WolfVilla Hotels
Database Management System

CSC 440 Database Management Systems

Project Report #2

Andrew Poe, Felix Kim, John Hutcherson, Guoyang Di

Assumptions

1. **A presidential suite has dedicated room service and catering staff as required, but other room types can also have them if the customer opts in at check in.**
2. **"Payment info" only consists of: payment method, card number, billing address, and SSN of payer.**
3. **The phone, laundry, and restaurant bills are treated as tabs, which the customer can accrue expenses in over their visit, while room service and catering are treated as services that are either provided for the entire visit, or not, and as such are opted in at check in time.**
4. **A hotel has at most one manager, and a manager manages exactly one hotel.**
5. **A staff member works at exactly one hotel.**
6. **The hotel name is not necessarily unique among different hotels, hence the need for a "Hotel ID" key attribute.**
7. **Catering and room service are two separate services.**
8. **The "Staff ID" is unique for every staff member throughout the entire hotel chain, assigned by a global authority for the entire hotel chain.**
9. **Similarly, the global authority assigns each customer a unique ID and every hotel room throughout the entire hotel chain has a unique room ID.**
10. **A customer is able to check into as many hotel rooms as he/she wants. At check-in, the customer says how many people they wish to check into a room (but a room has at most one listed customer). This gives the database the ability to handle situations where a customer has more than 4 people in their party.**
11. **A specific room may only have one customer checked in at a time, that is, the check in table only keeps track of currently checked in guests, not those who used to be checked in, or will be checking in in the future.**
12. **Pricing is static year round. We do not deal with such things as seasonality or day of the week when dealing with pricing.**
13. **Room pricing is determined on a case by case basis, allowing a hotel to have different room rates even among rooms with the same type. This allows more freedom to price more desirable rooms of similar types higher than less desirable ones. Rooms with an ocean view are an example of this.**
14. **Email addresses may be shareable, e.g. spouses or business team members may share a single email address in their communications with the hotels.**
15. **For the purposes of billing, there is no distinction made between credit and debit cards.**
16. **A room has at most one staff member from room service assigned to it. As well as, at most one catering staff assigned to it.**
17. **A room is only present in the CheckIn relation during the time period that a check in for the room is valid. Also, when a customer checks into a room, their information is added to the assigned CheckIn relation, and when they check out, their information is deleted from the CheckIn relation.**
18. **A staff must be at least 16 years old to work at a hotel.**
19. **Each hotel has its own unique phone number and unique physical address.**
20. **Any bill cannot fall below 0; a credit can only be applied to a customer's account if there is a corresponding charge whose magnitude is greater than or equal to it.**

1) Database Schema

Hotel(hotelID, name, address, phone)

- hotelID -> name, address, phone
  - hotelID is a unique value assigned to each hotel, so this holds.
- Address -> hotelID, name, phone
  - We have an assumption stating that no hotels share an address, so this is a unique value, and the relation holds.
- Phone -> hotelID, name, address
  - Similarly, we have an assumption that no hotel's share the same phone number, so this is a unique value, and the relation holds.

- This relation is in BCNF, and therefore 3NF, because all of the functional dependencies have a single key attribute on the left hand side.
  1. Name doesn't determine anything, since hotels can share the same name.

Staff(staffID, ssn, name, age, gender, phone, address, staffType, hotelID)

- staffID -> ssn, name, age, gender, phone, address, staffType, hotelID
  - staffID is a unique identifier, so this relation holds.
- ssn -> staffID, name, age, gender, phone, address, staffType, hotelID
  - ssn is also a unique identifier, so this relation holds.

- This relation is in BCNF, and therefore 3NF, because all of the functional dependencies have a super key on the left hand side.
  1. Name doesn't determine anything, since staff can share the same name.
  2. It's possible for two or more staff to share the same phone number and address, so neither of those determine anything.
  3. Similarly for age and gender, they can, and likely will, be shared among many staff.

RSAssignedTo(staffID, roomID, customerID)

- roomID -> staffID, customerID
  - A room can only have one staff/customer assigned

- This relation is in BCNF and thus in 3NF because it contains one key attribute on the left side.
  1. staffID is unable to determine anything, as a Room Service Staff may be assigned to multiple rooms/customers.
  2. customerID is similarly unable to determine anything, as a customer is allowed to have multiple rooms.

CAssignedTo(staffID, roomID, customerID)

- roomID-> staffID, customerID
    - A room can only have one staff/customer assigned

- This relation is in BCNF and thus in 3NF because it contains one key attribute on the left side.
    1. staffID is unable to determine anything, as a Room Service Staff may be assigned to multiple rooms/customers.
    2. customerID is similarly unable to determine anything, as a customer is allowed to have multiple rooms.

Customer(customerID, name, gender, address, phone, email)

- customerID -> name, gender, address, phone, email
    - A customer has all of this information, and is only allowed to have 1 of each

- This relation is in BCNF and thus in 3NF because it contains one key attribute on the left side.
    1. Multiple customers may share any or even all of these additional attributes other than customerID.

CheckIn(customerID, roomID, checkOutDateTime, checkInDateTime, numberInParty, hasCatering, hasRoomService, restaurantBill, laundryBill, phoneBill, paymentMethod, ssnOfPayer)

- roomID-> All other attributes
    - Since by our assumptions a room can only have one customer checked in at a time, and is either checked in or not, roomID is all that is needed to uniquely identify everything else.

- This relationship is in BCNF, and thus in 3NF, because the left hand side of the only functional dependency is a key.
    1. None of the other information on the right hand side of the functional dependency determines anything else because multiple check-in's can have the same information for these fields, however unlikely that would be.
    2. customerID on it's own is not a key because a customer is free to check in to any number of hotel rooms at any number of hotels.
    3. ssnOfPayer is unable to determine the payment method, as a payer is able to pay for multiple rooms using different payment information each time.
    4. customerID is unable to determine the numberInParty, as a customer is able to check into multiple rooms with different party sizes each time. This allows a customer with a large family to split their family between multiple rooms for example.
    5. Our assumptions state that a checkin only keeps track of currently checked in guests. When a guest checks out their check in is deleted, which causes the room to be available for check in again. This works the same way for dates in the future, they are not allowed.

CardNumber(roomID, billingAddress, cardNumber)

- roomID-> cardNumber, billingAddress
    - As a specific check in has one payment method, a single credit card has one cardNumber and billingAddress associated with it. Since a room is either checked in or not, roomID is all that is needed to identify a specific check in.

- This relationship is in BCNF, and thus in 3NF, because the left hand side of the only functional dependency is a key.
    1. cardNumber or billingAddress cannot determine anything else, as a payer is allowed to pay for multiple rooms, and may use a different form of payment for each room. Additionally, they may have multiple credit cards, so billing address would be unable to determine cardNumber.

Room(hotelID, roomNumber, roomType, rate, maxOccupancy, roomID)

- hotelID, roomNumber -> roomType, rate, maxOccupancy, roomID
    - A specific hotel room is able to have a set of custom room specifications, therefore a specific room determines type, rate, and occupancy. Also, the hotelID and roomNumber combined uniquely determine the roomID.
- roomID -> hotelID, roomNumber, roomType, rate, maxOccupancy
    - roomID is a unique identifier assigned by the database to each tuple, so it determines everything else. This is designated the primary key.

- This relation is in BCNF, and thus in 3NF, because the left side of each relation is a super key.
    1. A hotel Room may have a custom rate set based on some other criteria an administrator notices, such as rooms with a view; therefore, roomType does not explicitly determine rate.
    2. maxOccupancy is similarly undetermined by roomType, as maxOccupancy can be set on a room by room basis independent of anything else.
    3. roomNumber alone cannot be used to uniquely identify a room, as many hotels can share the same room numbers. roomID fills this particular requirement, specifically for referential integrity.

2) Design Decisions

Hotel(hotelID, name, address, phone)

hotelID (Primary Key) - unique identifier for a hotel
name (NOT NULL) - for identification of the hotel
address (UNIQUE, NOT NULL) - address of hotel
phone (UNIQUE, NOT NULL) - way to contact hotel

Staff(staffID, ssn, name, age, gender, phone, address, staffType, hotelID)

staffID (Primary Key) - unique identifier for staff
ssn (UNIQUE, NOT NULL) - used to keep track of a staff member
name (NOT NULL) - for identification
age (NOT NULL, age>15) - age of staff employee
gender (NOT NULL, IN {‘M’, ‘F’}) - gender of the employee
phone (NOT NULL) - phone number of employee
address (NOT NULL) - where the employee lives
staffType (NOT NULL, IN {'Manager', 'Front Desk', 'Catering', 'Room Service'}) - used to identify type of staff member
hotelID (FOREIGN KEY – Referential Integrity) - used to identify the hotel that the staff member works at

Customer(customerID, name, gender, address, phone, email)

customerID (Primary Key) - unique identifier for the customer
name (NOT NULL) - for identification
gender (NOT NULL, IN {‘M’, ‘F’}) - gender of customer
address (NOT NULL) - address of the customer
phone (NOT NULL) - phone number of the customer
email (NOT NULL) - email of the customer

CheckIn(customerID, roomID, checkOutDateTime, checkInDateTime, numberInParty, hasCatering, hasRoomService, restaurantBill, laundryBill, phoneBill, paymentMethod, ssnOfPayer)

customerID (FOREIGN KEY - Referential integrity) - part of unique identifier and refers to entities within the database (Customer)
roomID (FOREIGN KEY - Referential integrity, PRIMARY KEY) - part of unique identifier and refers to entities within the database (Room)
checkOutDateTime (NOT NULL) - when the customer checks out
checkInDateTime (NOT NULL) - when the customer checks in
numberInParty (NOT NULL, IN {1, 2, 3, 4}) - number of people in a room
hasCatering (NOT NULL, IN(‘T’, ‘F’)) - keep track if room has catering
hasRoomService (NOT NULL, IN (‘T’, ‘F’)) - keep track if room has room service
restaurantBill(NOT NULL, Default: 0.0, CHECK(restaurantBill >= 0)) - restaurant bill for customer
laundryBill (NOT NULL, Default: 0.0, CHECK(laundryBill >= 0)) - laundry bill for customer
phoneBill (NOT NULL, Default: 0.0, CHECK(phoneBill >= 0)) - phone bill for customer
paymentMethod (NOT NULL, IN {‘card’, ‘cash’, ‘check’}) - method of payment for customer
ssnOfPayer (NOT NULL) - for identification of the payer

CardNumber(roomID, billingAddress, cardNumber)

roomID(FOREIGN KEY - Referential integrity, PRIMARY KEY) - part of unique identifier and refers to entities within the database (Room)
billingAddress (NOT NULL) - billing address of the customer
cardNumber (NOT NULL) - the card number of the customer

RSAssignedTo(staffID, roomID, customerID)

staffID (FOREIGN KEY – Referential integrity) - refers to entities within the database (Staff)
roomID (FOREIGN KEY – Referential integrity, PRIMARY KEY) - unique identifier and refers to entities within the database (Room)
customerID (FOREIGN KEY – Referential integrity) - refers to entities within the database (Customer)

CAssignedTo(staffID, roomNumber, customerID)

staffID (FOREIGN KEY – Referential integrity) - refers to entities within the database (Staff)
roomID (FOREIGN KEY – Referential Integrity, PRIMARY KEY) - unique identifier and refers to entities within the database (Room)
customerID (NOT NULL, Referential integrity) - refers to entities within the database (Customer)

Room(hotelID, roomNumber, roomType, rate, maxOccupancy, roomID)

hotelID (PRIMARY KEY, FOREIGN KEY - referential integrity) - non unique hotel that the room belongs to
roomNumber (PRIMARY KEY - referential integrity) - non unique room number
roomID (PRIMARY KEY) - unique identifier and refers to entities with the database (Room)
roomType (NOT NULL, IN {“Economy”, “Deluxe”, “Executive Suite”, “Presidential Suite”}) - the type of a room
rate (NOT NULL) - rate of a room
maxOccupancy (NOT NULL) - max party size for a room

3) SQL Statements

Drop Table

```
DROP TABLE rsassignedto;
DROP TABLE cassignedto;
DROP TABLE CardNumber;
DROP TABLE Staff;
DROP TABLE Checkin;
DROP TABLE Customer;
DROP TABLE Room;
DROP TABLE Hotel;
```

Drop Sequence

```
DROP SEQUENCE staff_seq;
DROP SEQUENCE hotel_seq;
DROP SEQUENCE customer_seq;
DROP SEQUENCE room_seq;
```

Create Table

```
CREATE TABLE Hotel (
        hotelID INT,
        name VARCHAR(30) NOT NULL,
        address VARCHAR(50) NOT NULL UNIQUE,
        phone VARCHAR(16) NOT NULL UNIQUE,
        PRIMARY KEY (hotelID)
);

CREATE TABLE Staff (
        staffID INT,
        ssn INT NOT NULL,
        name VARCHAR2(30) NOT NULL,
        age INT NOT NULL,
        gender CHAR(1) NOT NULL,
        phone VARCHAR2(16) NOT NULL,
        address VARCHAR2(50) NOT NULL,
        staffType VARCHAR2(20) NOT NULL,
        hotelID INT,
        PRIMARY KEY (staffID),
        FOREIGN KEY (hotelID) REFERENCES hotel(hotelID) ON DELETE CASCADE,
        CONSTRAINT checkStaffType CHECK (staffType IN ('Manager', 'Front Desk', 'Catering', 'Room
```

```
                Service')),
        CONSTRAINT checkStaffGender CHECK (gender IN ('M','F')),
        CONSTRAINT checkStaffAge CHECK (age > 15)
);

CREATE TABLE Customer (
        customerID INT,
        name VARCHAR2(30) NOT NULL,
        gender CHAR(1) NOT NULL,
        address VARCHAR2(50) NOT NULL,
        phone VARCHAR2(16) NOT NULL,
        email VARCHAR2(40) NOT NULL,
        PRIMARY KEY (customerID),
        CONSTRAINT checkCustomerGender CHECK (gender IN('M', 'F'))
);

CREATE TABLE Room (
        hotelID INT,
        roomNumber INT NOT NULL,
        roomType VARCHAR2(20),
        rate FLOAT NOT NULL,
        maxOccupancy INT NOT NULL,
        roomID INT UNIQUE NOT NULL,
        PRIMARY KEY (hotelID, roomNumber, roomID),
        FOREIGN KEY (hotelID) REFERENCES hotel(hotelID) ON DELETE CASCADE,
        CONSTRAINT checkRoomType CHECK (roomType IN ('Economy', 'Deluxe', 'Executive Suite',
                'Presidential Suite')),
        CONSTRAINT occupancy CHECK (maxOccupancy BETWEEN 1 AND 4),
        CONSTRAINT minRate CHECK (rate > 0)
);

CREATE TABLE CheckIn (
        customerID INT,
        roomID INT,
        checkOutDateTime DATE NOT NULL,
        checkInDateTime DATE NOT NULL,
        numberInParty INT NOT NULL,
        hasCatering CHAR(1) NOT NULL,
        hasRoomService CHAR(1) NOT NULL,
        restaurantBill FLOAT DEFAULT 0.0,
        laundryBill FLOAT DEFAULT 0.0,
        phoneBill FLOAT DEFAULT 0.0,
        paymentMethod VARCHAR2(6) NOT NULL,
        ssnOfPayer INT NOT NULL,
```

```
        PRIMARY KEY (roomID),
        FOREIGN KEY (customerID) REFERENCES customer(customerID) ON DELETE CASCADE,
        FOREIGN KEY (roomID) REFERENCES room(roomID) ON DELETE CASCADE,
        CONSTRAINT checkHasCatering CHECK (hasCatering IN ('T','F')),
        CONSTRAINT checkHasRoomService CHECK (hasRoomService IN ('T','F')),
        CONSTRAINT checkNumberInParty CHECK (numberInParty BETWEEN 1 AND 4),
        CONSTRAINT method CHECK (paymentMethod IN('cash', 'check', 'credit')) ,
        CONSTRAINT minBill CHECK (restaurantBill >= 0 AND laundryBill>=0 AND phoneBill>=0),
        CONSTRAINT checkTime CHECK (checkOutDateTime > checkInDateTime) ,
        CONSTRAINT checkSSN CHECK (ssnOfPayer > 0)
);

CREATE TABLE CardNumber (
        roomID INT,
        billingAddress VARCHAR2(50) NOT NULL,
        cardNumber VARCHAR2(16) NOT NULL,
        PRIMARY KEY (roomID),
        FOREIGN KEY (roomID) REFERENCES Room(roomID) ON DELETE CASCADE
);

CREATE TABLE RSAssignedTo (
        staffID INT,
        roomID INT,
        customerID INT,
        PRIMARY KEY(roomID),
        FOREIGN KEY (customerID) REFERENCES Customer(customerID) ON DELETE CASCADE,
        FOREIGN KEY (staffID) REFERENCES staff(staffID) ON DELETE CASCADE,
        FOREIGN KEY (roomID) REFERENCES Room(roomID) ON DELETE CASCADE
);

CREATE TABLE CAssignedTo (
        staffID INT,
        roomID INT,
        customerID INT,
        PRIMARY KEY(roomID),
        FOREIGN KEY (customerID) REFERENCES Customer(customerID) ON DELETE CASCADE,
        FOREIGN KEY (staffID) REFERENCES staff(staffID) ON DELETE CASCADE,
        FOREIGN KEY (roomID) REFERENCES Room(roomID) ON DELETE CASCADE
);
```

Create Sequence

```
CREATE SEQUENCE hotel_seq MINVALUE 1 START WITH 1;
CREATE SEQUENCE staff_seq MINVALUE 1 START WITH 1;
CREATE SEQUENCE customer_seq MINVALUE 1 START WITH 1;
CREATE SEQUENCE room_seq MINVALUE 1 START WITH 1;
```

Load Data

```
INSERT INTO Hotel (hotelID, name, address, phone) VALUES (hotel_seq.nextval,'Wolf Villas Carrie', '3904 Lex St.; Carrie, SC', '396-768-3985');
INSERT INTO Hotel (hotelID, name, address, phone) VALUES (hotel_seq.nextval,'Good Night Hotel', '9997 Kirkland Lane, Brookfield, WI 53045', '345-388-4325');
INSERT INTO Hotel (hotelID, name, address, phone) VALUES (hotel_seq.nextval,'Peaceful Sleep Hotel', '798 Windfall Lane, Bartlett, IL 60103', '1-357-453-7643');
INSERT INTO Hotel (hotelID, name, address, phone) VALUES (hotel_seq.nextval,'Old Tavern', '65 Trusel Street, Fishers, IN 46037', '400-848-4956');

INSERT INTO Room (hotelID, roomNumber, roomType, rate, maxOccupancy, roomID) VALUES (1, 102, 'Economy', 89.0, 2, room_seq.nextval);
INSERT INTO Room (hotelID, roomNumber, roomType, rate, maxOccupancy, roomID) VALUES (2, 567, 'Executive Suite', 112.0, 4, room_seq.nextval);
INSERT INTO Room (hotelID, roomNumber, roomType, rate, maxOccupancy, roomID) VALUES (3, 993, 'Presidential Suite', 130.0, 4, room_seq.nextval);
INSERT INTO Room (hotelID, roomNumber, roomType, rate, maxOccupancy, roomID) VALUES (4, 413, 'Deluxe', 100.0, 3, room_seq.nextval);
INSERT INTO Room (hotelID, roomNumber, roomType, rate, maxOccupancy, roomID) VALUES (1, 993, 'Presidential Suite', 230.0, 4, room_seq.nextval);
INSERT INTO Room (hotelID, roomNumber, roomType, rate, maxOccupancy, roomID) VALUES (1, 413, 'Deluxe', 120.0, 4, room_seq.nextval);
INSERT INTO Room (hotelID, roomNumber, roomType, rate, maxOccupancy, roomID) VALUES (1, 101, 'Economy', 89.0, 2, room_seq.nextval);


INSERT INTO Staff (hotelID, staffID, ssn, name, age, gender, phone, address, staffType) VALUES (1, staff_seq.nextval, 943785749, 'Steven Lowe', 24, 'M', '336-384-3945', '3468 Stewart Road; Lala, Florida 30693', 'Manager');
INSERT INTO Staff (hotelID, staffID, ssn, name, age, gender, phone, address, staffType) VALUES (1, staff_seq.nextval, 847263749, 'John Pip', 30, 'F', '1-332-654-7544', '605 Backstreet; Winston-Hill, California 25095', 'Front Desk');
INSERT INTO Staff (hotelID, staffID, ssn, name, age, gender, phone, address, staffType) VALUES (1, staff_seq.nextval, 153850694, 'Rocky Pak', 72, 'F', '1-234-674-2953', '545 Cribbs St.; Huhu, ND 74629', 'Room Service');
```

```
INSERT INTO Staff (hotelID, staffID, ssn, name, age, gender, phone, address, staffType) VALUES (4, staff_seq.nextval, 172839474, 'Simon Pi', 55, 'M', '859-463-5768', '907 Quail Haven; Ralloo, FL 13505', 'Room Service');
INSERT INTO Staff (hotelID, staffID, ssn, name, age, gender, phone, address, staffType) VALUES (1, staff_seq.nextval, 746334006, 'Jeo Path', 24, 'M', '221-384-3945',  '3468 Stewart Road; Lala, Florida 30693', 'Catering');
INSERT INTO Staff (hotelID, staffID, ssn, name, age, gender, phone, address, staffType) VALUES (1, staff_seq.nextval, 564883772, 'Mary Ison', 30, 'F', '414-633-5234', '605 Backstreet; Winston-Hill, California 25095', 'Catering');
INSERT INTO Staff (hotelID, staffID, ssn, name, age, gender, phone, address, staffType) VALUES (2, staff_seq.nextval, 968477357, 'Tarry Hill', 72, 'M', '826-234-6633', '545 Cribbs St.; Huhu, ND 74629', 'Catering');
INSERT INTO Staff (hotelID, staffID, ssn, name, age, gender, phone, address, staffType) VALUES (3, staff_seq.nextval, 365758695, 'Marina Harris', 55, 'F', '1-222-567-4535', '907 Quail Haven; Ralloo, FL 13505', 'Catering');

INSERT INTO Customer (customerID, name, gender, address, phone, email) VALUES (customer_seq.nextval, 'Matthew Johnson', 'M', '1001 Bull Road; Yak, GA 20594', '694-693-7940','mjohnson@uwoe.com');
INSERT INTO Customer (customerID, name, gender, address, phone, email) VALUES (customer_seq.nextval, 'Jim Stevens', 'M', '734 Yellmans Dr.; Ciy, NY 21537', '184-684-9490', 'jimmysjunkmail@hotmail.com');
INSERT INTO Customer (customerID, name, gender, address, phone, email) VALUES (customer_seq.nextval, 'Matthew Johnson', 'M', '313 Red Tiger, Private Dr., Hawaii 74850', '232-777-5499', 'matthewj1264@gmail.com');
INSERT INTO Customer (customerID, name, gender, address, phone, email) VALUES (customer_seq.nextval, 'Kat Felion', 'F', '1161 Likoo; Awall, Texas 12545', '758-857-1849', 'kfaacha12@hotmail.com');

INSERT INTO CheckIn(customerID, roomID, checkOutDateTime, checkInDateTime, numberInParty, hasCatering, hasRoomService, paymentMethod, ssnOfPayer) VALUES (1, 1, TO_DATE('2016/12/22 12:00:00', 'yyyy/mm/dd hh24:mi:ss'), TO_DATE('2016/12/20 12:00:00', 'yyyy/mm/dd hh24:mi:ss'), 2, 'F', 'T', 'credit', 935774336);
INSERT INTO CheckIn(customerID, roomID, checkOutDateTime, checkInDateTime, numberInParty, hasCatering, hasRoomService, paymentMethod, ssnOfPayer) VALUES (2, 2, TO_DATE('2016/10/21 12:00:00', 'yyyy/mm/dd hh24:mi:ss'), TO_DATE('2016/10/20 12:00:00', 'yyyy/mm/dd hh24:mi:ss'), 3, 'T', 'F', 'credit', 935774336);
INSERT INTO CheckIn(customerID, roomID, checkOutDateTime, checkInDateTime, numberInParty, hasCatering, hasRoomService, paymentMethod, ssnOfPayer) VALUES (3, 3, TO_DATE('2016/10/25 12:00:00', 'yyyy/mm/dd hh24:mi:ss'), TO_DATE('2016/10/20 12:00:00', 'yyyy/mm/dd hh24:mi:ss'), 4, 'T', 'T', 'cash', 574993882);
INSERT INTO CheckIn(customerID, roomID, checkOutDateTime, checkInDateTime, numberInParty, hasCatering, hasRoomService, paymentMethod, ssnOfPayer) VALUES (4, 4, TO_DATE('2016/12/04
```

12:00:00', 'yyyy/mm/dd hh24:mi:ss'), TO_DATE('2016/11/08 12:00:00', 'yyyy/mm/dd hh24:mi:ss'), 2, 'F', 'F', 'check', 921645339);
INSERT INTO CheckIn(customerID, roomID, checkOutDateTime, checkInDateTime, numberInParty, hasCatering, hasRoomService, paymentMethod, ssnOfPayer) VALUES (1, 5, TO_DATE('2016/12/22 12:00:00', 'yyyy/mm/dd hh24:mi:ss'), TO_DATE('2016/12/20 12:00:00', 'yyyy/mm/dd hh24:mi:ss'), 2, 'T', 'T', 'credit', 935774336);
INSERT INTO CheckIn(customerID, roomID, checkOutDateTime, checkInDateTime, numberInParty, hasCatering, hasRoomService, paymentMethod, ssnOfPayer) VALUES (2, 6, TO_DATE('2016/10/21 12:00:00', 'yyyy/mm/dd hh24:mi:ss'), TO_DATE('2016/10/20 12:00:00', 'yyyy/mm/dd hh24:mi:ss'), 3, 'T', 'T', 'credit', 935774336);
INSERT INTO CheckIn(customerID, roomID, checkOutDateTime, checkInDateTime, numberInParty, hasCatering, hasRoomService, paymentMethod, ssnOfPayer) VALUES (3, 7, TO_DATE('2016/10/21 12:00:00', 'yyyy/mm/dd hh24:mi:ss'), TO_DATE('2016/10/20 12:00:00', 'yyyy/mm/dd hh24:mi:ss'), 3, 'T', 'T', 'credit', 935774336);

INSERT INTO CardNumber (roomID, billingAddress, cardNumber) VALUES (1, '24 Middle River St., Wayne, NJ 07470', '8888888888888888');
INSERT INTO CardNumber (roomID, billingAddress, cardNumber) VALUES (2, '86 East Shipley Lane, Livingston, NJ 07039', '7777777777777777');
INSERT INTO CardNumber (roomID, billingAddress, cardNumber) VALUES (5, '7958 Rockland Ave., Palmetto, FL 34221', '1111111111111111');
INSERT INTO CardNumber (roomID, billingAddress, cardNumber) VALUES (6, '394 Wazzoo Drive; Maya, OH 9302', '2222222222222222');

INSERT INTO RSAssignedTo(staffID, roomID, customerID) VALUES (3, 3, 3);
INSERT INTO RSAssignedTo(staffID, roomID, customerID) VALUES (3, 5, 1);
INSERT INTO RSAssignedTo(staffID, roomID, customerID) VALUES (4, 1, 1);
INSERT INTO RSAssignedTo(staffID, roomID, customerID) VALUES (4, 6, 2);

INSERT INTO CAssignedTo(staffID, roomID, customerID) VALUES (5, 3, 3);
INSERT INTO CAssignedTo(staffID, roomID, customerID) VALUES (5, 5, 1);
INSERT INTO CAssignedTo(staffID, roomID, customerID) VALUES (6, 2, 2);
INSERT INTO CAssignedTo(staffID, roomID, customerID) VALUES (6, 6, 2);

Selects

Note: All result sets from select statements have been formatted for readability.

SQL> SELECT * FROM Hotel;

HOTELID	NAME	ADDRESS	PHONE
1	Wolf Villas Carrie	3904 Lex St.; Carrie, SC	396-768-3985
2	Good Night Hotel	9997 Kirkland Lane, Brookfield, WI 53045	345-388-4325
3	Peaceful Sleep Hotel	798 Windfall Lane, Bartlett, IL 60103	1-357-453-7643
4	Old Tavern	65 Trusel Street, Fishers, IN 46037	400-848-4956

SQL> SELECT * FROM Staff;

STAFFID	SSN	NAME	AGE	G	PHONE	ADDRESS	STAFFTYPE	HOTELID
1	943785749	Steven Lowe	24	M	336-384-3945	3468 Stewart Road; Lala, Florida 30693	Manager	1
2	847263749	John Pip	30	F	1-332-654-7544	605 Backstreet; Winston-Hill, California 25095	Front Desk	1
3	153850694	Rocky Pak	72	F	1-234-674-2953	545 Cribbs St.; Huhu, ND 74629	Room Service	1
4	172839474	Simon Pi	55	M	859-463-5768	907 Quail Haven; Raloo, FL 13505	Room Service	4
5	746334006	Jeo Path	24	M	221-384-3945	3468 Stewart Road; Lala, Florida 30693	Catering	1
6	564883772	Mary Ison	30	F	414-633-5234	605 Backstreet; Winston-Hill, California 25095	Catering	1
7	968477357	Tarry Hill	72	M	826-234-6633	545 Cribbs St.; Huhu, ND 74629	Catering	2
8	365758695	Marina Harris	55	F	1-222-567-4535	907 Quail Haven; Raloo, FL 13505	Catering	3

SQL> SELECT * FROM RSAssignedTo;

STAFFID	ROOMID	CUSTOMERID
3	3	3
3	5	1
4	1	1
4	6	2

SQL> SELECT * FROM CAssignedTo;

STAFFID	ROOMID	CUSTOMERID
5	3	3
5	5	1
6	2	2
6	6	2

SQL> SELECT * FROM Customer;

CUSTOMERID	NAME	G	ADDRESS	PHONE	EMAIL
1	Matthew Johnson	M	1001 Bull Road; Yak, GA 20594	694-693-7940	mjohnson@uwoe.com
2	Jim Stevens	M	734 Yellmans Dr.; City, NY 21537	184-684-9490	jimmysjunkmail@hotmail.com
3	Matthew Johnson	M	313 Red Tiger, Private Dr., Hawaii 74850	232-777-5499	matthewj1264@gmail.com
4	Kat Felion	F	1161 Likoo; Awall, Texas 12545	758-857-1849	kfaacha12@hotmail.com

SQL> SELECT * FROM CheckIn;
(hasCatering and hasRoomService only show up as ‘H’ in the queries because they are designated as VARCHAR2(1). This means that the column width is only 1, so it only shows the first letter of the variable name)

```
CUSTOMERID ROOMID CHECKOUTD CHECKINDA NUMBER  H H RESTAURANT LAUNDRY PHONE PAYMEN SSNOFPAYER
                                      INPARTY     BILL       BILL    BILL
---------- ------ --------- --------- ------- - - ---------- ------- ----- ------ ----------
         1      1 22-DEC-16 20-DEC-16       2 F T          0       0     0 credit  935774336
         2      2 21-OCT-16 20-OCT-16       3 T F          0       0     0 credit  935774336
         3      3 25-OCT-16 20-OCT-16       4 T T          0       0     0 cash    574993882
         4      4 04-DEC-16 08-NOV-16       2 F F          0       0     0 check   921645339
         1      5 22-DEC-16 20-DEC-16       2 T T          0       0     0 credit  935774336
         2      6 21-OCT-16 20-OCT-16       3 T T          0       0     0 credit  935774336
         3      7 21-OCT-16 20-OCT-16       3 T T          0       0     0 credit  935774336
```

SQL> SELECT * FROM CardNumber;

```
   ROOMID BILLINGADDRESS                                CARDNUMBER
---------- ------------------------------------------- ----------------
         1 24 Middle River St., Wayne, NJ 07470        8888888888888888
         2 86 East Shipley Lane, Livingston, NJ 07039  7777777777777777
         5 7958 Rockland Ave., Palmetto, FL 34221      1111111111111111
         6 394 Wazzoo Drive; Maya, OH 9302             2222222222222222
```

SQL> SELECT * FROM Room;

```
  HOTELID ROOMNUMBER ROOMTYPE                 RATE MAXOCCUPANCY     ROOMID
---------- ---------- -------------------- ---------- ------------
         1        102 Economy                      89            2          1
         2        567 Executive Suite             112            4          2
         3        993 Presidential Suite          130            4          3
         4        413 Deluxe                      100            3          4
         1        993 Presidential Suite          230            4          5
         1        413 Deluxe                      120            4          6
         1        101 Economy                      89            2          7
```

4) Interactive SQL Queries

4.1 Queries for tasks and operations

Staff Info Operations

Enter in a new hotel staff member:

SQL > INSERT INTO Staff (hotelID, staffID, ssn, name, age, gender, phone, address, staffType) VALUES (2, staff_seq.nextval, '476940923', 'Nancy Taylor', 31, 'F', '336444563', '3438 Tolly Avenue; Manchu, PA 30693', 'Catering');

1 row created.

Update some information for hotel staff member:

SQL > UPDATE Staff SET ssn=892957493, age=32 WHERE staffID=2;

1 row updated.

Delete information of hotel staff member:
SQL > DELETE from Staff WHERE staffID=2;

1 row deleted.

Room Info Operations
Add in a new room:
SQL > INSERT INTO Room (roomID, hotelID, roomNumber, roomType, rate, maxOccupancy) VALUES (room_seq.nextval, 2, 101, 'Deluxe', 100.0, 3);

1 row created.

Update information about a room:
SQL > UPDATE Room SET rate=95.0 WHERE roomID=3;

1 row updated.

Delete a room:
SQL > DELETE from Room WHERE roomID=3;

1 row deleted.

Check what rooms are available given a roomType (in this case, 'Economy'), maxOccupancy (in this case, 2):
SQL > SELECT DISTINCT room.roomID FROM Room, CheckIn WHERE maxOccupancy>1 AND roomType='Economy' AND Room.roomID NOT IN (SELECT roomID from CheckIn);

no rows selected
(All rooms are currently occupied according to our bulk data, so this finds no rooms)

Assign a room (NOTE: only do this after room availability has been checked):
SQL > INSERT INTO CheckIn(customerID, roomID, checkOutDateTime, checkInDateTime, numberInParty, hasCatering, hasRoomService, paymentMethod, ssnOfPayer) VALUES (3, 7, TO_DATE('2016/10/21 12:00:00', 'yyyy/mm/dd hh24:mi:ss'), TO_DATE('2016/10/20 12:00:00', 'yyyy/mm/dd hh24:mi:ss'), 1, 'T', 'T', 'credit', 935774336);

1 row created.
(If roomID 7 wasn't already checked in this is what we would get, we currently get a constraint violation, since we are trying to add a checkin for a room that already has someone checked in.)

Release a room (from the assumptions, a customer can only book a fixed room for only one time range at a time):
SQL > DELETE from CheckIn WHERE roomID=4;

1 row deleted.

Customer Info Operations

Enter in a new customer:

SQL > INSERT INTO Customer (customerID, name, gender, address, phone, email) VALUES (customer_seq.nextval, 'Bob Felion', 'M', '1161 Likoo; Awall, Texas 12545', '758-857-1849', 'BobFelton@hotmail.com');

1 row created.

Update information for a customer:

SQL > UPDATE Customer SET name='Katherine Felion', address='1161 Likoo Avenue; Awall, Texas 12545' WHERE customerID=4;

1 row updated.

Delete a customer:

SQL > DELETE from Customer WHERE customerID=3;

1 row deleted.

Service/Billing Info Operations

Enter / Update a service (In this case, add 90 dollars to the restaurant bill of the customer checked into roomID=2)

SQL > UPDATE CheckIn SET restaurantBill=(select restaurantBill from checkin where customerID=2 and roomID=2)+90.0 WHERE roomID=2;

1 row updated.

Update a checkIn to have catering

SQL > UPDATE CheckIn SET hasCatering='T' WHERE roomID=1;

1 row updated.

Delete a service (In this case, remove 90 dollars from the restaurant bill of a customer checked into room 2)

SQL > UPDATE CheckIn SET restaurantBill=(select restaurantBill from checkin where customerID=2 and roomID=2)-90.0 WHERE customerID=2 and roomID=2;

1 row updated.

Update a checkin to remove catering

SQL > UPDATE CheckIn SET hasCatering='F' WHERE customerID=1 and roomID=1;

1 row updated.

Generate a new billing account for a particular customer

SQL > INSERT INTO CardNumber (roomID, billingAddress, cardNumber) VALUES (7, '394 Wazzoo Drive; Maya, OH 9302', 4444444444444444);

1 row created.

Delete a billing account for a particular customer
SQL > DELETE from CardNumber WHERE roomID=1;

1 row deleted.

Generate a billing report for a particular customer
SQL > SELECT rate, restaurantBill, laundryBill, phoneBill, hasCatering, hasRoomService, checkInDateTime, checkOutDateTime FROM CheckIn, Room WHERE customerID=1 AND CheckIn.roomID=Room.roomID;

```
      RATE RESTAURANTBILL LAUNDRYBILL  PHONEBILL H H CHECKINDA CHECKOUTD
---------- ------------- ----------- ---------- - - --------- ---------
        89             0           0          0 F T 20-DEC-16 22-DEC-16
       230             0           0          0 T T 20-DEC-16 22-DEC-16
```

(With the information from this table we can easily build a total bill for a customer in our implementation. Java makes it easy to find the number of days between two dates, which can be multiplied by the room rates and added to the other bills. The number of days can also be multiplied by a set rate for room service or catering if those services were set to 'T'. This may not produce an exact total figure all at once, but it does give all the necessary information to do so, without needing to do the absolutely enormous SQL statement that would be needed to produce such a figure. Further, we have no idea how to extract the number of days between two dates in SQL, while this is extremely simple to do on the implementation side.)

Delete a billing report (Notice that this deletes billing info for ALL the rooms a customer may have checked into) for a particular customer
SQL > DELETE from CheckIn WHERE roomID =4 ;

1 row deleted.

<u>Reporting Operations</u>
Report the occupancy by room type, date range, hotel

SELECT DISTINCT room.roomID
FROM room, checkin
WHERE room.roomID = checkin.roomID AND room.hotelID=1 AND roomType='Deluxe' AND checkInDateTime BETWEEN TO_DATE('2016/09/21 12:00:00', 'yyyy/mm/dd hh24:mi:ss') AND TO_DATE('2017/01/21 12:00:00', 'yyyy/mm/dd hh24:mi:ss');

```
    ROOMID
----------
         6
```

Return list of occupants of hotel
SQL > SELECT DISTINCT CheckIn.customerID
FROM Customer, CheckIn, Room
WHERE Customer.customerID=CheckIn.customerID AND
CheckIn.roomID=Room.roomID AND
Room.hotelID=1;

```
CUSTOMERID
----------
         1
         2
         3
```

Return percentage of rooms occupied in hotel
Explanation: This is a fraction where the query in the numerator gives the count of the rooms in the hotel with hotelID=1 *that have been checked into* (and therefore are occupied), and the query in the denominator gives the count of the total number of rooms in the hotel with hotelID=1.
SQL > SELECT COUNT(*) FROM CheckIn, Room WHERE checkIn.roomID=Room.roomID AND hotelID=1;

```
  COUNT(*)
----------
         4
```

SQL > SELECT COUNT(*) FROM Room WHERE hotelID=1;

```
  COUNT(*)
----------
         4
```

(The top number can then be divided by the bottom number to get the % of rooms in the given hotel that are currently occupied. In our bulk data every room is occupied, so this works out to 100% occupation as expected.)

Return list of staff members with the specific job title (all attributes of HotelStaff relation is used)
SQL > SELECT staffID FROM Staff WHERE staffType='Manager' AND hotelID = 1;

```
   STAFFID
----------
         1
```

SQL > SELECT staffID FROM Staff WHERE staffType='Front Desk' AND hotelID = 1;

```
   STAFFID
----------
         2
```

SQL > SELECT staffID FROM Staff WHERE staffType='Room Service' AND hotelID = 1;

```
   STAFFID
----------
         3
         4
```

SQL > SELECT staffID FROM Staff WHERE staffType='Catering' AND hotelID = 1;

```
   STAFFID
----------
         5
         6
         7
         8
```

Return list of customers assigned to the particular *Catering* staff member (all attributes of Customer relation is used)

SQL > SELECT Customer.customerID
FROM Customer, CAssignedTo
WHERE staffID=5 AND Customer.customerID=CAssignedTo.customerID;

```
CUSTOMERID
----------
         3
         1
```

Return list of customers assigned to the particular *Room Service* staff member (all attributes of Customer relation is used)

SQL > SELECT Customer.customerID
FROM Customer, RSAssignedTo
WHERE staffID=3 AND Customer.customerID=RSAssignedTo.customerID;

```
CUSTOMERID
----------
         3
         1
```

4.2 Autotrace and Indexes for two tables

SQL> SET AUTOTRACE ON;

Query #1 Return list of staff members with the who work at hotelID=1 and the Front Desk
// First, run the query without having created the index
SQL> SELECT staffID FROM Staff WHERE staffType='Front Desk' AND hotelID = 1;

```
   STAFFID
----------
         2


Execution Plan
----------------------------------------------------------
Plan hash value: 1776966647

--------------------------------------------------------------------------
| Id  | Operation         |Name |Rows| Bytes | Cost (%CPU)| Time       |
--------------------------------------------------------------------------
|   0 |SELECT STATEMENT |      |  1 |    38 |     3   (0)| 00:00:01   |
|*  1 |TABLE ACCESS FULL|STAFF|  1 |    38 |     3   (0)| 00:00:01   |
--------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   1 - filter("STAFFTYPE"='Front Desk' AND "HOTELID"=1)

Note
-----
   - dynamic sampling used for this statement (level=2)
```

// Create the index
SQL > CREATE INDEX StaffTypeIndex ON Staff(staffType);

Index created.

// Now run the query having created the index
SQL > SELECT staffID FROM Staff WHERE staffType='Front Desk';

```
   STAFFID
----------
         2


Execution Plan
----------------------------------------------------------
Plan hash value: 3193903112

-------------------------------------------------------------------------------------------------
| Id  | Operation                    | Name           | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |                |     1 |    25 |     2   (0)| 00:00:01 |
|   1 |  TABLE ACCESS BY INDEX ROWID| STAFF          |     1 |    25 |     2   (0)| 00:00:01 |
|*  2 |   INDEX RANGE SCAN           | STAFFTYPEINDEX |     1 |       |     1   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   2 - access("STAFFTYPE"='Front Desk')

Note
-----
   - dynamic sampling used for this statement (level=2)
```

Query #2: Return ID's of all customers who are male

// First, run the query without having created the index

SQL > SELECT customerID FROM customer WHERE gender='M';

```
CUSTOMERID
----------
         1
         2
         3


Execution Plan
----------------------------------------------------------
Plan hash value: 2844954298

--------------------------------------------------------------------------------
| Id  | Operation         | Name     | Rows  | Bytes | Cost (%CPU)| Time     |
--------------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |          |     3 |    48 |     3   (0)| 00:00:01 |
|*  1 |  TABLE ACCESS FULL| CUSTOMER |     3 |    48 |     3   (0)| 00:00:01 |
--------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   1 - filter("GENDER"='M')

Note
-----
   - dynamic sampling used for this statement (level=2)
```

// Create the index

SQL > CREATE INDEX GenderIndex ON Customer(gender);

Index Created.

// Now run the query having created the index
SQL > SELECT customerID FROM customer WHERE gender='M';

```
CUSTOMERID
----------
         1
         2
         3


Execution Plan
----------------------------------------------------------
Plan hash value: 2186662808

-------------------------------------------------------------------------------------------------
| Id  | Operation                    | Name        | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |             |     3 |    48 |     2   (0)| 00:00:01 |
|   1 |  TABLE ACCESS BY INDEX ROWID| CUSTOMER    |     3 |    48 |     2   (0)| 00:00:01 |
|*  2 |   INDEX RANGE SCAN           | GENDERINDEX |     1 |       |     1   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   2 - access("GENDER"='M')

Note
-----
   - dynamic sampling used for this statement (level=2)
```

4.3 Query correctness proofs

Query #1
Return list of customers (by customer ID) assigned to the particular *Catering* staff member
SQL > SELECT Customer.customerID
FROM Customer, CAssignedTo
WHERE staffID=2 AND Customer.customerID=CAssignedTo.customerID;

1) The relational algebra expression is:

$$\pi_{\text{Customer.customerID}}(\text{Customer} \bowtie_{\textit{staffID = 2 AND Customer.customerID=CAssignedTo.customerID}} \text{(CAssignedTo)})$$

2) Customer and CAssignedTo will be paired to each other by the theta-join in the relational algebra expression above if a tuple in Customer has a customerID attribute value that is equal to the customerID attribute value of a tuple in CAssignedTo (to get the list of all customers who have catering paired with their catering info). In addition, the staffID attribute value of CAssignedTo must be equal to 2 (to filter the previous list to only those customers paired with their catering info where the catering staff member has a staffID of 2). Since only the customer ID of each customer is being requested, the projection operator with the customerID attribute is applied after the theta-join is carried out. This is exactly what the query should return.

Query #2

Return billing report list for a particular customer

SQL > SELECT rate, restaurantBill, laundryBill, phoneBill, hasCatering, hasRoomService, checkInDateTime, checkOutDateTime FROM CheckIn, Room WHERE customerID=1 AND CheckIn.roomID=Room.roomID;

1) The corresponding relational algebra expression is:

πrate, restaurantBill, laundryBill, phoneBill, hasCatering, hasRoomService, checkInDateTime, checkOutDateTime(CheckIn $\bowtie_{customerID=1\ AND\ CheckIn.roomID=Room.roomID}$ (Room))

2) CheckIn and Room will be paired to each other by the theta-join in the relational algebra expression above if a tuple in CheckIn has a roomID attribute value that is equal to the roomID attribute value of a tuple in Room (to get the list of all rooms checked into paired with the corresponding check-in info). In addition, the customerID attribute value of CheckIn must be equal to 1 (to filter the previous list to only those customers which have a customerID of 1). The projection operator with the customer's bill attributes is applied after the theta-join is carried out; it extracts the rate, restaurantBill, laundryBill, phoneBill, hasCatering, hasRoomService, checkInDateTime, and checkOutDateTime. This is exactly what the query should return.