**ECE 456 – Assignment 3**

Ankit Srivastava, 20298545

Usman Khan, 20266360

DBMS: PostgreSQL

Programming Lanugage: Java

NOTE: The code segments contained in this document are snippets of the overall code. The full project with all the supporting code has been submitted to the assignment dropbox on LEARN.

Database create script:

-- Drop any existing tables if they exist

DROP TABLE IF EXISTS Booking;

DROP TABLE IF EXISTS Guest;

DROP TABLE IF EXISTS Room;

DROP TABLE IF EXISTS Hotel;

DROP TABLE IF EXISTS BillingLog;

DROP TRIGGER IF EXISTS check\_booking\_conflicts ON Booking;

DROP FUNCTION IF EXISTS raise\_booking\_exception;

-- Create new tables

CREATE TABLE Hotel (

hotelID SERIAL PRIMARY KEY,

hotelName VARCHAR(30),

city CHAR(9) CONSTRAINT city\_constraint CHECK (

city IN ('Guelph', 'Kitchener', 'Waterloo')

)

);

CREATE TABLE Room (

hotelID INTEGER,

roomNo CHAR(4),

price NUMERIC(5, 2),

type CHAR(6),

PRIMARY KEY (hotelID, roomNo),

FOREIGN KEY (hotelID)

REFERENCES Hotel (hotelID)

ON UPDATE RESTRICT

ON DELETE RESTRICT,

CONSTRAINT price\_constraint CHECK (price BETWEEN 50.00 AND 250.00),

CONSTRAINT type\_contraint CHECK (

type IN ('Single', 'Double', 'Queen', 'King')

)

);

CREATE TABLE Guest (

guestID SERIAL PRIMARY KEY,

guestName VARCHAR(30),

guestAddress VARCHAR(50),

guestAffiliation VARCHAR(30)

);

CREATE TABLE Booking (

bookingID SERIAL PRIMARY KEY,

hotelID INTEGER,

roomNo CHAR(4),

guestID INTEGER,

startDate DATE,

endDate DATE,

FOREIGN KEY (hotelID, roomNo)

REFERENCES Room (hotelID, roomno)

ON UPDATE RESTRICT

ON DELETE RESTRICT,

FOREIGN KEY (guestID)

REFERENCES Guest (guestID)

ON UPDATE RESTRICT

ON DELETE RESTRICT

);

CREATE TABLE BillingLog (

bookingID INTEGER,

hotelName VARCHAR(30),

city CHAR(9),

roomNo CHAR(4),

guestName VARCHAR(30),

guestAddress VARCHAR(50),

type CHAR(6),

price NUMERIC(5, 2),

startDate DATE,

endDate DATE,

numberOfDaysStayed INTEGER,

total NUMERIC(10, 2)

);

-- Function to call on INSERT/UPDATE which raises an exception if a booking

-- conflict is found.

CREATE OR REPLACE FUNCTION raise\_booking\_exception() RETURNS TRIGGER AS $$

BEGIN

IF TG\_OP = 'INSERT'

AND (SELECT COUNT(\*)

FROM booking

WHERE hotelid = NEW.hotelid

AND roomno = NEW.roomno

AND (

(NEW.startdate >= startdate AND NEW.startdate < enddate)

OR (NEW.enddate > startdate AND NEW.enddate <= enddate)

OR (NEW.startdate < startdate AND NEW.enddate > enddate)

OR (NEW.startdate = startdate AND NEW.enddate = enddate)

)

) = 0 THEN

RETURN NEW;

ELSIF TG\_OP = 'UPDATE'

AND (SELECT COUNT(\*)

FROM booking

WHERE hotelid = NEW.hotelid

AND roomno = NEW.roomno

AND (

(NEW.startdate >= startdate AND NEW.startdate < enddate)

OR (NEW.enddate > startdate AND NEW.enddate <= enddate)

OR (NEW.startdate < startdate AND NEW.enddate > enddate)

OR (NEW.startdate = startdate AND NEW.enddate = enddate)

)

-- Filter out the updated row itself

AND NOT (

hotelid = OLD.hotelid

AND roomno = OLD.roomno

AND guestid = OLD.guestid

AND startdate = OLD.startdate

AND enddate = OLD.enddate

)

) = 0 THEN

RETURN NEW;

ELSE

RAISE EXCEPTION USING

errcode = 'BADBK',

message = 'Conflict with existing booking';

END IF;

END;

$$ LANGUAGE plpgsql;

-- Insert/Update Trigger

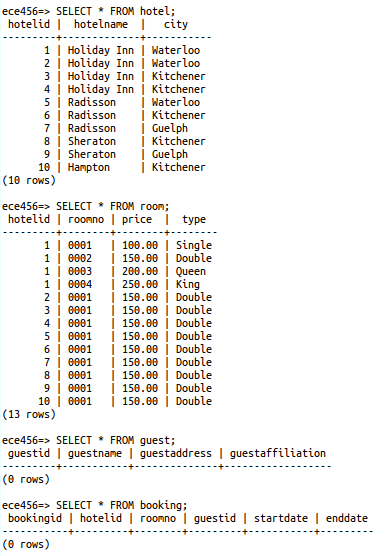
CREATE TRIGGER check\_booking\_conflicts

BEFORE INSERT OR UPDATE ON booking

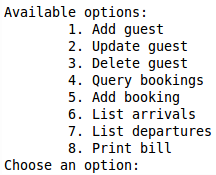
FOR EACH ROW

EXECUTE PROCEDURE raise\_booking\_exception();

Initial database tables:



Initial console output:



**Module 1 – Code:**

// ----- Module 1: Guest Registration -----

/\*\*

\* Adds a new guest.

\* **@param** guestName The full name of the guest.

\* **@param** guestAddress The full address.

\* **@param** guestAffiliation A guest's affiliation.

\* **@throws** SQLException

\*/

**public** **int** addGuest(String guestName, String guestAddress,

String guestAffiliation) **throws** SQLException {

Connection connection = **null**;

PreparedStatement preparedStatement = **null**;

**int** numGuestsAdded = 0;

**try** {

connection = databaseManager.getConnection();

preparedStatement = connection.prepareStatement(

"INSERT INTO guest " +

"(guestName, guestAddress, guestAffiliation) " +

"VALUES (?, ?, ?)");

preparedStatement.setString(1, guestName);

preparedStatement.setString(2, guestAddress);

preparedStatement.setString(3, guestAffiliation);

numGuestsAdded = preparedStatement.executeUpdate();

} **finally** {

databaseManager.closeQuietly(preparedStatement);

databaseManager.closeQuietly(connection);

}

**return** numGuestsAdded;

}

/\*\*

\* Updates the name, address, affiliation of a guest.

\* **@param** guestID The ID of the guest to update.

\* **@param** guestName The new name of the guest.

\* **@param** guestAddress The new address of the guest.

\* **@param** guestAffiliation The new affiliation of the guest.

\* **@return** The number of guests updated.

\* **@throws** SQLException

\*/

**public** **int** updateGuest(**int** guestID, String guestName, String guestAddress,

String guestAffiliation) **throws** SQLException {

**int** numUpdated = 0;

Connection connection = **null**;

PreparedStatement preparedStatement = **null**;

**try** {

connection = databaseManager.getConnection();

preparedStatement = connection.prepareStatement(

"UPDATE guest SET " +

"guestName = ?, " +

"guestAddress = ?, " +

"guestAffiliation = ? " +

"WHERE guestID = ?");

preparedStatement.setString(1, guestName);

preparedStatement.setString(2, guestAddress);

preparedStatement.setString(3, guestAffiliation);

preparedStatement.setInt(4, guestID);

numUpdated = preparedStatement.executeUpdate();

} **finally** {

databaseManager.closeQuietly(preparedStatement);

databaseManager.closeQuietly(connection);

}

**return** numUpdated;

}

/\*\*

\* Deletes a guest.

\* **@param** guestID The ID of the guest to be deleted.

\* **@return** The number of guests deleted.

\* **@throws** SQLException

\*/

**public** **int** deleteGuest(**int** guestID) **throws** SQLException {

**int** numDeleted = 0;

Connection connection = **null**;

PreparedStatement preparedStatement = **null**;

**try** {

connection = databaseManager.getConnection();

preparedStatement = connection.prepareStatement(

"DELETE FROM guest WHERE guestID = ?");

preparedStatement.setInt(1, guestID);

numDeleted = preparedStatement.executeUpdate();

} **finally** {

databaseManager.closeQuietly(preparedStatement);

databaseManager.closeQuietly(connection);

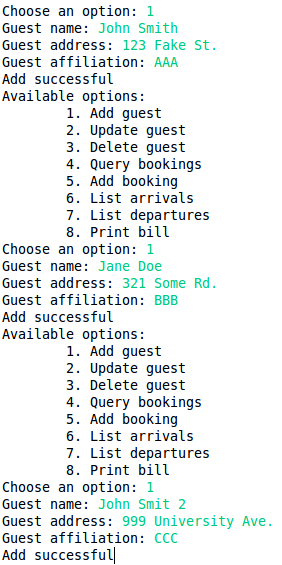
}

**return** numDeleted;

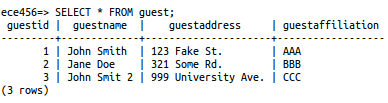
}

**Module 1 – Output:**

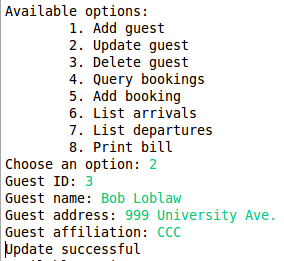
The following shows the additions of 3 guests to the database.



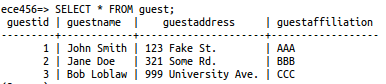
The database after the 3 guests have been added successfully.



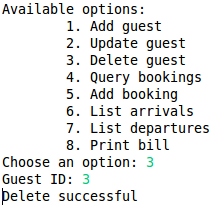
Updating the name of guest with guestID = 3.



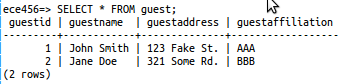
Database after the successful update operation.



Deleting guest with guestID = 3.



Database after successful delete operation.



**Module 2 – Code:**

**// ----- Module 2: Booking Query -----**

/\*\*

\* Returns available rooms. Paramters set to null will not be considered

\* when retrieving the list of available rooms.

\* **@param** startDate The start date of the desired time interval.

\* **@param** endDate The end date of the desired time interval.

\* **@param** hotelName The name of the hotel to look for rooms in.

\* **@param** city The city within which to limit the search.

\* **@param** price The price of available rooms.

\* **@param** type The type of the available rooms ("King", "Queen", "Double",

\* "Single")

\* **@return** A List of Map objects, where the map key value pairs are the

\* column name and column value.

\* **@throws** SQLException

\*/

**public** List<Map<String, Object>> getAvailableRooms(String startDate,

String endDate,

String hotelName,

String city,

Double price,

String type)

**throws** SQLException {

Connection connection = **null**;

Statement satement = **null**;

ResultSet resultSet = **null**;

List<Map<String, Object>> resultList = **null**;

// List of conditions that make up the WHERE clause of query

List<String> clauseList = **new** ArrayList<String>();

// Construct query

StringBuilder query = **new** StringBuilder("SELECT hotel.hotelID, ");

query.append("hotel.hotelName, hotel.city, room.roomNo, room.price, ");

query.append("room.type ");

query.append("FROM room ");

query.append("INNER JOIN hotel ON (hotel.hotelID = room.hotelID) ");

**if** (!startDate.isEmpty() && !endDate.isEmpty()) {

StringBuilder subquery = **new** StringBuilder();

subquery.append("NOT EXISTS (SELECT \* ");

subquery.append("FROM booking ");

subquery.append("WHERE booking.hotelID = hotel.hotelID ");

subquery.append("AND booking.roomNo = room.roomNo ");

subquery.append("AND ((booking.startDate BETWEEN '");

subquery.append(startDate);

subquery.append("' AND '");

subquery.append(endDate);

subquery.append("' OR booking.endDate BETWEEN '");

subquery.append(startDate);

subquery.append("' AND '");

subquery.append(endDate);

subquery.append("') OR (");

subquery.append("booking.startDate <= '");

subquery.append(startDate);

subquery.append("' AND endDate >= '");

subquery.append(endDate);

subquery.append("')))");

clauseList.add(subquery.toString());

}

**if** (!hotelName.isEmpty())

clauseList.add("hotel.hotelName = '" + hotelName + "'");

**if** (!city.isEmpty())

clauseList.add("hotel.city = '" + city + "'");

**if** (price != **null**)

clauseList.add("room.price = " + price);

**if** (!type.isEmpty())

clauseList.add("room.type = '" + type + "'");

**if** (clauseList.size() > 0) {

query.append("WHERE ");

query.append(Util.*concatWithAnd*(clauseList));

}

**try** {

connection = databaseManager.getConnection();

satement = connection.createStatement();

resultSet = satement.executeQuery(query.toString());

resultList = Util.*convertResultSetToList*(resultSet);

} **finally** {

databaseManager.closeQuietly(resultSet);

databaseManager.closeQuietly(satement);

databaseManager.closeQuietly(connection);

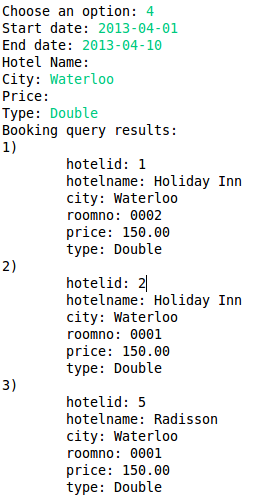
}

**return** resultList;

}

**Module 2 – Output:**

Booking query. Fields which are left blank are not considered when querying for available rooms.



**Module 3 – Code:**

**// ----- Module 3: Booking Registration -----**

/\*\*

\* Adds a new booking for a hotel room.

\* **@param** hotelID The ID of the hotel.

\* **@param** roomNo The room number of the room being booked.

\* **@param** guestID The ID of the guest.

\* **@param** startDate The start date of the booking.

\* **@param** endDate The end date of the booking.

\* **@return** The bookingID of the newly created booking.

\* **@throws** SQLException

\*/

**public** **int** addBooking(**int** hotelID, String roomNo, **int** guestID,

Date startDate, Date endDate) **throws** SQLException {

**int** newBookingID;

Connection connection = **null**;

PreparedStatement preparedStatement = **null**;

ResultSet rs = **null**;

**try** {

connection = databaseManager.getConnection();

preparedStatement = connection.prepareStatement(

"INSERT INTO booking " +

"(hotelID, roomNo, guestID, startDate, endDate) " +

"VALUES (?, ?, ?, ?, ?)",

PreparedStatement.*RETURN\_GENERATED\_KEYS*);

preparedStatement.setInt(1, hotelID);

preparedStatement.setString(2, roomNo);

preparedStatement.setInt(3, guestID);

preparedStatement.setDate(4, startDate);

preparedStatement.setDate(5, endDate);

preparedStatement.executeUpdate();

// Get bookingID of newly inserted row

rs = preparedStatement.getGeneratedKeys();

rs.next();

newBookingID = rs.getInt(1);

} **finally** {

databaseManager.closeQuietly(rs);

databaseManager.closeQuietly(preparedStatement);

databaseManager.closeQuietly(connection);

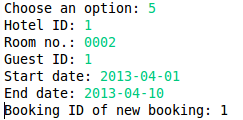
}

**return** newBookingID;

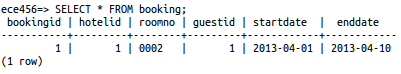
}

**Module 3 – Output:**

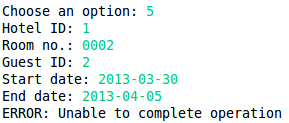
Adding a booking.



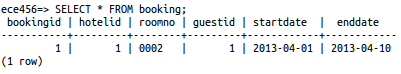
Database after successfully adding a new booking.



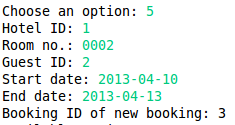
Trying to add a new booking that conflicts with an existing booking. A trigger on the booking table is used for detecting and rejecting conflicting bookings.



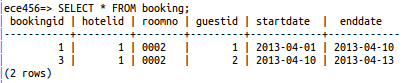
Database after the failed booking add operation.



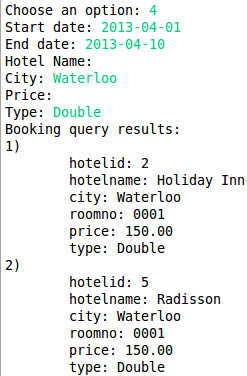
Adding another booking that does NOT conflict with any existing bookings.



Database after the 2nd booking was added successfully.



Running the same booking query that was run in the Module 2 section now no longer show the room that has been booked. (i.e. Number of available rooms went from 3 to 2 since one of the rooms is now booked)



**Module 4 – Code:**

**// ----- Module 4: Room Maintenance and Billing -----**

/\*\*

\* Returns a list of arrivals for the given date.

\* **@param** hotelID The ID of the hotel.

\* **@param** date The date we are checking arrivals for.

\* **@return** A List of Map object containing the arrivals.

\* **@throws** SQLException

\*/

**public** List<Map<String, Object>> getArrivals(**int** hotelID, Date date)

**throws** SQLException {

Connection connection = **null**;

PreparedStatement preparedStatement = **null**;

ResultSet resultSet = **null**;

List<Map<String, Object>> resultList = **null**;

**try** {

connection = databaseManager.getConnection();

preparedStatement = connection.prepareStatement(

"SELECT booking.bookingID, guest.guestID, " +

"hotel.hotelName, booking.roomNo, guest.guestName " +

"FROM booking " +

"INNER JOIN hotel ON hotel.hotelID = booking.hotelID " +

"INNER JOIN guest on guest.guestID = booking.guestID " +

"WHERE booking.hotelID = ? AND booking.startDate = ?");

preparedStatement.setInt(1, hotelID);

preparedStatement.setDate(2, date);

resultSet = preparedStatement.executeQuery();

resultList = Util.*convertResultSetToList*(resultSet);

} **finally** {

databaseManager.closeQuietly(resultSet);

databaseManager.closeQuietly(preparedStatement);

databaseManager.closeQuietly(connection);

}

**return** resultList;

}

/\*\*

\* Returns a list of departures for the given date.

\* **@param** hotelID The ID of the hotel.

\* **@param** date The date we are checking departures for.

\* **@return** A List of Map object containing the departures.

\* **@throws** SQLException

\*/

**public** List<Map<String, Object>> getDepartures(**int** hotelID, Date date)

**throws** SQLException {

Connection connection = **null**;

PreparedStatement preparedStatement = **null**;

ResultSet resultSet = **null**;

List<Map<String, Object>> resultList = **null**;

**try** {

connection = databaseManager.getConnection();

preparedStatement = connection.prepareStatement(

"SELECT booking.bookingID, guest.guestID, " +

"hotel.hotelName, booking.roomNo, guest.guestName " +

"FROM booking " +

"INNER JOIN hotel ON hotel.hotelID = booking.hotelID " +

"INNER JOIN guest on guest.guestID = booking.guestID " +

"WHERE booking.hotelID = ? AND booking.endDate = ?");

preparedStatement.setInt(1, hotelID);

preparedStatement.setDate(2, date);

resultSet = preparedStatement.executeQuery();

resultList = Util.*convertResultSetToList*(resultSet);

} **finally** {

databaseManager.closeQuietly(resultSet);

databaseManager.closeQuietly(preparedStatement);

databaseManager.closeQuietly(connection);

}

**return** resultList;

}

**public** List<Map<String, Object>> getBill(**int** bookingID)

**throws** SQLException {

Connection connection = **null**;

PreparedStatement preparedStatement = **null**;

ResultSet resultSet = **null**;

List<Map<String, Object>> resultList = **null**;

**try** {

connection = databaseManager.getConnection();

// Retrieve billing information

preparedStatement = connection.prepareStatement(

"SELECT booking.bookingID, hotel.hotelName, hotel.city, " +

"room.roomNo, guest.guestName, guest.guestAddress, " +

"room.type, room.price, " +

"booking.startDate, booking.endDate, " +

"booking.endDate - booking.startDate AS numberOfDaysStayed, " +

"(booking.endDate - booking.startDate) \* price AS total " +

"FROM booking " +

"INNER JOIN hotel ON hotel.hotelID = booking.hotelID " +

"INNER JOIN room ON room.hotelID = booking.hotelID AND " +

"room.roomNo = booking.roomNo " +

"INNER JOIN guest ON guest.guestID = booking.guestID " +

"WHERE booking.bookingID = ?");

preparedStatement.setInt(1, bookingID);

resultSet = preparedStatement.executeQuery();

resultList = Util.*convertResultSetToList*(resultSet);

// Log billing information

preparedStatement = connection.prepareStatement(

"INSERT INTO billinglog (" +

"SELECT booking.bookingID, hotel.hotelName, hotel.city, " +

"room.roomNo, guest.guestName, guest.guestAddress, " +

"room.type, room.price, " +

"booking.startDate, booking.endDate, " +

"booking.endDate - booking.startDate AS numberOfDaysStayed, " +

"(booking.endDate - booking.startDate) \* price AS total " +

"FROM booking " +

"INNER JOIN hotel ON hotel.hotelID = booking.hotelID " +

"INNER JOIN room ON room.hotelID = booking.hotelID AND " +

"room.roomNo = booking.roomNo " +

"INNER JOIN guest ON guest.guestID = booking.guestID " +

"WHERE booking.bookingID = ? " +

")");

preparedStatement.setInt(1, bookingID);

preparedStatement.executeUpdate();

} **finally** {

databaseManager.closeQuietly(resultSet);

databaseManager.closeQuietly(preparedStatement);

databaseManager.closeQuietly(connection);

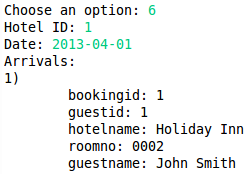
}

**return** resultList;

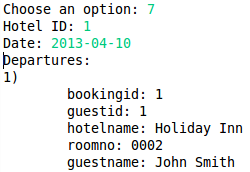
}

**Module 4 – Output:**

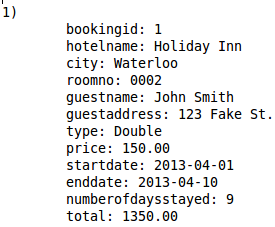
Listing all arrivals for the day 2013-04-01 for hotel with hotelID = 1.



Listing all departures for the day 2013-04-10 for hotel with hotelID = 1.



Printing the bill for bookingID = 1.



Every time a bill is generated, all relevant fields are logged in the BillingLog table. The following shows the corresponding entry in the BillingLog table for the bill above.

