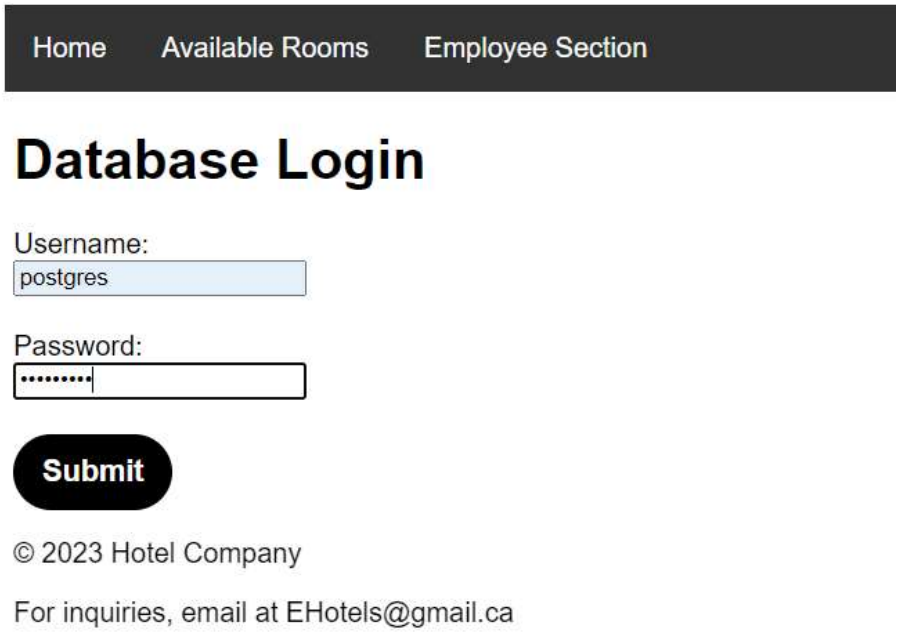
Hotel Service

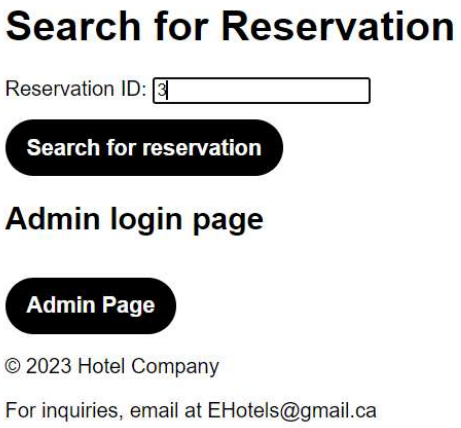
How to use the service

1) First step is to login into the database (username "postgres", password being your password)



Reservation

B) you can go to the "Employee section" this section can be used to search for the agreements



After having searched for a agreement ID, you can either deny, view or accept the agreement

Search Results

Agreement Number	Start Date	End Date	Accept Reservation	Deny Reservation	View Reservation	Left Room
agreement_num: 3	startdate: 2018-01-01	enddate: 2018-12-31	Accept Reservation	Deny Reservation	View Reservation	Left Room

If you decide to view the agreement, you will see information such as customerSSN, room info etc…

Search Results

Agreement Number	Name of the hotel	Room number	Room type	Room price	Room capacity
1	hotel101	1	"sea view"	140	1

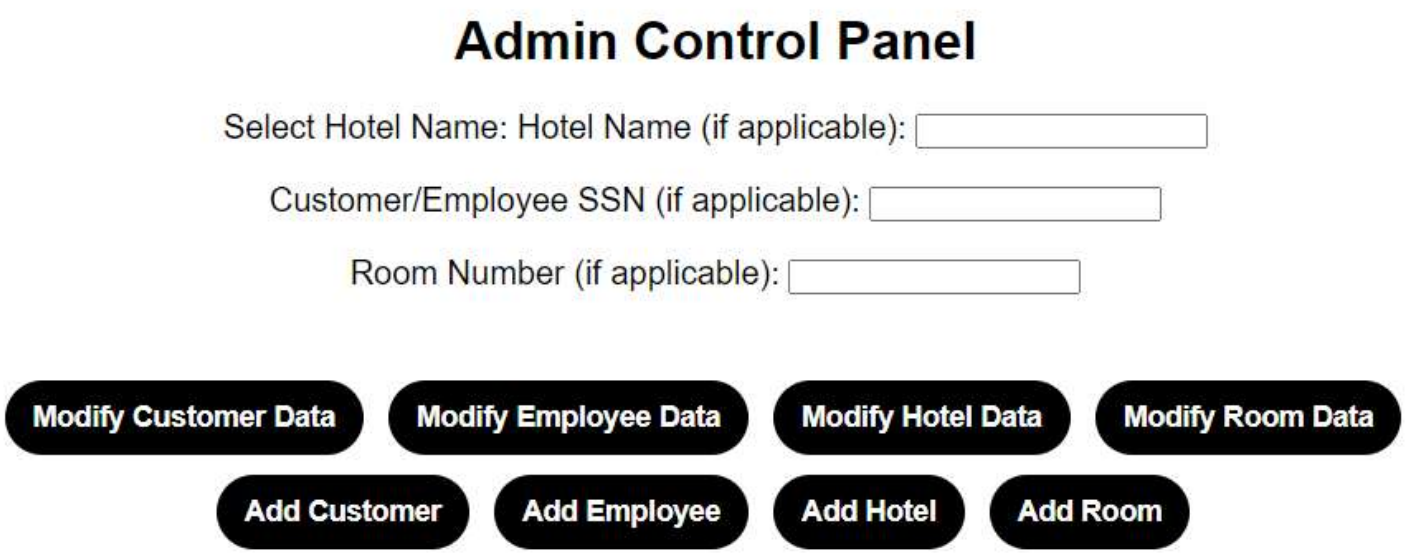
If you decide to accept the agreement, the room will change from ‘Reserved’ to ‘Used’

For inquiries, email at EHotels@gmail.ca

Reservation Accepted

Admin Control Panel

For an Administrator to have access to the database, they must first log in using their credentials. Once in, they will be greeted by the following page:



As per the project requirements, the Admin is able to modify/insert/delete data related to customers, employees, hotels and individual rooms

Unfortunately, as we've had issues using drop-down menus and the like, we have opted to instead rely on a text-based prompt system.

To search for customers, one must only type in the customer's Social Security Number. Any Hotel Name input will be ignored

To search for employees, one must only type in the employee's SSN. Sorting by Hotel/Hotel + SSN was considered but current schema setup and lack of time led us to querying using the employee's SSN

To search for hotels, one must only type in the hotel name.

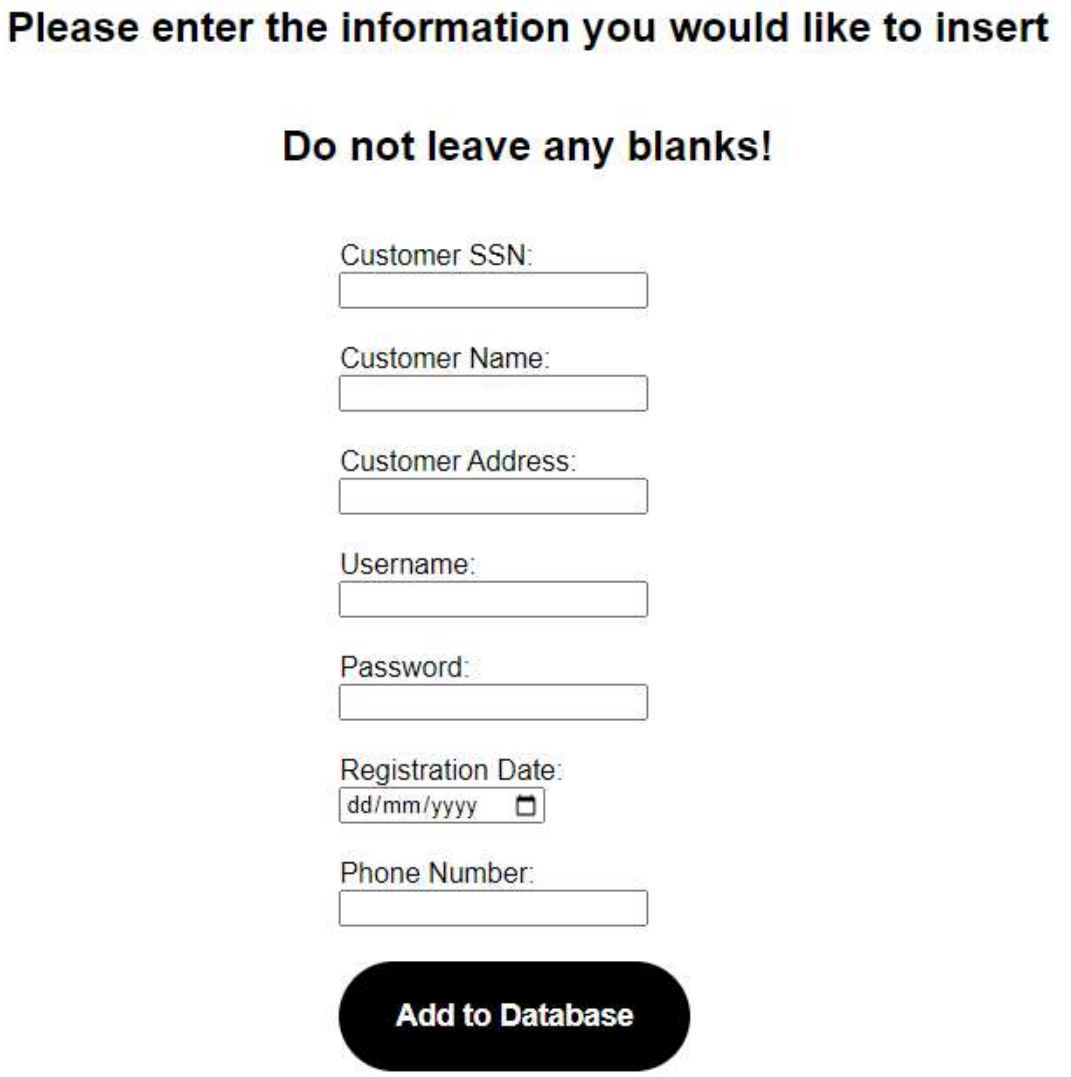
To search for rooms, one must type BOTH Hotel Name and Room Number. This is because Rooms have a multi-variable primary key consisting of Hotel Name & Room Number

An example result is as follows:

Search Results

Customer SSN	Customer Name	Customer Address	Username	Password	Registration Date	Phone Number	Delete from DB
145	customer1	address1	username1	password1	2018-01-01	123456789	Confirm Delete?
Modify SSN	Modify Name	Modify Address	Modify Username	Modify Password	Modify Registration	Modify Phone Number	Delete

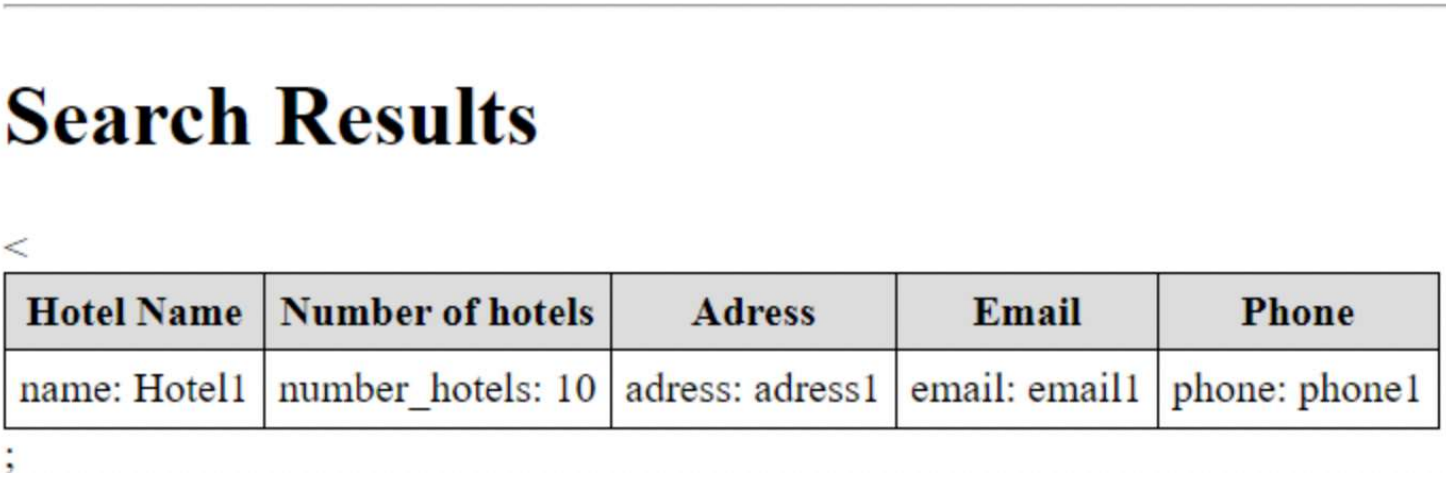
From the result, the Admin can decide to either modify a specific value or delete the entire row (i.e. all data for that object's instance in the DB). Otherwise, the Admin can choose to click on any of the Add X buttons to be taken to a different page to enter and insert data into the DB. Here is an example of such a page:



If the Admin wants to instead modify any data in the DB, they can choose to click on any of the modification buttons and they will be taken to a page similar to the following:

With modified value being Modified X (where X is the variable we want to modify). The box changes to a calendar if the item is of a date data type.

Searching



```
EXAMPLE OF SEARCH QUERY FOR A HOTEL:
SELECT *
FROM search_room
WHERE hotelname = 'hotel1'
```

How the servlet work:

AdminLogInServlet: is used for logging in as the admin
AdminPanelServlet: is used for modifying data
AdminModifyServlet: is used for modifying data
AdminModifyFinishServlet: is used for modifying data
AdminDeleteServlet: is used for modifying data
SignupServlet: used to sign up as a customer
LoginServlet: is used to login as the database (username:postgres, password:UNIQUE)
LogoutServlet: is used to logout of the database (just resets the username and password)
ReservationServlet: Is used to search for the reservationID's, can then use the 3 buttons to deny, accept or view
SearchServlet: is used to search for a specific room

DenyReservationServlet : Change a reservation from ___ to "free"
AcceptReservationServlet : Change a reservation from ___ to "used"
ViewReservationServlet : Allow you to view room and client

How JSDC connects to SQL

Application.java is used for connecting to the database

createTable: Used to create a table

"name" will be the name of the table

"variables" variables will be a string array of the variables in the table in the format (NAME TYPE),(NAME TYPE), ETC... examples: (name VARCHAR(20)), (age INT)

"location" will be the location of the table (schema)

insertIntoTable: Used to insert a row into a table

@param tableName: Name of the table

@param tableCollumn: A array with the name of the collumns

@param values: A array with the values to be inserted (make sure they have '' around them if they are strings, example: 'Bob')

createSchema: create a schema

@param schemaName: Name of the Schema

selectFromTable: Used to get information
@param tableName: Name of table
@param variable: The variables to be selected
@param schema: The location of the schema
@param where: specifics of what your serching for (example: name = 'Bob', age = 20")

updateRow: used to update a row
@param tableName Name of the table
@param schema Name of the schema (Hotels)
@param values values to be changed (ex: room_status = 'free', room_annimities = 'test')
@param where specifics of what your serching for (example: agreement_num = '1')

databaseRefresh: used to delete a database

foreignSelect: used when you want to connect a foreign key to a 2 tables (example in main)
*
* @param tableName Table names
* @param variable Name of column
* @param schema Name of the schema
* @param where specifics of what your serching for (example: name = 'Bob', age = 20")
*/

main: Used to create the database and test some stuff

Refresh.sql is used when you want to delete the whole sql database (used when I was changing the SQL alot)
Project.sql is used when you want to create the whole database, it also creates a bunch of testing items (hotel,rooms,etc)

Test Querys

:
Query 1: Inserting a row into the database

```
String[][] testQuery6 = {{"name", "number_hotels", "adress", "email",
"phone"},{"'Hotel1'","'10'","'adress1'","'email1'","'phone1'"}};

app.insertIntoTable("Hotels", testQuery6[0],testQuery6[1], "Hotels");
```

This would insert into the table “Hotels” in the scheme “Hotels
It would add ‘Hotel1’ as the name of the hotel, ‘10’ as the number of hotels, ‘adress1’ as the address, etc…

Query2: When you want to search with the help of a foreign key
Searching for rooms which are currently under a agreement (used when accepting, denying, viewing a reservation)

```
        String[] tablename = {"agreement", "rooms"};
        String[] select = {"\"Hotels\".agreement.agreement_num",
"\"Hotels\".rooms.hotel_name", "\"Hotels\".rooms.room_num", "\"Hotels\".rooms.room_type",
"\"Hotels\".rooms.room_price", "\"Hotels\".rooms.room_capacity", "\"Hotels\".agreement.ssn"};
        String where[] = {"\"Hotels\".rooms.room_num = \"Hotels\".agreement.room",
"\"Hotels\".rooms.hotel_name = \"Hotels\".agreement.hotel"};

        List<String> allBoockings = app.foreignSelect(tablename, select,"Hotels", where);
```

This would use the tables agreement and rooms
It would only keep the items in the select array
It would do this search in the scheme “Hotels”
It would search for the items for which the room_num is the same as the one in the agreement, and for which they both have the same hotel name
This would print out the following to the database

SELECT ("Hotels".agreement.agreement_num, "Hotels".rooms.hotel_name, "Hotels".rooms.room_num, "Hotels".rooms.room_type, "Hotels".rooms.room_price, "Hotels".rooms.room_capacity, "Hotels".agreement.ssn) **FROM** "Hotels".agreement, "Hotels".rooms **WHERE** "Hotels".rooms.room_num = "Hotels".agreement.room **AND** "Hotels".rooms.hotel_name = "Hotels".agreement.hotel;

If you wanted to get a specific agreement number, then you would then

```
String query = "\"Hotels\".agreement.agreement_num: " + agreementNum;
        for (String s: allBoockings ) {
            if (s.startsWith(query))
             System.out.println(s);
             Foundboocking.add(s);
        }
```

This would try and find a booking with the agreement number which is underlined

Query 3: Updating a row

```
String[][] testQuery2 = {{"room_status = 'reserved'", "room_annimities =
'test3'"},{"agreement_num = '1'", "room_type = 'five'"}};
        app.updateRow("rooms", "Hotels", testQuery2[0], testQuery2[1]);
```

This would update the row for which the table is “rooms” and the schema “Hotels”
It would change the room_status to ‘reserved’ and change a room_annimities to ‘test3’
It would do this change for all agreements that are ‘1’ and for which the room has a capacity of five people
This would print out the following SQL code

UPDATE "Hotels".rooms **SET** room_status = 'reserved', room_annimities = 'test3' **WHERE** agreement_num = '1' **AND** agreement_num = '1';

Query 4: Getting information from the database

```
String[][] testQueryNull = {{"room_num", "room_type", "room_price", "room_capacity",
"room_status", "room_annimities", "hotel_name", "agreement_num"},{"room_status = 'used'"}};

            app.selectFromTable("Rooms", testQueryNull[0], "Hotels", testQueryNull[1] );
```

This would search the table “rooms”, and the schema “Hotels”
It would only keep the information "room_num", "room_type", "room_price", "room_capacity", "room_status", "room_annimities", "hotel_name", "agreement_num"
It would only keep the information from rooms that are currently ‘used’

This is would print the SQL code
SELECT (room_num, room_type, room_price, room_capacity, room_status, room_annimities, hotel_name, agreement_num) **FROM** "Hotels".Rooms **WHERE** room_status = 'used';

Query 5: Deleting from the database

```
String ssnToDelete = "ssne = "+request.getParameter( s: "employeeSSN");
app.deleteFromTable( tableName: "Hotels.customer",ssnToDelete);
```

This would first select the employee with an SSNE equal to ssnToDelete. It will look for said employee in the table “employee” and the schema “Hotels”. Once it finds the employee with the associated ssnToDelete, the employee instance (or row) would be removed from the database.

This code is the equivalent of:
DELETE FROM “Hotels”.employee
Where ssne = ssnToDelete;