VIETNAM NATIONAL UNIVERSITY, HO CHI MINH CITY
UNIVERSITY OF TECHNOLOGY
FACULTY OF COMPUTER SCIENCE AND ENGINEERING

DATABASE SYSTEM LAB (CO2014)

RESORT MANAGEMENT

Class CC03

Instructor: Nguyen Thanh Binh
Trinh Son Lam 1852502

HO CHI MINH CITY, NOVEMBER 2022

Contents

1 Part1: Create Database System

1.1 Create database table and constrains

1.1.1 Branch

```sql
CREATE TABLE Branch(
    number_id SERIAL NOT NULL,
    branch_id TEXT GENERATED ALWAYS AS ('CN' || number_id::text) STORED,
    province TEXT,
    address TEXT,
    phone_num VARCHAR(12),
    email TEXT,
    PRIMARY KEY(branch_id)
);
```

Figure 1: TABLE 1

1.1.2 BranchPicture

```sql
CREATE TABLE BranchPicture(
    branch_id TEXT ,
    link_image TEXT,
    PRIMARY KEY(branch_id,link_image),
    FOREIGN KEY(branch_id) REFERENCES Branch(branch_id)
);
```

Figure 2: TABLE 2

1.1.3 Zone

```sql
CREATE TABLE Zone(
    branch_id TEXT,
    name_zone TEXT,
    PRIMARY KEY(branch_id, name_zone),
    FOREIGN KEY(branch_id) REFERENCES Branch(branch_id)
);
```

Figure 3: TABLE 3

1.1.4 RoomType

```sql
CREATE TABLE RoomType(
    roomtype_id SERIAL NOT NULL,
    room_name TEXT,
    area_square TEXT,
    num_guests INT NOT NULL CHECK(num_guests > 0 AND num_guests <11),
    description TEXT,
    PRIMARY KEY(roomtype_id)
);
```

Figure 4: TABLE 4

1.1.5 BedInfo

```sql
CREATE TABLE BedInfo(
    roomtype_id SERIAL,
    bed_size NUMERIC(2,1),
    quantity INT NOT NULL DEFAULT 1 CHECK(quantity > 0 AND quantity < 11),
    PRIMARY KEY(roomtype_id,bed_size),
    FOREIGN KEY(roomtype_id) REFERENCES RoomType(roomtype_id)
);
```

Figure 5: TABLE 5

1.1.6 BranchHavRoomType

```sql
CREATE TABLE BranchHavRoomType(
    roomtype_id SERIAL,
    branch_id TEXT,
    price INT NOT NULL,
    PRIMARY KEY(roomtype_id, branch_id),
    FOREIGN KEY(roomtype_id) REFERENCES RoomType(roomtype_id),
    FOREIGN KEY(branch_id) REFERENCES Branch(branch_id)
);
```

Figure 6: TABLE 6

1.1.7 Room

```sql
CREATE TABLE Room(
    branch_id TEXT,
    num_room VARCHAR(3),
    roomtype_id SERIAL,
    name_zone TEXT,
    PRIMARY KEY(branch_id,num_room),
    FOREIGN KEY (branch_id) REFERENCES Branch(branch_id),
    FOREIGN KEY(branch_id,name_zone) REFERENCES Zone(branch_id,name_zone),
    FOREIGN KEY(roomtype_id) REFERENCES RoomType(roomtype_id)
);
```

Figure 7: TABLE 7

1.1.8 SpicimenType

```sql
CREATE TABLE SpicimenType(
    spicimen_id VARCHAR(6) CHECK(spicimen_id ~ 'VT[0-9]{4}'),
    spicimen_name TEXT,
    PRIMARY KEY(spicimen_id)
);
```

Figure 8: TABLE 8

1.1.9 SpicTypeInRoomType

```sql
CREATE TABLE SpicTypeInRoomType(
    spicimen_id VARCHAR(6),
    roomtype_id SERIAL,
    quantity_spic INT NOT NULL DEFAULT 1 CHECK(quantity_spic > 0 AND quantity_spic < 21),
    PRIMARY KEY(spicimen_id,roomtype_id),
    FOREIGN KEY(spicimen_id) REFERENCES SpicimenType(spicimen_id),
    FOREIGN KEY(roomtype_id) REFERENCES RoomType(roomtype_id)
);
```

Figure 9: TABLE 9

1.1.10 Material

```sql
CREATE TABLE Material(
    branch_id TEXT,
    spicimen_id VARCHAR(6),
    num_material INT CHECK(num_material > 0),
    status TEXT,
    num_room VARCHAR(3),
    PRIMARY KEY(branch_id,spicimen_id,num_material),
    FOREIGN KEY(branch_id) REFERENCES Branch(branch_id),
    FOREIGN KEY(spicimen_id) REFERENCES SpicimenType(spicimen_id),
    FOREIGN KEY(branch_id,num_room) REFERENCES Room(branch_id,num_room)
);
```

Figure 10: TABLE 10

1.1.11 Provider

```sql
CREATE TABLE Provider(
    provider_id VARCHAR(7) CHECK(provider_id ~ 'NCC[0-9]{4}'),
    provider_name TEXT,
    provider_email TEXT,
    provider_address TEXT,
    PRIMARY KEY(provider_id)
);
```

Figure 11: TABLE 11

1.1.12 ProviderMaterial

```sql
CREATE TABLE ProviderMaterial(
    provider_id VARCHAR(7),
    spicimen_id VARCHAR(6),
    branch_id TEXT,
    PRIMARY KEY(spicimen_id,branch_id),
    FOREIGN KEY(provider_id) REFERENCES Provider(provider_id),
    FOREIGN KEY(spicimen_id) REFERENCES SpicimenType(spicimen_id),
    FOREIGN KEY(branch_id) REFERENCES Branch(branch_id)
);
```

Figure 12: TABLE 12

1.1.13 Customer

```sql
CREATE TABLE Customer(
    customer_id VARCHAR(8) CHECK(customer_id ~ 'KH[0-9]{6}'),
    cccd_cmnd VARCHAR(12) NOT NULL UNIQUE,
    customer_name TEXT,
    customer_phone TEXT UNIQUE,
    customer_email TEXT UNIQUE,
    username TEXT UNIQUE,
    cus_password TEXT,
    points INT NOT NULL DEFAULT 0 CHECK(points >= 0),
    customer_type INT NOT NULL DEFAULT 1 CHECK(customer_type > 0 AND customer_type < 5),
    PRIMARY KEY(customer_id)
);
```

Figure 13: TABLE 13

1.1.14 ServicePackage

```sql
CREATE TABLE ServicePackage(
    package_name TEXT,
    num_days INT NOT NULL CHECK(num_days > 0 AND num_days < 101),
    num_guests INT NOT NULL CHECK(num_guests > 0 AND num_guests < 11),
    price INT NOT NULL,
    PRIMARY KEY(package_name)
);
```

Figure 14: TABLE 14

1.1.15 BillService

```sql
CREATE TABLE BillService(
    customer_id VARCHAR(8),
    package_name TEXT,
    time_buying TIMESTAMP,
    time_starting DATE CHECK(time_starting > time_buying),
    total_price INT ,
    PRIMARY KEY(customer_id,package_name,time_buying),
    FOREIGN KEY(customer_id) REFERENCES Customer(customer_id),
    FOREIGN KEY(package_name) REFERENCES ServicePackage(package_name)
);
```

Figure 15: TABLE 15

1.1.16 TypeForDiscount

```sql
CREATE TABLE TypeForDiscount(
    customer_type INT,
    discount REAL,
    point_type INT,
    PRIMARY KEY(customer_type)
);
```

Figure 16: TABLE 16

1.1.17 BillBooking

```sql
CREATE OR REPLACE FUNCTION Generate_Room_Booking_ID(timestamp,INT) RETURNS text AS
$$
    SELECT (to_char($1, 'DD') || to_char($1, 'MM') || to_char($1, 'YYYY'))||LPAD($2::text, 6, '0');
$$ LANGUAGE sql IMMUTABLE;
CREATE TABLE BillBooking(
    book_id SERIAL NOT NULL,
    booking_id VARCHAR(16) GENERATED ALWAYS AS ('DP'|| Generate_Room_Booking_ID(time_booking,book_id)) STORED,
    time_booking TIMESTAMP,
    time_checkin DATE CHECK(time_checkin > time_booking),
    time_checkout DATE CHECK(time_checkout > time_checkin),
    bill_status INT CHECK(bill_status >= 0 AND bill_status <= 3),
    bill_price INT NOT NULL DEFAULT 0,
    customer_id VARCHAR(8),
    package_name TEXT,
    PRIMARY KEY(booking_id),
    FOREIGN KEY(customer_id) REFERENCES Customer(customer_id),
    FOREIGN KEY(package_name) REFERENCES ServicePackage(package_name)
);
```

Figure 17: TABLE 17

1.1.18 RentingRoom

```sql
CREATE TABLE RentingRoom(
    booking_id VARCHAR(16),
    branch_id TEXT,
    num_room VARCHAR(3),
    PRIMARY KEY(booking_id,branch_id,num_room),
    FOREIGN KEY(booking_id) REFERENCES BillBooking(booking_id),
    FOREIGN KEY(branch_id,num_room) REFERENCES Room(branch_id,num_room)
);
```

Figure 18: TABLE 18

1.1.19 PaymentBill

```sql
CREATE TABLE PaymentBill(
    bill_num SERIAL NOT NULL,
    bill_id VARCHAR(16) GENERATED ALWAYS AS ('HD'|| Generate_Room_Booking_ID(timer_checkout,bill_num)) STORED,
    timer_checkin TIMESTAMP,
    timer_checkout TIMESTAMP,
    booking_id VARCHAR(16),
    PRIMARY KEY(bill_id),
    FOREIGN KEY(booking_id) REFERENCES BillBooking(booking_id)
);
```

Figure 19: TABLE 19

1.1.20 Company

```sql
CREATE TABLE Company(
    company_id VARCHAR(6) CHECK(company_id ~ 'DN[0-9]{4}'),
    company_name TEXT,
    PRIMARY KEY(company_id)
);
```

Figure 20: TABLE 20

1.1.21 Service

```sql
CREATE TABLE Service(
    service_id VARCHAR(6) CHECK(service_id ~ 'DV[RSCMB][0-9]{3}'),
    service_type VARCHAR(1) CHECK(SUBSTRING(service_id,3,1) = service_type),
    num_guests INT,
    stype TEXT,
    company_id VARCHAR(6),
    PRIMARY KEY(service_id),
    FOREIGN KEY(company_id) REFERENCES Company(company_id)
);
```

Figure 21: TABLE 21

1.1.22 ServiceSpa

```sql
CREATE TABLE ServiceSpa(
    service_id VARCHAR(6) CHECK(SUBSTRING(service_id,1,3) = 'DVS'),
    spa_type TEXT,
    PRIMARY KEY(service_id,spa_type),
    FOREIGN KEY(service_id) REFERENCES Service(service_id)
);
```

Figure 22: TABLE 22

1.1.23 TpeSouvenir

```sql
CREATE TABLE TypeSouvenir(
    service_id VARCHAR(6) CHECK(SUBSTRING(service_id,1,3) = 'DVM'),
    souv_type TEXT,
    PRIMARY KEY(service_id,souv_type),
    FOREIGN KEY(service_id) REFERENCES Service(service_id)
);
```

Figure 23: TABLE 23

1.1.24 BrandSouvenir

```sql
CREATE TABLE BrandSouvenir(
    service_id VARCHAR(6) CHECK(SUBSTRING(service_id,1,3) = 'DVM'),
    souv_brand TEXT,
    PRIMARY KEY(service_id,souv_brand),
    FOREIGN KEY(service_id) REFERENCES Service(service_id)
);
```

Figure 24: TABLE 24

1.1.25 AreaPlace

```sql
CREATE TABLE AreaPlace(
    branch_id TEXT,
    stt_num INT NOT NULL DEFAULT 1 CHECK( stt_num >= 1 AND stt_num <= 50),
    area_length TEXT,
    area_width TEXT,
    area_price INT NOT NULL,
    description TEXT,
    service_id VARCHAR(6),
    name_shop TEXT,
    logo_shop TEXT,
    PRIMARY KEY(branch_id,stt_num),
    FOREIGN KEY(branch_id) REFERENCES Branch(branch_id),
    FOREIGN KEY(service_id) REFERENCES Service(service_id)
);
```

Figure 25: TABLE 25

1.1.26 ShopPicture

```sql
CREATE TABLE ShopPicture(
    branch_id TEXT,
    stt_num INT,
    picture TEXT,
    PRIMARY KEY(branch_id,stt_num,picture),
    FOREIGN KEY(branch_id,stt_num) REFERENCES AreaPlace(branch_id,stt_num)
);
```

Figure 26: TABLE 26

1.1.27 TimelineShop

```sql
CREATE TABLE TimelineShop(
    branch_id TEXT,
    stt_num INT,
    time_starting TIME,
    time_closing TIME,
    PRIMARY KEY(branch_id,stt_num,time_starting),
    FOREIGN KEY(branch_id,stt_num) REFERENCES AreaPlace(branch_id,stt_num)
);
```

Figure 27: TABLE 27

Then when creating all of table we have a set of tables there:

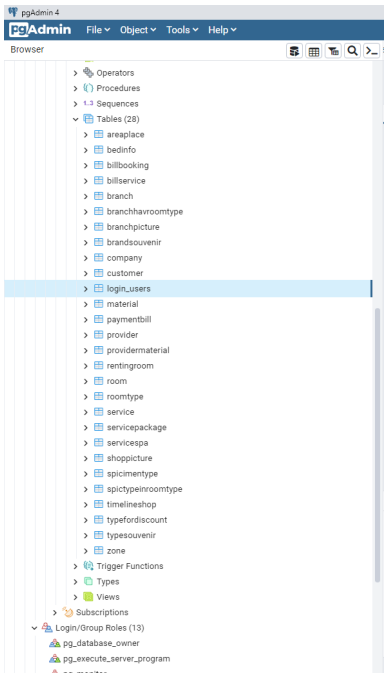


Figure 28: TABLES

1.2 Insert data to tables

A data file have already create with name "insert_data" for adding them to database system. Then we have some data that provide for querying in the DBMS. There are some pictures when you query data

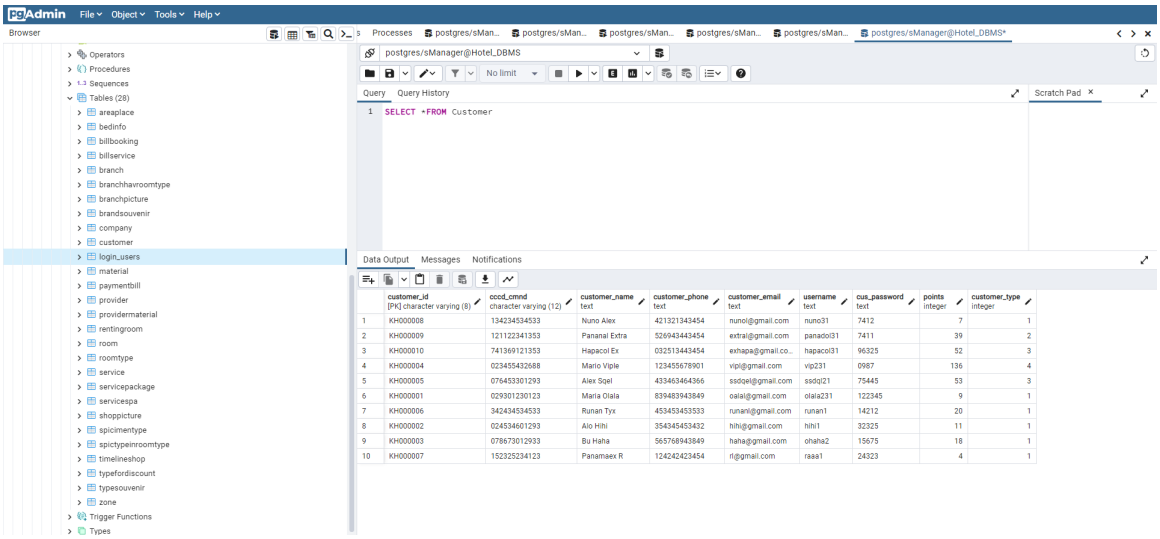


Figure 29: Query from Customer table

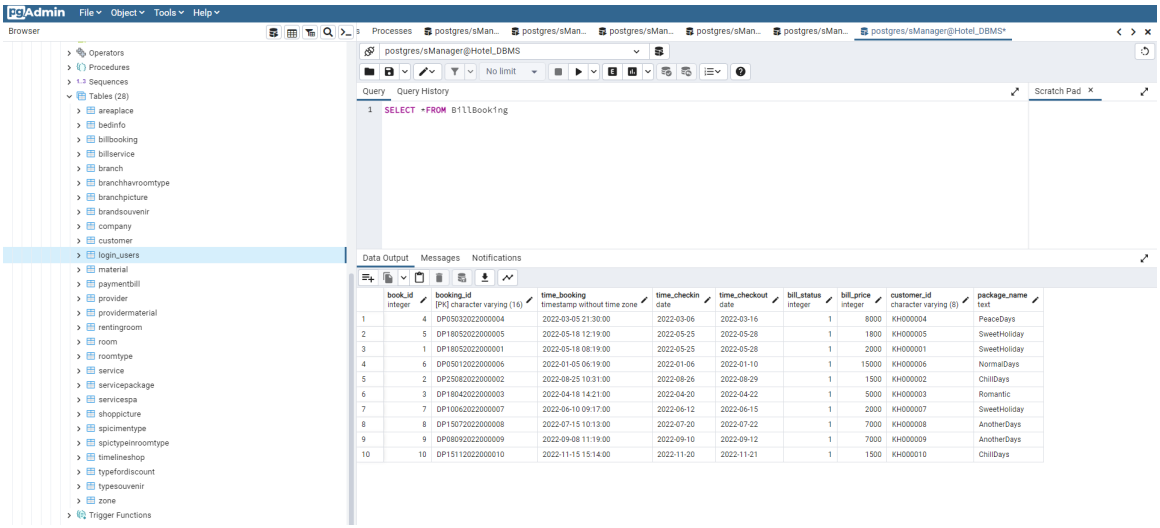


Figure 30: Query from BillBooking

2 Part2: Store Procedure/Function and Trigger

2.1 Store Procedure/Function

2.1.1 Procedure/Function for ServicePackage(GoiDichVu)

```
CREATE OR REPLACE FUNCTION AddDate(_days INT, _datetime DATE) RETURNS DATE AS
$$
  SELECT _datetime + _days;
```

```
$$ LANGUAGE sql IMMUTABLE;

CREATE OR REPLACE FUNCTION AddDateInt(_days INT, _datetime INT) RETURNS INT AS
$$
  SELECT _datetime + _days;
$$ LANGUAGE sql IMMUTABLE;

CREATE TYPE mytype AS(
    package_name TEXT,
    num_guests INT,
    time_starting DATE,
    time_expiring DATE,
    remaining_days INT
);
CREATE OR REPLACE FUNCTION PackageInfo(_customer_id TEXT)
  RETURNS mytype
AS
$$
DECLARE tableshow mytype;
DECLARE tempvar INT;
DECLARE tempvar1 DATE;
DECLARE tempvar2 DATE;
BEGIN
    tableshow.package_name = BillService.package_name FROM BillService WHERE
    BillService.customer_id = _customer_id;
    tableshow.num_guests = ServicePackage.num_guests FROM ServicePackage WHERE
    ServicePackage.package_name = tableshow.package_name;
    tableshow.time_starting = BillService.time_starting FROM BillService WHERE
    BillService.customer_id = _customer_id ;
    tableshow.time_expiring = (tableshow.time_starting+365);
    tempvar= ServicePackage.num_days FROM ServicePackage WHERE ServicePackage.
    package_name = tableshow.package_name;
    tempvar1 = BillBooking.time_checkout FROM BillBooking WHERE BillBooking.
    customer_id = _customer_id;
    tempvar2 = BillBooking.time_checkin FROM BillBooking WHERE BillBooking.
    customer_id = _customer_id;
    IF(AddDate(365,tableshow.time_starting) - CURRENT_DATE -
       AddDateInt(tempvar ,- (tempvar1 - tempvar2))) > 0 THEN
       tableshow.remaining_days = AddDateInt(tempvar, - (tempvar1 - tempvar2)) ;
    ELSE
      tableshow.remaining_days = AddDate(365,tableshow.time_starting) -
    CURRENT_DATE ;
    END IF;
    RETURN tableshow;
END;
$$ LANGUAGE 'plpgsql';
```

To create function to view package service we need to have some information as follows:

- We need to have *package_name* from BillService(DonDichVu) where *customer_id* of BillService is the same with *customer_id* from input user.
- With *num_guests* we query from ServicePackage(GoiDichVu) where *package_name* of ServicePackage is the same one from BillService.
- With *time_starting* we query from BillService where *customer_id* of BillService is the same with *customer_id* from input user.
- With *time_expiring* we calculate with *time_starting* plus 365 days.
- With *remaining_days* we use formula with if the remaining days from *time_expiring* minus *current_time* minus usage days in package that is greater than 0, the remaining days is equal remaining usage days in package, otherwise is 0.

2.1.2 Procedure/Function for StatisticGuests

```
CREATE OR REPLACE FUNCTION TextDate(_year TEXT) RETURNS DATE AS
$$
```

```
  SELECT TO_DATE(_year::TEXT,'YYYY');
$$ LANGUAGE sql IMMUTABLE;

CREATE OR REPLACE FUNCTION MonthDate(_month TIMESTAMP) RETURNS TEXT AS
$$
  SELECT TO_CHAR(_month,'MM');
$$ LANGUAGE sql IMMUTABLE;

CREATE OR REPLACE FUNCTION StatisticGuests(_branch_id TEXT,_year TEXT)
  RETURNS TABLE(
    months TEXT,
    sum_num_guests BIGINT
  )
AS
$$
BEGIN
  RETURN QUERY
  SELECT
    MonthDate(BillBooking.time_booking),
    COUNT(BillBooking.booking_id)
  FROM
    RentingRoom JOIN PaymentBill ON RentingRoom.booking_id = PaymentBill.booking_id
    JOIN BillBooking ON PaymentBill.booking_id = BillBooking.booking_id
  WHERE
    RentingRoom.branch_id = _branch_id AND DATE_TRUNC('year',BillBooKing.
    time_booking) = TextDate(_year)
  GROUP BY MonthDate(BillBooking.time_booking);
END;
$$ LANGUAGE 'plpgsql';
```

To create function to view statistical guests we need to have some information as follows: We chose *month* and *sum_num_guests* from joining some tables as RentingRoom(PhongThue) with PaymentBill(HoaDonThanhToan) where *booking_id* is the same as well as with joining BillBooking where the condition is *branch_id* from RentingRoom and the year are the same with input from user by grouping follow month in a year.

2.2 Triggers

2.2.1 Trigger to update values

- UpdateSumServiceBill(TongTienGoiDichVu), we find *discount* from joining BillServive(DonDichVu) to Customer through *customer_id* and joining with TypeForDiscount through *customer_type* where BillService has a new *custoemer_id* and then we use calculation as picture.

```
    CREATE OR REPLACE FUNCTION UpdateSumServiceBill()
RETURNS TRIGGER
AS
$updateServiceBill$
BEGIN
  UPDATE BillService
  SET total_price = ServicePackage.price - ServicePackage.price * (SELECT
    discount FROM BillService JOIN Customer
                        ON BillService.customer_id = Customer.customer_id
                        JOIN TypeForDiscount ON Customer.customer_type =
    TypeForDiscount.customer_type
                        WHERE BillService.customer_id = NEW.customer_id LIMIT
    1)
  FROM ServicePackage
  WHERE
    ServicePackage.package_name = BillService.package_name AND BillService.
    customer_id = NEW.customer_id;
  RETURN NULL;
END;
$updateServiceBill$ LANGUAGE plpgsql;

CREATE OR REPLACE TRIGGER UpdateTotalServiceBill
  AFTER INSERT
  ON BillService
  FOR EACH ROW
```

```
  EXECUTE FUNCTION UpdateSumServiceBill();

```

- UpdateSumPayment(TongTienDonDatPhong), we need some information like *discount* from RentingRoom(PhongThue) to BillBooking(DonDatPhong) through *booking_id* and joining with Customer through *customer_id* where BillBooking has a new *booking_id* and then we calculate as formula as picture.

```
CREATE OR REPLACE FUNCTION UpdateSumPayment()
RETURNS TRIGGER
AS
$UpdateBillBooking$
BEGIN
  UPDATE BillBooking
  SET bill_price = BillBooking.bill_price - BillBooking.bill_price *(SELECT discount FROM RentingRoom
                JOIN BillBooking ON RentingRoom.booking_id = BillBooking.booking_id
                JOIN Customer ON Customer.customer_id = BillBooking.customer_id
                JOIN TypeForDiscount ON TypeForDiscount.customer_type = Customer.customer_type
                WHERE NEW.booking_id = BillBooking.booking_id LIMIT 1)
  FROM RentingRoom
  WHERE NEW.booking_id = BillBooking.booking_id;
  RETURN NULL;
END;
$UpdateBillBooking$ LANGUAGE plpgsql;

CREATE OR REPLACE TRIGGER UpdateSumPaymentBill
AFTER INSERT
ON RentingRoom
FOR EACH ROW
EXECUTE FUNCTION UpdateSumPayment();

```

- UpdatePointCustomer(DiemCuaKhachHang), we need to find *bill_price* from joining BillBooking to Customer through *customer_id* where *bill_status* = 1 and BillBooking has a new *booking_id* and then divide 1000 and next take the result is plus initial point.

```
    CREATE OR REPLACE FUNCTION UpdatePointCustomer()
RETURNS TRIGGER
AS
$UpdatCustomer$
BEGIN
  UPDATE Customer
  SET points = Customer.points + (SELECT bill_price FROM BillBooking JOIN Customer ON
                    BillBooking.customer_id = Customer.customer_id
                   WHERE NEW.bill_status = 1 AND NEW.booking_id = BillBooking.booking_id LIMIT 1)/1000
  FROM BillBooking
  WHERE NEW.customer_id = Customer.customer_id AND NEW.booking_id = BillBooking.booking_id AND NEW.bill_status = 1;
  RETURN NULL;
END;
$UpdatCustomer$ LANGUAGE plpgsql;

CREATE OR REPLACE TRIGGER UpdatePointClient
AFTER UPDATE
ON BillBooking
FOR EACH ROW
EXECUTE FUNCTION UpdatePointCustomer();

```

- UpdateTypeCustomer(LoaiKhachHang), we need to calculate the point of customer and then using it to comparison with some rules to update type of customers

```
CREATE OR REPLACE FUNCTION UpdateTypeCustomer()
RETURNS TRIGGER
AS
$UpdateCustomer$
BEGIN
  UPDATE Customer
  SET customer_type = (SELECT customer_type FROM TypeForDiscount WHERE NEW.
    points >= point_type
            ORDER BY customer_type DESC LIMIT 1)
  WHERE NEW.customer_id = Customer.customer_id;
  RETURN NULL;
END;
$UpdateCustomer$ LANGUAGE plpgsql;

CREATE OR REPLACE TRIGGER UpdateTypeClient
AFTER UPDATE
ON Customer
FOR EACH ROW
WHEN(NEW.points <> OLD.points)
EXECUTE FUNCTION UpdateTypeCustomer();


```

3 Part3: Build an application

A link video demo(backup in drive):
https://drive.google.com/drive/folders/1cdX3ZRnXvM25cO2UBAvyqK9pslph4xVD?usp=sharing

3.1 Create user

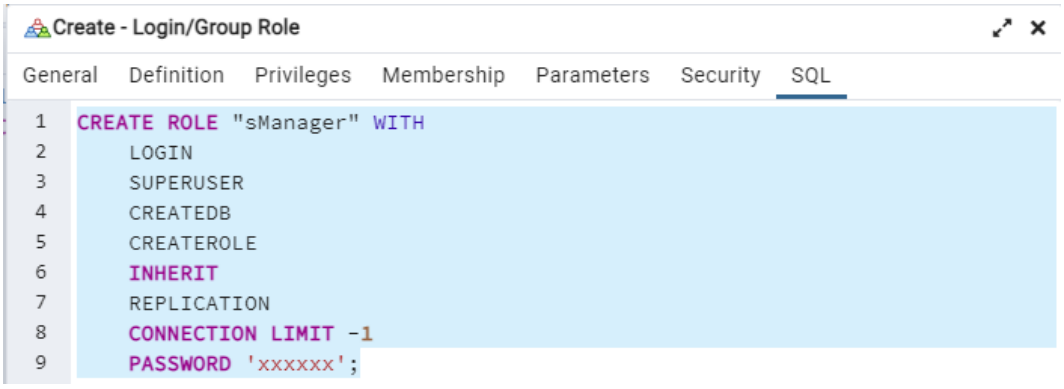


Figure 31: Create DBA user

3.2 Some features

3.2.1 Login/Logout

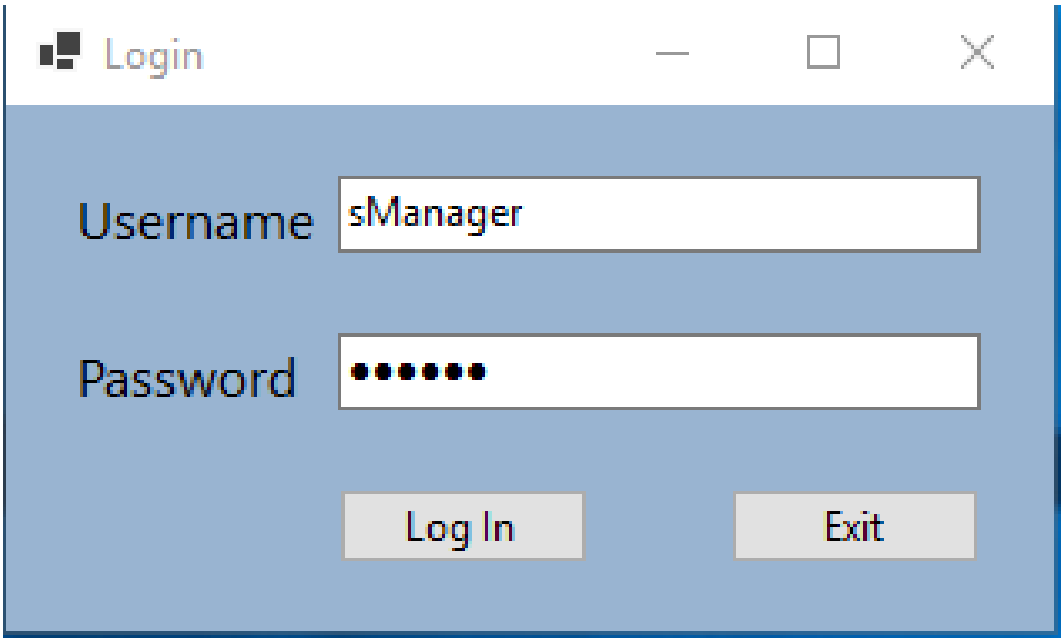
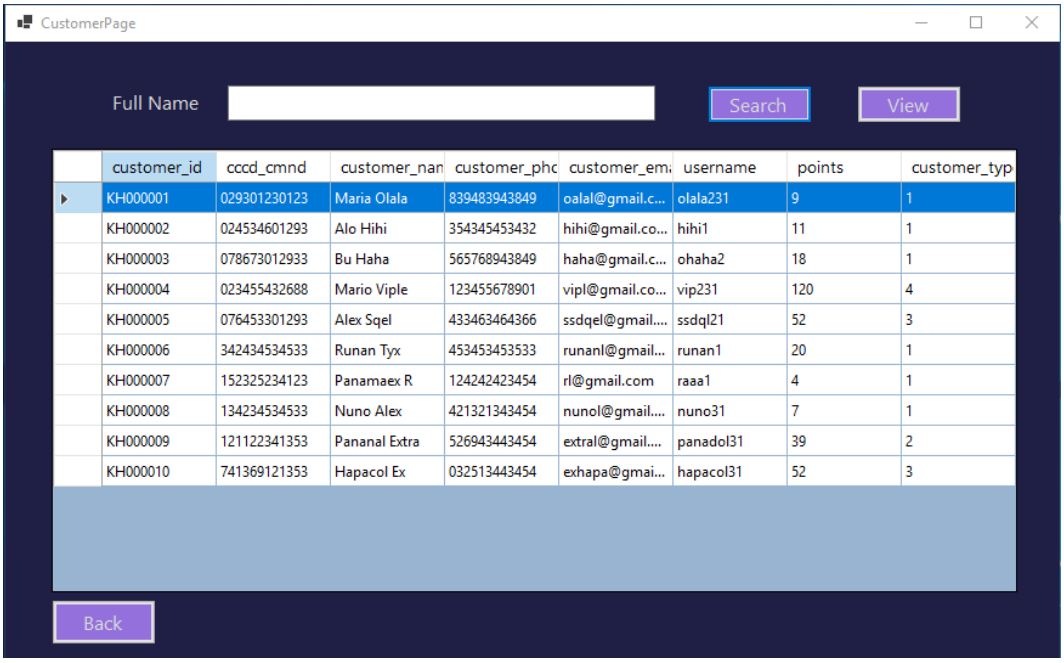


Figure 32: Login/Logout UI

3.2.2 Features

- View information of customer



customer_id	cccd_cmnd	customer_nan	customer_phc	customer_em	username	points	customer_typ
KH000001	029301230123	Maria Olala	839483943849	oalal@gmail.c...	olala231	9	1
KH000002	024534601293	Alo Hihi	354345453432	hihi@gmail.co...	hihi1	11	1
KH000003	078673012933	Bu Haha	565768943849	haha@gmail.c...	ohaha2	18	1
KH000004	023455432688	Mario Viple	123455678901	vipl@gmail.co...	vip231	120	4
KH000005	076453301293	Alex Sqel	433463464366	ssdqel@gmail....	ssdql21	52	3
KH000006	342434534533	Runan Tyx	453453453533	runanl@gmail...	runan1	20	1
KH000007	152325234123	Panamaex R	124242423454	rl@gmail.com	raaa1	4	1
KH000008	134234534533	Nuno Alex	421321343454	nunol@gmail....	nuno31	7	1
KH000009	121122341353	Pananal Extra	526943443454	extral@gmail....	panadol31	39	2
KH000010	741369121353	Hapacol Ex	032513443454	exhapa@gmai...	hapacol31	52	3

Figure 33: Information customer

- Search information with some characters. And then view details a customer.

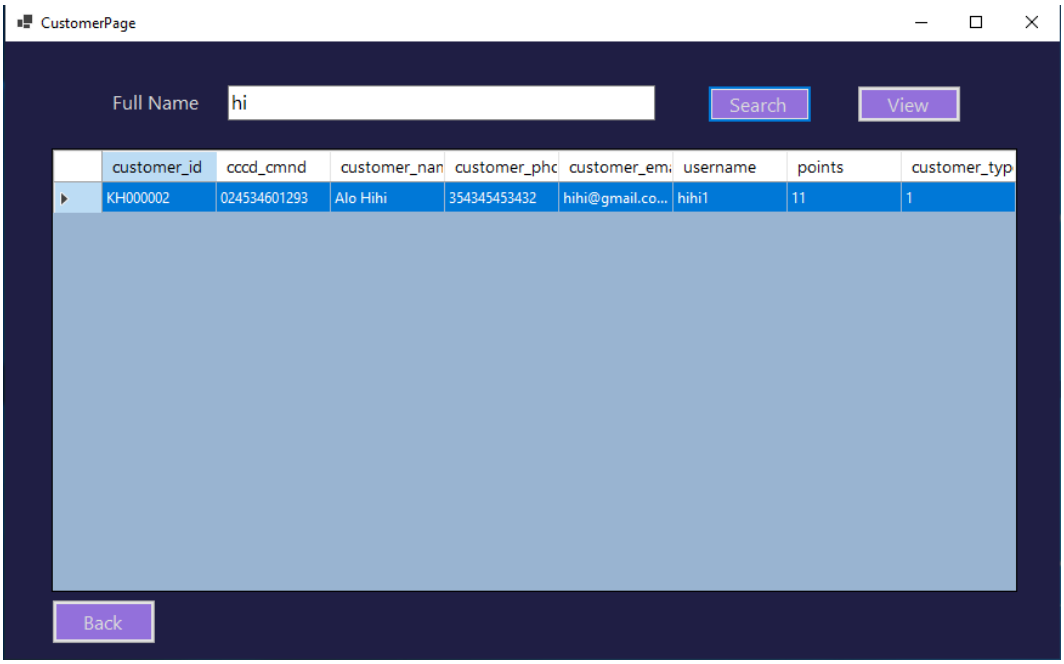


Figure 34: Search information

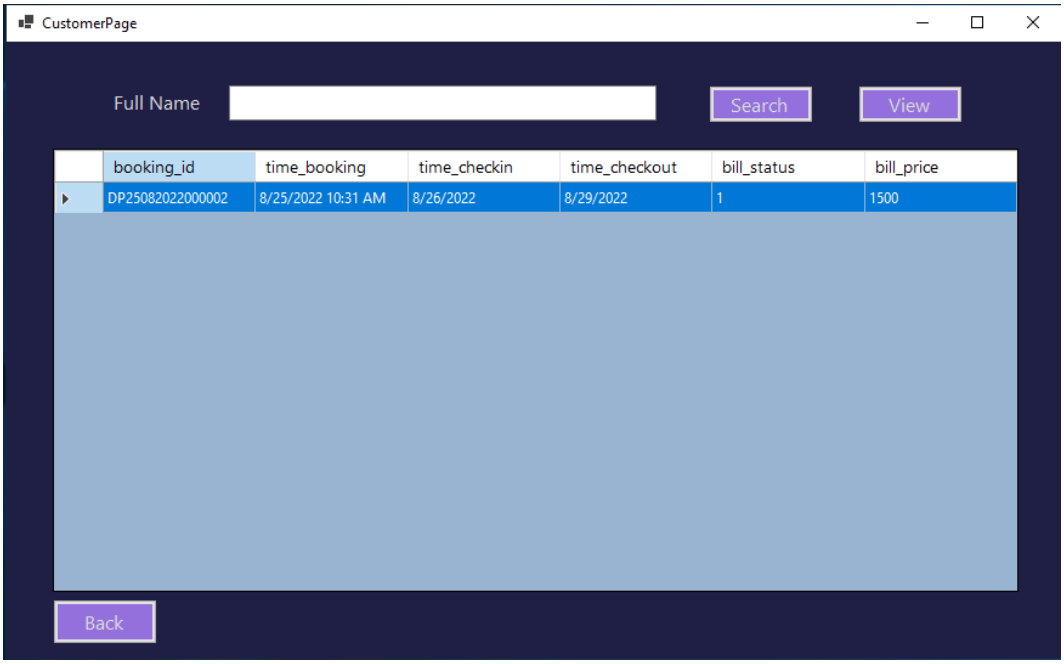


Figure 35: View details

- Add new information of room type

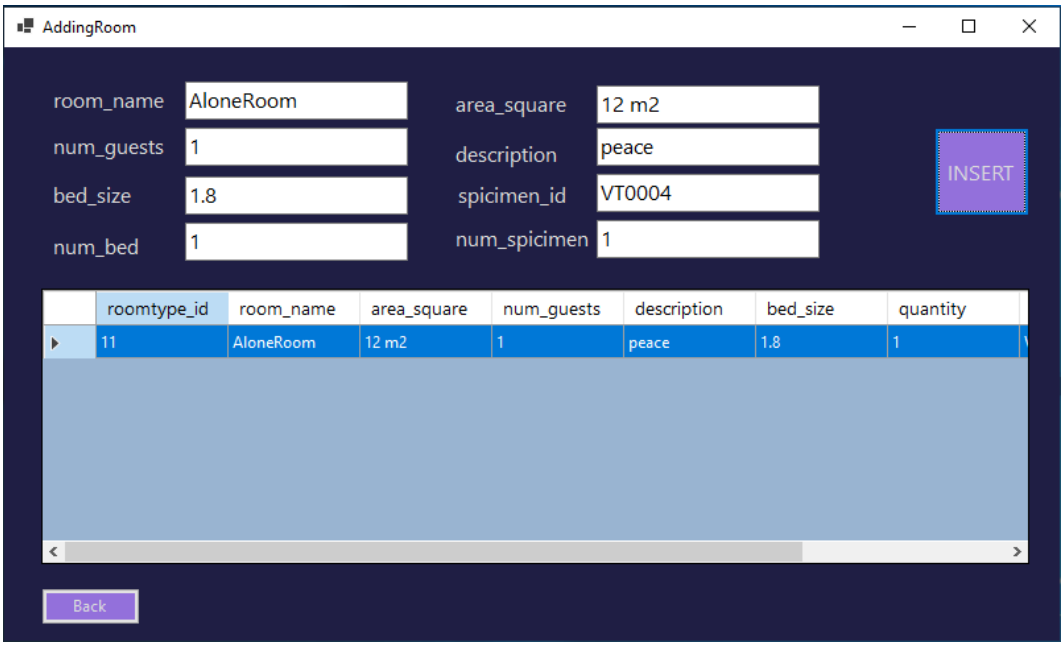


Figure 36: Add new information of room type

- View statistical guests in a branch in year

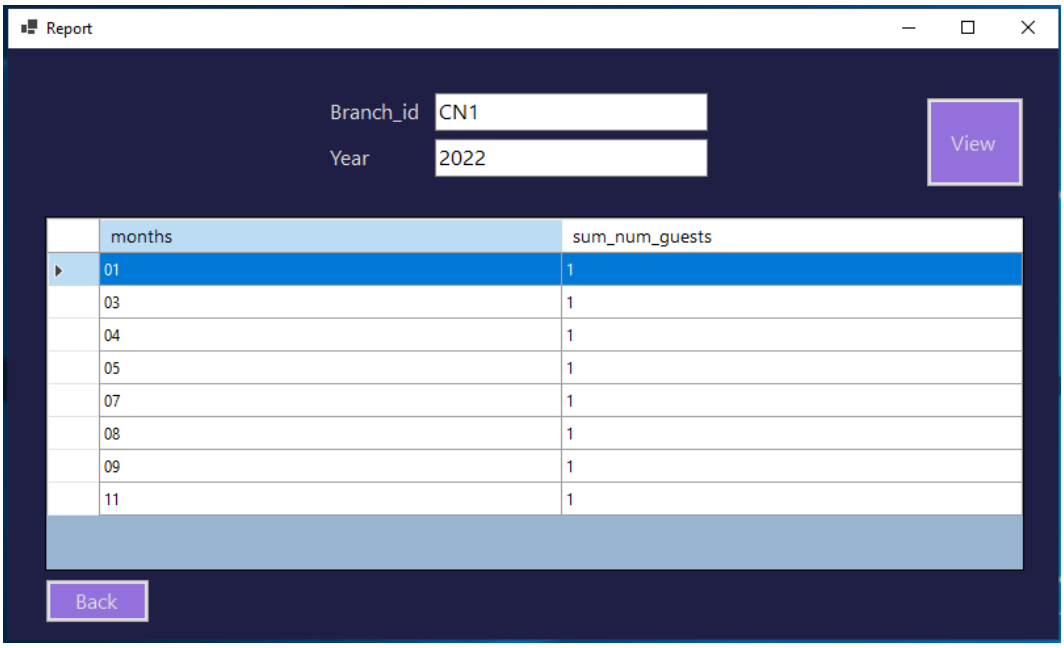
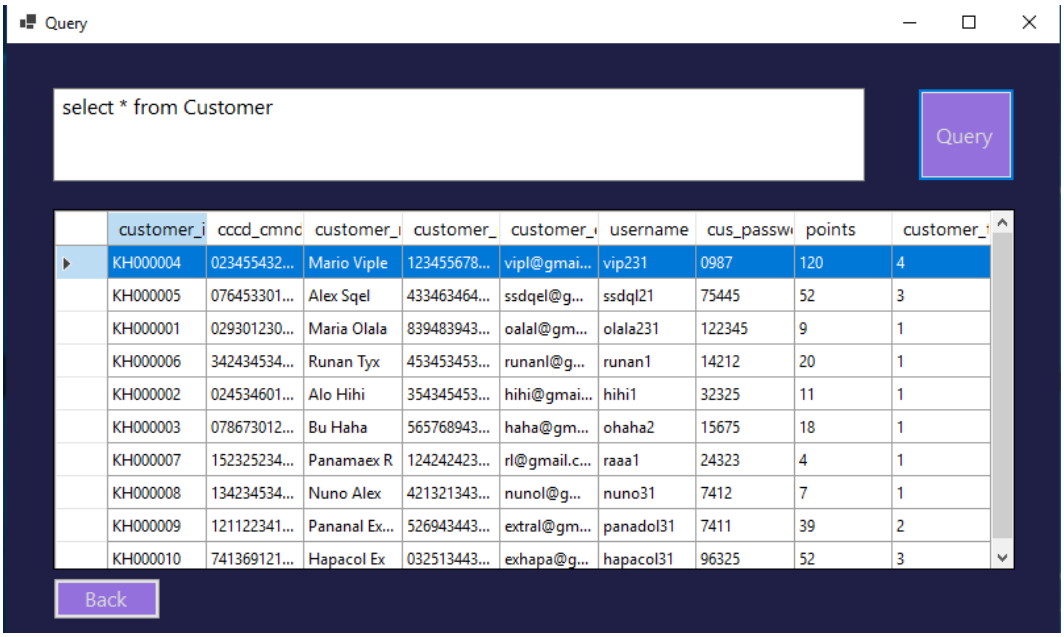


Figure 37: Add new information of room type

- In addition one feature for Admin to query on app



customer_i	cccd_cmnc	customer_	customer_	customer_	username	cus_passw	points	customer_
KH000004	023455432...	Mario Viple	123455678...	vipl@gmai...	vip231	0987	120	4
KH000005	076453301...	Alex Sqel	433463464...	ssdqel@g...	ssdql21	75445	52	3
KH000001	029301230...	Maria Olala	839483943...	oalal@gm...	olala231	122345	9	1
KH000006	342434534...	Runan Tyx	453453453...	runanl@g...	runan1	14212	20	1
KH000002	024534601...	Alo Hihi	354345453...	hihi@gmai...	hihi1	32325	11	1
KH000003	078673012...	Bu Haha	565768943...	haha@gm...	ohaha2	15675	18	1
KH000007	152325234...	Panamaex R	124242423...	rl@gmail.c...	raaa1	24323	4	1
KH000008	134234534...	Nuno Alex	421321343...	nunol@g...	nuno31	7412	7	1
KH000009	121122341...	Pananal Ex...	526943443...	extral@gm...	panadol31	7411	39	2
KH000010	741369121...	Hapacol Ex	032513443...	exhapa@g...	hapacol31	96325	52	3

Figure 38: Query on app