rows: 25 ▾ Filter rows: Search this table Sort by key: None ▾

Extra options

	EnterpriseID	EnterpriseName
<input type="checkbox"/>	Edit Copy Delete DN0001	Michelin Star
<input type="checkbox"/>	Edit Copy Delete DN0002	Restful Spa
<input type="checkbox"/>	Edit Copy Delete DN0003	Circle K
<input type="checkbox"/>	Edit Copy Delete DN0004	Luxury Souvenir
<input type="checkbox"/>	Edit Copy Delete DN0005	Bartender

20. service (ServiceID, ServiceType, GuestNum, Style, EnterpriseID)

In this table, we add 20 entries.

Showing rows 0 - 19 (20 total, Query took 0.0946 seconds.)

```
SELECT * FROM `service`
```

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

Show all | Number of rows: 25 ▾ Filter rows: Search this table Sort by key: None ▾

Extra options

	ServiceID	ServiceType	GuestNum	Style	EnterpriseID
<input type="checkbox"/>	Edit Copy Delete DVB001	B	50	extra chill	DN0005
<input type="checkbox"/>	Edit Copy Delete DVB002	B	50	extra chill	DN0005
<input type="checkbox"/>	Edit Copy Delete DVB003	B	50	extra chill	DN0005
<input type="checkbox"/>	Edit Copy Delete DVB004	B	50	extra chill	DN0005

21. spa_service (ServiceID, ProvidedService)

In this table, we add 4 entries

The screenshot shows the MySQL Workbench interface with the following details:

- Query results: "Showing rows 0 - 3 (4 total, Query took 0.0002 seconds.)"
- SQL query: "SELECT * FROM `spa_service`"
- Table structure: "ServiceID" and "ProvidedService".
- Table data:

ServiceID	ProvidedService
DVS001	Face treatment
DVS002	Body treatment
DVS003	Nail treatment
DVS004	Hair treatment

22. souvenir_category (ServiceID, Category)

In this table, we add 4 entries.

The screenshot shows the MySQL Workbench interface with the following details:

- Query results: "Showing rows 0 - 3 (4 total, Query took 0.0002 seconds.)"
- SQL query: "SELECT * FROM `souvenir_category`"
- Table structure: "ServiceID" and "Category".
- Table data:

ServiceID	Category
DVM001	toy
DVM002	keychain
DVM003	bottle
DVM004	hat



23. souvenir_brand (ServiceID, Brand)

In this table, we add 4 entries

Showing rows 0 - 3 (4 total, Query took 0.0002 seconds.)

```
SELECT * FROM `souvenir_brand`
```

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

Show all | Number of rows: 25 Filter rows: Search this table Sort by key: None

Extra options

	ServiceID	Brand
<input type="checkbox"/>	DVM001	My Kingdom
<input type="checkbox"/>	DVM002	Kawaii
<input type="checkbox"/>	DVM003	Lock and Lock
<input type="checkbox"/>	DVM004	Non Son

24. block (BranchID, BlockID, Length, Width, RentalPrice, Description, ServiceID, StoreName, Logo)

In this table, we add 20 entries

Showing rows 0 - 19 (20 total, Query took 0.0131 seconds.)

```
SELECT * FROM `block`
```

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

Show all | Number of rows: 25 Filter rows: Search this table Sort by key: None

Extra options

	BranchID	BlockID	Length	Width	RentalPrice	Description	ServiceID	StoreName	Logo
<input type="checkbox"/>	CN1	1	20	4	10000	serve food	DVR001	Restaurant Branch 1	logoDVR001.png
<input type="checkbox"/>	CN1	2	20	4	10000	spa your day	DVS001	Spa Branch 1	logoDVS001.png
<input type="checkbox"/>	CN1	3	20	4	10000	sell convienience	DVC001	Convienience Store Branch 1	logoDVC001.png
<input type="checkbox"/>	CN1	4	20	4	10000	sell souvenir	DVM001	Souvenir Shop Branch 1	logoDVM001.png



25. store_image(BranchID, BlockID, Image)

In this table, we add 20 entries.

Showing rows 0 - 19 (20 total, Query took 0.0002 seconds.)

```
SELECT * FROM `store_image`
```

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

Show all | Number of rows: 25 ▾ Filter rows: Search this table Sort by key: None ▾

Extra options

	BranchID	BlockID	Image
<input type="checkbox"/>	Edit Copy Delete	CN1	1 storeimage1cn1.png
<input type="checkbox"/>	Edit Copy Delete	CN1	2 storeimage2cn1.png
<input type="checkbox"/>	Edit Copy Delete	CN1	3 storeimage3cn1.png
<input type="checkbox"/>	Edit Copy Delete	CN1	4 storeimage4cn1.png

26. active_time (BranchID, BlockID, openTime, closeTime)

In this table, we add 20 entries.

Showing rows 0 - 19 (20 total, Query took 0.0002 seconds.)

```
SELECT * FROM `active_time`
```

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

Show all | Number of rows: 25 ▾ Filter rows: Search this table Sort by key: None ▾

Extra options

	BranchID	BlockID	openTime	closeTime
<input type="checkbox"/>	Edit Copy Delete	CN1	07:00:00	21:00:00
<input type="checkbox"/>	Edit Copy Delete	CN1	07:00:00	21:00:00
<input type="checkbox"/>	Edit Copy Delete	CN1	07:00:00	21:00:00
<input type="checkbox"/>	Edit Copy Delete	CN1	07:00:00	21:00:00



PART 2: STORE PROCEDURE, FUNCTION, TRIGGER

I. Stored Procedure/Function

1. Write procedure/function GoiDichVu to show valid service packets of a customer.

Input: *CustomerID*

Output: <*Package_Name*, *Number_of_Guest*, *Start_Date*, *Expired_Date*,
Remaining_Day>

```
DELIMITER //
DROP PROCEDURE IF EXISTS GoiDichVu //
CREATE PROCEDURE GoiDichVu
(
    in customer_id varchar(10)
)
BEGIN

    SELECT packet_bill.PackageName AS Package_Name,
           service_packet.GuestNum AS Number_Of_Guest,
           packet_bill.StartDate AS Start_Date,
           (packet_bill.StartDate + INTERVAL 1 YEAR) AS Expire_Date,
           packet_bill.RemainDay AS Remaining_Day
    FROM packet_bill INNER JOIN service_packet
    ON packet_bill.PackageName = service_packet.PackageName
    WHERE packet_bill.CustomerID = customer_id
        HAVING Expire_Date > NOW();

END; //
DELIMITER ;
```

The GoiDichVu procedure will be written as above, the procedure has an IN parameter *customer_id* and returns a table which has 5 columns <*Package_Name*, *Number_of_Guest*, *Start_Date*, *Expire_Date*, *Remaining_Day*> as a result.

In the procedure, the *Expire_Date* is calculated by taking the *StartDate* and adding 1 Year to it. To retrieve the number of guests in a package, we have to perform an INNER JOIN of *packet_bill* and *service_packet*. Finally, we choose only the rows which satisfy the criteria: Package belongs to the given Customer and isn't expired.

The screenshot shows a MySQL Workbench interface. At the top, there is a code editor with the following SQL command:

```
654
655 • CALL GoiDichVu('KH000020');
656
```

Below the code editor is a result grid. The grid has the following columns: Package_Name, Number_Of_Guest, Start_Date, Expire_Date, and Remaining_Day. There is one row of data:

Package_Name	Number_Of_Guest	Start_Date	Expire_Date	Remaining_Day
Package A	2	2022-01-02 00:00:00	2023-01-02 00:00:00	14

2. Write procedure/function ThongKeLuotKhach to show statistics of a branch in a year.

Input: *Branch ID, Year*

Output: <*Month, Total_Guest*>

```
DELIMITER //
CREATE PROCEDURE ThongKeLuotKhach
(
    in branch_id varchar(6),
    in chosen_year int unsigned
)
BEGIN

    DROP TABLE IF EXISTS tmp;
    DROP TABLE IF EXISTS all_months;

    CREATE TABLE all_months (
        month_num INT
    );

    INSERT INTO all_months VALUES
    (1),(2),(3),(4),(5),(6),(7),(8),(9),(10),(11),(12);

    CREATE TEMPORARY TABLE tmp AS
        SELECT booking.*
```



```
FROM booking INNER JOIN booking_room
ON booking.BookingID = booking_room.BookingID
AND booking_room.BranchID = branch_id AND booking.status = 1 AND
YEAR(bookings.CheckIn) = chosen_year;

SELECT
all_months.month_num AS Month,
CASE
WHEN temp.total_guest is NULL THEN 0
ELSE temp.total_guest
END AS total_guest
FROM (
    SELECT DATE_FORMAT(CheckIn, '%m') AS review_month, SUM(GuestNum) AS
total_guest
    FROM tmp
    GROUP BY DATE_FORMAT(CheckIn, '%m')
) as temp
RIGHT JOIN all_months ON all_months.month_num=temp.review_month
ORDER BY all_months.month_num ASC;

DROP TABLE IF EXISTS all_months;
DROP TABLE IF EXISTS tmp;
END; //
DELIMITER ;
```

First, a table named “*all_month*” is created to store all the months of a year. Without this table, we cannot display the month which doesn’t have guests.

Next, we need to find “*booking*” which has the status of 1 (paid) and the *CheckIn* year is the same as the input year. However, we cannot identify which branch a booking receipt belongs to using only the “*booking*” table. Therefore, a temporary table “*tmp*” is created by using an INNER JOIN between “*booking*” and “*booking_room*” which has the same *BookingID*.

Finally, the result table is created by using the RIGHT JOIN between “*all_months*” and “*temp*” we just created, group by month of *CheckIn* date. The total number of guests in 1 month will be calculated using the aggregate function *SUM()*, the month which doesn’t have any guest will display as 0.

589 • CALL ThongKeLuotKhach('CN1', 2022);

<

Result Grid | Filter Rows: Export:

	Month	total_guest
▶	1	17
	2	8
	3	7
	4	15
	5	4
	6	7
	7	19
	8	0
	9	16
	10	9
	11	0
	12	9

II. Trigger

1. Trigger that update value

a. Update service packet bill

```
DELIMITER $$  
CREATE TRIGGER calculate_packet_bill BEFORE INSERT ON `packet_bill`  
FOR EACH ROW FOLLOWING check_packet_bill  
BEGIN  
    DECLARE customer_type INT ;  
    DECLARE total_day INT;
```



```
SET customer_type = (SELECT CustomerType FROM customer
                      WHERE customer.CustomerID = NEW.CustomerID);

SET total_day = (SELECT DayNum FROM service_packet
                  WHERE service_packet.PackageName = NEW.PackageName);

IF (customer_type = 3) THEN
    SET total_day = total_day + 1;
END IF;

IF (customer_type = 4) THEN
    SET total_day = total_day + 2;
END IF;

SET NEW.RemainDay = total_day;
END$$
DELIMITER ;
```

Based on the information given on Assignment 1, only VIP (*CustomerType*=3) and SUPER VIP (*CustomerType*=4) customers can earn the promo when buying a service packet. The promo will add 1 or 2 days to the service packet based on *CustomerType* and won't discount the price of the packet.

Therefore, trigger will be created BEFORE INSERT ON “*packet_bill*” and adds 1 day to the *RemainDay* of “*packet_bill*” if *CustomerType* is 3 or 2 days if *CustomerType* is 4. The price will stay the same.

b. Update total price of Booking

```
DELIMITER $$
CREATE TRIGGER calculate_booking_price BEFORE INSERT ON `booking_room`
FOR EACH ROW
BEGIN
    DECLARE customer_type INT ; DECLARE total_price INT; DECLARE
appliedPackage INT;
    DECLARE package_name VARCHAR(45); DECLARE duration INT; DECLARE
booking_date DATETIME;
    DECLARE customer_id VARCHAR(10);
    DROP TEMPORARY TABLE IF EXISTS booking_temp;
    CREATE TEMPORARY TABLE booking_temp AS( SELECT * FROM booking WHERE
booking.BookingID = NEW.BookingID);
    SET booking_date = (SELECT BookingDate FROM booking_temp); SET
customer_id = (SELECT CustomerID FROM booking_temp);
```



```
SET package_name = (SELECT PackageName from booking_temp);
SET duration = (SELECT TIMESTAMPDIFF(DAY, booking_temp.CheckIn,
booking_temp.CheckOut) FROM booking_temp);
SET customer_type = (SELECT CustomerType FROM customer INNER JOIN
booking_temp
ON customer.CustomerID = booking_temp.CustomerID);

SET total_price = (SELECT roomtype_branch.RentalPrice
FROM room INNER JOIN roomtype_branch
ON room.RoomID = roomtype_branch.RoomID and
room.BranchID = roomtype_branch.BranchID
WHERE room.RoomNumber = NEW.RoomNumber and
room.BranchID = NEW.BranchID) ; -- get room price
SET total_price = total_price * duration;
SET appliedPackage = 0;
IF NOT(package_name IS NULL OR package_name = '') THEN -- if
package is applied
    SELECT COUNT(*) INTO appliedPackage
    FROM
    (SELECT packet_bill.PackageName AS Package_Name,
    service_packet.GuestNum AS Number_Of_Guest,
    packet_bill.StartDate AS Start_Date,
    (packet_bill.StartDate + INTERVAL 1 YEAR) AS Expire_Date,
    packet_bill.RemainDay AS Remaining_Day
    FROM packet_bill INNER JOIN service_packet
    ON packet_bill.PackageName = service_packet.PackageName
    INNER JOIN booking_temp
    ON packet_bill.CustomerID = booking_temp.CustomerID
    AND booking_temp.BookingDate >= packet_bill.StartDate
    AND booking_temp.PackageName = packet_bill.PackageName
    AND booking_temp.GuestNum <=
service_packet.GuestNum -- check if number of guest is valid
    AND (packet_bill.RemainDay) >= TIMESTAMPDIFF(DAY,
booking_temp.CheckIn, booking_temp.CheckOut) -- check if remaining day of
a packet is valid
    HAVING Expire_Date > booking_date) AS tmp; -- check if packet is
expired
    IF (appliedPackage <> 0) THEN -- if exist a valid
package
        UPDATE booking SET TotalPay = 0 WHERE booking.BookingID =
NEW.BookingID ;
        UPDATE booking SET Status = 1 WHERE booking.BookingID =
NEW.BookingID;
```



```
UPDATE packet_bill
SET packet_bill.RemainDay = packet_bill.RemainDay - duration -- 
subtract using day from packet
WHERE CustomerID = customer_id AND PackageName =
package_name
AND (packet_bill.StartDate + INTERVAL 1 YEAR) > booking_date ;
ELSE
    SIGNAL SQLSTATE '12345'
    SET MESSAGE_TEXT = 'Cannot applied package';
END IF ;
ELSE          -- if package is not applied
IF(customer_type = 1) THEN
    UPDATE booking SET TotalPay = total_price WHERE booking.BookingID
= NEW.BookingID ;
    END IF;
    IF(customer_type = 2) THEN
        UPDATE booking SET TotalPay = total_price * 0.9 WHERE
booking.BookingID = NEW.BookingID ;
        END IF;
        IF(customer_type = 3) THEN
            UPDATE booking SET TotalPay = total_price * 0.9 WHERE
booking.BookingID = NEW.BookingID ;
            END IF;
            IF(customer_type = 4) THEN
                UPDATE booking SET TotalPay = total_price * 0.9 WHERE
booking.BookingID = NEW.BookingID ;
                END IF;
                END IF;
                DROP TEMPORARY TABLE IF EXISTS booking_temp;
END$$
DELIMITER ;
```

The trigger “*calculate_booking_price*” is created to update the *TotalPay* of “*booking*” when customer books room and applied package and promotion for *CustomerType* (*Loyal*, *VIP*, *SUPERVIP*)

The initial price for the booking will be calculated by taking *RentalPrice* of that room type at that branch multiplied by the number of days that customer stays.

This trigger has 2 cases:



- When a packet is applied in booking: In this situation, the trigger will check if the applied package is valid or not: *RemainDay* is enough to use the packet on this booking , the number of guests allowed on this packet is valid and the packet is not expired. If the applied packet is valid, status of this booking is set to 1 (paid) and the customer doesn't have to pay money for this booking so *TotalPay* is set to 0 then *RemainDay* of packet of this customer will be subtracted based on duration between *CheckIn* and *CheckOut* of this booking. If the applied packet is not valid, an error will be announced "Cannot applied package", the current booking room will not be inserted into the "*booking_room*" table and the customer will have to book another room or create another booking without using the package.
- When packet is not applied in booking: In this situation, *CustomerType* of the customer will be checked. If *CustomerType* is 2 (Loyal) , *TotalPay* will be discounted by 10%. If *CustomerType* is 3 (VIP) , *TotalPay* will be discounted by 15%. If *CustomerType* is 4 (SUPERVIP) , *TotalPay* will be discounted by 20%. Then the *TotalPay* of this booking will be updated accordingly.

c. Update customer's point when complete booking payment or when buy service packet

In this situation, we will create 2 trigger to cover all cases:

```
DELIMITER $$  
CREATE TRIGGER update_point_packet_bill AFTER INSERT ON `packet_bill`  
FOR EACH ROW  
BEGIN  
    DECLARE new_point INT;  
    SET new_point = floor(NEW.TotalPay/1000); -- calculate point  
  
    UPDATE customer  
    SET Point = Point + new_point -- update customer's point  
    WHERE CustomerID = NEW.CustomerID;  
END$$  
DELIMITER ;
```

In the first situation, the customer's point will be updated when buying a service packet.



The trigger “*update_point_packet_bill*” is triggered after a *packet_bill* is inserted, it will calculate points based on *TotalPay* of that *packet_bill* and update customer’s *Point* accordingly.

```
DELIMITER $$  
CREATE TRIGGER update_point_booking AFTER UPDATE ON `booking`  
FOR EACH ROW  
BEGIN  
    DECLARE new_point INT;  
    IF (NEW.Status <> OLD.Status AND NEW.Status = 1) THEN  
        SET new_point = floor(NEW.TotalPay/1000);  
        UPDATE customer  
        SET Point = Point + new_point  
        WHERE CustomerID = NEW.CustomerID;  
    END IF;  
END$$  
DELIMITER ;
```

In the second situation, customer’s point will be updated when customer pay for a booking after the booking room is success (the booking’s status is changed to paid)

The trigger “*update_point_booking*” is triggered after a booking updates its status to paid, it will check if the status is updated or not and if it is updated to paid (1), then it will calculate point and add to customer’s *Point*.

d. Update Customer’s type based on Customer’s point

```
DELIMITER $$  
CREATE TRIGGER update_customer_type BEFORE UPDATE ON `customer`  
FOR EACH ROW  
BEGIN  
    IF (NEW.Point <> OLD.Point) THEN  
        IF (NEW.Point < 50 ) THEN  
            SET NEW.CustomerType = 1;  
        ELSEIF (NEW.Point < 100) THEN  
            SET NEW.CustomerType = 2;  
        ELSEIF (NEW.Point < 1000) THEN  
            SET NEW.CustomerType = 3;  
        ELSE  
            SET NEW.CustomerType = 4;  
        END IF;  
    END IF;  
END$$
```



```
END$$
```

```
DELIMITER ;
```

In this situation, we do not need to check customer's *CustomerType* based on customer's *Point* when a customer is inserted because the *Point* is always 0, so we only need to check when customer's Point is updated.

The trigger “*update_customer_type*” will be triggered if customer's *Point* is updated and will update *CustomerType* if condition is satisfied.

2. Check constraint: Customer can not buy the same service packet if not expired

```
DELIMITER $$  
CREATE TRIGGER check_packet_bill BEFORE INSERT ON `packet_bill`  
FOR EACH ROW  
BEGIN  
    DECLARE current_package INT;  
    SELECT COUNT(*) INTO current_package  
    FROM  
        ( SELECT * FROM packet_bill  
        WHERE NEW.CustomerID = packet_bill.CustomerID  
            AND NEW.PackageName = packet_bill.PackageName  
            AND NEW.StartDate < (packet_bill.StartDate + INTERVAL 1 YEAR)  
        ) AS tmp;  
    IF (current_package > 0) THEN  
        SIGNAL SQLSTATE '12345'  
        SET MESSAGE_TEXT = 'Cannot buy the same package when package is not  
        expire';  
    END IF;  
END$$  
DELIMITER ;
```

The trigger “*check_packet_bill*” will be triggered before a *packet_bill* is inserted. It will check if there is any current package with the same *PackageName*, belongs to the same Customer and is not expired (its *ExpiredDate* is larger than *StartDate* of the newly added packet). If there is a current package still not expired, the customer can not insert the *packet_bill* and an error will be shown “Cannot buy the same package when package is not expired”.



PART 3: BUILD WEB APPLICATION

Github link: <https://github.com/namkha1032/databaseAssignment.git>

Demo video link:

https://drive.google.com/file/d/12OZ9qt0niS5NGghTFi79Q9_uW_xxaunN/view?usp=sharing

In this part, we decide to use HTML/CSS for Front-end and Django for Back-end to design a Web application

We connect our application database with MySQL database in settings.py as below

```
 DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'qlks',
        'USER': 'root',
        'HOST': 'localhost',
        'PORT': 3306,
        'PASSWORD': '',
    }
}
```

Also, we create models.py as our application's model to interact with our MySQL database, you can see more about this models.py in our code



```
base > models.py > User
 1  from django.db import models
 2  from django.contrib.auth.models import AbstractUser
 3
 4  # Create your models here.
 5
 6  class User(AbstractUser):
 7      username = models.CharField(max_length=200, null=True, unique = True)
 8      email = models.EmailField(null =True)
 9
10      USERNAME_FIELD = 'username'
11      REQUIRED_FIELDS = []
12
13
14
15  class Branch(models.Model):
16      branchid = models.CharField(db_column='BranchID', primary_key=True, max_length=6) # Field name made lowercase.
17      province = models.CharField(db_column='Province', max_length=50) # Field name made lowercase.
18      address = models.CharField(db_column='Address', unique=True, max_length=100) # Field name made lowercase.
19      phonenum = models.CharField(db_column='PhoneNum', unique=True, max_length=12) # Field name made lowercase.
20      email = models.CharField(db_column='Email', unique=True, max_length=64) # Field name made lowercase.
21
22      class Meta:
23          managed = False
24          db_table = 'branch'
25
26  class Customer(models.Model):
27      customerid = models.CharField(db_column='CustomerID', primary_key=True, max_length=10) # Field name made lowercase.
28      citizenid = models.CharField(db_column='CitizenID', unique=True, max_length=45) # Field name made lowercase.
29      fullname = models.CharField(db_column='FullName', max_length=45) # Field name made lowercase.
30      phone = models.CharField(db_column='Phone', unique=True, max_length=12) # Field name made lowercase.
31      email = models.CharField(db_column='Email', unique=True, max_length=45, blank=True, null=True) # Field name made lowercase.
32      username = models.CharField(db_column='Username', unique=True, max_length=45, blank=True, null=True) # Field name made lowercase.
33      password = models.CharField(db_column='Password', max_length=45, blank=True, null=True) # Field name made lowercase.
34      point = models.PositiveIntegerField(db_column='Point', blank=True, null=True) # Field name made lowercase.
35      customertype = models.PositiveIntegerField(db_column='CustomerType', blank=True, null=True) # Field name made lowercase.
36
37      class Meta:
38          managed = False
39          db_table = 'customer'
40
41
42  class Booking(models.Model):
43      bookingId = models.CharField(db_column='BookingID', primary_key=True, max_length=20) # Field name made lowercase.
44      bookingdate = models.DateTimeField(db_column='BookingDate') # Field name made lowercase.
45      guestnum = models.PositiveIntegerField(db_column='GuestNum') # Field name made lowercase.
46      checkin = models.DateTimeField(db_column='CheckIn') # Field name made lowercase.
47      checkout = models.DateTimeField(db_column='Checkout') # Field name made lowercase.
48      status = models.PositiveIntegerField(db_column='Status') # Field name made lowercase.
49      totalpay = models.PositiveIntegerField(db_column='TotalPay') # Field name made lowercase.
50      customerid = models.ForeignKey('Customer', models.DO_NOTHING, db_column='CustomerID') # Field name made lowercase.
51      packagename = models.ForeignKey('ServicePacket', models.DO_NOTHING, db_column='PackageName', blank=True, null=True) # Field name made lowercase.
52
53      class Meta:
```

1. Create user

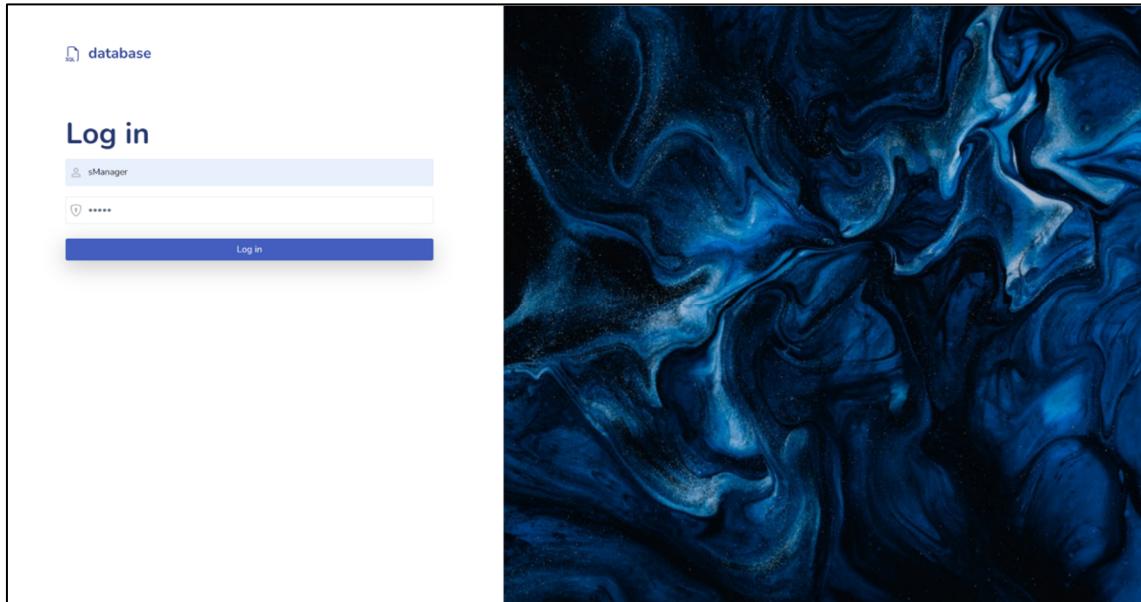
By typing into terminal : *python manage.py createsuperuser*

Then input the username and password as below, I have created a user with username sManager and this user has all access to the database.

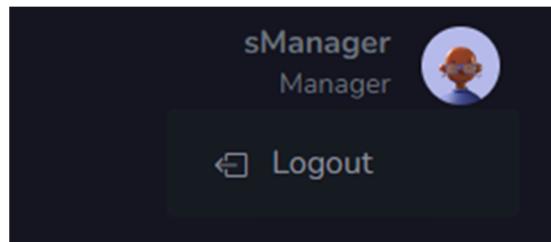
```
PS C:\Users\Admin\Desktop\DB PROJ\final\project> python manage.py createsuperuser
Username: sManager
Password:
Password (again):
This password is too short. It must contain at least 8 characters.
This password is too common.
This password is entirely numeric.
Bypass password validation and create user anyway? [y/N]: y
Superuser created successfully.
```



The picture below shows that the user sManager is created successfully in our database.



We login into the website using the sManager account that we have created above. When login successfully, we will be redirected to the homepage which shows the information of all customers. On the top-right, we have the avatar of the user, click on the avatar to see the Logout option.





2. Main functionality

a. View Customer's information

The screenshot shows a web application titled "Customers information". On the left, there is a sidebar with navigation links: "Customers", "Rooms", and "Statistics". The main content area displays a table of customer data with the following columns: Customer ID, Citizen ID, Full Name, Phone, Email, Username, Point, Rank, and Action. The table contains 10 entries, each with a "View" button in the Action column. At the bottom of the table, it says "Showing 1 to 10 of 80 entries". The browser address bar shows "127.0.0.1:1000".

Customer ID	Citizen ID	Full Name	Phone	Email	Username	Point	Rank	Action
KH000001	079046706997	Luke Skywalker	0903389043	lukeskywalker@gmail.com	lukeskywalker	72	2	<button>View</button>
KH000002	079524695121	Darth Vader	0912992124	darthvader@gmail.com	darthvader	78	2	<button>View</button>
KH000003	079908782048	Leia Organa	0924263016	leiaorgana@gmail.com	leiaorgana	74	2	<button>View</button>
KH000004	079547532623	Owen Lars	0938605584	owenlars@gmail.com	owenlars	78	2	<button>View</button>
KH000005	079746815551	Beru Whitesun Lars	0999034571	beruwhitesunlars@gmail.com	beruwhitesunlars	79	2	<button>View</button>
KH000006	079456497468	Biggs Darklighter	0999118912	biggsdarklighter@gmail.com	biggsdarklighter	72	2	<button>View</button>
KH000007	079650138125	Obi-Wan Kenobi	0951299887	obi-wankenobi@gmail.com	obi-wankenobi	71	2	<button>View</button>
KH000008	079954409937	Anakin Skywalker	0940637913	anakinskywalker@gmail.com	anakinskywalker	65	2	<button>View</button>
KH000009	079044640228	Wile E. Coyote	0979920623	wileecoyote@gmail.com	wileecoyote	77	2	<button>View</button>
KH000010	079733289101	Chewbacca	0938668133	chewbacca@gmail.com	chewbacca	72	2	<button>View</button>

When login successfully, we will be redirected to the homepage which shows the information of all customers. On this page, we can select the number of entries we want to appear.

b. Search Customer and View Booking's information

The screenshot shows the same web application as before, but with a search filter applied. In the top right corner, there is a search bar with the text "Luke". The table below shows only one entry matching the search term, with a "View" button in the Action column. At the bottom, it says "Showing 1 to 1 of 1 entries (filtered from 80 total entries)".

Customer ID	Citizen ID	Full Name	Phone	Email	Username	Point	Rank	Action
KH000001	079046706997	Luke Skywalker	0903389043	lukeskywalker@gmail.com	lukeskywalker	72	2	<button>View</button>

We can type into the search box to search for the Customer we are looking for, and in the column “Action”, there is a button “View”, click on this button to view customer's booking information.



The screenshot shows a table titled "Bookings of customer: Luke Skywalker". The table has columns: Booking ID, Booking date, Number of guests, Check in date, Check out date, Status, and Total pay. There are 8 entries listed:

Booking ID	Booking date	Number of guests	Check in date	Check out date	Status	Total pay
DP2612202300015	26-12-2022 10:51	7	27-12-2022	28-12-2022	1	6500
DP2412202300004	24-12-2023 12:03	7	26-12-2023	26-12-2023	1	7000
DP2201202300014	22-01-2023 09:26	5	23-01-2023	24-01-2023	1	4500
DP1805202300121	18-05-2023 14:18	5	19-05-2023	20-05-2023	2	4500
DP1710202300122	17-10-2023 19:41	2	18-10-2023	19-10-2023	3	1800
DP1505202300092	15-05-2023 13:56	2	16-05-2023	17-05-2023	1	2000
DP0110202300007	01-10-2023 02:33	6	08-10-2023	09-10-2023	1	6000
DP0112202300010	01-12-2023 11:45	7	02-12-2023	03-12-2023	1	7000

The picture above is the booking information of customer Luke Skywalker, it includes all the necessary information of Customer's booking. And only the booking with the CheckIn date in the future will be shown.

c. Add a new room type

We can click on the “Rooms” button on the left side bar to redirect to the “Add room type” page.

The screenshot shows a form titled "Add new room type". It has fields for Room name, Area, Number of guests, Description, Number of beds, and Number of supplies. The Number of supplies section lists items like Cabinet, Chair, Clock, Desk, Lamp, Mirror, Television, and Wardrobe, each with a quantity input field. At the bottom are "Submit" and "Reset" buttons.



This is the “Add new room type” form that we can use to create a new room type by adding necessary information.

The screenshot shows a web-based application interface for managing room types. On the left, there's a sidebar with navigation links: 'Customers', 'Rooms' (which is selected), and 'Statistics'. The main content area has a title 'Add new room type'. It contains several input fields:

- Room name:** VP
- Area:** 5
- Number of guests:** 5
- Description:** Nice room
- Number of beds:** 1.5m (1), 2.0m (0)
- Number of supplies:** Cabinet (1), Chair (0), Clock (1), Desk (0), Lamp (1), Mirror (0), Television (0), Wardrobe (0)

At the bottom right of the form are 'Submit' and 'Reset' buttons.

When submitted, it will return a message “Added successfully !!!” and we can see in our database that the new room type has been added.

The screenshot shows the same application interface after a successful submission. The main content area now displays a green success message: 'Added successfully !!!'. The rest of the form fields are identical to the previous screenshot, showing the same room details and supply counts.

	RoomID	RoomName	Area	NumGuest	Description
▶	1	Single	30	1	room for 1 guest
	2	Double	40	2	room for 2 guests
	3	Triple	50	3	room for 3 guests
	4	Quad	60	4	room for 4 guests
*	5	VIP	5	5	Nice room
	NULL	NULL	NULL	NULL	NULL

d. Customer's Statistics

```
@login_required(login_url='login')
def statistic(request):
    branches = Branch.objects.all()
    branch_id = 'CN1'
    year = 2022
    data_list = "["
    if request.method == 'POST':
        branch_id = request.POST.get('branch_id')
        year = request.POST.get('year')

    cursor = connection.cursor()
    cursor.callproc("ThongKeLuotKhach", [branch_id, year]) #call procedure
    results = cursor.fetchall()

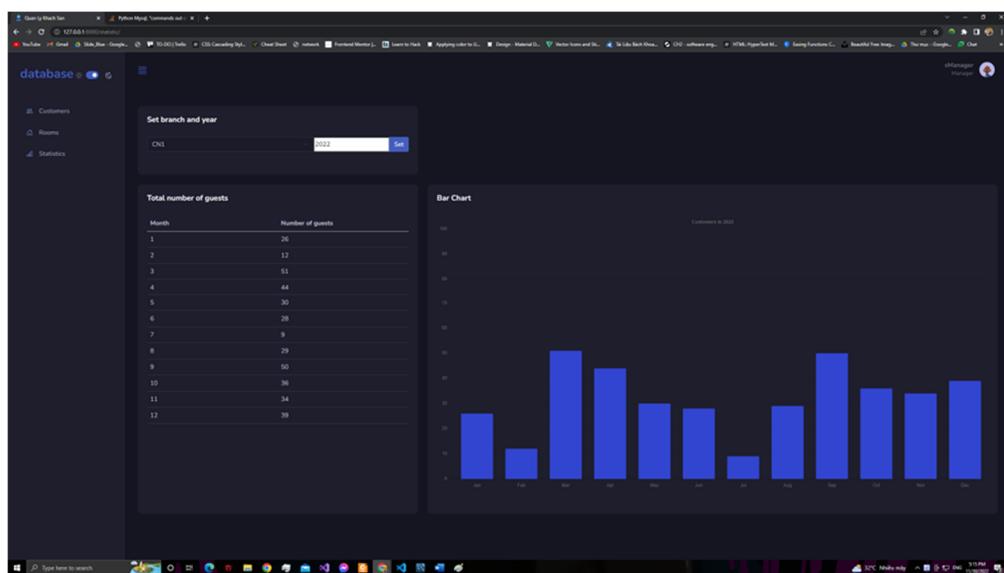
    for result in results:
        data_list += str(result[1]) + ','

    data_list = data_list[:-1];
    data_list += ']';
    cursor.close()

    context = {'branches': branches, 'results': results, 'year': year, 'branch_id': branch_id, 'data_list': data_list}
    return render(request, 'db-statistic.html', context)
```

To implement this functionality, we have to call the procedure ThongKeLuotKhach from our database, the highlighted code above is used to call the procedure without having to write any SQL code.

To go to the Statistics page, we click on “Statistic” button on the left side bar





Here, we can see the statistics illustrated in the bar chart and in the table as in the picture above.

In the “Set branch and year” section, we can choose which year or which branch we want to see the statistics and click “Set”, this will show us the following statistics of the branch and year we had chosen.

This is the statistics of branch CN3 in 2022

