Hotel Management System

For Wolf Inns, a popular Hotel Chain

CSC 540 Database Systems

Project Report #3

Srilatha Bekkem

How to compile and run:
There are 5 java files i.e. Main.java, Information_Processing.java, Service.java, Reports.java, Billing_func.java.

1. From local command prompt :

> pscp <Local Path where the files are kept>/<Filename>.javar <unityid>@remote.eos.ncsu.edu:

2. In eos:

javac Information_Processing.java
javac Service.java
javac Reports.java
javac Billing_func.java
javac Main.java
java Main

General Structure :

Our codes are modular. Each of the major parts of the functionalities are written in separate classes. All these classes are accessed from the Main class.

In the section below , we put brief description of each of the classes :

1. Main.java: This class has main method which calls other classes i.e. Information_Processing,Billing_func,Services,Reports.
2. Information_Processing.java:This class does information processing functionalities i.e. insert/update/delete information about hotel/room/staff/customer and checking available rooms, checking requested rooms are available or not, assigning and releasing rooms.
3. Billing_func.java:This class does billing functionalities i.e. to generate bill and print itemized receipt.
4. Services.java:This class does service functionalities i.e. insert/update/delete service and to enter service used by customer, enter the staff id serving the customer
5. Reports.java: This class does reports functionalities i.e. to report occupancy by city, hotel, room type and date range. To show information of staff grouped by their role. To list the staff that served a particular customer during their stay. To generate revenue earned by the hotel given specific date range.

Application of Transactions

Transaction Control (Billing Section)
In the Billing Section, transaction control has been implemented in the following manner .

If the payment mode is selected as credit, then the system will first validate the existence of the card in the system, match its details with the customer name and then then apply 5% discount to the final bill amount .
If all these functionalities work fine, then only we commit the transaction or else we perform a rollback.

Code in billing that consists transaction:

```
if (bill_amt <= balance) {
      str = "UPDATE bill,card"
      + " SET bill.discount=5, bill.Payment_type='hotel credit
        card', bill.total= bill.amount-(bill.amount*5/100), "
      + " card.balance=
card.Balance-(bill.amount-(bill.amount*5/100))"
      + " where bill.customer_id=? and card_no=?;";
      try {
            //setting auto commit to false
            conn.setAutoCommit(false);
            stmt = conn.prepareStatement(str);
            stmt.setInt(1, customer_id);
            stmt.setLong(2, card_num_ver);
            action = stmt.executeUpdate();


            /*
            * After payment via credit card, updating the
            * credit balance. Transaction
            * control implemented here
            */
            str = "update customer set card_no=? where
customer_id=?;";
            try {
            stmt = conn.prepareStatement(str);
            stmt.setLong(1, card_num_ver);
            stmt.setInt(2, customer_id);
            action = stmt.executeUpdate();
            } catch (SQLException e) {
            e.printStackTrace();
```

```
        }
        //committing the transaction
        conn.commit();
        //setting auto commit to true
        conn.setAutoCommit(true);
        System.out.println("the transaction happened successfully");
        action = 0;
} catch (SQLException e) {
        if (conn != null) {
        try {
              System.out.println("Failed to Update Bill with
              Paying through Credit Card Info");
              //rolling back in case of any error
              conn.rollback();
              //and then auto committing it
              conn.setAutoCommit(true);
              return;
              } catch (SQLException e1) {

                    e.printStackTrace();
                    return;
              }
        }
}
```

Transaction Control (Information Processing section)

In the Information Processing section, transaction control has been implemented in the following manner .

1. Entering the customer information i.e. basic details and the room in which customer checks in.
2. the front desk staff who assigns the room to customer-- a record of is inserted into serves i.e. the staff serving customer
3. checking if the room is presidential
4. if presidential then assigned dedicated catering and service staff
5. checking if the catering and service staff are not assigned to other presidential rooms.

If any one of the above fails then the customer record is not entered into the database and statements are rolled back otherwise commits and sets auto commit to true.

Code in information processing that consists transaction:

```
str = "insert into customer (name, dob, phone_number, email,
```

```
ssn, room_no, hotel_id, "
+ "guest_count, check_in, start_date)
 values(?,?,?,?,?,?,?,?,?,?)";
try {
      //setting auto commit to false
      conn.setAutoCommit(false);
      stmt = conn.prepareStatement(str);
      stmt.setString(1, customer_name);
      stmt.setString(2, dob);
      stmt.setLong(3, phone_no);
      stmt.setString(4, email_id);
      stmt.setInt(5, ssn);
      stmt.setInt(6, room_no);
      stmt.setInt(7, hotel_id);
      stmt.setInt(8, guest_count);
      stmt.setString(9, time);
      stmt.setString(10, date);
      action = 0;
      action = stmt.executeUpdate();
if (action == 1)
      System.out.println("inserted successfully");

if (action == 1) {
      try {
            // to return the generated customer_id for
            //the customer
            str = "select customer_id from customer where
            name=? and dob=? and phone_number=? and room_no=?
            and hotel_id=?";
            stmt = conn.prepareStatement(str);
            stmt.setString(1, customer_name);
            stmt.setString(2, dob);
            stmt.setLong(3, phone_no);
            stmt.setInt(4, room_no);
            stmt.setInt(5, hotel_id);
            rs = stmt.executeQuery();
            while (rs.next()) {
                  customer_id = rs.getInt(1);
                  System.out.println("the Id of the customer:  "
                  + rs.getInt(1));
                  System.out.println(" The customer booked the
                  room successfully");
            }
```

```
}catch (SQLException e) {
      if (conn != null) {
            try {
                  System.out.println("there is some
                  issue with details of customer");
                  //rolling back in case of any error
                  conn.rollback();
                  //and then auto committing it
                  conn.setAutoCommit(true);
                  return;
            } catch (SQLException e1) {

                  e.printStackTrace();
                  return;
            }
      }
}
// insert into serves the front desk representative id
//and customer id
str = "insert ignore into serves
      (staff_id,customer_id) values(?,?)";
try {
      stmt = conn.prepareStatement(str);
      stmt.setInt(1, staff_id);
      stmt.setInt(2, customer_id);
      stmt.executeUpdate();
} catch (SQLException e) {
      if (conn != null) {
            try {
                  System.out.println("there is issue with
                  the staff id entered");
                  //rolling back in case of any error
                  conn.rollback();
                  //and then auto committing it
                  conn.setAutoCommit(true);
                  return;
            } catch (SQLException e1) {

                  e.printStackTrace();
                  return;
            }
      }
}
```

```
            // checking if customer has booked presidential
            str = "select category_name from room where room_no=?
            and hotel_id=?";
            try {
                  stmt = conn.prepareStatement(str);
                  stmt.setInt(1, room_no);
                  stmt.setInt(2, hotel_id);
                  rs = stmt.executeQuery();
                  if (rs.next()) {
                        // the room customer booked presidential room,
                        // so catering and service staff
                        // must be assigned
                        if (rs.getString(1).equals("presidential")) {
                              System.out.println("Assign catering
                              staff and service staff to the
                        customer:");
                              System.out.println("Enter the
following:");
                              System.out.println("catering staff Id: ");
                              catering = reader.nextInt();
                              System.out.println("service staff Id: ");
                              service = reader.nextInt();
                              str = "update presidential
                              set catering_staff_id=?,
                              service_staff_id=? where room_no=? and
                              hotel_id=?";
                              try {
                                    stmt = conn.prepareStatement(str);
                                    stmt.setInt(1, catering);
                                    stmt.setInt(2, service);
                                    stmt.setInt(3, room_no);
                                    stmt.setInt(4, hotel_id);
                                    action = stmt.executeUpdate();
                                    System.out.println("successfully
                                    assigned staff");
                              } catch (SQLException e) {
                                    if (conn != null) {
                                          try {
                                                System.out.println("
                                                there is issue setting
                                                presidential room
                                                dedicated staff");
                                                //rolling back in case
```

```
                                        //of any error
                                        conn.rollback();
                                        //and then auto
                                        //committing it
                                        conn.setAutoCommit(true);
                                        return;
                                  } catch (SQLException e1) {

                                        e.printStackTrace();
                                        return;
                                  }
                            }
                      }
                      if (action == 1) {
                            try {
                                  // inserting catering staff
                                  // id and customer id in serves
                                  str = "insert ignore into
                                  serves (staff_id,customer_id)
                                  values(?,?)";

                            stmt = conn.prepareStatement(str);
                                  stmt.setInt(1, catering);
                                  stmt.setInt(2, customer_id);
                                  stmt.executeUpdate();
                            } catch (SQLException e) {
                                  if (conn != null) {
                                        try {

System.out.println("there is issue with entering record into
serves table");
//rolling back in case of any error
conn.rollback();
//and then auto committing it

conn.setAutoCommit(true);
return;
} catch (SQLException e1) {

      e.printStackTrace();
      return;
      }
}
```

```
}
try {
      // inserting service staff id and customer id in serves
      str = "insert ignore into serves (staff_id,customer_id)
values(?,?)";
      stmt = conn.prepareStatement(str);
      stmt.setInt(1, service);
      stmt.setInt(2, customer_id);
      stmt.executeUpdate();
} catch (SQLException e) {
      if (conn != null) {
            try {
                  System.out.println("there is issue with
                  entering record into serves table");
                  //rolling back in case of any error
                  conn.rollback();
                  //and then auto committing it
                  conn.setAutoCommit(true);
                  return;
            } catch (SQLException e1) {

                  e.printStackTrace();
                  return;
            }
      }
}}}}

} catch (SQLException e) {
      if (conn != null) {
            try {
                  System.out.println("there is issue with querying
                  the database about the room type");
                  //rolling back in case of any error
                  conn.rollback();
                  //and then auto committing it
                  conn.setAutoCommit(true);
                  return;
            } catch (SQLException e1) {

                  e.printStackTrace();
                  return;
            }
      }
```

```
}}
//committing the transaction
conn.commit();
//setting autocommit to true
conn.setAutoCommit(true);
} catch (SQLException e) {
      if (conn != null) {
            try {
                  System.out.println("there is issue with the details
                  of the customer");
                  //rolling back in case of any error
                  conn.rollback();
                  //and then auto committing it
                  conn.setAutoCommit(true);
                  return;
            } catch (SQLException e1) {

                  e.printStackTrace();
                  return;
            }
      }
}
```

High Level Design Decisions

Billing Section

This particular section deals with printing itemized bill and performing calculations for final bill amount after taking the mode of payment as input from the customers.

We will first take the customer_id as input, that is the ID of the customer for whom the billing tasks will be performed.

Switch cases have been used to perform the rest of the Billing tasks.

If the customer just wants to print the itemised bill, then the code just performs that actions, and prints the itemized bill of room, services etc.

If the customer wants to have the final bill calculation, then the system first asks for the mode of payment (credit / debit/cash) .

If the payment is via hotel credit card, then we perform card number validation and balance check in the system. If everything matches, then we apply 5% discount on the initial bill amount, then update the system with the same. In this case, we also reduce the card balance by the specific amount and also update the card in the customer table.We also update the payment_type of the customer as hotel credit card.

While updating the card, in the customer table, we have implemented transaction control. That is, if the bill payment via credit card is successful, then only we commit the transaction and update the card details in customer or else we rollback the connection.

For debit or cash mode of payment, we do not apply any discount and update the payment mode as debit .

```
                /*
                 * If the end date is found to be not null, that is,
the customer has already
                 * checked out, then only we proceed with the final
bill calculation. Here we
                 * are not taking into account any other
consideration other than end date not
                 * equals null.
                 */
                if (end_date != null) {
                     /*
                      * Calculating the initial bill amount before
any payment method got selected.
                      */
                     System.out.println("Calculating Initial Bill
Amount.....");
                     str = "UPDATE bill SET bill.amount= (select
rate*(datediff(End_date,Start_date)) from customer natural join room
where"
                                  + " customer_id=?)+(select
coalesce(sum(amount),0) from uses where customer_id=?),
bill_date=(select end_date"
                                  + " from customer where
customer_id=?) WHERE customer_id =?;";
                     try {

                          stmt = conn.prepareStatement(str);

                          stmt.setInt(1, customer_id);
```

```
                              stmt.setInt(2, customer_id);
                              stmt.setInt(3, customer_id);
                              stmt.setInt(4, customer_id);
                              action = stmt.executeUpdate();
                              if (action == 1) {
                                   System.out.println("Initial Billing
Calculation Performed");

                              }
                              action = 0;
                         } catch (SQLException e) {
                              e.printStackTrace();
                              System.out.print("Initial Billing Amount
Calculation failed.");

                         }

                         /*
```

Information Processing:

We have used switch block for the user to choose from the listed functionalities information processing does.

When user chooses option 10 i.e. entering a customer, along with customer details, room number and hotel id are asked for check in. If the room is available then entry is made. If the room is presidential then the user needs to assign catering staff and service staff.

TRIGGERS:

```
-- trigger before insert into staff to check that a hotel a hotel has
only one manager

delimiter //
create trigger before_insert_staff
before insert on staff
for each row
begin
if new.title='manager' then
```

```
if(exists(select * from staff where title='manager' and
hotel_id=new.hotel_id)) then
signal sqlstate '45000' set message_text= 'A manager for this hotel
is already present.
To add a new manager delete the old manager';
end if;
end if;
end;

-- trigger after insert into staff, to insert record into catering
staff / service staff if the staff role is catering or service
respectively

delimiter //
create trigger insert_staff
after insert on staff
for each row
begin
if new.title='catering' then
insert into catering_staff values (new.staff_id);
end if;
if new.title='service' then
insert into service_staff values(new.staff_id);
end if;
end;

-- trigger before insert into room, to calculate the nightly rate of
the room

delimiter //
create trigger insert_before_room
before insert on room
for each row
begin
set new.availability=1;
set new.rate = (select a.costfactor+b.costfactor from
(select costfactor from location where city=(select city from hotel
where hotel_id=new.hotel_id) )as a,
(select costfactor from cost where (category_name,max_limit)=(select
category_name,max_limit
from cost where category_name=new.category_name and
max_limit=new.max_limit) )as b);
end;
```

```
-- trigger after insert into room, to insert a record into
presidential if the room is of presidential category

delimiter //
create trigger insert_after_room
after insert on room
for each row
begin
if new.category_name = 'presidential' then
insert into presidential(room_no,hotel_id) values(new.room_no,
new.hotel_id);
end if;
end;

-- trigger after update on cost, to update the cost of the room when
costfactor of the room type and occupancy changes

delimiter //
create trigger update_cost
after update on cost
for each row
begin
update room,(select
room_no,hotel_id,location.costfactor+cost.costfactor as costfactor
from room
natural join hotel natural join location join cost
using(max_limit,category_name) where
cost.max_limit=new.max_limit and
cost.category_name=new.category_name)as a
set rate=a.costfactor
where room.room_no=a.room_no and room.hotel_id=a.hotel_id;
end;


-- trigger after updating location, to update the nightly rate of the
room when the cost factor in the location changes.

delimiter //
create trigger update_location
after update on location
for each row
begin
```

```
update room,(select
room_no,hotel_id,location.costfactor+cost.costfactor as costfactor
from
room natural join hotel natural join location join cost
using(max_limit,category_name) where
location.city=new.city)as a
set rate=a.costfactor
where room.room_no=a.room_no and room.hotel_id=a.hotel_id;
end;

-- trigger before update on room, to update the nightly rate as the
occupancy and room type changes, and to insert/delete a room if it is
presidential/not after change.

delimiter //
create trigger update_room
before update on room
for each row
begin
if old.max_limit<> new.max_limit or
old.category_name<>new.category_name then
set new.rate = (select a.costfactor+b.costfactor from
(select costfactor from location where city=(select city from hotel
where hotel_id=new.hotel_id) )as a,
(select costfactor from cost where (category_name,max_limit)=(select
category_name,max_limit
from cost where category_name=new.category_name and
max_limit=new.max_limit) )as b);
end if;
if old.category_name='presidential' and
(old.category_name <> new.category_name) and
(exists(select * from presidential where room_no=new.room_no and
hotel_id=new.hotel_id)) then
delete from presidential where room_no=new.room_no and
hotel_id=new.hotel_id;
elseif (new.category_name='presidential') and
(not exists(select * from presidential where room_no=new.room_no and
hotel_id=new.hotel_id))then
insert into presidential(room_no,hotel_id)
values(new.room_no,new.hotel_id);
end if;
end;
```

```
-- trigger after customer_insert, to change the availability of the
room to no and insert a new record into bill.

delimiter //
create trigger insert_customer
after insert on customer
for each row
begin
declare price float;
select rate into price from room where room_no=new.room_no and
hotel_id=new.hotel_id;
update room set availability=0 where room_no=new.room_no and
hotel_id=new.hotel_id;
insert into bill (customer_id,rate) values(new.customer_id,price);
end;



-- trigger before insert customer, to check if the no of guests are
not more than the allowed occupancy.

delimiter //
create trigger before_insert_customer
before insert on customer
for each row
begin
if new.guest_count > (select max_limit from room where
room_no=new.room_no and hotel_id=new.hotel_id) then
signal sqlstate '45000' set message_text='The guest count is more
than the occupancy capacity of the room.';
end if;
end;


-- trigger before deletion of room, only room that is available can
be deleted, considering that the room not available is booked by a
customer.

delimiter //
create trigger delete_room
```

```
before delete on room
for each row
begin
if old.availability=0 then
signal sqlstate '45000' set message_text= 'There is currently a
customer in the room,
try deleting after the customer checks out';
end if;
end;

-- trigger before customer delete, only customer who has checked out
can be deleted.

delimiter //
create trigger delete_customer
before delete on customer
for each row
begin
if old.end_date is null then
signal sqlstate '45000' set message_text= 'the customer needs to
checkout the room before deletion';
end if;
end;

-- update customer, if the customer checks out then, room
availability is made 1, if room is presidential then assigned
catering and service staff are released and bill date is updated to
same as check out date.

delimiter //
create trigger update_customer
after update on customer
for each row
begin
if new.end_date is not null and old.end_date is null then
update room set availability=1 where room_no=old.room_no and
hotel_id=old.hotel_id;
update bill set bill_date=new.end_date where
customer_id=new.customer_id;
if(exists(select * from presidential where room_no=old.room_no and
hotel_id=old.hotel_id) )then
update presidential set catering_staff_id=null,service_staff_id=null
where room_no=old.room_no and hotel_id=old.hotel_id;
```

```
end if;
end if;
end;

-- updation of presidential, to check that the staff assigned to
presidential room are not serving other presidential rooms.

delimiter //
create trigger update_presidential
before update on presidential
for each row
begin
if new.catering_staff_id is not null and
new.catering_staff_id<>old.catering_staff_id then
if (not exists(select * from staff where
staff_id=new.catering_staff_id and hotel_id=new.hotel_id)) then
signal sqlstate '45000' set message_text= 'choose the catering staff
that works in the hotel the room is in';
end if;
end if;
if new.service_staff_id<> null and
new.service_staff_id<>old.service_staff_id then
if (not exists(select * from staff where
staff_id=new.service_staff_id and hotel_id=new.hotel_id)) then
signal sqlstate '45000' set message_text= 'choose the catering staff
that works in the hotel the room is in';
end if;
end if;
end;

-- trigger before delete on staff, only delete staff who are not
assigned to any presidential room service.

delimiter //
create trigger delete_staff
before delete on staff
for each row
begin
if (exists(select * from presidential where
catering_staff_id=old.staff_id))
then  signal sqlstate '45000' set message_text= 'the staff is already
assigned to serve a customer,
delete after changing the status of staff';
```

```
elseif(exists(select * from presidential where
service_staff_id=old.staff_id))
then  signal sqlstate '45000' set message_text= 'the staff is already
assigned to serve a customer,
delete after changing the status of staff';
end if;
end;
```

Services

We have used switch case for this part. In this we can enter a new service, update charge of service, delete a service, entering the service used and entering the service staff of the customer.

Inputs like service name, charge, customer id is asked from the user for it to get updated in the database.