***Hotel Management Database System***

*For WolfVillas*

*CSC 440 Database Systems*

*Project Report #2*

*Joanna Pollard, Mohamad Saleh, Anusha Balaji, Alex Harrell*

Project report #2 (520 points plus peer evaluations) *Please include one or more separate pages with a complete list of your assumptions about the project domain (as described in the* [*narrative*](http://courses.ncsu.edu/csc440/lec/001/wrap/narrative.pdf)*). Your entire report can be graded only with respect to your assumptions, and is not going to be graded in their absence.*

**Assumptions on attributes:**

* Staff attribute “**Hotel currently serving**” will have the value of “hotel ID”
* **Customer ID** assigned to customers is unique across the hotel chain and hence, customers can be identified with their ID across the whole hotel chain.
* **Hotel Room Number**: Every room in a hotel has a unique room number.
* Possible **services availed** by customers during stay **and their respective codes** include:
  + Room services (**0**), taxi (**1**), restaurant services (**2**), in-store/in-hotel purchases (**3**), phone services (**4**), and laundry services (**5**).
* Only the customer associated with the reservation can authorize charges made on the reservation from services availed.
* We are going to use integers to represent **room categories**. Possible room types in the hotel chain and their corresponding codes:
  + **Economy** (represented by **1**)
  + **Deluxe** (represented by **2**)
  + **Executive Suite** (represented by **3**)
  + **Presidential Suite** (represented by **4**)
* Possible categories for staff/possible values for job titles:
  + **Manager** will have job title **0**
  + **Front Desk Representative** will have job title **1**
  + **Room Service Staff** will have job title **2**
  + **Catering Staff** will have job title **3**
  + **Laundry Staff** will have job title **4**
* Nightly rate is static and dependant only on the room number
* Presidential suite is assigned one room service staff and one catering staff

**Assumptions on design:**

* Check in/Check out details fall under “**Reservation**” entity
* Front desk representatives handle **billing**; other staff can only “charge”/ add to the bill - they can’t “bill” customers.
* Customers **cannot** book a hotel/reserve a room. They have to call or meet the Front Desk Representative for a reservation as customers don’t have access to staff and room availability information.
* **Hotel managers** are able to access any data that front desk representatives and other staff can access. They are also able to complete tasks related to that data, such as bill a customer or make a reservation. This is in addition to what they can already access.
* **Customers can avail only one service** at a particular date and time (down to the second). For example, on 1/1/2016 at 12:00:00, a customer can request restaurant services. The customer has to wait until 1/1/2016 at 12:00:01 to avail another request. This restriction is plausible due to real life constraints: a customer physically cannot make multiple requests at one time.
* Customers with **multiple reservations** with the same check-in date must have them all in the same hotel. In other words, they can’t have concurrent reservations at different hotels.
* The **front desk representative** handles all “billing staff” duties. There’s no “billing staff” in our system.
* The customer associated with a reservation pays the bill.
* **A staff member can work at only one hotel, meaning that a hotel manager can manage only 1 hotel**.
  + Staff members (including hotel managers) must be employed at **exactly one hotel** at once.
  + To be in the database system, staff must belong to a hotel. So, if an employee leaves/is fired, all their information is deleted from the system and when they are re-hired, their information must be collected again and they will be assigned a **new ID**.
  + When an employee is transferred/promoted to a different hotel/role, the same holds: all their information including ID is deleted from the system and when they are re-hired, their information must be collected again and they will be assigned a **new ID.**
  + **This ensures an employee serves exactly one hotel with exactly one role at once. At any single point in time, a hotel can be traced to exactly 1 manager and vice versa.**
  + However, a starking drawback with this assumption is that there’s no employment history maintained for any employee and any role in the hotel system.

1. (80 points) Derive a global relational database schema from the schemas you obtained at step 9 in project report 1. Normalize to at least 3NF. Submit your database schema and the explanations why your relation schemas are in at least 3NF. Note: to conclude that your relation schemas satisfy (at least) 3NF (and to get full credit for this item), you need to consider and discuss *all* functional dependencies that the instructor can assume might hold on your schemas, rather than only those dependencies that are obvious and do hold on your schemas.

*hotel*(hotelID, address, name, phoneNumber)

**id -> address, name, phoneNumber**

This holds because every hotel has a unique id that determines the hotel’s address, name, and phone number. Since **id** is a key and, therefore, a superkey, this makes hotel conform to BCNF and hence, 3NF.

**Address -> id, name, phoneNumber,**

This holds because every hotel has a unique address and address may be used to determine the values of other attributes (hotel id, name, phoneNumber). Hence, the relation is in BCNF and 3NF.

**phoneNumber -> id, address, name** also holds as every hotel in the chain has a unique phone number. Since phonenumber can determine hotel id, it can also determine hotel’s name, address (by **id -> address, name, phoneNumber**). Since phoneNumber functionally determines other attributes of hotel, it’s a superkey and the relation is in BCNF and 3NF.

This relation is in 3NF also because:

* **name -> address, name -> phoneNumber, and name -> id** do not hold because multiple hotels can have the same name and have different values for the other attributes; **name** does not functionally determine any of the other attributes and hence, not a superkey for any of these relations. However, since **id** is prime as it’s a key, this is not a 3NF violation.

*employs*(staffID, hotelID)

**staffID -> hotelID**

This holds because every Staff entity knows the hotel they currently serve and every staff can serve exactly one hotel at once. Since staffID is a key (and of course, a superkey), this relation is in BCNF (and 3NF).

This relation is in 3NF also because:

* **hotelID -> staffID** does not hold as a hotel does not keep track of how many employees it has nor any information regarding them. However, since staffID is the key of this relation and hence, prime, this relation is not a 3NF violation.

*customer*(customerId, name, SSN, email, address, phoneNumber, gender, creditCard, billingAddress)

**customerID -> name, SSN, email, address, phoneNumber, gender, creditCard, billingAddress**

**SSN -> customerID**

The relation is in 3NF because:

* The first FD makes the relation conform to BCNF and 3NF because customerID is a key (and a superkey) that functionally determines the values of any and all of the other attributes of the relation.
* The relation is in 3NF because the right side of the second FD is part of a key (in the first FD) and prime.
* No combination of name, email, address, phoneNumber, gender, creditCard, billingAddress could functionally determine customerID nor values of any other attributes because many customers could share these attributes. For example, name -> SSN does not hold as multiple customers can share the same name and phoneNumber -> name does not hold as multiple customers can share the same phone number. For the same reason, these attributes alone and combinations of these (without customerID) cannot functionally determine any other attributes.

*staff*(staffID, name, address, age, dept, phoneNumber, gender, jobTitle, SSN, hotelID, salary)

**staffID -> name, address, age, department, phoneNumber, gender, jobTitleType, SSN, hotelID, salary**

**SSN -> staffID**

The relation is in 3NF because:

* For the first FD, staff id is a key, and is therefore a superkey, ensuring the relation is in BCNF
* For the second FD, because the right side (staff ID) is part of key (in the first FD) and prime, the relation doesn’t violate 3NF.
* No combination of name, address, age, department, phoneNumber, gender, jobTitleType, SSN, hotelID, salary could functionally determine staffID or SSN because multiple staff members could share all of these attributes (they could have the same name, age, and salary while working in the same hotel or department) and have unique staffID and SSN. For the same reason, these attributes alone (without staffID) cannot functionally determine the other attributes.

*manager*(staffId)

**staffID -> staffID**

Since manager is a subclass of staff, the staffID serves as the key of the relation and functionally determines other attributes of the entity-set. Therefore, this relation is in BCNF (and thus 3NF).

*jobTitleType*(id, jobTitle)

**id -> jobTitle**

This relation is in BCNF (and thus 3NF) because id functionally determines the jobTitle. As stated in the assumptions, Manager ID starts with 1, Front Desk Representative ID starts with 2, Room Service Staff ID starts with 3, Catering Staff ID starts with 4, and Laundry Staff ID starts with 5.

This relation is in 3NF also because:

* **jobTitle -> id** also holds as there is a one-one relationship between jobType and id. When given the jobType, its id can also be found based on the mapping above. Additionally, since id is the key of this relation and hence, prime, this relation is not a 3NF violation.

*frontDeskRepresentative*(staffID)

**staffID -> staffID**

Since frontDeskRepresentative is a subclass of staff, the staffID serves as the key of the relation and functionally determines other attributes of the entity-set. This relation is in BCNF (and thus 3NF) because it contains one attribute which is the key.

*room*(number, hotelID, rate, availability, occupancy, roomType)

**number, hotelID -> rate, availability, occupancy, roomType**

This relation is in 3NF because:

* The relation is in BCNF because (room)number and hotelID combined are a key, and therefore a superkey, of the above FD
* No combination of rate, availability, occupancy, or roomType could functionally determine room number, hotel ID, nor any other attributes because there could be multiple rooms in the same hotel that have the same rate, availability, and occupancy, as well as be of the same room type.
* **roomType -> occupancy** does not hold because multiple room of the same type can have different maximum occupancy. Occupancy may also depend upon availability.
* **roomType -> rate** does not hold because different rooms that are smoke/pet free could cost more, despite being the same room type.

roomType(id, roomType)

**id -> roomType**

This relation is in BCNF (and thus 3NF) because id functionally determines the room type. As stated in the assumptions, economy rooms’ id’s start with 1, deluxe rooms start with 2, executive suites start with 3 and presidential suites start with 4.

This relation is in 3NF also because:

* **roomType -> id** also holds as there is a one-one relationship between roomType and id. When given the roomType, its id can also be found based on the mapping above. Additionally, since id is the key of this relation and hence, prime, this relation is not a 3NF violation.

*presidentialSuite*(number, hotelID, roomStaffID, cateringStaffID)

**number, hotelID-> roomStaffID, cateringStaffID**

This relation is in 3NF because:

* Because number and hotelID combined are a key and, therefore, a superkey of the relation, it is in BCNF and in 3NF.
* **roomStaffID -> cateringStaffID and** **cateringStaffID -> roomStaffID** do not hold because room staff and catering staff are assigned independently of each other to a suite. So, different combinations of room staff and catering staff could be paired and assigned to a suite.
* **roomStaffID, cateringStaffID -> number, hotelID** does not hold because the same pair of room staff and catering staff could be assigned to multiple presidential suites.
* However, **roomStaffID, cateringStaffID -> hotelID** holds because every staff tuple has a value for hotel served. Even this FD does not make the relation violate 3NF because hotelID is a prime attribute.
* As shown above, no subset of the key above nor any other combination of the attributes will functionally determine or uniquely identify a presidential suite. So, no other FD’s hold for this relation.

*reservation*(hotelID, roomNumber, customerID, checkInDate, checkOutDate, balance)

**hotelID, roomNumber, customerID, checkInDate -> checkOutDate, balance**

This relation is in 3NF because:

* Because hotelID, roomNumber, customerID, and checkInDate combined are a key, and therefore a superkey, of **reservation**, the relation is in BCNF and hence, in 3NF.
* **hotelID, customerID, checkInDate -> checkOutDate** does not hold because because there could be several reservations in the same hotel by the customer with the same check in date, but different check out dates.
* **hotelID, customerID -> roomNumber** does not hold because one customer could book multiple rooms at various dates.
* **hotelID, roomNumber, customerID -> checkInDate** nor **hotelID, roomNumber, customerID -> checkOutDate** hold because a customer can have multiple reservations for the same room in the same hotel, at different dates in this system.
* **checkInDate, checkOutDate -> hotelId** or **roomNumber** or **balance** or **customerID** because there could multiple reservations across the hotel chain with the same check in and check out dates.
* No subset of the key above nor any other combination of the attributes will functionally determine or uniquely identify a reservation because multiple reservations could share several of these attributes.

*charges*(customerID, roomNumber, checkInDate, dateAndTime, staffID, serviceType, amount)

**customerID, roomNumber, checkInDate, dateAndTime -> staffID, serviceType, amount**

This relation is in BCNF/3NF because:

* Because the left side is a key, and therefore a superkey, of the FD, this relation is in BCNF. The reason this is a key is because a customerId, roomNumber, and checkInDate is what makes up a reservation. A reservation combined with a specific dateAndTime (since a customer can avail just one charge per specific date and time, according to our assumptions) will give us the specific charge, which holds the information about the staff member who created the charge, the serviceType, and the amount charged.
* No subset of the key above nor any other combination of the attributes will functionally determine or uniquely identify a charge on a reservation because of various reasons: there could be multiple reservations on the same day by a customer, or multiple charges to a single reservation, or same charges at the same time applied to different reservations.

serviceTypes(id, serviceType)

**id -> serviceType**

This relation is in BCNF (and thus 3NF) because (service)id functionally determines the serviceType. As stated in the assumptions, we are going to map particular services to particular ids: phone services (**1**), restaurant services (**2**), laundry services (**3**), and in-store/in-hotel purchases (**4**).

This relation is in 3NF also because:

* **serviceType -> id** also holds. Since there is a one-one relationship between serviceType and id, when given the serviceType, we can also generate the id based on the mapping above. Additionally, since id is the key of this relation and hence, prime, this relation is not a 3NF violation.

1. (120 points) Describe any design decisions for the global schema. Identify and explain all integrity constraints of the following types: NOT NULL, key, and referential integrity. Describe which attributes are allowed to be NULL, why, and what a NULL value means for each attribute on which it is allowed. Submit all your descriptions and explanations.

*hotel*(hotelID, address, name, phoneNumber)

* hotelID (Primary Key) - Unique identifier and cannot be NULL
* Address (NOT NULL) - to identify the address of the hotel
* name (NOT NULL) - to identify the name of the hotel
* phoneNumber (NOT NULL) - to identify the phone number of the hotel

*employs*(staffID, hotelID)

* staffID(Referential integrity, Primary Key) - unique identifier for a staff member that refers to a value that also exists in the staff entity
* hotelID(Referential integrity, Primary key) - - unique identifier that refers to a value that also exists in the hotel entity

*customer*(customerId, name, SSN, email, address, phoneNumber, gender, creditCard, billingAddress)

* customerID(primary key) - unique identifier
* name (NOT NULL) - identify the name of the customer
* SSN (NOT NULL) - identify the SSN of the customer
* Email (NOT NULL) - identify the email address of the customer
* Address (NOT NULL) - identify the address of the customer
* phoneNumber (NOT NULL) - identify the phone number of the customer
* Gender (NOT NULL) - identify the gender of the customer
* creditCard (NOT NULL) - identify the credit card number of the customer
* billingAddress (NOT NULL) - identify the billing address of the customer

*staff*(staffID, name, address, age, dept, phoneNumber, gender, jobTitle, SSN, hotelID, salary)

* staffID (primary key) - unique identifier
* name (NOT NULL) - identify the name of the staff member
* address (NOT NULL) - identify the address of the staff member
* age (NOT NULL) - identify the age of the staff member
* dept (NOT NULL) - identify the department the staff member works in
* phoneNumber (NOT NULL) - identify the phone number of the staff member
* gender (NOT NULL) - identify the gender of the staff member
* jobTitle (NOT NULL, referential identity) - identify the job title of the staff member, refers to a possible value in the jobTitleType relation
* SSN (NOT NULL) - identify the SSN of the staff member
* hotelID (NOT NULL, referential integrity) - specify the hotel that the staff member works at
* salary (NOT NULL) - specify the salary of the staff member

*manager*(staffId)

* staffId (primary key, referential integrity) - unique identifier of a manager that refers to a value in the staff entity

*jobTitleType*(id, jobTitle)

* id (primary key) - unique identifier of a type of job title
* jobTitle (NOT NULL) - specify what job title the id stands for

*frontDeskRepresentative*(staffID)

* staffId (primary key, referential integrity) - unique identifier of a front desk representative that refers to a value in the staff entity

*room*(number, hotelID, rate, availability, occupancy, roomType)

* number (primary key) - identify the number associated with the room
* hotelID (primary key, referential identity) - identify the hotel associated with the room, refers to a value in the hotel entity
* rate (NOT NULL) - specify the nightly cost of the room
* availability (NOT NULL) - specify whether or not the room is available to reserve
* occupancy (NOT NULL) - specify the number of people currently staying in the room (between 0 and max occupancy for room)
* roomType (NOT NULL, referential integrity) - specify what type of room it is (ex: Economy, Deluxe, Executive Suite, Presidential Suite), refers to a value found in the room type relation

roomType(id, roomType)

* id (primary key) - unique identifier
* roomType (NOT NULL) - specify what room type the id stands for

*presidentialSuite*(number, hotelID, roomStaffID, cateringStaffID)

* number (primary key, referential integrity) - identify the number associated with the room
* hotelID (primary key, referential identity) - identify the hotel associated with the room, refers to a value in the hotel entity
* roomStaffID (NOT NULL, referential integrity) - specifies the room staff member that is assigned to a presidential suite, refers to a value in the staff entity
* cateringStaffID (NOT NULL, referential integrity) - specifies the catering staff member that is assigned to a presidential suite, refers to a value in the staff entity

*reservation*(roomNumber, customerID, checkInDate, checkoutDate, balance)

* roomNumber (primary key, referential integrity) - unique identifier of room number associated with reservation, refers to a value in the rooms entity
* customerID (primary key, referential integrity) - unique identifier of the customer associated with reservation, refers to a value in the customers entity
* checkInDate (primary key) - specifies the check in date associated with the reservation
* checkOutDate - specifies the check out date associated with the reservation
  + This value can be NULL because a customer could be staying in a hotel for an “extended stay” where they are unsure of their check out date
* balance (NOT NULL) - specifies balance customer associated with reservation owes

*charges*(customerID, roomNumber, checkInDate, dateAndTime, staffID, serviceType, amount)

* roomNumber (primary key, referential integrity) - unique identifier of room number associated with reservation, refers to a value in the rooms entity
* customerID (primary key, referential integrity) - unique identifier of the customer associated with reservation, refers to a value in the customers entity
* checkInDate (primary key) - specifies the check in date associated with the reservation
* dateAndTime (primary key) - specifies the date and time of a specific transaction for a service availed.
* staffID (NOT NULL, referential integrity) - identifies staff member who created this charge, refers to a value in the staff entity
* serviceType (NOT NULL, referential identity) - identifies type of service associated with this charge, refers to a value in the serviceTypes relation
* Amount (NOT NULL) - specifies amount charged to reservation for this service

serviceTypes(id, serviceType)

* id (primary key) - unique identifier
* serviceType (NOT NULL) - specify what service type the id stands for

1. (80 points) Using SqlPlus, for all your relation schemas create base relations with the right attribute domains and with *all* the integrity constraints you listed in item 2 of this report. Populate the base relations with 4-8 rows each. Show what is in each table by printing out, for each table, the answer to the "SELECT \* FROM" query. Submit the printouts of (1) all your "CREATE TABLE" statements, of (2) all your "SELECT \*" queries, and of (3) the answers to all the queries in SqlPlus.

1. (240 points) Write interactive SQL queries for each operation in the [**narrative**](http://courses.ncsu.edu/csc440/lec/001/wrap/narrative.pdf), and test the queries in SqlPlus:

4.1 (140 points). Make assumptions to justify the constants in your queries: for example, for the operation "show all staff members who are receptionists" assume that the database has information about three receptionists. Use these fictitious constants to ask specific SQL queries and to make specific updates on your stored relations. For the queries, choose the constants so that a non-empty answer set is returned; you may need to add additional rows to do so. For each operation in Tasks and Operations in the [**narrative**](http://courses.ncsu.edu/csc440/lec/001/wrap/narrative.pdf), submit the query or update (whichever is appropriate) for the operation and the printout of the SqlPlus response to the query/update. Note: A lot of points will be taken off if you miss some of the operations in the [narrative](http://courses.ncsu.edu/csc440/lec/001/wrap/narrative.pdf).

*Note: Before executing the following queries, assume the relations are populated according to ALL\_TABLES.txt (attached at the end of the document).*

Manager can enter information about staff; manager adds a new staff member (Tom Sawyer) to the database (staff table):

* **SQL>** INSERT INTO staff(staffID, name, address, age, dept, phoneNumber, gender, jobTitle, SSN, hotelID, salary) VALUES (staff\_seq.NEXTVAL, 'Tom Sawyer', '100 Treasure Island, NC', 21, 'Catering', 1213435665, 'Male', 3, 123123123, 1, 30000);
* **Sqlplus output**: “1 row created.”

Manager can update information about an already existing staff member with new information given their **id**; specifically, manager wants to edit salary for Jasmine Jackson (StaffID: 2) from hotel 0:

* **SQL>** Update staff Set salary=35000 Where staffID=2;
* **Sqlplus output**: “1 row updated.”

Manager can delete information about staff; manager wants to remove staff member Tom Sawyer (StaffID: 8) from hotel 1:

* **SQL>** Delete from staff Where staffID=8;
* **Sqlplus output**: “1 row deleted.”

Staff can enter information about rooms; front desk representative wants to add a new economy room (Number: 11) to hotel 0:

* **SQL>** Insert into room(num, hotelID, rate, availability, occupancy, roomType) VALUES (11, 0, 99, 'Available', 0, 1);
* **Sqlplus output**: “1 row created.”

Staff can update information about rooms; manager changes the occupancy of an already existing room (room# 2 in hotel 1) by setting it to 5:

* **SQL>** Update room Set occupancy = 5 Where num=2 and hotelID=0;
* **Sqlplus output**: “1 row updated.”

Staff can delete information about rooms; front desk representative wants to delete room# 11 from hotel 0:

* **SQL>** Delete from room where num=11 and hotelID=0;
* **Sqlplus output**: “1 row deleted.”

Staff can enter information about customers;

* **SQL>** INSERT INTO customer(customerID, name, SSN, email, address, phoneNumber, gender, creditCard, billingAddress) VALUES (customer\_seq.nextval, 'Thomas Bradley', 123454687, 'tbradley@gmail.com', '45 Garden Drive Raleigh, NC 38473', 1231561324, 'Male', 6915911824503118, '45 Garden Drive Raleigh, NC 38473');
* **Sqlplus output**: “1 row created.”

Staff can update information about customers:

* **SQL>** UPDATE customer SET email = ‘tbradley[@yahoo.com](mailto:jshmoe2@gmail.com)’ WHERE customerID=9;
* **Sqlplus output**: “1 row updated.”

Staff can delete information about customers:

* **SQL>** DELETE FROM customer WHERE customerID = 9;
* **Sqlplus output**: “1 row deleted.”

Staff can check available rooms in their hotel: Assume Benny Johnson from hotel 1 is logged in.

* **SQL>** SELECT roomType, num FROM room WHERE availability = ‘Available’ AND hotelID=1;
* **Sqlplus output**:

ROOMTYPE NUM

------------------ ----------------

0 1

1 2

2 3

3 4

Manager/Front Desk Representative can update the room and hotel for a reservation:

* **SQL>**  UPDATE reservation SET roomNumber = 4 WHERE roomNumber = 1 and hotelID=0 and customerID=0 and checkInDate='02/09/2017';
* **Sqlplus output**: “1 row updated.”

Staff can enter phone bills into reservations:

* **SQL>** INSERT INTO charges(roomNumber, hotelID, customerID, checkInDate, dateAndTime, staffID, serviceType, amount) VALUES (4, 0, 3, '03/30/2017', '4/01/2017 11:02PM', 2, 4, 51);
* **Sqlplus output**: “1 row created.”

Staff can update phone bills in reservations:

* **SQL>** UPDATE charges SET amount=100 WHERE roomNumber = 4 and hotelID=0 and customerID = 3 and checkInDate = '03/30/2017' and serviceType=4 and DATEANdTime='4/01/2017 11:02PM' and staffID=2;
* **Sqlplus output**: “1 row updated.”

Staff can enter laundry service charges:

* **SQL>** INSERT INTO charges(roomNumber, hotelID, customerID, checkInDate, dateAndTime, staffID, serviceType, amount) VALUES (4, 0, 3, '03/30/2017', '4/01/2017 3:00PM', 2, 5, 49);
* **Sqlplus output**: “1 row created.”

Staff can update laundry service charges:

* **SQL>** Update charges set amount=75 where roomNumber = 4 and hotelID=0 and customerID = 3 and checkInDate = '03/30/2017' and serviceType=5 and DATEAndTime='4/01/2017 3:00PM' and staffID=2;
* **Sqlplus output**: “1 row updated.”

Staff can enter restaurant bills into reservations:

* **SQL>** INSERT INTO charges(roomNumber, hotelID, customerID, checkInDate, dateAndTime, staffID, serviceType, amount) VALUES (4, 0, 3, '03/30/2017', '4/02/2017 5:00PM', 2, 2, 49)
* **Sqlplus output**: “1 row created.”

Staff can update restaurant bills in reservations:

* Update charges set amount=50 where roomNumber = 4 and hotelID=0 and customerID = 3 and checkInDate = '03/30/2017' and serviceType=2 and DATEAndTime='4/01/2017 11:02PM' and staffID=2;
* **Sqlplus output**: “1 row updated.”

Front Desk Representatives can generate reservation (billing included in reservation):

* **SQL>** INSERT INTO reservation(roomNumber, hotelID, customerID, checkInDate, checkOutDate, balance) VALUES (5, 0, 7, ‘02/09/2017’, ‘02/12/2017’, 0);
* **Sqlplus output**: “1 row created.”
* **SQL>** Update room set occupancy=’Unavailable’ where number=5 and hotelID=0;
* **Sqlplus output**: “1 row updated.”

Front Desk Representatives can maintain reservation (access/modify billing information):

* **SQL>** UPDATE reservation SET balance = 100, WHERE customerID = 7 AND roomNumber = 5 AND checkInDate = ‘02/09/2017’;
* **Sqlplus output:** “1 row updated.”

Front Desk Representatives can close reservations (billing included in reservation):

* **SQL>** DELETE FROM reservation WHERE customerID = 7 AND roomNumber = 5 AND checkInDate = ‘02/09/2017’;
* **Sqlplus output:** “1 row deleted”

Manager/Front desk representative can get occupancy by room type:

* **SQL>** SELECT roomType, SUM(occupancy) AS occupancy FROM room WHERE hotelID = 0 GROUP BY roomType
* **Sqlplus output:**

ROOMTYPE OCCUPANCY

------------------- -------------------

0 2

1 2

2 6

3 8

Manager/Front desk representative can get occupancy by date range:

* **SQL>** SELECT SUM(o.occupancy) AS occupancy FROM reservation r INNER JOIN room o ON r.roomNumber = o.num WHERE r.checkInDate BETWEEN ‘08-FEB-17’ AND ‘31-MAR-17’;
* **Sqlplus output:**

OCCUPANCY

-------------------

11

Manager/Front desk representative can get occupancy by hotel:

* **SQL>** SELECT hotelID, SUM(occupancy) AS occupancy FROM room GROUP BY hotelID
* **Sqlplus output:**

HOTELID OCCUPANCY

------------------- -------------------

0 18

1 18

Manager/WolfVillas Management can access percentage of rooms occupied in a hotel:

* **SQL>** SELECT hotelID, SUM(CASE WHEN availability != ‘Available’ THEN 1 ELSE 0 END) \* 100 / (SELECT Count(\*) FROM room) AS PercentFull FROM room GROUP BY hotelID
* **Sqlplus output:**

HOTELID PERCENTFULL

------------------- ----------------------

0 100

1 100

Manager/WolfVillas Management can view information on staff grouped by their role:

* **SQL>** SELECT jobTitle, name FROM staff GROUP BY (jobTitle, name);
* **Sqlplus output:**

JOBTITLE NAME

------------------- -------------------------

0 John Doe

0 Benny Johnson

1 Jasmine Jackson

1 Jason Richardson

2 SeanMcRonald

2 Jessica Candy

3 Nathan Drake

3 Derrick Cox

Manager/Front desk representative can access the customers served by a particular catering/room service staff member (with staffid =4)

* **SQL>** Select c.customerID, c.Name from (Reservation r INNER JOIN (presidentialSuite p ON r.roomNumber = p.num AND p.hotelID=r.hotelID) where roomStaffID=4;
* **Sqlplus output**:

CUSTOMERID NAME

----------------------------- ---------------------------------------

7 Andrew Bradley

4.2 (70 points). Turn on Oracle auto-trace for your queries (from item 4.1 above) and examine the outputs. Find two queries for which autotrace shows full table scans; print the autotrace outputs for these queries. For these two queries, create appropriate indexes; print the autotrace output that shows the use of the indexes. For each query, submit the printouts of: (1) the SQL query, (2) the execution plan (autotrace) for the query which shows at least one full table scan, (3) your index-creation statement in SQL, and (4) the execution plan (autotrace) for the query where the full table scan has been replaced with a use of your index.

1. The first index was added to Room in the “availability” column.

SQL> Set autotrace on;

SQL> SELECT roomType, num FROM room WHERE availability=‘Available’ AND hotelID=1;

ROOMTYPE NUM

---------- ----------

0 1

1 2

2 3

3 4

Execution Plan

---------------------------------------------------------------------

Plan hash value: 640342614

---------------------------------------------------------------------

| Id | Operation | Name | Rows | Bytes | Cost (%CPU)| Time |

---------------------------------------------------------------------

| 0 | SELECT STATEMENT | | 3 | 84 | 3 (0)| 00:00:01 |

|\* 1 | TABLE ACCESS FULL| ROOM | 3 | 84 | 3 (0)| 00:00:01 |

---------------------------------------------------------------------

Predicate Information (identified by operation id):

---------------------------------------------------

1 - filter("HOTELID"=1 AND "AVAILABILITY"=U'Available')

SQL> Create index RoomAvbty ON room (availability);

(Output): Index created.

SQL> SELECT roomType, num FROM room WHERE availability = ‘Available’ AND hotelID=1;

*Note: Slightly formatted output for more readability.*

ROOMTYPE NUM

---------- ----------

0 1

1 2

2 3

3 4

Execution Plan

----------------------------------------------------------

Plan hash value: 1177936153

--------------------------------------------------------------------------------

| Id | Operation | Name |Rows| Bytes | Cost (%CPU) |Time |

------------------------------------------------------------------------------

| 0 | SELECT STATEMENT | | 3 | 84 | 2 (0)| 00

:00:01 |

|\* 1 | TABLE ACCESS BY INDEX ROWID| ROOM | 3 | 84 | 2 (0)| 00

:00:01 |

|\* 2 | INDEX RANGE SCAN | ROOMAVTY | 8 | | 1 (0)| 00

:00:01|

-----------------------------------------------------------------------------

Predicate Information (identified by operation id):

---------------------------------------------------

1 - filter("HOTELID"=1)

2 - access("AVAILABILITY"=U'Available')

2) The second index was added to staff on the “jobTitle” column and third was added to staff on the “name” column to facilitate grouping for the following query.

SQL> SELECT jobTitle, name FROM staff GROUP BY (jobTitle, name);

JOBTITLE NAME

------------------- --------------------------

0 John Doe

0 Benny Johnson

1 Jasmine Jackson

1 Jason Richardson

2 SeanMcRonald

2 Jessica Candy

3 Nathan Drake

3 Derrick Cox

8 rows selected.

Execution Plan

---------------------------------------------------------------------

Plan hash value: 1777826082

---------------------------------------------------------------------| Id | Operation | Name | Rows | Bytes | Cost (%CPU)| Time |

---------------------------------------------------------------------| 0 | SELECT STATEMENT | | 8 | 240 | 4 (25)| 00:00:01 |

| 1 | HASH GROUP BY | | 8 | 240 | 4 (25)| 00:00:01 |

| 2 | TABLE ACCESS FULL| STAFF | 8 | 240 | 3 (0)| 00:00:01 |

---------------------------------------------------------------------

SQL> create index role on staff (jobTitle);

Index created.

SQL> create index name on staff (name);

Index created.

SQL> SELECT jobTitle, name FROM staff GROUP BY (jobTitle, name);

JOBTITLE NAME

------------------- --------------------------

0 John Doe

0 Benny Johnson

1 Jasmine Jackson

1 Jason Richardson

2 SeanMcRonald

2 Jessica Candy

3 Nathan Drake

3 Derrick Cox

8 rows selected.

Execution Plan

---------------------------------------------------------------------

Plan hash value: 3177460008

---------------------------------------------------------------------

| Id | Operation | Name | Rows | Bytes | Cost (%CPU) | Time |

--------------------------------------------------------------------

| 0 | SELECT STATEMENT | | 8 | 240 | 4 (50) | 00:00:01 |

| 1 | HASH GROUP BY | | 8 | 240 | 4 (50) | 00:00:01 |

| 2 | VIEW | index$\_join$\_001 | 8 | 240 | 3 (34) | 00:00:01 |

|\* 3 | HASH JOIN | | | |

| |

| 4 | INDEX FAST FULL SCAN| ROLE | 8 | 240 | 1 (0) | 00:00:01 |

| 5 | INDEX FAST FULL SCAN| NAME | 8 | 240 | 1 (0) | 00:00:01 |

---------------------------------------------------------------------

Predicate Information (identified by operation id):

---------------------------------------------------

3 - access(ROWID=ROWID)

**Report 2 Corrections - Group G**

4.3 (30 points). For any two of your SQL queries ("select" statements only, rather than "insert/delete/update") **with joins**, explain why the queries are correct - one explanation per query. If you come up with erroneous solutions prior to the correct one, also include those as part of the explanation. For each query, submit the query itself and your explanation. Your explanations will consist of two parts:

* + the corresponding relational algebra expression, **and**
  + a *correctness proof* that shows that the query answer always corresponds to the query specification;
  + for example, a correctness proof of the query
  + "SELECT e.Lname, d.DeptName FROM Employee e, Department d WHERE e.DeptNo = d.DeptNo,"
  + with specification
  + "return the last names of all employees together with the names of their departments,"
  + can be as follows:
  + "Suppose *e* is any tuple in the Employee relation, and *d* is any tuple in the Department relation, such that the value *e.DeptNo* is the same as the value *d.DeptNo*. Each such combination of tuples *(e, d)* gives personal information about one employee, together with all information on the department for that employee. For each such combination *(e, d)*, the query returns the value of Lname and the value of DeptName. These values are the last name of the employee and the name of the department for the employee. But this is exactly what our query should return; see the specification."

SELECT SUM(o.occupancy) AS occupancy FROM reservation r INNER JOIN room o ON r.roomNumber = o.num WHERE r.checkInDate BETWEEN ‘08-FEB-17’ AND ‘31-MAR-17’;

πroom.occupancy (σ08-FEB-17 > checkInDate > 31-MAR-17( reservation |><|reservation.roomNumber = room.num room))

Suppose that r is any tuple in the reservation relation and o is any tuple in the room relation, such that r.roomNumber equals o.num. (room number matching). The combination of (r,o) gives information about reservations, most importantly their time frame, and information about the room, most importantly the occupancy of each room. For each set (r,o) the query returns the room occupancy and the check in time for a visiting groups reservation. By restricting the query with a date range, we can limit our timeframe to be within those dates (using the built in date functionality in SQL) Then our query returns the sum of all of the occupancies which meet the prior date requirements based on the reservation time frame and have a correlating room occupancy.

SELECT DISTINCT ch.customerID, c.name FROM Charges ch JOIN Customer c ON ch.customerID=c.customerID WHERE ch.staffID = 2;

πcharges.customerID, customers.name (σstaffID=2( charges |><|charges.customerID=c.customerID customers))

Suppose that ch is any tuple in the charges relation and c is any tuple in the customer relation, such that ch.customerID equals c.customerID, where the relations are “joined by” customer ID). The combination of (ch,c) gives information about both the charges and the customer, including customer name, email, id and hotel id, amount, staffID associated with a charge. Further filtering the entries to match the given the staffID yields customers that the particular staff personnel serves. Distinct customer names and ids are produced as output for the query to eliminate duplicate entries resulting from a staff charging the same customer twice or more.