

UNIVERSITY OF SCIENCE AND TECHNOLOGY
INFORMATION AND TECHNOLOGY DEPARTMENT



Database IT3290E

Capstone Project Report
Hotel Booking Management

Lecturer: Dr. Vu Tuyen Trinh

Student: Nguyen Tuan Minh - 20184294

Nguyen Hoang Phuong – 2018430

Nguyen Tuan Dung - 20184247

Ha Noi, 2021

Table of Contents

1. Project description	4
1.1 Problems	4
1.2 Description	4
2. Use case diagram	5
2.1 Sign up:	5
2.2 Sign in:	6
2.3 Search hotel	6
2.4 Book.....	6
2.5 See and adjust the receipts	6
2.6 Manage account settings	6
2.7 Add hotel	7
2.8 Manage the own hotel.....	7
3. Entity relation diagram.....	8
4. Relation diagram.....	10
5. Stored procedure	12
5.1. Login	12
5.2. SignUpUser.....	13
5.3. SignUpManager	15
5.4. UserUpdateInfo	16
5.5. ManagerUpdateDetail	16
5.6 ManagerAddRoom	17
5.7. InsertHotel	17
5.8. FilteredSearch	18
5.9. ConfirmBooking.....	22
5.10. CancelReceiptHotel.....	24
5.11. CancelReceiptUser	25
5.12. UpdateReceiptStatusHotel	26
5.13. UpdateReceiptStatusUser	27
5.14. UpdateReceiptStatus	27
5.15. UserViewManagerDetail	28

5.16. ManagerViewUserDetail	29
6. <i>Index</i>	29
7. <i>Contribution</i>	31

1. Project description

1.1 Problems

User story 1: The hotel managers want to find the system that they can post their hotel information for renting on, and connect them to the customers who are finding places satisfying their needs. A system that helps hotel managers with not only adding, and managing their hotels, rooms but also showing the information of each of booking order (checkin date, check out date, roomID, ...) as well as customer's contact information is what they need now.

User story 2: When people have a plan to go on a traveling or business trip, they want to find a room or a place that meets their requirements. A system that helps people finding a suitable place to stay with no difficulty in registering, searching, booking and canceling is what they need now. This system should help them searching with a filter and show them brief information on a hotel (address, star, price, ...), and booking procedure must be as easiest as possible. The system should help them alter the information of their own, and also the receipt history should be available

1.2 Description

In this project, we built a hotel booking application that meet the stated criteria in the problem section

For the Database Management System we chose MySQL as it is easy to use and has large community for problem solving

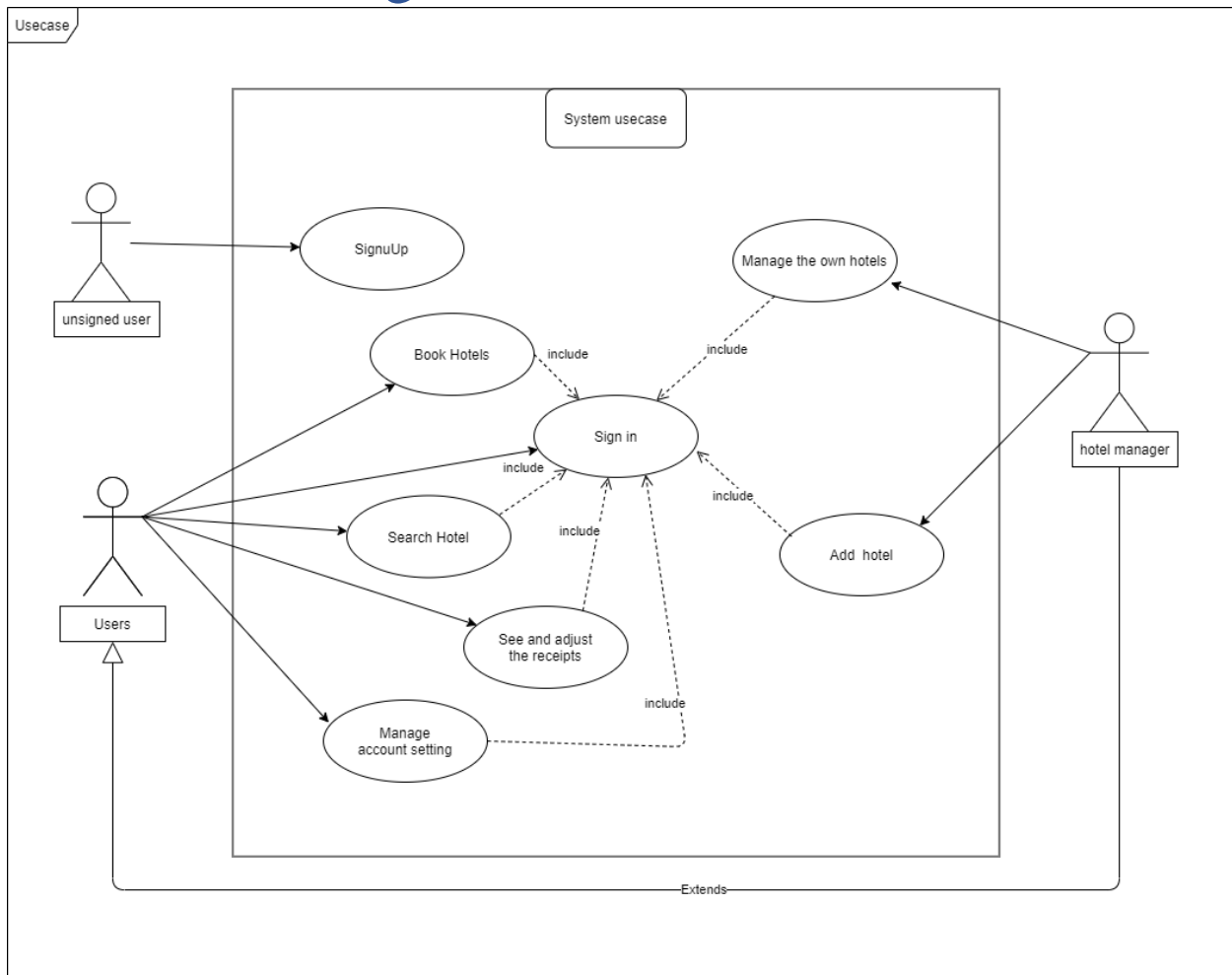
The database was built using the top-down database design method. From the problems, we determined a set of entities, and this set was show on the entity relation diagram (ERD). After had considered carefully about relationships in ERD, we came up with the relation diagram (RD). The relation schema in database was created after the RD had been modified correctly

Based on the use case diagram, we decided to create several stored procedures in order to generalize the interaction between application and database

Indexing was also added with the hope of improving the system

Finally, with the database ready, we constructed an application using Java and Javafx with all the function that will be mentioned in the use case diagram

2. Use case diagram



2.1 Sign up:

Unsigned users will sign up to participate in the system.

Procedure used:

- +) signUpUser(user)
- +) signUpHotelManager(newManager, newHotel)

2.2 Sign in:

Users or hotel managers must log in the system to use their function.

Procedure used:

- +) Login(accountNameString, passwordString)

2.3 Search hotel

Users search the hotel that meet their demand of location can addition some convinient filters.

Procedure used:

- +) filteredSearch(destination, hotelname, star, filteringString)

2.4 Book

After searching, users can book the hotel that they want.

Procedure used:

- +) confirmBooking(numberOfRoom, hotelID, checkinDate, checkOutDate, userID)

2.5 See and adjust the receipts

User will have the client info which have severals receipts which is the history of booking.

Procedured used:

- +) CancelReceiptUser(receiptID, userID)
- +) UserViewManagerDetail(receiptID)

2.6 Manage account settings

User can chage their information including password in client site

Procedured used:

- +) UserUpdateInfo(userID)

2.7 Add hotel

Hotel manager add more hotel if they want to.

Procedures used:

- + Insert Hotel(hotelnameString, hotelAddressFull, streetId)

2.8 Manage the own hotel

Hotel manager adjust the information of the hotel, cancel the receipt and also can add more room.

Procedures used:

- + managerUpdateDetail(hotelID, starInt, currentPrice, filterString)

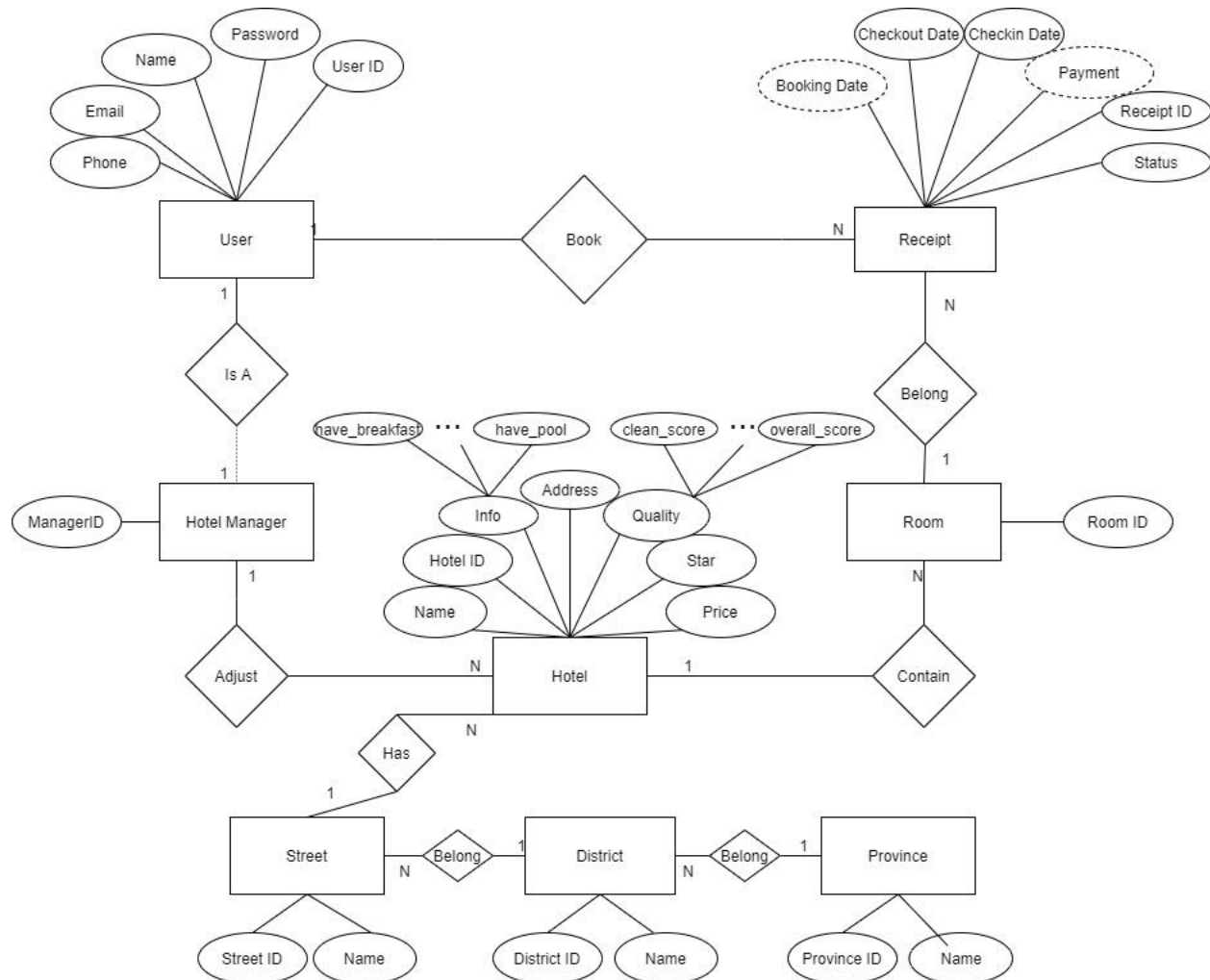
- + managerAddRoom(hotelID)

- + managerViewUserDetail(receiptID)

- + cancelReceiptHotel(receiptID, hotelID)

3. Entity relation diagram

After problem analyzing, we came up with a list of entities that our system has to take into consider as follow:



List of entities:

User: An entity that includes all the registered user and hotel manager

In relation with **Receipt**, one user can book many receipts, and one receipt belongs to one user

In relation with **Hotel manager**, one user can be a hotel manager or not, and one hotel manager must be a user

Hotel manager: An entity that is a user who want his or her hotel(s) to use our service, hotel manager can interact with all the functions that a normal user can do

In relation with **User**, one hotel manager is a User, and one user can be a Hotel manager or not

In relation with **Hotel**, one hotel manager can manage many hotels, and one hotel can only be managed by a Hotel manager

Hotel: an entity represents the realife hotel

In relation with **Room**, one hotel has many rooms, and one room belongs to a Hotel

In relation with **Hotel manager**, one hotel can only be managed by a Hotel manager, and one hotel manager can manage many hotels

In relation with **Street**, one hotel can be allocated on a specific street, and on a street there can be many hotels

Room: an entity represents room of a hotel

In relation with **Hotel**, one Room belongs to a hotel, and one hotel can have many rooms

In relation with **Receipt**, one room can be included in many receipts, and one receipt belongs to one room

Receipt: an entity represents receipt that was created after user's booking

In relation with **User**, one receipt belongs to one user, and one user can have many receipts

In relation with **Room**, one receipt belongs to one room, one room can be included in many receipt

Street: an entity represents a street in realife with name and id

In relation with **Hotel**, on a street there can be many hotels, and one hotel can only be allocated on a specific street

In relation with **District**, one street belongs to one district, and one district has many streets

District: an entity represents a district in real life with name and id

In relation with **Street**, on a district there can be many streets, and one street can only be allocated on a specific district

In relation with **Province**, one district belongs to one province, and one province has many districts

Province: an entity represents a province in real life with name and id

In relation with **District**, on a province there can be many districts, and one district can only be allocated on a specific province

4. Relation diagram

Based on completed ERD, we decided to transform it to RD by following a rule:

For 1-N relation, 2 tables will be created with their attributes as in the ERD and the entity representing the N relation will have the primary key of the entity representing the 1 relation as Foreign Key, and the connector between 2 tables is 1 to Many connector

For example, relation between hotel and room is 1-N, so room table will have a Foreign Key hotel_id (Primary key of hotel entity)

Three special cases of transforming:

The relation between User and Hotel Manager is 1-1(optional) so we created the table HotelManager with manager_id as primary key and user_id as Foreign Key and the connect between HotelManager and User is 1 optional – 1 mandatory

The info attribute of hotel entity is a complex attribute, and we think that if this complex attribute is being considered during the query of Hotel table, the query will get much slower, therefore, we separate it into a difference table named Info. Connection between Hotel and Info will be set as 1 – 1 connector

The quality attribute of hotel entity is handled with same way as info attribute creating a table Quality. Connection between Hotel and Info will be set as 1 – 1 connector

Based on the ERD and the rule implied, we came up with these tables and a RD:

User (**userID**, name, phone, email, password)

Recepit (**receiptID**, bookingDate, checkinDate, checkoutDate, payment, status, **userID**, **roomID**)

HotelManager (**managerID**, **userID**)

Hotel (**hotelID**, name, address, star, price, **streetID**, **manager_ID**)

Quality (hotelID, clean_score, meal_score, location_score, sleep_score, room_score, service_score, facility_score, overall_score, review_score, num_review)

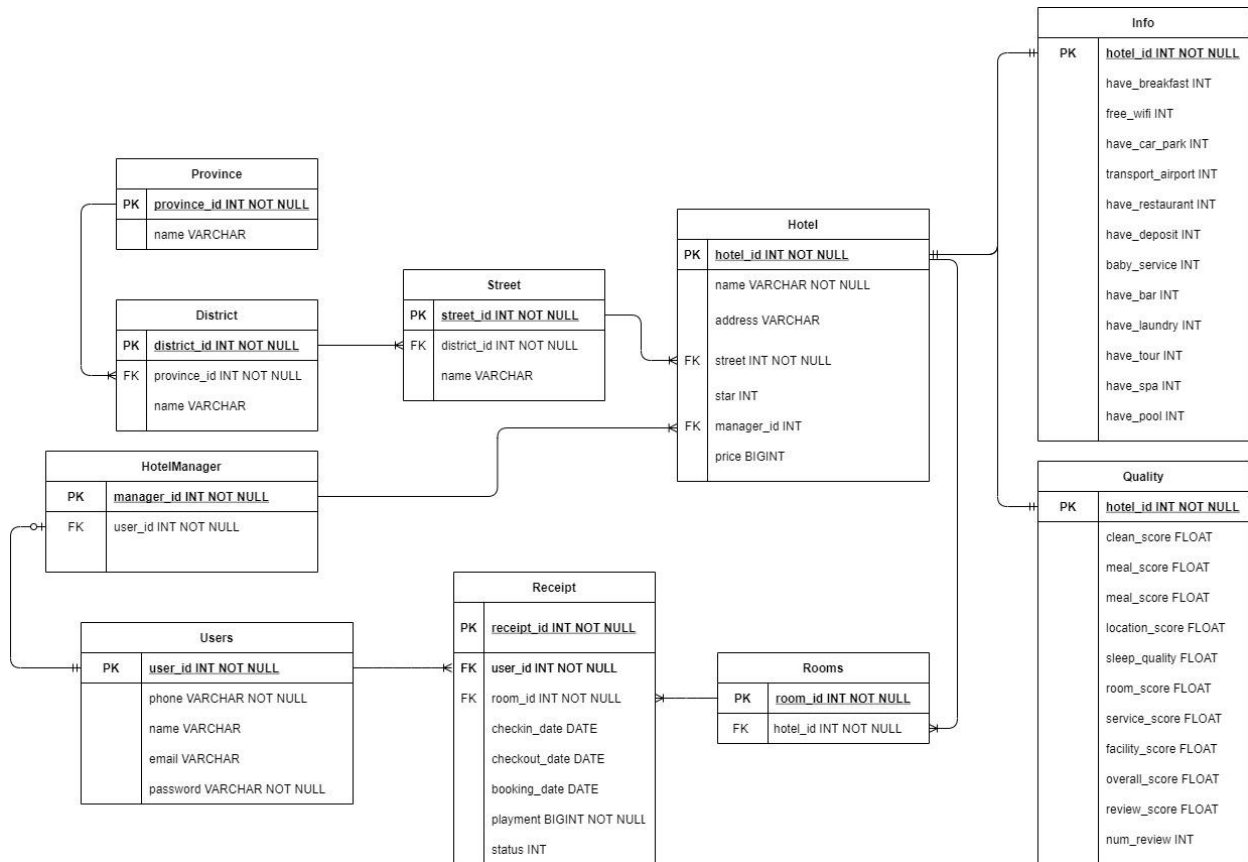
Info (**hotelID**, have_breakfast, free_wifi, have_car_park, transport_airport, have_restauran, have_deposit, baby_service, have_bar, have_laundry, have_tour, have_spa, have_pool)

Room (**roomID**, **hotelID**)

Street (**streetID**, name, **districtID**)

District (**districtID**, name, **provinceID**)

Province (**provinceID**, name)



5. Stored procedure

In our database project, we created stored procedure using explicitly for each use-case.

5.1. Login

- Login procedure finds any user if there is a match with accountNameString and passwordString; if no user is found, it queries -1.
- If a user can be founded, it continues to search if that user is a hotel manager and queries the information of that manager; else, it queries the user information.

```

CREATE DEFINER=`root`@`localhost` PROCEDURE `Login`
(IN `accountNameString` TEXT, IN `passwordString` TEXT)
BEGIN
DECLARE user1 INT DEFAULT -1;
DECLARE manager INT DEFAULT 0;
SELECT
    id
INTO user1 FROM
    user
WHERE
    username = accountNameString
    AND password = passwordString;
IF user1 != -1 THEN
    SELECT COUNT(*) INTO manager FROM user,hotelmanager WHERE user.id = hotelmanag
er.user_id AND user.id = user1;
    IF manager != 0 THEN
        SELECT * FROM user, hotelmanager WHERE user.id = hotelmanager.user_id AND use
r.id = user1;
    ELSE
        SELECT * FROM user where id = user1;
    END IF;
ELSE
    SELECT -1;
END IF;
END

```

5.2. SignUpUser

- User can sign up by providing the fields name, phone, accountName, email, phone and password.
- The procedure first checks for if there is an existing phone or accountName in the database (See Fig. CheckExistAccount)
 - If that phone number existed, it stores in status value 2
 - If that accountName existed, it stores in status value 1.
 - If there is no equivalent phone or accountName in the database, the status is 0.

```

CREATE DEFINER=`root`@`localhost` PROCEDURE `CheckExistAccount`
(IN `accountNameString` VARCHAR(40),
IN `phoneString` VARCHAR(40), OUT `status` INT)
BEGIN
DECLARE tmpusername VARCHAR(40);
DECLARE tmpphone VARCHAR(40);
SET status = 0;
SELECT phone INTO tmpphone FROM user WHERE phone = phoneString;
IF tmpphone = phoneString
THEN SET status = 2;
END IF;
SELECT username INTO tmpusername FROM user WHERE username = accountNameString;
IF tmpusername = accountNameString
THEN SET status = 1;
END IF;
END

```

- SignUpUser procedure takes the status value after execute the CheckExistAccount; if status is 0, insert the user information to the database. (See Fig. SignUpUser

```

CREATE DEFINER=`root`@`localhost` PROCEDURE `SignUpUser`
(IN `nameString` VARCHAR(40),
IN `phoneString` VARCHAR(40),
IN `emailString` VARCHAR(40),
IN `accountNameString` VARCHAR(40),
IN `passwordConfirm` VARCHAR(40))
BEGIN
CALL checkExistAccount(accountNameString, phoneString, @status);
SELECT @status;
IF @status = 0
THEN
INSERT INTO user(name,phone,email,username,password) VALUES (nameString,phon
eString,emailString,accountNameString,passwordConfirm);
END IF;
END

```

5.3. SignUpManager

- A hotel manager also needs to define those fields as a user and provides the relevant fields of the hotel such as hotelNameString, hotelAddressFull, streetID.
- First the procedure executes CheckExistAccount (see Fig. CheckExistAccount) and stores the account status.
- Next it checks for status and adds the user to the database; then it calls the procedure InsertHotel (see part 4.7) to insert the hotel, if the hotel status is zero which means cannot add this hotel, we need to delete the user we added earlier.
- The procedure will query 0 if there is an existed hotel, 1 if there is an existed account name, 2 if there is an existed phone number.

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `SignUpManager`
(IN `nameString` VARCHAR(40),
IN `phoneString` VARCHAR(40),
IN `emailString` VARCHAR(40),
IN `accountNameString` VARCHAR(40),
IN `passwordConfirm` VARCHAR(40),
IN `hotelNameString` VARCHAR(40),
IN `hotelAddressFull` VARCHAR(40), IN `streetID` INT)
BEGIN
    DECLARE userID INT;
    CALL checkExistAccount(AccountNameString, phoneString, @accountstatus);
    IF @accountstatus = 0 THEN
        INSERT INTO user(name,phone,email,username,password) VALUES (nameString,
phoneString,emailString,accountNameString,passwordConfirm);
        SELECT id INTO userID FROM user WHERE phone = phoneString;
        INSERT INTO hotelmanager(user_id) VALUES (userID);
        CALL InsertHotel(hotelNameString, hotelAddressFull, streetID, userID, @h
otelstatus);
        IF @hotelstatus = 0 THEN
            DELETE user, hotelmanager FROM user INNER JOIN hotelmanager ON hotel
manager.user_id = user.id WHERE user.id = userID;
        END IF;
    ELSE
        SELECT @accountstatus;
    END IF;
END
```

5.4. UserUpdateInfo

- A user or a manager can change nameString, phoneString, emailString, passwordString by userID (see Fig. UserUpdateInfo).

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `UpdateUserData`  
(IN `userID` INT,  
IN `nameString` VARCHAR(40),  
IN `phoneString` VARCHAR(40),  
IN `emailString` VARCHAR(40),  
IN `passwordString` VARCHAR(40))  
BEGIN  
UPDATE user SET  
    name = nameString,  
    phone = phoneString,  
    email=emailString,  
    password = passwordString  
WHERE id = userID;  
END
```

5.5. ManagerUpdateDetail

- A manager can update the detail of a hotel by providing new star, currentPrice or information in FilterString (see Fig. ManagerUpdateDetail).
- The procedure first updates the star, price with hotelID; next execute the query statement by concat it with FilterString and hotelID.

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `ManagerUpdateDetail`  
(IN `HotelID` INT,  
IN `starInt` INT,  
IN `currentPrice` BIGINT,  
IN `FilterString` TEXT)  
BEGIN  
UPDATE hotel SET star= starInt , price = currentPrice WHERE id= HotelID;  
SET @querysm = (SELECT CONCAT('UPDATE hotelinfo SET ', FilterString, " WHERE id  
=", HotelID, ";"));  
PREPARE smt FROM @querysm;  
EXECUTE smt;  
END
```


5.6 ManagerAddRoom

- A manager can add a room to a hotel by hotelID(see Fig. ManagerAddRoom)

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `ManagerAddRoom` (IN `HotelID` INT)
BEGIN
    INSERT INTO room(hotel_id) VALUES(HotelID);
END
```

5.7. InsertHotel

- The procedure first checks if there is existed hotel name by function CheckExistHotelName (see Fig. CheckExistHotelName)

```
CREATE DEFINER=`root`@`localhost` FUNCTION `CheckExistHotelName`
(`HotelNameString` TEXT) RETURNS int(11)
BEGIN
    DECLARE count INT;
    DECLARE HotelStatus INT;
    SELECT COUNT(*) INTO count FROM hotel WHERE name = HotelNameString;
    IF count = 1 THEN
        SET HotelStatus = 0;
    ELSE
        SET HotelStatus = -1;
    END IF;
    RETURN HotelStatus;
END
```

- If the function returns -1 then we can add the hotel and initiate some default value for hotel information.

```

CREATE DEFINER=`root`@`localhost` PROCEDURE `InsertHotel`
(IN `hotelNameString` VARCHAR(40),
IN `hotelAddressFull` VARCHAR(40),
IN `streetID` INT, IN `managerID` INT, OUT `hotelstatus` INT)
BEGIN
    DECLARE hotelID INT;
    SET hotelstatus = CheckExistHotelName(hotelNameString);
    IF hotelstatus = -1 THEN
        INSERT INTO hotel(name,address,star,street_id,manager_id)
        VALUES (hotelNameString,hotelAddressFull,1,streetID,managerID);
        SELECT id INTO hotelID FROM hotel WHERE name = hotelNameString AND manag
er_id = managerID;
        INSERT INTO hotelinfo VALUES(hotelID,0,0,0,0,0,0,0,0,0,0,0,0);
    END IF;
END

```

5.8. FilteredSearch

- Since the search query for the database can be long and complex, we divided the query into substring and handle them individually and combine them at the end.

```

CREATE DEFINER=`root`@`localhost` PROCEDURE `FilteredSearch`
(IN `destination` VARCHAR(40),
IN `hotelname` TEXT,
IN `star` INT,
IN `FilterString` TEXT)
BEGIN
    DECLARE flagDestinationEmpty INT DEFAULT 1;
    DECLARE flagHotelNameEmpty INT DEFAULT 1;
    DECLARE flagStarEmpty INT DEFAULT 1;
    SET @querysmt = 'SELECT hotelinfo.id FROM hotelinfo';
    SET @querysmt1 = ' JOIN hotel ON (hotelinfo.id = hotel.id) JOIN street ON (s
treet.id = hotel.street_id) JOIN district ON (district.id = street.district_id)
JOIN province ON (province.id = district.province_id)';
    SET @conditionDestination = GetDestinationString(destination);
    SET @conditionHotelName = GetHotelNameString(hotelname);
    SET @conditionStar = (SELECT CONCAT(" star = '", star, "'"));

```

- First, we generate the string `quersmt` and `quersmt1` contains the `SELECT` `FROM` part and the join condition with `hotel` table and `province` table since we offer search by province, name and star along with search by filter.
 - `flagDestinationEmpty`, `flagHotelNameEmpty`, `flagStarEmpty` are the condition check if the user input those options.
 - `ConditionDestination` is a substring for compare with input destination generated by `GetDestinationString` function (see Fig. `GetDestinationString`).
 - `ConditionHotelName` is a substring for compare with input hotelname generated by `GetHotelNameString` function (see Fig. `GetHotelNameString`).
 - `ConditionStar` is a substring for compare with input star.

```
CREATE DEFINER=`root`@`localhost` FUNCTION `GetDestinationString`(`destination`
TEXT) RETURNS text CHARSET utf8
BEGIN
  SET @destination1 = (SELECT CONCAT("%",destination, "%"));
  SET @destination2 = (SELECT CONCAT("",destination, "%"));
  SET @destination3 = (SELECT CONCAT("%",destination, ""));
  SET @conditionDestination = (SELECT CONCAT(' ( province.name like ',@destina
tion1,' OR province.name LIKE ',@destination2,' OR province.name LIKE ',@destina
tion3," OR province.name LIKE "",destination,")'));
  RETURN (@conditionDestination);
END
```

```
CREATE DEFINER=`root`@`localhost` FUNCTION `GetHotelNameString`(`hotelname` TEX
T) RETURNS text CHARSET utf8
BEGIN
  SET @hotelname1 = (SELECT CONCAT("%",hotelname, "%"));
  SET @hotelname2 = (SELECT CONCAT("",hotelname, "%"));
  SET @hotelname3 = (SELECT CONCAT("%",hotelname, ""));
  SET @conditionHotelName = (SELECT CONCAT(' ( hotel.name like ',@hotelname1,'
OR hotel.name LIKE ',@hotelname2,' OR hotel.name LIKE ',@hotelname3," OR hotel.n
ame LIKE "",hotelname,")'));
  RETURN (@conditionHotelName);
END
```

- Next, we check for status of each flag and concat the substring to a complete query.
- Finally, we prepare the query and execute it.

```
IF (destination != '') OR (hotelname != '') OR (star != 0) THEN
  IF (destination != '') THEN
    SET flagDestinationEmpty = 0;
  END IF;
  IF (hotelname != '') THEN
    SET flagHotelNameEmpty = 0;
  END IF;
  IF (star != 0) THEN
    SET flagStarEmpty = 0;
  END IF;
  SET @querysm1 = (SELECT CONCAT(@querysm1, @querysm1));
END IF;
```

```

    IF (FilterString != '') THEN
        SET @querysm = (SELECT CONCAT(@querysm, ' WHERE ', FilterString));
        IF (flagDestinationEmpty = 0) THEN
            SET @querysm = (SELECT CONCAT(@querysm, ' AND ', @conditionDestina
tion));
        END IF;
        IF (flagHotelNameEmpty = 0) THEN
            SET @querysm = (SELECT CONCAT(@querysm, ' AND ', @conditionHotelNa
me));
        END IF;
        IF (flagStarEmpty = 0) THEN
            SET @querysm = (SELECT CONCAT(@querysm, ' AND ', @conditionStar));
        END IF;
    ELSE
        IF (flagDestinationEmpty = 0) THEN
            SET @querysm = (SELECT CONCAT(@querysm, ' WHERE ', @conditionDesti
nation));
        IF (flagHotelNameEmpty = 0) THEN
            SET @querysm = (SELECT CONCAT(@querysm, ' AND ', @conditionHot
elName));
        END IF;
        IF (flagStarEmpty = 0) THEN
            SET @querysm = (SELECT CONCAT(@querysm, ' AND ', @conditionSta
r));
        END IF;
    ELSE
        IF (flagHotelNameEmpty = 0) THEN
            SET @querysm = (SELECT CONCAT(@querysm, ' WHERE ', @conditionH
otelName));
        IF (flagStarEmpty = 0) THEN
            SET @querysm = (SELECT CONCAT(@querysm, ' AND ', @conditio
nStar));
        END IF;
    ELSE
        IF (flagStarEmpty = 0) THEN
            SET @querysm = (SELECT CONCAT(@querysm, ' WHERE ', @condit
ionStar));
        END IF;
    END IF;
    END IF;
    PREPARE smt FROM @querysm;
    EXECUTE smt;
END

```

5.9. ConfirmBooking

- To book a room, a user or manager need to find the number of room available for his/her booking time. Since we create a receipt for each room which being booked, we use a cursor for the roomID that available.
- For each roomID the cursor fetch to the procedure, it inserts the data to the receipt table and increment to counter until it hit the number of room.
 - If the counter equals to number if room set done to 1 and exit the LOOP and close the cursor.
 - If the number of rooms is bigger than the available rooms, the procedure queries minus 1.

```
CREATE DEFINER='root'@'localhost' PROCEDURE `ConfirmBooking`  
  (IN `numberOfRoom` INT,  
   IN `hotelID` INT,  
   IN `checkinDate` DATE,  
   IN `checkoutDate` DATE,  
   IN `userID` INT)  
BEGIN  
  DECLARE done INT DEFAULT 0;  
  DECLARE counter INT DEFAULT 0;  
  DECLARE roomID INT ;  
  DECLARE DateNow DATE;  
  DECLARE RoomPrice INT;  
  DECLARE curRoomID CURSOR  
  FOR (SELECT id FROM room WHERE hotel_id = hotelID  
  AND id NOT IN  
    (SELECT room_id FROM receipt  
    WHERE status !=-1  
    AND hotel_id = hotelID  
    AND ((checkin_date <= checkinDate AND checkout_date >= checkoutDate)  
    OR (checkin_date >= checkinDate AND checkin_date <= checkoutDate)  
    OR (checkout_date >= checkinDate AND checkout_date <= checkoutDate))  
    )  
  );  
  DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;
```

```

        SET RoomPrice = GetPriceForRoom(hotelID, checkinDate, checkoutDate);
SELECT CURDATE() INTO DateNow;
    IF (numberOfRoom <= GetNumberOfAvailableRooms(hotelID, checkinDate, checkout
Date)) THEN
        OPEN curRoomID;
        GetRoomID: LOOP
            FETCH curRoomID INTO roomID;
            IF done = 1 THEN
                LEAVE GetRoomID;
            END IF;
            IF counter < numberOfRoom THEN
                INSERT INTO receipt(user_id, room_id,checkin_date,checkout_date,
booking_date,payment,status) VALUES(userID, roomID,checkinDate,checkoutDate,Date
Now, RoomPrice, 0);
                SET counter = counter + 1;
            ELSE
                SET done = 1;
            END IF;
        END LOOP GetRoomID;
        CLOSE curRoomID;
    ELSE
        SELECT -1;
    END IF;
END

```

- The room price is created by the GetPriceForRoom function (see Fig. GetPriceForRoom)

```

CREATE DEFINER=`root`@`localhost` FUNCTION `GetPriceForRoom`(`hotelID` INT, `che
ckinDate` DATE, `checkoutDate` DATE) RETURNS bigint(20)
BEGIN
DECLARE RoomPrice INT;
SELECT
    price * DATEDIFF(checkoutDate, checkinDate)
INTO RoomPrice FROM
    hotel
WHERE
    id = hotelID;
RETURN(RoomPrice);
END

```

- The available room is generated by GetNumberOfAvailableRooms function (see Fig. GetNumberOfAvailableRooms)

```
CREATE DEFINER=`root`@`localhost` FUNCTION `GetNumberOfAvailableRooms`
(`hotelID` INT, `checkinDate` DATE, `checkoutDate` DATE) RETURNS int(11)
BEGIN
DECLARE availableRoom INT;
SELECT
    COUNT(id)
INTO availableRoom FROM
    room
WHERE
    hotel_id = hotelID
    AND id NOT IN (SELECT
        room_id
    FROM
        receipt
    WHERE
        status != 2 AND hotel_id = hotelID
        AND ((checkin_date <= checkinDate
            AND checkout_date >= checkoutDate)
        OR (checkin_date >= checkinDate
            AND checkin_date <= checkoutDate)
        OR (checkout_date >= checkinDate
            AND checkout_date <= checkoutDate)));
RETURN(availableRoom);
END
```

5.10. CancelReceiptHotel

- A manager can set the status of a receipt to -1 by the receiptID and hotelID. Then the manager calls the ReceiptStatusHotel (see Fig. ReceiptStatusHotel) procedure to update all the status.
- Then the procedure queries the receipt information and HotelID of that receipt for the application.


```

CREATE DEFINER=`root`@`localhost` PROCEDURE `CancelReceiptHotel`
(IN `receiptID` INT, IN `HotelID` INT)
BEGIN
    UPDATE receipt SET status = -1 WHERE id = receiptID;

    CALL UpdateReceiptStatusHotel(HotelID);

    SELECT receipt.*,hotel.id FROM receipt
    JOIN room ON (receipt.room_id = room.id)
    JOIN hotel ON (room.hotel_id = hotel.id)
    WHERE hotel_id = HotelID;

END

```

5.11. CancelReceiptUser

- A user can set the status of a receipt to -1 by the receiptID and userID. Then the user calls the ReceiptStatusUser (see Fig. ReceiptStatusUser) procedure to update all the status.
- Then the procedure queries the receipt information and HotelID of that receipt for the application.

```

CREATE DEFINER=`root`@`localhost` PROCEDURE `CancelReceiptUser`
(IN `receiptID` INT, IN `userID` INT)
BEGIN
    UPDATE receipt SET status = -1 WHERE id = receiptID;

    CALL UpdateReceiptStatusUser(userID);

    SELECT receipt.*,hotel.id FROM receipt
    JOIN room ON (receipt.room_id = room.id)
    JOIN hotel ON (room.hotel_id = hotel.id)
    WHERE user_id = userID;

END

```

5.12. UpdateReceiptStatusHotel

- To update the status of all receipts in hotel, we need to scan for each row of the table receipt of that hotel so that we created a cursor for the receiptID of that table.
- We created a table of all the receipts for that hotel, and open the cursor to apply the procedure UpdateReceiptStatus (see Fig. UpdateReceiptStatus) for each receiptID.
- But the cursor will return an error if we don't declare a handler for not found exception. The handler helps us to leave the loop we've created

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `UpdateReceiptStatusHotel`  
(IN `HotelID` INT)  
BEGIN  
  DECLARE done INT DEFAULT 0;  
  DECLARE receiptStatus INT;  
  DECLARE receiptID INT;  
  DECLARE DateNow DATE;  
  DECLARE curReceipt CURSOR  
  FOR  
  (SELECT receipt.id FROM receipt  
   JOIN room ON (receipt.room_id = room.id)  
   JOIN hotel ON (room.hotel_id = hotel.id)  
   WHERE hotel_id = HotelID);  
  DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;  
  OPEN curReceipt;  
getReceiptID: LOOP  
  FETCH curReceipt INTO receiptID;  
  IF done = 1 THEN  
    LEAVE getReceiptID;  
  END IF;  
  SELECT CURDATE() INTO DateNow;  
  CALL UpdateReceiptStatus(receiptID, DateNow);  
END LOOP getReceiptID;  
CLOSE curReceipt;  
END
```

5.13. UpdateReceiptStatusUser

- To update the status of all receipts for user, we need to scan for each row of the table receipt of that user so that we created a cursor for the receiptID of that table.
- We created a table of all the receipts for that user, and open the cursor to apply the procedure UpdateReceiptStatus (see Fig. UpdateReceiptStatus) for each receiptID.
- But the cursor will return an error if we don't declare a handler for not found exception. The handler helps us to leave the loop we've created

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `UpdateReceiptStatusUser` (IN `userID`
` INT)
BEGIN
DECLARE done INT DEFAULT 0;
DECLARE receiptStatus INT;
DECLARE receiptID INT;
DECLARE DateNow DATE;
DECLARE curReceipt CURSOR
FOR
(SELECT receipt.id FROM receipt
JOIN room ON (receipt.room_id = room.id)
JOIN hotel ON (room.hotel_id = hotel.id)
WHERE user_id = userID);
DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;
OPEN curReceipt;
getReceiptID: LOOP
    FETCH curReceipt INTO receiptID;
    IF done = 1 THEN
        LEAVE getReceiptID;
    END IF;
    SELECT CURDATE() INTO DateNow;
    CALL UpdateReceiptStatus(receiptID, DateNow);
END LOOP getReceiptID;
CLOSE curReceipt;
END
```

5.14. UpdateReceiptStatus

- The procedure first queries the status, checkin_date, checkout_date of that receipt.

- Then the procedure will set the status for that receipt by checking the conditions.
 - Status is 2 when the receipt is finished.
 - Status is 1 when the receipt is in booking time.

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `UpdateReceiptStatus`(IN `receiptID`
INT, IN `datetime` DATE)
BEGIN
  DECLARE receiptstatus INT;
  DECLARE checkinDate DATE;
  DECLARE checkoutDate DATE;
  SELECT status, checkin_date, checkout_date INTO receiptstatus, checkinDate, ch
  eckoutDate FROM receipt WHERE id = receiptID;
  IF receiptstatus = 0 THEN
    IF checkinDate <= datetime THEN
      UPDATE receipt SET status = 1 WHERE id = receiptID;
    END IF;
  ELSE
    IF checkoutDate < datetime THEN
      UPDATE receipt SET status = 2 WHERE id = receiptID;
    END IF;
  END IF;
END
```

5.15. UserViewManagerDetail



- A user can view the manager information of the hotel the

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `UserViewManagerDetail`
(IN `receiptID` INT)
BEGIN
  SELECT user.*, hotelmanager.id
  FROM user, hotelmanager
  WHERE hotelmanager.id = (SELECT manager_id FROM hotel WHERE id = GetHotelIDByRec
  eiptID(receiptID))
  AND hotelmanager.user_id = user.id;
END
```

5.16. ManagerViewUserDetail

- A manager can view the user that book the receipt in the application by simply select the user_id corresponding to that receiptID.

SQL ▾

 Copy  Caption ...

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `ManagerViewUserDetail`  
(IN `receiptID` INT)  
BEGIN  
  DECLARE userID INT;  
  SELECT user_id INTO userID FROM receipt WHERE id = receiptID;  
  SELECT * FROM user where id = userID;  
END
```

6. Index

For our project, the indexing technique is used to improve searching hotel by filter as this function takes province.name, hotel.name, hotel.star, and attributes in HotelInfo to search for a hotel.hotel_id. Indexes will be created as follow:

```
CREATE INDEX idx_hotelname ON hotel(name) USING BTREE
```

```
CREATE INDEX idx_provincename ON province(name) USING HASH
```

```
CREATE INDEX idx_hotelstar ON hotel(star) USING BTREE
```

```
CREATE INDEX idx_have_breakfast ON hotelinfo(have_breakfast)  
USING BTREE
```

```
CREATE INDEX idx_free_wifi ON hotelinfo(free_wifi) USING BTREE
```

```
CREATE INDEX idx_have_car_park ON hotelinfo(have_car_park)
USING BTREE
```

```
CREATE INDEX idx_transport_airport ON hotelinfo(transport_airport)
USING BTREE
```

```
CREATE INDEX idx_have_restauran ON hotelinfo(have_restauran)
USING BTREE
```

```
CREATE INDEX idx_have_deposit ON hotelinfo(have_deposit) USING
BTREE
```

```
CREATE INDEX idx_baby_service ON hotelinfo(baby_service) USING
BTREE
```

```
CREATE INDEX idx_have_bar ON hotelinfo(have_bar) USING BTREE
```

```
CREATE INDEX idx_have_laundry ON hotelinfo(have_laundry) USING
BTREE
```

```
CREATE INDEX idx_have_tour ON hotelinfo(have_tour) USING
BTREE
```

```
CREATE INDEX idx_have_spa ON hotelinfo(have_spa) USING BTREE
```

```
CREATE INDEX idx_have_pool ON hotelinfo(have_pool) USING
BTREE
```

The idea of creating 12 different indexes for 12 attributes in the HotelInfo is because when searching using filter, not all 12 attributes will be considered in the filter, therefore, creating index of 12 attributes will restrict the result and leads to fault answer.

For example: `SELECT hotel_id FROM hotel JOIN hotelinfo ON (hotel.hotel_id = hotelinfo.hotel_id) WHERE have_breakfast = 1 AND free_wifi = 1.`

➔ This query only needs to use indexes of have_breakfast and free_wifi so creating seperated indexes is necessary

We have also conducted some results to compare the improvement between using indexes and not using them:

```
CALL FilteredSearch('Hà nội','khách sạn',0,'have_car_park = 1 AND have_deposit  
= 1 AND have_bar = 1')
```

Searching with province.name, hotel.name, hotel.star, and attributes in HotelInfo get result much better at 54% faster (0,0037s comparing to 0,0057s when searching in database without using indexes)

```
CALL FilteredSearch('khách sạn',0,'have_car_park = 1 AND have_deposit = 1  
AND have_bar = 1')
```

Searching with hotel.name and attributes in HotelInfo get a significant result at 559% faster (0,0044 comparing to 0,029s when searching in database without using indexes)

We believe when the database scale up, the improvement will be much more significant

7. Contribution

ERD and RD Designing, Report writing: All members

Nguyễn Tuấn Minh 20184294: model, front-end, use-case, index

Nguyễn Tuấn Dũng 20184247: Crawling data, stored procedure, index

Nguyễn Hoàng Phương 20184302: stored procedure, index, model, front-end