Wolf Inns
Hotel Chain Management System

CSC 540
Database Management Concepts and Systems

Project Report # 2

Team C

Abhay Soni	(asoni3)
Aurora Tiffany-Davis	(attiffan)
Manjusha Trilochan Awasthi	(mawasth)
Samantha Scoggins	(smscoggi)

Table of Contents

Project Assumptions

1. Customers

    1.1. A customer has exactly one phone number (that the system tracks at any one time).
    1.2. A customer has exactly one email address (that the system tracks at any one time).
    1.3. All Customers will have a United States Social Security Number.

2. Staff

    2.1. Each staff member has exactly one job title and department.
    2.2. A staff member's job title indicates what they are expected to do within the system. However, a manager may need to fill in for a colleague, and so for example a manager may provide any type of service regardless of job title. For example, if all staff members except for the manager are dedicated to presidential suites, then the hotel manager would have to run the rest of the hotel.
    2.3. A staff member's department is merely an attribute and is not expected to influence how the staff member interacts with the system. This attribute has no functional impact to the system, and is maintained purely due to its presence in the project narrative.
    2.4. Each staff member is assigned to at most one hotel.
    2.5. A hotel has exactly one manager, and a manager manages exactly one hotel. Therefore, if you wish to fire a hotel manager, you must update the hotel information with a new manager within the same transaction.
    2.6. Because "Front desk representatives… bill customers", "Billing Staff" is just another job title meaning "Front Desk Representative".
    2.7. Staff members know their own staff ID as well as the hotel ID of the hotel they are currently serving.
    2.8. A presidential suite is assigned exactly one room service staff member and exactly one catering staff member at check-in.
    2.9. Staff ID is unique across all hotels.
    2.10. If room category is presidential suite, then the room will be considered unavailable unless there is both a not-yet-dedicated room service staff member, and a not-yet-dedicated catering staff member assigned to the hotel.

3. People

    3.1. Several staff members and/or customers may have the same name, or the same phone number, or the same address, or the same email.

4. System Interaction

    4.1. The system has no UI.

5. Rooms

5.1. Room requests may involve more than room category (“Basic”, “Deluxe”, etc.). That is, assigning rooms to customers according to their request criteria may include some or all of: hotel location, room category, room (nightly) rate, maximum occupancy.
5.2. Room prices vary by location and class of service, but are not strictly defined by location and class of service. That is, each room may be given a unique price when the room is added to the system.
5.3. Each hotel may have any number of rooms of any given category. For example, “Presidential Suite” is not necessarily unique within a hotel.
5.4. Reports of occupancy are expected to report occupancy by room, and not by guest (e.g. “4 rooms occupied” rather than “6 guests staying”).
5.5. “Total occupancy” refers to the total number of rooms occupied.
5.6. Maximum occupancy is not determined by room category (“Basic”, “Deluxe”, etc).

6. Services

6.1. Each instance of a service rendered for a customer is provided by one staff member. That is, staff members do not team up to bring room service, etc.
6.2. The only standard services offered by the hotel are phone bills, dry cleaning, gyms, room service, and catering. All other services are considered “special requests”.
6.2.1. Every Wolf Inns hotel charges the same rate/minute for phone use.
6.2.2. Every Wolf Inns hotel charges the same rate/item for dry cleaning.
6.2.3. Every Wolf Inns hotel charges the same rate/use for gyms.
6.2.4. Every Wolf Inns hotel charges the same rate/meal for catering.
6.2.5. Every Wolf Inns hotel charges the same rate for special requests.
6.2.6. No Wolf Inns hotel charges customers for room service (ordering a meal to be delivered to your room is handled as a catering charge).
6.3. All services must be provided by a staff member with a job title matching the service provided. There are only two exceptions to this rule. First, a manager may provide any type of service. Second, phone service or a special request may be provided by any staff member.
6.4. All services must be provided by a staff member assigned to the hotel where the customer’s stay is taking place.
6.5. A staff member dedicated to the service of one room may not provide a service to another room.
6.6. Wolf Inns does not provide the service of “splitting your bill”.
6.6.1. Each instance of a service rendered is associated with a single hotel room stay. For example, multiple occupied rooms cannot jointly request catering. That is, there is one bill per room.
6.6.2. Each hotel bill has one customer responsible for payment.

7. Check-In / Check-Out

7.1. A check-in cannot proceed without necessary billing information available:
- 7.1.1. Customer SSN
- 7.1.2. Customer name
- 7.1.3. Payment method
- 7.1.4. Card Type if payment is card:
  - 7.1.4.1. Accepted values: ‘VISA’, ’MASTERCARD’, ’HOTEL’
- 7.1.5. Card # if payment method is card
- 7.1.6. Customer billing address if payment method is card

7.2. A check-in can proceed without unnecessary billing information available:
- 7.2.1. Card type if payment method is not card
- 7.2.2. Card # if payment method is not card
- 7.2.3. Customer billing address if payment method is not card

7.3. The customer who checks in is responsible for paying for their stay. Information about guests physically staying in the rooms is not tracked, with the exception of tracking the number of such guests.

7.4. The customer may not wish to use the same method of payment for multiple Wolf Inns hotel stays.

7.5. A bill, and therefore a customer stay, is considered to take place in a single room. That is, if a customer books many rooms (family reunion, etc), separate bills will be generated for each room.

7.6. Each customer stay will be identified by a generated ID which is unique across all stays regardless of which hotel the stay takes place in.

7.7. Check-Out Time and End-Date will be entered at the time when a Customer checks out. And for this reason, these attributes are allowed to be NULL prior to the guest checking out.

8. Reports

8.1. Updating basic information about rooms may include updating a room’s nightly rate. Therefore, at the end of each stay we must preserve the total amount owed for the stay. This is in order to accurately generate information about revenue earned from past stays.

8.2. In the operation “For each customer stay, return information on all the staff members serving the customer during the stay”, “for each customer stay” can best be translated to “for a given customer stay”, rather than “for every single customer stay”. This assumption is made due to the perceived usefulness of the former, and the likely overwhelming length of the latter, once the Wolf Inns hotel chain is blessed with many customer stays.

9. Removing Records

9.1. Aside from the assumptions listed below, removing a record from the Wolf Inns database system will be isolated and will not “cascade” removal to other records.
9.2. If a hotel is removed from the Wolf Inns system, then all rooms within that hotel are also removed.
9.3. If a room is removed from the Wolf Inns system, then all information associated with this room is removed from the system, entirely. For example, all stays associated with that room are also removed.
9.4. If a customer is removed from the Wolf Inns system, then all information about this customer is removed from the system, entirely. For example, all stays associated with that customer are also removed.
9.5. If a stay or a service type is removed from the Wolf Inns system, then any record of an instance of a service provided that is associated with that stay or service type shall also be removed.
9.6. If a staff member is removed from the Wolf Inns system, then any record of an instance of a service provided by that staff member shall have the staff ID associated with the record set to null.
9.7. Any hotel, room, customer, staff member, or service type associated with a current guest stay may not be deleted.
9.8. Special note: As noted in a previous project assumption, a hotel has exactly one manager, and a manager manages exactly one hotel. Therefore, if you wish to fire a hotel manager, you must update the hotel information with a new manager within the same transaction. If this is not done, then removing a manager may result in also removing a hotel. Please note that this would indicate a serious problem, and is not intended to occur.

10. Updating Records

10.1. Any room associated with a current guest stay may not be updated.
10.2. Any staff member currently dedicated to serving a presidential suite may not have their job title changed.
10.3. While the Wolf Inns system supports updating most aspects of a record, the primary key of any record can never be changed - this is not supported by the Wolf Inns system.
10.3.1. One example is the SSN of a customer. A person’s social security number cannot be changed, and it is for this reason that SSN was chosen as the primary key for the Customers relation.
10.3.2. Other examples of primary keys, which the Wolf Inns system does not support changing, are similar. These are aspects of a record which by definition define the record, and so if they are incorrect, the record itself can be considered invalid.
10.3.3. Such examples are the name of a service type, and the combination of room number and hotel ID for a room.

11. Miscellaneous

11.1. Wolf Inns does not accept reservations.
11.2. Wolf Inns will never be exposed to a hacking attempt, such that security related concerns, including access control, is not a necessary concern.
11.3. Wolf Inns handles employee payroll, etc. through a separate system.
11.4. If information that Wolf Inns should maintain is computable from other already explicitly stored information, then it is considered “maintained” and there is no need to explicitly store it.
11.5. If a question about a customer, stay, hotel, etc. is not required by the project narrative, then there is no need to support it.
11.6. A hotel’s name needn’t be unique, for example we may have several hotels named / branded “Econo Wolf”, or “Wolf Deluxe Stay”.
11.7. A phone number cannot belong to more than one hotel.

Global Schema

"→" means "functionally determines"
"↠" means "does not functionally determine"

1. Hotels

    1.1. Schema

    Hotels (ID, Name, StreetAddress, City, State, PhoneNum, ManagerID)

    1.2. Functional Dependencies

    1.2.1. ID → Name, StreetAddress, City, State, PhoneNum, ManagerID

    1.2.2. StreetAddress, City, State → ID, Name, PhoneNum, ManagerID
    - This means that StreetAddress, City, State is a key, however it is not the primary key for practical concerns, namely that if it were the primary key it would be more cumbersome for tuples in other relations to refer to a unique hotel with a foreign key.

    1.2.3. PhoneNum → ID, Name, StreetAddress, City, State, ManagerID
    - This means that PhoneNum is a key, however it is not the primary key for practical concerns, namely that the phone number might change.

    1.2.4. ManagerID → ID, Name, StreetAddress, City, State, PhoneNum
    - While Managers might change, there is exactly one ManagerID per Hotel at any given time. So, inputting ManagerID will output a unique ID for a hotel. This means that ManagerID is a key, however it is not the primary key for practical concerns, namely that the Manager might change.

    1.2.5. Name ↠ StreetAddress, City, State
    - Name might be "Econo Wolf" or "Wolf Deluxe Stay" and these names may not be unique for a particular hotel in the hotel chain. This is one of our project assumptions.

    1.2.6. StreetAddress ↠ City State
    - StreetAddress might be "123 Main Street", and this address may not be unique to a particular city.

    1.2.7. City ↠ State
    - City might be "Franklin" and state might be "Arkansas" or "Georgia".

1.3. 3NF Satisfaction

There are four functional dependencies in the Hotels schema.

1.3.1. ID → Name, StreetAddress, City, State, PhoneNum, ManagerID

- The left hand side of this functional dependency is ID, which is a superkey for the schema. Therefore this FD does not violate 3NF (or even BCNF).

1.3.2. StreetAddress, City, State → ID, PhoneNum, ManagerID

- The left hand side of this functional dependency is StreetAddress, City, State, which is a superkey for the schema (since ID is a superkey). Therefore this FD does not violate 3NF (or even BCNF).

1.3.3. PhoneNum → ID, Name, StreetAddress, City, State, ManagerID

- The left hand side of this functional dependency is PhoneNum, which is a superkey for the schema (since ID is a superkey). Therefore this FD does not violate 3NF (or even BCNF).

1.3.4. ManagerID → ID, Name, StreetAddress, City, State, PhoneNum

- The left hand side of this functional dependency is ManagerID, which is a superkey for the schema (since ID is a superkey). Therefore this FD does not violate 3NF (or even BCNF).

2. Rooms

2.1. Schema

Rooms (RoomNum, HotelID, Category, MaxOcc, NightlyRate, DRSStaff, DCStaff)

Note: “DRSStaff” is the room service staff member who is dedicated to serving a given room.
Note: “DCStaff” is the catering staff member who is dedicated to serving a given room.

2.2. Functional Dependencies

2.2.1. RoomNum, HotelID → Category, MaxOcc, NightlyRate, DRSStaff, DCStaff

2.2.2. DRSStaff, DCStaff ↠ RoomNum, HotelID

- A room may have NULL values for DRSStaff and DCStaff (for example if the room is not a presidential suite, or if the room is not occupied, then there is no need to have dedicated staff for the room).

2.2.3. Category ↠ MaxOcc

- Room category (“Basic”, “Deluxe”, etc) does not determine maximum occupancy. Maximum occupancy is not determined by room category (“Basic”, “Deluxe”, etc). This is one of our project assumptions. For example, we may have a Basic room for 2 people, or a Basic room for 4 people.

2.2.4. Category ↠ NightlyRate

- Room category (“Basic”, “Deluxe”, etc) does not determine nightly rate. Each room may be given a unique price when the room is added to the system. This is one of our project assumptions. For example, one room might have a spectacular view, and thus have an exorbitant nightly rate.

2.3. 3NF Satisfaction

There is one functional dependency in the Rooms schema.

2.3.1. RoomNum, HotelID → Category, MaxOcc, NightlyRate

- The left hand side of this functional dependency is RoomNum, HotelID, which is a superkey for the schema. Therefore this FD does not violate 3NF (or even BCNF).

3. Staff

3.1. Schema

Staff (ID, Name, DOB, JobTitle, Dep, PhoneNum, Address, HotelID)

Note: “DOB” is date of birth
Note: “Dep” is department

3.2. Functional Dependencies

3.2.1. ID → Name, DOB, JobTitle, Dep, PhoneNum, Address, HotelID

- One item to note here is that staff ID does functionally determine hotel ID, because Each staff member is assigned to at most one hotel. This is one of our project assumptions.

3.2.2. Name ↠ ID

- Several staff members may have the same name. This is one of our project assumptions.

3.2.3. PhoneNum ↠ ID

- Several staff members may have the same phone number. This is one of our project assumptions.

3.2.4. Address ↠ ID

- Several staff members may have the same address. This is one of our project assumptions.

3.2.5. JobTitle ↠ Dep, Dep ↠ JobTitle

- A staff member’s Dep is merely an attribute and is not expected to influence how the staff member interacts with the system. This is one of our project assumptions.

3.3. 3NF Satisfaction

There is one functional dependency in the Staff schema.

3.3.1. ID → Name, DOB, JobTitle, Dep, PhoneNum, Address, HotelID

- The left hand side of this functional dependency is ID, which is a superkey for the schema. Therefore this FD does not violate 3NF (or even BCNF).

4. Customers

4.1. Schema

Customers (SSN, Name, DOB, PhoneNum, Email)

Note: “DOB” is date of birth

4.2. Functional Dependencies

4.2.1. SSN → Name, DOB, PhoneNum, Email

4.2.2. Name ↠ SSN

- Several customers may have the same name. This is one of our project assumptions.

4.2.3. PhoneNum ↠ SSN

- Several customers may have the same phone number. This is one of our project assumptions.

4.2.4. Email ↠ SSN

- Several customers may have the same email address. This is one of our project assumptions.

4.3. 3NF Satisfaction

There is one functional dependency in the Customers schema.

4.3.1. ID → Name, DOB, JobTitle, Dep, PhoneNum, Address, HotelID

- The left hand side of this functional dependency is ID, which is a superkey for the schema. Therefore this FD does not violate 3NF (or even BCNF).

5. Stays

5.1. Schema

Stays (ID, StartDate, RoomNum, HotelID, CustomerSSN, NumGuests, CheckInTime, CheckOutTime, EndDate, AmountOwed, PaymentMethod, CardType, CardNumber, BillingAddress)

5.2. Functional Dependencies

5.2.1. ID → StartDate, RoomNum, HotelID, CustomerSSN, NumGuests, CheckInTime, CheckOutTime, EndDate, AmountOwed, PaymentMethod, CardType, CardNumber, BillingAddress

5.2.2. StartDate, CheckInTime, RoomNum, HotelID → ID, CustomerSSN, NumGuests, CheckOutTime, EndDate, AmountOwed, PaymentMethod, CardType, CardNumber, BillingAddress

- This means that StartDate, RoomNum, HotelID is a key, however it is not the primary key for practical concerns, namely that if it were the primary key it would be more cumbersome for tuples in other relations to refer to a unique stay with a foreign key.

5.2.3. CardType, CardNumber ↛ BillingAddress

- The customer may keep the same credit card, but change their billing address, between one stay and the next.

5.2.4. CustomerSSN ↛ CardType, CardNumber, BillingAddress

- The customer may not wish to use the same method of payment for multiple Wolf Inns hotel stays. This is one of our project assumptions.

5.2.5. CardNumber ↛ CardType

- We make no assumption that all credit card companies hold a unique block of card numbers.

5.3. 3NF Satisfaction

There are two functional dependencies in the Stays schema.

5.3.1. ID → StartDate, RoomNum, HotelID, CustomerSSN, NumGuests, CheckInTime, CheckOutTime, EndDate, AmountOwed, PaymentMethod, CardType, CardNumber, BillingAddress

- The left hand side of this functional dependency is ID, which is a superkey for the schema. Therefore this FD does not violate 3NF (or even BCNF).

5.3.2. StartDate, CheckInTime, RoomNum, HotelID → ID, CustomerSSN, NumGuests, CheckOutTime, EndDate, AmountOwed, PaymentMethod, CardType, CardNumber, BillingAddress

- The left hand side of this functional dependency is StartDate,CheckInTime RoomNum, HotelID, which is a superkey for the schema. Therefore this FD does not violate 3NF (or even BCNF).

6. ServiceTypes

6.1. Schema

ServiceTypes (<u>Name</u>, Cost)

6.2. Functional Dependencies

6.2.1. Name → Cost

6.3. 3NF Satisfaction

There is one functional dependency in the ServiceTypes schema.

6.3.1. Name → Cost

- The left hand side of this functional dependency is Name, which is a superkey for the schema. Therefore this FD does not violate 3NF (or even BCNF).

7. Provided

7.1. Schema

Provided (ID, StayID, StaffID, ServiceName)

7.2. Functional Dependencies

7.2.1. ID → StayID, StaffID, ServiceName

7.2.2. StayID ↠ StaffID

- A customer may be served by several staff members during a single stay.

7.2.3. StayID ↠ ServiceName

- A customer may receive several types of service during a single stay.

7.2.4. StaffID ↠ ServiceName

- A staff member's job title indicates what they are expected to do within the system. However, a manager may need to fill in for a colleague, and so for example a manager may provide any type of service regardless of job title. This is one of our project assumptions.

7.3. 3NF Satisfaction

There is one functional dependency in the Provided schema.

7.3.1. ID → StayID, StaffID, ServiceName

- The left hand side of this functional dependency is ID, which is a superkey for the schema. Therefore this FD does not violate 3NF (or even BCNF).

Global Schema Documentation

1. Design Decisions

1.1. Mechanical Approach

Translations from a global view of the local E/R diagrams, to global relation schemas were done mechanically.

1.1.1. Each entity set was translated into a relation schema with the same set of attributes as shown in the entity set.
1.1.2. Each many to one relationship was translated into additional attributes from the “one” side of the relationship, shown in the schema representing the “many” side of the relationship.
1.1.3. Each many to many relationship was translated into a relation schema with attributes representing the keys for the participating entity sets.

1.2. NULLS approach for subclasses

1.2.1. There are 4 entity sets which have an “ISA” relationship to some other entity set.

1.2.1.1. Presidential “ISA” Rooms
1.2.1.2. RoomService “ISA” Staff
1.2.1.3. Catering “ISA” Staff
1.2.1.4. Managers “ISA” Staff

1.2.2. None of the subclass entity sets require a schema of their own for any functional reason. The DB system design team prefers the conceptually simpler approach of minimizing the number of relations, and so has chosen the NULLS approach.

1.2.2.1. Presidential

The relationship between a presidential suite room and the staff which is dedicated to it is handled by the “DRSStaff” and “DCStaff” attributes of Rooms. In this case, the choice of the NULLS approach does increase the presence of NULL values in the database. However, this is considered an acceptable sacrifice in light of the minimization of relations and the conceptual simplicity of the NULLS approach.

1.2.2.2. RoomService

The relationship between a presidential suite room and the staff which is dedicated to it is handled by the “DRSStaff” and “DCStaff” attributes of Rooms. This is discussed above.

1.2.2.3. Catering

The relationship between a presidential suite room and the staff which is dedicated to it is handled by the “DRSStaff” and “DCStaff” attributes of Rooms. This is discussed above.

1.2.2.4. Managers

The relationship between a manager and the hotel which they manage is handled by the “HotelID” attribute of Staff, and by the “ManagerID” attribute of Hotels. Because it is already the case all Staff members may be associated with a hotel, using the NULLS approach does not actually increase the presence of NULL values in the database.

2. Integrity Constraints

2.1. Hotels (ID, Name, StreetAddress, City, State, PhoneNum, ManagerID)

Attribute	DataType	NULL	Key	Referential Integrity
ID	INT AUTO_INC	NOT NULL (implicit: primary key)	PRIMARY KEY (unique identifier)	(no)
Name	VARCHAR (255)	NOT NULL (every hotel has a name)	(no)	(no)
StreetAddress	VARCHAR (255)	NOT NULL (every hotel has an address)	UNIQUE KEY (there cannot be multiple hotels at the same physical location) Constraint Name: UC_HACS	(no)
City	VARCHAR (255)	NOT NULL (every hotel has a city)		(no)
State	CHAR (2)	NOT NULL (every hotel has a state)		(no)
PhoneNum	BIGINT	NOT NULL (every hotel has a phone number)	UNIQUE KEY (there cannot be multiple hotels with the same phone number) Constraint Name: UC_HPN	(no)
ManagerID	INT	NOT NULL (A hotel has exactly one manager)	UNIQUE KEY (a manager manages exactly one hotel) Constraint Name: UC_HMID	REFERENCES Staff.ID Constraint Name: FK_HMID

2.2. Rooms (RoomNum, HotelID, Category, MaxOcc, NightlyRate, DRSStaff, DCStaff)

Attribute	DataType	NULL	Key	Referential Integrity
RoomNum	INT	NOT NULL (implicit: in primary key)	PRIMARY KEY (unique identifier)	(no)
HotelID	INT	NOT NULL (implicit: in primary key)		REFERENCES Hotels.ID Constraint Name: FK_ROOMHID
Category	VARCHAR (255)	NOT NULL (every room has a category)	(no)	(no)
MaxOcc	INT	NOT NULL (every room has a maximum occupancy)	(no)	(no)
NightlyRate	DOUBLE	NOT NULL (every room has a nightly rate)	(no)	(no)
DRSStaff	INT	NULL allowed (there is no dedicated staff member for this room - either because it is not a presidential suite or because it is not occupied)	(no)	REFERENCES Staff.ID Constraint Name: FK_ROOMDRSID
DCStaff	INT	NULL allowed (there is no dedicated staff member for this room - either because it is not a presidential suite or because it is not occupied)	(no)	REFERENCES Staff.ID Constraint Name: FK_ROOMDCID

2.3. Staff (ID, Name, DOB, JobTitle, Dep, PhoneNum, Address, HotelID)

Attribute	DataType	NULL	Key	Referential Integrity
ID	INT AUTO_INC	NOT NULL (implicit: in primary key)	PRIMARY KEY (unique identifier)	(no)
Name	VARCHAR (255)	NOT NULL (every staff member has a name)	(no)	(no)
DOB	DATE	NOT NULL (every staff member has a DOB)	(no)	(no)
JobTitle	VARCHAR (255)	NOT NULL (every staff member has a job title)	(no)	(no)
Dep	VARCHAR (255)	NOT NULL (every staff member has a Dep)	(no)	(no)
PhoneNum	BIGINT	NOT NULL (every staff member has a phone number)	(no)	(no)
Address	VARCHAR (255)	NOT NULL (every staff member has an address)	(no)	(no)
HotelID	INT	NULL is allowed (staff member is not currently assigned to any particular hotel)	(no)	REFERENCES Hotels.ID Constraint Name: FK_STAFFHID

2.4. Customers (SSN, Name, DOB, PhoneNum, Email)

Attribute	DataType	NULL	Key	Referential Integrity
SSN	BIGINT	NOT NULL (implicit: in primary key)	PRIMARY KEY (unique identifier)	(no)
Name	VARCHAR (255)	NOT NULL (every customer has a name)	(no)	(no)
DOB	DATE	NOT NULL (every customer has a DOB)	(no)	(no)
PhoneNum	BIGINT	NOT NULL (every customer has a phone number)	(no)	(no)
Email	VARCHAR (255)	NOT NULL (every customer has an email address)	(no)	(no)

2.5. Stays (ID, StartDate, RoomNum, HotelID, CustomerSSN, NumGuests, CheckInTime, CheckOutTime, EndDate, AmountOwed, PaymentMethod, CardType, CardNumber, BillingAddress)

Attribute	DataType	NULL	Key	Referential Integrity
ID	INT AUTO_INC	NOT NULL (implicit: in primary key)	PRIMARY KEY (unique identifier)	(no)
StartDate	DATE	NOT NULL (every stay has start date at check-in)	UNIQUE KEY (there cannot be multiple stays in the same room of the same hotel starting on the same date and check in time) Constraint Name: UC_STAYKEY	(no)
CheckInTime	TIME	NOT NULL (every stay has a check-in time at check-in)		(no)
RoomNum	INT	NOT NULL (every stay is in a particular room)		REFERENCES Rooms.RoomNum, Rooms.HotelID Constraint Name: K_STAYRID
HotelID	INT	NOT NULL (every stay is in a particular hotel)		
CustomerSSN	BIGINT	NOT NULL (every stay is associated with a customer)	(no)	REFERENCES Customers.SSN Constraint Name: FK_STAYCSSN
NumGuests	INT	NOT NULL (every stay is associated with a number of guests at check-in)	(no)	(no)
CheckOutTime	TIME	NULL is allowed (stay has not yet ended)	(no)	(no)
EndDate	DATE	NULL is allowed (stay has not yet ended)	(no)	(no)
AmountOwed	DOUBLE	NULL is allowed (stay has not yet ended)	(no)	(no)
PaymentMethod	ENUM (‘CASH’, ’CARD’)	NOT NULL (payment methods is established at check-in)	(no)	(no)
CardType	ENUM (‘VISA’, ’MASTERCARD’, ’HOTEL’)	NULL is allowed (payment method is not card)	(no)	(no)
CardNumber	BIGINT	NULL is allowed (payment method is not card)	(no)	(no)
BillingAddress	VARCHAR (255)	NULL is allowed (payment method is not card)	(no)	(no)

2.6. ServiceTypes (Name, Cost)

Attribute	DataType	NULL	Key	Referential Integrity
Name	ENUM (‘Phone’, ‘Dry Cleaning’, ‘Gym’, ‘Room Service’, ‘Catering’, ‘Special Request’)	NOT NULL (implicit: in primary key)	PRIMARY KEY (unique identifier)	(no)
Cost	INT	NOT NULL (every service type has a cost)	(no)	(no)

2.7. Provided (ID, StayID, StaffID, ServiceName)

Attribute	DataType	NULL	Key	Referential Integrity
ID	INT AUTO_INC	NOT NULL (implicit: in primary key)	PRIMARY KEY (unique identifier)	(no)
StayID	INT	NOT NULL (every instance of a provided service is provided for a customer stay)	(no)	REFERENCES Stays.ID Constraint Name: FK_PROVSTAYID
StaffID	INT	NULL is allowed (staff member associated with the service provided has been deleted from the system)	(no)	REFERENCES Staff.ID Constraint Name: FK_PROVSTAFFID
ServiceName	ENUM (‘Phone’, ‘Dry Cleaning’, ‘Gym’, ‘Room Service’, ‘Catering’, ‘Special Request’)	NOT NULL (every instance of a provided stay is of a certain type)	(no)	REFERENCES ServiceTypes.Name Constraint Name: FK_PROVSERV

Populated Relations

Note: MariaDB commands given using MariaDB accounts of various team members due to shared work. We have verified that these commands work when given in the proper sequence all on the same MariaDB account.

1. Customers

1.1. CREATE TABLE statement

```
MariaDB [attiffan]> CREATE TABLE Customers (
    -> SSN BIGINT NOT NULL,
    -> Name VARCHAR(255) NOT NULL,
    -> DOB DATE NOT NULL,
    -> PhoneNum BIGINT NOT NULL,
    -> Email VARCHAR(255) NOT NULL,
    -> PRIMARY KEY (SSN)
    -> );
Query OK, 0 rows affected (0.01 sec)
```

1.2. SELECT * query and response

```
MariaDB [smscoggi]> SELECT * FROM Customers;
+-----------+-----------------+------------+------------+---------------------------+
| SSN       | Name            | DOB        | PhoneNum   | Email                     |
+-----------+-----------------+------------+------------+---------------------------+
| 111038548 | Jay Sharp       | 1956-07-09 | 9191237548 | jay.sharp@gmail.com       |
| 222075875 | Jenson Lee      | 1968-09-25 | 9194563217 | jenson.lee@gmail.com      |
| 333127845 | Benjamin Cooke  | 1964-01-07 | 9191256324 | benjamin.cooke@gmail.com  |
| 444167216 | Joe Bradley     | 1954-04-07 | 9194587569 | joe.bradley@gmail.com     |
| 555284568 | Isaac Gray      | 1982-11-12 | 9194562158 | issac.gray@gmail.com      |
| 666034568 | Conor Stone     | 1975-06-04 | 9194567216 | conor.stone@gmail.com     |
| 777021654 | Elizabeth Davis | 1964-07-26 | 9195432187 | elizabeth.davis@gmail.com |
| 888090545 | Gary Vee        | 1996-03-10 | 9199237455 | gary.vee@gmail.com        |
| 888091545 | Natasha Moore   | 1966-08-14 | 9194562347 | natasha.moore@gmail.com   |
| 888092545 | Gary Vee        | 1996-03-10 | 9199237455 | gary.vee@gmail.com        |
+-----------+-----------------+------------+------------+---------------------------+
10 rows in set (0.00 sec)
```

2. ServiceTypes

2.1. CREATE TABLE statement

```
MariaDB [smscoggi]> CREATE TABLE ServiceTypes (
    -> Name ENUM('Phone','Dry Cleaning','Gym','Room Service','Catering','Special Request') NOT NULL,
    -> Cost INT NOT NULL,
    -> PRIMARY KEY (Name)
    -> );
Query OK, 0 rows affected (0.00 sec)
```

2.2. SELECT * query and response

```
MariaDB [smscoggi]> SELECT * FROM ServiceTypes;
+-----------------+------+
| Name            | Cost |
+-----------------+------+
| Phone           |   25 |
| Dry Cleaning    |   20 |
| Gym             |   35 |
| Room Service    |   25 |
| Catering        |   50 |
| Special Request |   40 |
+-----------------+------+
6 rows in set (0.00 sec)
```

3. Staff

3.1. CREATE TABLE statement

```
MariaDB [smscoggi]> CREATE TABLE Staff (
    -> ID INT NOT NULL AUTO_INCREMENT,
    -> Name VARCHAR(255) NOT NULL,
    -> DOB DATE NOT NULL,
    -> JobTitle VARCHAR(255),
    -> Dep VARCHAR(255) NOT NULL,
    -> PhoneNum BIGINT NOT NULL,
    -> Address VARCHAR(255) NOT NULL,
    -> HotelID INT,
    -> PRIMARY KEY(ID)
    -> );
Query OK, 0 rows affected (0.01 sec)
```

Note:

- After the Hotels table is created, the Staff table must then be altered to add a foreign key constraint which references the Hotels table.
- The reason we don't simply wait to create the Staff table until after the Hotels table exists, is that the Hotels table also has a foreign key reference back to the Staff table.
- This circular reference exists to support the project assumptions "Each staff member is assigned to at most one hotel", and "A hotel has exactly one manager, and a manager manages exactly one hotel".
- We could enforce the latter assumption by ensuring that at all times, every hotel has exactly one staff member tuple which has the job title "Manager" and which points to the hotel via "HotelID". However, MariaDB does not support assertions, so this enforcement would need to come in at the application layer.
- We considered it more direct and safe to enforce the latter assumption by having an explicit attribute in the Hotels relation to indicate the hotel's manager.

```
MariaDB [attiffan]> ALTER TABLE Staff
    -> ADD CONSTRAINT FK_STAFFHID
    -> FOREIGN KEY (HotelID) REFERENCES Hotels(ID) ON DELETE SET NULL;
Query OK, 0 rows affected (0.02 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

Note:

- The population of the Staff table takes place after the sequence: create-staff, create-hotel, alter-staff.
- After the Hotels table is populated, the tuples within the Staff table must then be updated to reference the hotel to which each staff member is assigned. This then is another sequence: populate-staff, populate-hotel, update-staff.
- The SELECT * query and response below reflects a state where the full sequence: create-staff, create-hotel, alter-staff, populate-staff, populate-hotel, update-staff has already taken place.

3.2. SELECT * query and response

Note: We have added many more than the requested “4-8” staff members, in order to more thoroughly test interactions between relations.

```
MariaDB [smscoggi]> SELECT * FROM Staff;
+----+------------------+------------+---------------------------+-----+------------+-----------------------------------------------------+---------+
| ID | Name             | DOB        | JobTitle                  | Dep | PhoneNum   | Address                                             | HotelID |
+----+------------------+------------+---------------------------+-----+------------+-----------------------------------------------------+---------+
|  1 | Zoe Holmes       | 1980-10-02 | Manager                   | A   | 8141113134 | 123 6th St. Melbourne, FL 32904                     |       1 |
|  2 | Katelyn Weeks    | 1970-04-20 | Front Desk Representative | B   | 6926641058 | 123 6th St. Melbourne, FL 32904                     |       1 |
|  3 | Abby Huffman     | 1990-12-14 | Room Service              | C   | 6738742135 | 71 Pilgrim Avenue Chevy Chase, MD 20815             |       1 |
|  4 | Oliver Gibson    | 1985-05-12 | Room Service              | A   | 1515218329 | 70 Bowman St. South Windsor, CT 06074               |       1 |
|  5 | Michael Day      | 1983-02-25 | Catering                  | B   | 3294931245 | 4 Goldfield Rd. Honolulu, HI 96815                  |       1 |
|  6 | David Adams      | 1985-01-17 | Dry Cleaning              | C   | 9194153214 | 44 Shirley Ave. West Chicago, IL 60185              |       1 |
|  7 | Ishaan Goodman   | 1993-04-19 | Gym                       | A   | 5203201425 | 514 S. Magnolia St. Orlando, FL 32806               |       1 |
|  8 | Nicholas Read    | 1981-01-14 | Catering                  | B   | 2564132017 | 236 Pumpkin Hill Court Leesburg, VA 20175           |       1 |
|  9 | Dominic Mitchell | 1971-03-13 | Manager                   | A   | 2922497845 | 7005 South Franklin St. Somerset, NJ 08873          |       2 |
| 10 | Oliver Lucas     | 1961-05-11 | Front Desk Representative | A   | 2519881245 | 7 Edgefield St. Augusta, GA 30906                   |       2 |
| 11 | Molly Thomas     | 1987-07-10 | Room Service              | B   | 5425871245 | 541 S. Holly Street Norcross, GA 30092              |       2 |
| 12 | Caitlin Cole     | 1989-08-15 | Catering                  | B   | 4997845612 | 7 Ivy Ave. Traverse City, MI 49684                  |       2 |
| 13 | Victoria Medina  | 1989-02-04 | Dry Cleaning              | C   | 1341702154 | 8221 Trenton St. Jamestown, NY 14701                |       2 |
| 14 | Will Rollins     | 1982-07-06 | Gym                       | C   | 7071264587 | 346 Beacon Lane Quakertown, PA 18951                |       2 |
| 15 | Masen Shepard    | 1983-01-09 | Manager                   | A   | 8995412364 | 3 Fulton Ave. Bountiful, UT 84010                   |       3 |
| 16 | Willow Roberts   | 1987-02-08 | Front Desk Representative | A   | 5535531245 | 7868 N. Lees Creek Street Chandler, AZ 85224        |       3 |
| 17 | Maddison Davies  | 1981-03-07 | Room Service              | A   | 6784561245 | 61 New Road Ithaca, NY 14850                        |       3 |
| 18 | Crystal Barr     | 1989-04-06 | Catering                  | B   | 4591247845 | 9094 6th Ave. Macomb, MI 48042                      |       3 |
| 19 | Dayana Tyson     | 1980-05-05 | Dry Cleaning              | B   | 4072134587 | 837 W. 10th St. Jonesboro, GA 30236                 |       3 |
| 20 | Tommy Perry      | 1979-06-04 | Gym                       | B   | 5774812456 | 785 Bohemia Street Jupiter, FL 33458                |       3 |
| 21 | Joshua Burke     | 1972-01-10 | Manager                   | C   | 1245214521 | 8947 Briarwood St. Baldwin, NY 11510                |       4 |
| 22 | Bobby Matthews   | 1982-02-14 | Front Desk Representative | C   | 5771812456 | 25 W. Dogwood Lane Bemidji, MN 56601                |       4 |
| 23 | Pedro Cohen      | 1983-04-24 | Room Service              | C   | 8774812456 | 9708 Brickyard Ave. Elyria, OH 44035                |       4 |
| 24 | Alessandro Beck  | 1981-06-12 | Catering                  | A   | 5774812452 | 682 Glen Ridge St. Leesburg, VA 20175               |       4 |
| 25 | Emily Petty      | 1984-08-19 | Dry Cleaning              | A   | 5772812456 | 7604 Courtland St. Easley, SC 29640                 |       4 |
| 26 | Rudy Cole        | 1972-01-09 | Gym                       | A   | 5774812856 | 37 Marconi Drive Owensboro, KY 42301                |       4 |
| 27 | Blair Ball       | 1981-01-10 | Manager                   | A   | 8854124568 | 551 New Saddle Ave. Cape Coral, FL 33904            |       5 |
| 28 | Billy Lopez      | 1982-05-11 | Front Desk Representative | B   | 5124562123 | 99 Miles Road Danbury, CT 06810                     |       5 |
| 29 | Lee Ward         | 1983-06-12 | Room Service              | B   | 9209124562 | 959 S. Tailwater St. Ridgewood, NJ 07450            |       5 |
| 30 | Ryan Parker      | 1972-08-13 | Catering                  | B   | 1183024152 | 157 State Dr. Attleboro, MA 02703                   |       5 |
```

31	Glen Elliott	1971-09-14	Catering	B	6502134785	9775 Clinton Dr. Thornton, CO 80241	5
32	Ash Harrison	1977-02-15	Dry Cleaning	C	9192451365	9924 Jefferson Ave. Plainfield, NJ 07060	5
33	Leslie Little	1979-12-16	Gym	C	9192014512	7371 Pin Oak St. Dalton, GA 30721	5
34	Mason West	1970-10-17	Gym	C	6501231245	798 W. Valley Farms Lane Saint Petersburg, FL 33702	5
35	Riley Dawson	1975-01-09	Manager	C	1183021245	898 Ocean Court Hilliard, OH 43026	6
36	Gabe Howard	1987-03-01	Front Desk Representative	A	6501421523	914 Edgefield Dr. Hartselle, AL 35640	6
37	Jessie Nielsen	1982-06-02	Room Service	A	7574124587	7973 Edgewood Road Gallatin, TN 37066	6
38	Gabe Carlson	1983-08-03	Room Service	A	5771245865	339 Pine Lane Tampa, FL 33604	6
39	Carmen Lee	1976-01-04	Catering	A	9885234562	120 Longbranch Drive Port Richey, FL 34668	6
40	Mell Tran	1979-06-05	Dry Cleaning	A	9162451245	32 Pearl St. Peoria, IL 61604	6
41	Leslie Cook	1970-10-08	Gym	B	6501245126	59 W. High Ridge Street Iowa City, IA 52240	6
42	Rory Burke	1971-01-05	Manager	B	7702653764	9273 Ridge Drive Winter Springs, FL 32708	7
43	Macy Fuller	1972-02-07	Front Desk Representative	B	7485612345	676 Myers Street Baldwin, NY 11510	7
44	Megan Lloyd	1973-03-01	Room Service	B	7221452315	849 George Lane Park Ridge, IL 60068	7
45	Grace Francis	1974-04-09	Catering	B	3425612345	282 Old York Court Mechanicsburg, PA 17050	7
46	Macy Fuller	1975-05-02	Dry Cleaning	C	4665127845	57 Shadow Brook St. Hudson, NH 03051	7
47	Cory Hoover	1976-06-12	Gym	C	9252210735	892 Roosevelt Street Ithaca, NY 14850	7
48	Sam Graham	1977-07-25	Gym	C	7226251245	262 Bayberry St. Dorchester, MA 02125	7
49	Charlie Adams	1981-01-01	Manager	C	6084254152	9716 Glen Creek Dr. Newark, NJ 07103	8
50	Kiran West	1985-02-02	Front Desk Representative	C	9623154125	68 Smith Dr. Lexington, NC 27292	8
51	Franky John	1986-03-03	Room Service	A	8748544152	6 Shirley Road Fairborn, OH 45324	8
52	Charlie Bell	1985-04-04	Room Service	A	9845124562	66 Elm Street Jupiter, FL 33458	8
53	Jamie Young	1986-06-05	Catering	A	9892145214	8111 Birch Hill Avenue Ravenna, OH 44266	8
54	Jackie Miller	1978-08-06	Dry Cleaning	A	9795486234	9895 Redwood Court Glenview, IL 60025	8
55	Jude Cole	1979-03-07	Gym	A	9195642251	8512 Cambridge Ave. Lake In The Hills, IL 60156	8

55 rows in set (0.00 sec)

4. Hotels

4.1. CREATE TABLE statement

```
MariaDB [smscoggi]> CREATE TABLE Hotels (
    -> ID INT NOT NULL AUTO_INCREMENT,
    -> Name VARCHAR(255) NOT NULL,
    -> StreetAddress VARCHAR(255) NOT NULL,
    -> City VARCHAR(255) NOT NULL,
    -> State CHAR(2) NOT NULL,
    -> PhoneNum BIGINT Not Null,
    -> ManagerID INT Not Null,
    -> Primary Key(ID),
    -> CONSTRAINT UC_HACS UNIQUE (StreetAddress, City, State),
    -> CONSTRAINT UC_HPN UNIQUE (PhoneNum),
    -> CONSTRAINT UC_HMID UNIQUE (ManagerID),
    -> CONSTRAINT FK_HMID FOREIGN KEY (ManagerID) REFERENCES Staff(ID) ON DELETE CASCADE
    -> );
Query OK, 0 rows affected (0.00 sec)
```

4.2. SELECT * query and response

```
MariaDB [smscoggi]> SELECT * FROM Hotels;
+----+-----------------------+-----------------------+----------+-------+------------+-----------+
| ID | Name                  | StreetAddress         | City     | State | PhoneNum   | ManagerID |
+----+-----------------------+-----------------------+----------+-------+------------+-----------+
|  1 | The Plaza             | 768 5th Ave           | New York | NY    | 9194152368 |         1 |
|  2 | DoubleTree            | 4810 Page Creek Ln    | Raleigh  | NC    | 9192012364 |         9 |
|  3 | Ramada                | 1520 Blue Ridge Rd    | Raleigh  | NC    | 9190174632 |        15 |
|  4 | Embassy Suites        | 201 Harrison Oaks Blvd | Raleigh  | NC    | 6502137942 |        21 |
|  5 | Four Seasons          | 57 E 57th St          | New York | NY    | 6501236874 |        27 |
|  6 | The Pierre            | 2 E 61st St           | New York | NY    | 6501836874 |        35 |
|  7 | Fairfield Inn & Suites | 0040 Sellona St       | Raleigh  | NC    | 6501236074 |        42 |
|  8 | Mandarin Oriental     | 80 Columbus Cir       | New York | NY    | 6591236874 |        49 |
+----+-----------------------+-----------------------+----------+-------+------------+-----------+
8 rows in set (0.00 sec)
```

5. Rooms

5.1. CREATE TABLE statement

```
MariaDB [smscoggi]> CREATE TABLE Rooms (
    -> RoomNum INT NOT NULL,
    -> HotelID INT NOT NULL,
    -> Category VARCHAR(255) NOT NULL,
    -> MaxOcc INT NOT NULL,
    -> NightlyRate DOUBLE NOT NULL,
    -> DRSStaff INT,
    -> DCStaff INT,
    -> PRIMARY KEY(RoomNum,HotelID),
    -> CONSTRAINT FK_ROOMHID FOREIGN KEY (HotelID) REFERENCES Hotels(ID) ON DELETE CASCADE,
    -> CONSTRAINT FK_ROOMDRSID FOREIGN KEY (DRSStaff) REFERENCES Staff(ID) ON DELETE SET NULL,
    -> CONSTRAINT FK_ROOMDCID FOREIGN KEY (DCStaff) REFERENCES Staff(ID) ON DELETE SET NULL
    -> );
Query OK, 0 rows affected (0.01 sec)
```

5.2. SELECT * query and response

```
MariaDB [smscoggi]> SELECT * FROM Rooms;
+---------+---------+--------------------+--------+-------------+----------+---------+
| RoomNum | HotelID | Category           | MaxOcc | NightlyRate | DRSStaff | DCStaff |
+---------+---------+--------------------+--------+-------------+----------+---------+
|       1 |       1 | ECONOMY            |      3 |         150 |     NULL |    NULL |
|       1 |       2 | DELUXE             |      3 |         200 |     NULL |    NULL |
|       1 |       3 | PRESIDENTIAL_SUITE |      3 |         550 |     NULL |    NULL |
|       1 |       4 | ECONOMY            |      4 |         100 |     NULL |    NULL |
|       1 |       5 | DELUXE             |      3 |         300 |     NULL |    NULL |
|       1 |       6 | ECONOMY            |      2 |         220 |     NULL |    NULL |
|       1 |       7 | ECONOMY            |      2 |         125 |     NULL |    NULL |
|       1 |       8 | ECONOMY            |      2 |         200 |     NULL |    NULL |
|       2 |       1 | PRESIDENTIAL_SUITE |      4 |         450 |     NULL |    NULL |
|       2 |       2 | ECONOMY            |      3 |         125 |     NULL |    NULL |
|       2 |       3 | ECONOMY            |      2 |         350 |     NULL |    NULL |
|       2 |       4 | EXECUTIVE_SUITE    |      4 |         250 |     NULL |    NULL |
|       2 |       5 | EXECUTIVE_SUITE    |      4 |         400 |     NULL |    NULL |
|       2 |       6 | DELUXE             |      4 |         350 |     NULL |    NULL |
|       2 |       7 | EXECUTIVE_SUITE    |      4 |         400 |     NULL |    NULL |
|       2 |       8 | DELUXE             |      3 |         250 |     NULL |    NULL |
|       3 |       1 | EXECUTIVE_SUITE    |      4 |         300 |     NULL |    NULL |
|       3 |       2 | EXECUTIVE_SUITE    |      4 |         250 |     NULL |    NULL |
|       3 |       3 | DELUXE             |      3 |         450 |     NULL |    NULL |
|       3 |       5 | PRESIDENTIAL_SUITE |      4 |         500 |     NULL |    NULL |
|       3 |       8 | EXECUTIVE_SUITE    |      3 |         300 |     NULL |    NULL |
|       4 |       8 | PRESIDENTIAL_SUITE |      4 |         450 |       51 |      53 |
+---------+---------+--------------------+--------+-------------+----------+---------+
22 rows in set (0.00 sec)
```

6. Stays

6.1. CREATE TABLE statement

```
MariaDB [smscoggi]> CREATE TABLE Stays (
    -> ID INT NOT NULL AUTO_INCREMENT,
    -> StartDate DATE NOT NULL,
    -> CheckInTime TIME NOT NULL,
    -> RoomNum INT NOT NULL,
    -> HotelID INT NOT NULL,
    -> CustomerSSN BIGINT NOT NULL,
    -> NumGuests INT NOT NULL,
    -> CheckOutTime TIME,
    -> EndDate DATE,
    -> AmountOwed DOUBLE,
    -> PaymentMethod ENUM('CASH','CARD') NOT NULL,
    -> CardType ENUM('VISA','MASTERCARD','HOTEL'),
    -> CardNumber BIGINT,
    -> BillingAddress VARCHAR(255),
    -> PRIMARY KEY(ID),
    -> CONSTRAINT UC_STAYKEY UNIQUE (StartDate, CheckInTime,RoomNum, HotelID),
    -> CONSTRAINT FK_STAYRID FOREIGN KEY (RoomNum, HotelID) REFERENCES Rooms(RoomNum, HotelID) ON DELETE CASCADE,
    -> CONSTRAINT FK_STAYCSSN FOREIGN KEY (CustomerSSN) REFERENCES Customers(SSN) ON DELETE CASCADE
    -> );
Query OK, 0 rows affected (0.02 sec)
```

6.2. SELECT * query and response

```
MariaDB [smscoggi]> SELECT * FROM Stays;
+----+------------+-------------+---------+---------+-------------+-----------+--------------+------------+-----------+---------------+------------+---------------+-----------------------------------------------------+
| ID | StartDate  | CheckInTime | RoomNum | HotelID | CustomerSSN | NumGuests | CheckOutTime | EndDate    | AmountOwed | PaymentMethod | CardType   | CardNumber    | BillingAddress                                      |
+----+------------+-------------+---------+---------+-------------+-----------+--------------+------------+-----------+---------------+------------+---------------+-----------------------------------------------------+
|  1 | 2018-01-12 | 20:10:00    |       1 |       1 |   555284568 |         3 | 10:00:00     | 2018-01-20 |      1320 | CARD          | VISA       | 4400123454126587 | 7178 Kent St. Enterprise, AL 36330                  |
|  2 | 2018-02-15 | 10:20:00    |       3 |       2 |   111038548 |         2 | 08:00:00     | 2018-02-18 |       775 | CASH          | NULL       |          NULL | NULL                                                |
|  3 | 2018-03-01 | 15:00:00    |       1 |       3 |   222075875 |         1 | 13:00:00     | 2018-03-05 |      2109 | CARD          | HOTEL      | 1100214521684512 | 178 Shadow Brook St. West Chicago, IL 60185         |
|  4 | 2018-02-20 | 07:00:00    |       2 |       4 |   333127845 |         4 | 15:00:00     | 2018-02-27 |      1785 | CARD          | MASTERCARD | 4400124565874591 | 802B Studebaker Drive Clinton Township, MI 48035    |
|  5 | 2018-03-05 | 11:00:00    |       3 |       5 |   444167216 |         4 | 08:00:00     | 2018-03-12 |      3520 | CARD          | VISA       | 4400127465892145 | 83 Inverness Court Longwood, FL 32779               |
|  6 | 2018-03-01 | 18:00:00    |       1 |       6 |   666034568 |         1 | 23:00:00     | 2018-03-01 |        25 | CASH          | NULL       |          NULL | NULL                                                |
|  7 | 2018-01-20 | 06:00:00    |       2 |       7 |   777021654 |         3 | NULL         | NULL       |      NULL | CARD          | HOTEL      | 1100214532567845 | 87 Gregory Street Lawndale, CA 90260                |
|  8 | 2018-02-14 | 09:00:00    |       4 |       8 |   888091545 |         2 | NULL         | NULL       |      NULL | CARD          | VISA       | 4400178498564512 | 34 Hall Ave. Cranberry Twp, PA 16066                |
+----+------------+-------------+---------+---------+-------------+-----------+--------------+------------+-----------+---------------+------------+---------------+-----------------------------------------------------+
8 rows in set (0.01 sec)
```

7. Provided

7.1. CREATE TABLE statement

```
MariaDB [smscoggi]> CREATE TABLE Provided (
    -> ID INT NOT NULL AUTO_INCREMENT,
    -> StayID INT NOT NULL,
    -> StaffID INT,
    -> ServiceName ENUM('Phone','Dry Cleaning','Gym','Room Service','Catering','Special Request') NOT NULL,
    -> PRIMARY KEY(ID),
    -> CONSTRAINT FK_PROVSTAYID FOREIGN KEY (StayID) REFERENCES Stays(ID) ON DELETE CASCADE,
    -> CONSTRAINT FK_PROVSTAFFID FOREIGN KEY (StaffID) REFERENCES Staff(ID) ON DELETE SET NULL,
    -> CONSTRAINT FK_PROVSERV FOREIGN KEY (ServiceName) REFERENCES ServiceTypes(Name) ON DELETE CASCADE
    -> );
Query OK, 0 rows affected (0.01 sec)
```

7.2. SELECT * query and response

```
MariaDB [smscoggi]> SELECT * FROM Provided;
+----+--------+---------+--------------+
| ID | StayID | StaffID | ServiceName  |
+----+--------+---------+--------------+
|  1 |      1 |       7 | Gym          |
|  2 |      1 |       7 | Gym          |
|  3 |      1 |       5 | Catering     |
|  4 |      2 |      11 | Room Service |
|  5 |      3 |      19 | Dry Cleaning |
|  6 |      4 |      26 | Gym          |
|  7 |      5 |      32 | Dry Cleaning |
|  8 |      6 |      38 | Room Service |
|  9 |      7 |      48 | Gym          |
| 10 |      8 |      54 | Dry Cleaning |
+----+--------+---------+--------------+
10 rows in set (0.00 sec)
```

Queries / Updates

1. Queries / Updates

Note: Each new query and update is independent of the effect of previously run queries, and are executed on the same static database that exists after creating and populating tables as described in the previous section.

Note: Queries and updates are based on examples. We might say “In this example, we insert a new “ECONOMY” class room to the hotel with ID 8”. The values related to the examples used will be shown in **large bold text**, to help the reader distinguish values that would change based on what the user of the system actually wanted to accomplish using the query or update.

1.1. Information Processing

1.1.1. “*Enter/update/delete basic information about hotels, rooms, staff, and customers.*”

1.1.1.1. Insert a new hotel

Per our project assumptions, a hotel has exactly one manager, and a manager manages exactly one hotel. Therefore, when inserting a new hotel, we must also run an additional update in the same transaction, to promote a staff member to the manager role.

Also per our project assumptions, any staff member currently dedicated to serving a presidential suite may not have their job title changed. Therefore we must check to make sure this is not the case for the staff member who will be promoted to manager. This is enforced by using a SELECT statement to produce values to be inserted. If the WHERE clause evaluates to false then no values will be produced, and thus the insertion will not take place.

Finally, it should be noted that IF the manager of the new hotel was previously managing another hotel, that other hotel will need a new manager, within the same transaction. Because this could cause a chain reaction, it is more appropriately handled in application code than in SQL.

In this example, we insert a new hotel which gets the auto-incremented ID of 9, and we assign the staff member with ID 50 to be the manager of this new hotel.

```
INSERT INTO Hotels (Name, StreetAddress, City, State, PhoneNum, ManagerID)
SELECT "WolfInns Capital", "123 Main St", "Boston", "MA", 9196662555, Staff.ID
FROM Staff
WHERE
    Staff.ID = 50 AND
    Staff.ID NOT IN (SELECT DCStaff FROM Rooms WHERE DCStaff IS NOT NULL) AND
    Staff.ID NOT IN (SELECT DRSStaff FROM Rooms WHERE DRSStaff IS NOT NULL);

UPDATE Staff
SET JobTitle = "Manager", HotelID = 9
WHERE
    ID = 50 AND
    ID = (SELECT ManagerID FROM Hotels WHERE ID = 9);
```

```
MariaDB [smscoggi]> SELECT * FROM Hotels;
+----+-----------------------+-----------------------+----------+-------+------------+-----------+
| ID | Name                  | StreetAddress         | City     | State | PhoneNum   | ManagerID |
+----+-----------------------+-----------------------+----------+-------+------------+-----------+
|  1 | The Plaza             | 768 5th Ave           | New York | NY    | 9194152368 |         1 |
|  2 | DoubleTree            | 4810 Page Creek Ln    | Raleigh  | NC    | 9192012364 |         9 |
|  3 | Ramada                | 1520 Blue Ridge Rd    | Raleigh  | NC    | 9190174632 |        15 |
|  4 | Embassy Suites        | 201 Harrison Oaks Blvd | Raleigh | NC    | 6502137942 |        21 |
|  5 | Four Seasons          | 57 E 57th St          | New York | NY    | 6501236874 |        27 |
|  6 | The Pierre            | 2 E 61st St           | New York | NY    | 6501836874 |        35 |
|  7 | Fairfield Inn & Suites | 0040 Sellona St      | Raleigh  | NC    | 6501236074 |        42 |
|  8 | Mandarin Oriental     | 80 Columbus Cir       | New York | NY    | 6591236874 |        49 |
|  9 | WolfInns Capital      | 123 Main St           | Boston   | MA    | 9196662555 |        50 |
+----+-----------------------+-----------------------+----------+-------+------------+-----------+
9 rows in set (0.00 sec)
```

```
MariaDB [smscoggi]> SELECT * FROM Staff WHERE HotelID = 9;
+----+------------+------------+----------+-----+------------+---------------------------------+---------+
| ID | Name       | DOB        | JobTitle | Dep | PhoneNum   | Address                         | HotelID |
+----+------------+------------+----------+-----+------------+---------------------------------+---------+
| 50 | Kiran West | 1985-02-02 | Manager  | C   | 9623154125 | 68 Smith Dr. Lexington, NC 27292 |       9 |
+----+------------+------------+----------+-----+------------+---------------------------------+---------+
1 row in set (0.02 sec)
```

1.1.1.2. Update basic information about a hotel

Per our project assumption, a hotel has exactly one manager, and a manager manages exactly one hotel

When updating the manager of an existing hotel, we must also run additional updates in the same transaction, beforehand to demote the old manager (we choose to demote managers to the role of Front Desk Representative), and afterwards to promote a staff member to the manager role.

In this example, we update all basic information about the hotel with ID 6.

```
UPDATE Staff
SET JobTitle = "Front Desk Representative"
WHERE JobTitle = "Manager" AND HotelID = 6;
UPDATE Hotels
SET Name = "The Pirates", StreetAddress = "Penguin Island", City = "Nowhere", State = "AK",
PhoneNum = 9198675309, ManagerID = 2
WHERE ID = 6;
UPDATE Staff
SET JobTitle = "Manager", HotelID = 6
WHERE ID = 2;
```

```
MariaDB [smscoggi]> SELECT * FROM Hotels;
+----+-----------------------+-----------------------+----------+-------+------------+-----------+
| ID | Name                  | StreetAddress         | City     | State | PhoneNum   | ManagerID |
+----+-----------------------+-----------------------+----------+-------+------------+-----------+
|  1 | The Plaza             | 768 5th Ave           | New York | NY    | 9194152368 |         1 |
|  2 | DoubleTree            | 4810 Page Creek Ln    | Raleigh  | NC    | 9192012364 |         9 |
|  3 | Ramada                | 1520 Blue Ridge Rd    | Raleigh  | NC    | 9190174632 |        15 |
|  4 | Embassy Suites        | 201 Harrison Oaks Blvd | Raleigh  | NC    | 6502137942 |        21 |
|  5 | Four Seasons          | 57 E 57th St          | New York | NY    | 6501236874 |        27 |
|  6 | The Pirates           | Penguin Island        | Nowhere  | AK    | 9198675309 |         2 |
|  7 | Fairfield Inn & Suites | 0040 Sellona St       | Raleigh  | NC    | 6501236074 |        42 |
|  8 | Mandarin Oriental     | 80 Columbus Cir       | New York | NY    | 6591236874 |        49 |
+----+-----------------------+-----------------------+----------+-------+------------+-----------+
8 rows in set (0.00 sec)
```

```
MariaDB [smscoggi]> SELECT * FROM Staff WHERE HotelID = 6;
+----+---------------+------------+---------------------------+-----+------------+-----------------------------------------------+---------+
| ID | Name          | DOB        | JobTitle                  | Dep | PhoneNum   | Address                                       | HotelID |
+----+---------------+------------+---------------------------+-----+------------+-----------------------------------------------+---------+
|  2 | Katelyn Weeks | 1970-04-20 | Manager                   | B   | 6926641058 | 123 6th St. Melbourne, FL 32904               |       6 |
| 35 | Riley Dawson  | 1975-01-09 | Front Desk Representative | C   | 1183021245 | 898 Ocean Court Hilliard, OH 43026            |       6 |
| 36 | Gabe Howard   | 1987-03-01 | Front Desk Representative | A   | 6501421523 | 914 Edgefield Dr. Hartselle, AL 35640         |       6 |
| 37 | Jessie Nielsen | 1982-06-02 | Room Service              | A   | 7574124587 | 7973 Edgewood Road Gallatin, TN 37066         |       6 |
| 38 | Gabe Carlson  | 1983-08-03 | Room Service              | A   | 5771245865 | 339 Pine Lane Tampa, FL 33604                 |       6 |
| 39 | Carmen Lee    | 1976-01-04 | Catering                  | A   | 9885234562 | 120 Longbranch Drive Port Richey, FL 34668    |       6 |
| 40 | Mell Tran     | 1979-06-05 | Dry Cleaning              | A   | 9162451245 | 32 Pearl St. Peoria, IL 61604                 |       6 |
| 41 | Leslie Cook   | 1970-10-08 | Gym                       | B   | 6501245126 | 59 W. High Ridge Street Iowa City, IA 52240   |       6 |
+----+---------------+------------+---------------------------+-----+------------+-----------------------------------------------+---------+
8 rows in set (0.00 sec)
```

1.1.1.3. Delete a hotel

In this example, we delete the hotel with ID 5. Per our project assumptions, “Any hotel, room, customer, staff member, or service type associated with a current guest stay may not be deleted”. The hotel with ID 5 does not currently have a guest staying in it, so it is successfully deleted.

```
DELETE
FROM Hotels
WHERE ID = 5 AND ID NOT IN (
    SELECT HotelID FROM Stays WHERE CheckOutTime IS NULL OR EndDate IS NULL
);
```

```
MariaDB [smscoggi]> SELECT * FROM Hotels;
+----+-----------------------+-----------------------+----------+-------+------------+------------+
| ID | Name                  | StreetAddress         | City     | State | PhoneNum   | ManagerID |
+----+-----------------------+-----------------------+----------+-------+------------+------------+
|  1 | The Plaza             | 768 5th Ave           | New York | NY    | 9194152368 |          1 |
|  2 | DoubleTree            | 4810 Page Creek Ln    | Raleigh  | NC    | 9192012364 |          9 |
|  3 | Ramada                | 1520 Blue Ridge Rd    | Raleigh  | NC    | 9190174632 |         15 |
|  4 | Embassy Suites        | 201 Harrison Oaks Blvd | Raleigh  | NC    | 6502137942 |         21 |
|  6 | The Pierre            | 2 E 61st St           | New York | NY    | 6501836874 |         35 |
|  7 | Fairfield Inn & Suites | 0040 Sellona St       | Raleigh  | NC    | 6501236074 |         42 |
|  8 | Mandarin Oriental     | 80 Columbus Cir       | New York | NY    | 6591236874 |         49 |
+----+-----------------------+-----------------------+----------+-------+------------+------------+
7 rows in set (0.00 sec)
```

1.1.1.4. Insert a new room

In this example, we insert a new “ECONOMY” class room to the hotel with ID 8.

```
INSERT INTO Rooms(RoomNum, HotelID, Category, MaxOcc, NightlyRate) VALUES(5, 8, "ECONOMY", 3, 400);
```

```
MariaDB [smscoggi]> SELECT * FROM Rooms WHERE HotelID = 8;
+---------+---------+--------------------+--------+-------------+----------+----------+
| RoomNum | HotelID | Category           | MaxOcc | NightlyRate | DRSStaff | DCStaff |
+---------+---------+--------------------+--------+-------------+----------+----------+
|       1 |       8 | ECONOMY            |      2 |         200 |     NULL |     NULL |
|       2 |       8 | DELUXE             |      3 |         250 |     NULL |     NULL |
|       3 |       8 | EXECUTIVE_SUITE    |      3 |         300 |     NULL |     NULL |
|       4 |       8 | PRESIDENTIAL_SUITE |      4 |         450 |       51 |       53 |
|       5 |       8 | ECONOMY            |      3 |         400 |     NULL |     NULL |
+---------+---------+--------------------+--------+-------------+----------+----------+
5 rows in set (0.01 sec)
```

1.1.1.5. Update basic information about a room

In this example, we update all the basic information about room 2 in hotel 8, for example changing the class of room to “ECONOMY”.

As noted in our project assumptions:

- While the Wolf Inns system supports updating most aspects of a record, the primary key of any record can never be changed - this is not supported by the Wolf Inns system. Thus a room’s room number and hotel ID may not be updated.
- Any room associated with a current guest stay may not be updated.

Our example does not violate either of these assumptions, and so the update is applied successfully.

```
UPDATE Rooms
SET Category = "ECONOMY", MaxOcc = 2, NightlyRate = 200
WHERE
    RoomNum = 2 AND
    HotelID = 8 AND
    NOT EXISTS (
        SELECT *
        FROM Stays
        WHERE
            Stays.HotelID = Rooms.HotelID AND
            Stays.RoomNum = Rooms.RoomNum AND
            (CheckOutTime IS NULL OR EndDate IS NULL)
    );
```

```
MariaDB [smscoggi]> SELECT * FROM Rooms WHERE HotelID = 8;
+---------+---------+--------------------+--------+-------------+----------+---------+
| RoomNum | HotelID | Category           | MaxOcc | NightlyRate | DRSStaff | DCStaff |
+---------+---------+--------------------+--------+-------------+----------+---------+
|       1 |       8 | ECONOMY            |      2 |         200 |     NULL |    NULL |
|       2 |       8 | ECONOMY            |      2 |         200 |     NULL |    NULL |
|       3 |       8 | EXECUTIVE_SUITE    |      3 |         300 |     NULL |    NULL |
|       4 |       8 | PRESIDENTIAL_SUITE |      4 |         450 |       51 |      53 |
+---------+---------+--------------------+--------+-------------+----------+---------+
4 rows in set (0.00 sec)
```

1.1.1.6. Delete a room

In this example, we delete room 2 from hotel 5. Per our project assumptions, “Any hotel, room, customer, staff member, or service type associated with a current guest stay may not be deleted”. Room 2 in hotel 5 does not currently have a guest staying in it, so it is deleted successfully.

```
DELETE
FROM Rooms
WHERE Rooms.HotelID = 5 AND Rooms.RoomNum = 2 AND NOT EXISTS (
    SELECT *
    FROM Stays
    WHERE Stays.HotelID = 5 AND Stays.RoomNum = 5 AND (CheckOutTime IS NULL OR EndDate IS NULL)
);
```

```
MariaDB [smscoggi]> SELECT * FROM Rooms WHERE HotelID = 5;
+---------+---------+--------------------+--------+-------------+----------+---------+
| RoomNum | HotelID | Category           | MaxOcc | NightlyRate | DRSStaff | DCStaff |
+---------+---------+--------------------+--------+-------------+----------+---------+
|       1 |       5 | DELUXE             |      3 |         300 |     NULL |    NULL |
|       3 |       5 | PRESIDENTIAL_SUITE |      4 |         500 |     NULL |    NULL |
+---------+---------+--------------------+--------+-------------+----------+---------+
2 rows in set (0.00 sec)
```

1.1.1.7. Insert a new staff member

In this example, we insert a new staff member named “Thomas Atkinson”, a room service staff member, and assign him to no particular hotel - he is available to be assigned to any Wolf Inns hotel in the future.

```
INSERT INTO Staff(Name, DOB, JobTitle, Dep, PhoneNum, Address)
VALUES("Thomas Atkinson", 1990-10-01, "Room Service", "A", "9196492222", "10 New Road
Bombay, IN 10234");
```

```
MariaDB [smscoggi]> SELECT * FROM Staff WHERE HotelID IS NULL;
+----+-----------------+------------+--------------+-----+------------+------------------------------+---------+
| ID | Name            | DOB        | JobTitle     | Dep | PhoneNum   | Address                      | HotelID |
+----+-----------------+------------+--------------+-----+------------+------------------------------+---------+
| 56 | Thomas Atkinson | 0000-00-00 | Room Service | A   | 9196492222 | 10 New Road Bombay, IN 10234 |    NULL |
+----+-----------------+------------+--------------+-----+------------+------------------------------+---------+
1 row in set (0.00 sec)
```

1.1.1.8. Update basic information about a staff member

In this example, we update several pieces of basic information about Rudy Cole, the staff member with ID 26.

As noted in our project assumptions, any staff member currently dedicated to serving a presidential suite may not have their job title changed. The query accounts for this, and the update will simply not occur for any such staff member. In our example, Rudy Cole is not dedicated to serving a presidential suite, and so the update is successful.

```
UPDATE Staff
SET DOB = "1973-01-09", JobTitle = "Front Desk Representative", Dep = "B", PhoneNum =
5774812855, Address = "36 Marconi Drive Owensboro, KY 42301"
WHERE
    ID = 26 AND
    (
        ID NOT IN (SELECT DCStaff FROM Rooms WHERE DCStaff IS NOT NULL) OR
        "Front Desk Representative" = "Catering"
    ) AND
    (
        ID NOT IN (SELECT DRSStaff FROM Rooms WHERE DRSStaff IS NOT NULL) OR
        "Front Desk Representative" = "Room Service"
    );
```

```
MariaDB [smscoggi]> SELECT * FROM Staff WHERE ID = 26;
+----+-----------+------------+--------------------------+-----+------------+--------------------------------------+---------+
| ID | Name      | DOB        | JobTitle                 | Dep | PhoneNum   | Address                              | HotelID |
+----+-----------+------------+--------------------------+-----+------------+--------------------------------------+---------+
| 26 | Rudy Cole | 1973-01-09 | Front Desk Representative | B   | 5774812855 | 36 Marconi Drive Owensboro, KY 42301 |       4 |
+----+-----------+------------+--------------------------+-----+------------+--------------------------------------+---------+
1 row in set (0.00 sec)
```

1.1.1.9. Delete a staff member

In this example, we delete the staff member with ID 10. Per our project assumptions, “Any hotel, room, customer, staff member, or service type associated with a current guest stay may not be deleted”. Therefore this query will only delete the staff member if they are not dedicated to serving a room that has a guest staying in it. Staff member 10, Oliver Lucas, is not dedicated to serving a room that has a guest staying in it, so he is deleted successfully.

Note: In our subqueries, we need to check to ensure that DRSStaff, or DCStaff, is NOT NULL, because ID NOT IN ([*results including NULL*]) evaluates to UNKNOWN rather than to TRUE.

```
DELETE FROM Staff WHERE ID = 10 AND ID NOT IN (
    SELECT Rooms.DRSStaff AS X
    FROM Stays,Rooms
    WHERE Rooms.RoomNum = Stays.RoomNum
        AND Rooms.HotelID = Stays.HotelID
        AND (Stays.CheckOutTime IS NULL OR Stays.EndDate IS NULL)
        AND Rooms.DRSStaff IS NOT NULL
    UNION ALL
    SELECT Rooms.DCStaff AS X
    FROM(Stays,Rooms)
    WHERE Rooms.RoomNum = Stays.RoomNum
        AND Rooms.HotelID = Stays.HotelID
        AND (Stays.CheckOutTime IS NULL OR Stays.EndDate IS NULL)
        AND Rooms.DCStaff IS NOT NULL
);
```

The screenshot of the Staff relation after the update is cut off after staff ID 12, simply for readability.

```
MariaDB [smscoggi]> SELECT * FROM Staff;
+----+----------------+------------+---------------------------+-----+------------+-----------------------------------------------------+---------+
| ID | Name           | DOB        | JobTitle                  | Dep | PhoneNum   | Address                                             | HotelID |
+----+----------------+------------+---------------------------+-----+------------+-----------------------------------------------------+---------+
|  1 | Zoe Holmes     | 1980-10-02 | Manager                   | A   | 8141113134 | 123 6th St. Melbourne, FL 32904                     |       1 |
|  2 | Katelyn Weeks  | 1970-04-20 | Front Desk Representative | B   | 6926641058 | 123 6th St. Melbourne, FL 32904                     |       1 |
|  3 | Abby Huffman   | 1990-12-14 | Room Service              | C   | 6738742135 | 71 Pilgrim Avenue Chevy Chase, MD 20815             |       1 |
|  4 | Oliver Gibson  | 1985-05-12 | Room Service              | A   | 1515218329 | 70 Bowman St. South Windsor, CT 06074               |       1 |
|  5 | Michael Day    | 1983-02-25 | Catering                  | B   | 3294931245 | 4 Goldfield Rd. Honolulu, HI 96815                  |       1 |
|  6 | David Adams    | 1985-01-17 | Dry Cleaning              | C   | 9194153214 | 44 Shirley Ave. West Chicago, IL 60185              |       1 |
|  7 | Ishaan Goodman | 1993-04-19 | Gym                       | A   | 5203201425 | 514 S. Magnolia St. Orlando, FL 32806               |       1 |
|  8 | Nicholas Read  | 1981-01-14 | Catering                  | B   | 2564132017 | 236 Pumpkin Hill Court Leesburg, VA 20175           |       1 |
|  9 | Dominic Mitchell | 1971-03-13 | Manager                 | A   | 2922497845 | 7005 South Franklin St. Somerset, NJ 08873          |       2 |
| 11 | Molly Thomas   | 1987-07-10 | Room Service              | B   | 5425871245 | 541 S. Holly Street Norcross, GA 30092              |       2 |
| 12 | Caitlin Cole   | 1989-08-15 | Catering                  | B   | 4997845612 | 7 Ivy Ave. Traverse City, MI 49684                  |       2 |
```

1.1.1.10. Insert a new customer

In this example, we insert a new customer named “Robert Skaggs”.

```
INSERT INTO Customers (SSN, Name, DOB, PhoneNum, Email)
VALUES (123456789, "Robert Skaggs", "1980-01-01", 1234567890, "robert.skaggs@gmail.com");
```

```
MariaDB [smscoggi]> SELECT * FROM Customers;
+-----------+-----------------+------------+------------+---------------------------+
| SSN       | Name            | DOB        | PhoneNum   | Email                     |
+-----------+-----------------+------------+------------+---------------------------+
| 111038548 | Jay Sharp       | 1956-07-09 | 9191237548 | jay.sharp@gmail.com       |
| 123456789 | Robert Skaggs   | 1980-01-01 | 1234567890 | robert.skaggs@gmail.com   |
| 222075875 | Jenson Lee      | 1968-09-25 | 9194563217 | jenson.lee@gmail.com      |
| 333127845 | Benjamin Cooke  | 1964-01-07 | 9191256324 | benjamin.cooke@gmail.com  |
| 444167216 | Joe Bradley     | 1954-04-07 | 9194587569 | joe.bradley@gmail.com     |
| 555284568 | Isaac Gray      | 1982-11-12 | 9194562158 | issac.gray@gmail.com      |
| 666034568 | Conor Stone     | 1975-06-04 | 9194567216 | conor.stone@gmail.com     |
| 777021654 | Elizabeth Davis | 1964-07-26 | 9195432187 | elizabeth.davis@gmail.com |
| 888090545 | Gary Vee        | 1996-03-10 | 9199237455 | gary.vee@gmail.com        |
| 888091545 | Natasha Moore   | 1966-08-14 | 9194562347 | natasha.moore@gmail.com   |
| 888092545 | Gary Vee        | 1996-03-10 | 9199237455 | gary.vee@gmail.com        |
+-----------+-----------------+------------+------------+---------------------------+
11 rows in set (0.01 sec)
```

1.1.1.11. Update basic information about a customer

In this example, we update all the basic information that can be updated, about the customer Natasha Moore (SSN 888091545). As noted in our project assumptions, “While the Wolf Inns system supports updating most aspects of a record, the primary key of any record can never be changed - this is not supported by the Wolf Inns system.” Thus a customer’s SSN may not be updated.

```
UPDATE Customers
SET Name = "Natasha Moore-Jones", DOB = "1966-08-13", PhoneNum = 8285559090, email =
"natasha.moore.jones@gmail.com"
WHERE SSN = 888091545;
```

```
MariaDB [smscoggi]> SELECT * FROM Customers;
+-----------+---------------------+------------+------------+-------------------------------+
| SSN       | Name                | DOB        | PhoneNum   | Email                         |
+-----------+---------------------+------------+------------+-------------------------------+
| 111038548 | Jay Sharp           | 1956-07-09 | 9191237548 | jay.sharp@gmail.com           |
| 222075875 | Jenson Lee          | 1968-09-25 | 9194563217 | jenson.lee@gmail.com          |
| 333127845 | Benjamin Cooke      | 1964-01-07 | 9191256324 | benjamin.cooke@gmail.com      |
| 444167216 | Joe Bradley         | 1954-04-07 | 9194587569 | joe.bradley@gmail.com         |
| 555284568 | Isaac Gray          | 1982-11-12 | 9194562158 | issac.gray@gmail.com          |
| 666034568 | Conor Stone         | 1975-06-04 | 9194567216 | conor.stone@gmail.com         |
| 777021654 | Elizabeth Davis     | 1964-07-26 | 9195432187 | elizabeth.davis@gmail.com     |
| 888090545 | Gary Vee            | 1996-03-10 | 9199237455 | gary.vee@gmail.com            |
| 888091545 | Natasha Moore-Jones | 1966-08-13 | 8285559090 | natasha.moore.jones@gmail.com |
| 888092545 | Gary Vee            | 1996-03-10 | 9199237455 | gary.vee@gmail.com            |
+-----------+---------------------+------------+------------+-------------------------------+
10 rows in set (0.01 sec)
```

1.1.1.12. Delete a customer

In this example, we delete Joe Bradley (SSN 444167216) from the Wolf Inns database.

Per our project assumptions:

- Any hotel, room, customer, staff member, or service type associated with a current guest stay may not be deleted.
- If a customer is removed from the Wolf Inns system, then all information associated with this customer is removed from the system, entirely. For example, all stays associated with that customer are also removed.

In our example, Joe Bradley is not associated with a current stay, and so he is deleted successfully. There is a past stay associated with Joe Bradley, and this stay is removed from the system as expected.

```
DELETE FROM Customers WHERE
    SSN = 444167216 AND
    SSN NOT IN (
        SELECT CustomerSSN AS SSN FROM Stays WHERE
            CheckOutTime IS NULL OR
            EndDate IS NULL
    );
```

```
MariaDB [smscoggi]> SELECT * FROM Customers;
+-----------+----------------+------------+------------+--------------------------+
| SSN       | Name           | DOB        | PhoneNum   | Email                    |
+-----------+----------------+------------+------------+--------------------------+
| 111038548 | Jay Sharp      | 1956-07-09 | 9191237548 | jay.sharp@gmail.com      |
| 222075875 | Jenson Lee     | 1968-09-25 | 9194563217 | jenson.lee@gmail.com     |
| 333127845 | Benjamin Cooke | 1964-01-07 | 9191256324 | benjamin.cooke@gmail.com |
| 555284568 | Isaac Gray     | 1982-11-12 | 9194562158 | issac.gray@gmail.com     |
| 666034568 | Conor Stone    | 1975-06-04 | 9194567216 | conor.stone@gmail.com    |
| 777021654 | Elizabeth Davis| 1964-07-26 | 9195432187 | elizabeth.davis@gmail.com|
| 888090545 | Gary Vee       | 1996-03-10 | 9199237455 | gary.vee@gmail.com       |
| 888091545 | Natasha Moore  | 1966-08-14 | 9194562347 | natasha.moore@gmail.com  |
| 888092545 | Gary Vee       | 1996-03-10 | 9199237455 | gary.vee@gmail.com       |
+-----------+----------------+------------+------------+--------------------------+
9 rows in set (0.00 sec)
```

```
MariaDB [smscoggi]> SELECT * FROM Stays;
+----+------------+-------------+---------+---------+-------------+-----------+--------------+------------+------------+---------------+------------+------------------+---------------------------------------------------+
| ID | StartDate  | CheckInTime | RoomNum | HotelID | CustomerSSN | NumGuests | CheckOutTime | EndDate    | AmountOwed | PaymentMethod | CardType   | CardNumber       | BillingAddress                                    |
+----+------------+-------------+---------+---------+-------------+-----------+--------------+------------+------------+---------------+------------+------------------+---------------------------------------------------+
|  1 | 2018-01-12 | 20:10:00    |       1 |       1 |   555284568 |         3 | 10:00:00     | 2018-01-20 |       1320 | CARD          | VISA       | 4400123454126587 | 7178 Kent St. Enterprise, AL 36330                |
|  2 | 2018-02-15 | 10:20:00    |       3 |       2 |   111038548 |         2 | 08:00:00     | 2018-02-18 |        775 | CASH          | NULL       |             NULL | NULL                                              |
|  3 | 2018-03-01 | 15:00:00    |       1 |       3 |   222075875 |         1 | 13:00:00     | 2018-03-05 |       2109 | CARD          | HOTEL      | 1100214521684512 | 178 Shadow Brook St. West Chicago, IL 60185       |
|  4 | 2018-02-20 | 07:00:00    |       2 |       4 |   333127845 |         4 | 15:00:00     | 2018-02-27 |       1785 | CARD          | MASTERCARD | 4400124565874591 | 802B Studebaker Drive Clinton Township, MI 48035  |
|  6 | 2018-03-01 | 18:00:00    |       1 |       6 |   666034568 |         1 | 23:00:00     | 2018-03-01 |         25 | CASH          | NULL       |             NULL | NULL                                              |
|  7 | 2018-01-20 | 06:00:00    |       2 |       7 |   777021654 |         3 | NULL         | NULL       |       NULL | CARD          | HOTEL      | 1100214532567845 | 87 Gregory Street Lawndale, CA 90260              |
|  8 | 2018-02-14 | 09:00:00    |       4 |       8 |   888091545 |         2 | NULL         | NULL       |       NULL | CARD          | VISA       | 4400178498564512 | 34 Hall Ave. Cranberry Twp, PA 16066              |
+----+------------+-------------+---------+---------+-------------+-----------+--------------+------------+------------+---------------+------------+------------------+---------------------------------------------------+
7 rows in set (0.00 sec)
```

1.1.2. "*Check if room(s) and room type requested are available.*"

In this example, we check to see if we have any available presidential suites accommodating at least 4 people for less than $500 per night.We see that we do have 1 such room available.

Per our project assumptions, if room category is presidential suite, then the room will be considered unavailable unless there is both a not-yet-dedicated room service staff member, and a not-yet-dedicated catering staff member assigned to the hotel.

Note that making this query more specific (e.g. to also consider which hotel the customer prefers to stay in), or less specific (e.g. to not require a presidential suite) is trivial. For the sake of the reader's sanity, we do not list all possible combinations of query here.

```
SELECT RoomNum, HotelID, MaxOcc, NightlyRate FROM Rooms WHERE
    Category = "PRESIDENTIAL_SUITE" AND
    MaxOcc >= 4 AND NightlyRate < 500 AND
    NOT EXISTS (SELECT * FROM Stays WHERE
        Stays.HotelID = Rooms.HotelID AND
        Stays.RoomNum = Rooms.RoomNum AND
        (CheckOutTime IS NULL OR EndDate IS NULL)
    ) AND (
        Category <> "PRESIDENTIAL_SUITE" OR (
            EXISTS (SELECT * FROM Staff WHERE
                Staff.JobTitle = "Room Service" AND
                Staff.HotelID = Rooms.HotelID AND
                Staff.ID NOT IN (SELECT DRSStaff FROM Rooms WHERE DRSStaff IS NOT NULL)
            ) AND
            EXISTS (SELECT * FROM Staff WHERE
                Staff.JobTitle = "Catering" AND
                Staff.HotelID = Rooms.HotelID AND
                Staff.ID NOT IN (SELECT DCStaff FROM Rooms WHERE DCStaff IS NOT NULL)
            )
        )
    );
```

```
+---------+---------+--------------------+--------+-------------+
| RoomNum | HotelID | Category           | MaxOcc | NightlyRate |
+---------+---------+--------------------+--------+-------------+
|       2 |       1 | PRESIDENTIAL_SUITE |      4 |         450 |
+---------+---------+--------------------+--------+-------------+
1 row in set (0.00 sec)
```

1.1.3. "*Assign rooms to customers according to their requests and to availability.*"

Because there are other queries (discussed in a previous section of this report) that determine room availability, here we will assume that we have an available room to assign to a customer.

In this example, we assign room 1 in hotel 3 (a presidential suite) to Jay Sharp (who has SSN 111038548 and has a friend staying with him). Because this is a presidential suite, IF the room assignment is successful, then we must also assign dedicated room service staff (we choose Maddison Davies, staff ID 17) and catering staff (we choose Crystal Barr, staff ID 18). These multiple database updates must happen within the same transaction to ensure that presidential suites have dedicated staff to serve them whenever they are occupied.

If the assigned room were not a presidential suite, only one database update (the insert into Stays) would be needed.

Also, as noted in our project assumptions, certain billing information is required upfront in order to check a customer in.

A check-in cannot proceed without necessary billing information available:

- Customer SSN
- Customer name
- Payment method
- Card Type if payment is card
- Card # if payment method is card
- Customer billing address if payment method is card

A check-in can proceed without unnecessary billing information available:

- Card type if payment method is not card
- Card # if payment method is not card
- Customer billing address if payment method is not card

The query below enforces these assumptions. Customer SSN and payment method are enforced by NOT NULL constraints in the Stays table. Customer name is enforced by a NOT NULL constraint in the Customers table, combined with a foreign key constraint in the Stays table. What remains for the query itself to enforce is the presence of further billing information in the case that the customer chooses to pay with a credit card. This is enforced by using a SELECT statement to produce values to be inserted. If the WHERE clause evaluates to false then no values will be produced, and thus the insertion will not take place.

```
INSERT INTO Stays (StartDate, CheckInTime, RoomNum, HotelID, CustomerSSN, NumGuests, PaymentMethod,
CardType, CardNumber, BillingAddress)
SELECT CURDATE(), CURTIME(), RoomNum, HotelID, 111038548, 2, "CARD", "VISA", 0987654321098765, "123
Alphabet Street, Durham, NC, 27707"
FROM Rooms WHERE
    RoomNum = 1 AND HotelID = 3 AND (
        "CARD" <> "CARD" OR (
            "VISA" IS NOT NULL AND
            0987654321098765 IS NOT NULL AND
            "123 Alphabet Street, Durham, NC, 27707" IS NOT NULL
        )
    );


UPDATE Rooms SET DRSStaff = 17, DCStaff = 18 WHERE
    RoomNum = 1 AND
    HotelID = 3 AND
    EXISTS (SELECT * FROM Stays WHERE RoomNum = 1 AND HotelID = 3 AND EndDate IS NULL);
```

```
MariaDB [smscoggi]> SELECT * FROM Stays;
+----+------------+-------------+---------+---------+-------------+-----------+---------------+------------+-------------+---------------+----------+------------------+-----------------------------------------------------+
| ID | StartDate  | CheckInTime | RoomNum | HotelID | CustomerSSN | NumGuests | CheckOutTime  | EndDate    | AmountOwed  | PaymentMethod | CardType | CardNumber       | BillingAddress                                      |
+----+------------+-------------+---------+---------+-------------+-----------+---------------+------------+-------------+---------------+----------+------------------+-----------------------------------------------------+
|  1 | 2018-01-12 | 20:10:00    |       1 |       1 |   555284568 |         3 | 10:00:00      | 2018-01-20 |        1320 | CARD          | VISA     | 4400123454126587 | 7178 Kent St. Enterprise, AL 36330                  |
|  2 | 2018-02-15 | 10:20:00    |       3 |       2 |   111038548 |         2 | 08:00:00      | 2018-02-18 |         775 | CASH          | NULL     |             NULL | NULL                                                |
|  3 | 2018-03-01 | 15:00:00    |       1 |       3 |   222075875 |         1 | 13:00:00      | 2018-03-05 |        2109 | CARD          | HOTEL    | 1100214521684512 | 178 Shadow Brook St. West Chicago, IL 60185         |
|  4 | 2018-02-20 | 07:00:00    |       2 |       4 |   333127845 |         4 | 15:00:00      | 2018-02-27 |        1785 | CARD          | MASTERCARD | 4400124565874591 | 802B Studebaker Drive Clinton Township, MI 48035    |
|  5 | 2018-03-05 | 11:00:00    |       3 |       5 |   444167216 |         4 | 08:00:00      | 2018-03-12 |        3520 | CARD          | VISA     | 4400127465892145 | 83 Inverness Court Longwood, FL 32779               |
|  6 | 2018-03-01 | 18:00:00    |       1 |       6 |   666034568 |         1 | 23:00:00      | 2018-03-01 |          25 | CASH          | NULL     |             NULL | NULL                                                |
|  7 | 2018-01-20 | 06:00:00    |       2 |       7 |   777021654 |         3 | NULL          | NULL       |        NULL | CARD          | HOTEL    | 1100214532567845 | 87 Gregory Street Lawndale, CA 90260                |
|  8 | 2018-02-14 | 09:00:00    |       4 |       8 |   888091545 |         2 | NULL          | NULL       |        NULL | CARD          | VISA     | 4400178498564512 | 34 Hall Ave. Cranberry Twp, PA 16066                |
|  9 | 2018-03-17 | 10:46:46    |       1 |       3 |   111038548 |         2 | NULL          | NULL       |        NULL | CARD          | VISA     |  987654321098765 | 123 Alphabet Street, Durham, NC, 27707              |
+----+------------+-------------+---------+---------+-------------+-----------+---------------+------------+-------------+---------------+----------+------------------+-----------------------------------------------------+
9 rows in set (0.00 sec)
```

```
MariaDB [smscoggi]> SELECT * FROM Rooms WHERE HotelID = 3;
+---------+---------+---------------------+---------+-------------+----------+----------+
| RoomNum | HotelID | Category            | MaxOcc  | NightlyRate | DRSStaff | DCStaff  |
+---------+---------+---------------------+---------+-------------+----------+----------+
|       1 |       3 | PRESIDENTIAL_SUITE  |       3 |         550 |       17 |       18 |
|       2 |       3 | ECONOMY             |       2 |         350 |     NULL |     NULL |
|       3 |       3 | DELUXE              |       3 |         450 |     NULL |     NULL |
+---------+---------+---------------------+---------+-------------+----------+----------+
3 rows in set (0.00 sec)
```

1.1.4. "*Release rooms.*"

In this example, we release the room associated with stay ID 8 (which is room 4 in hotel 8).

There are two steps to a room release, both of which should be performed within the same transaction.

First, we set the CheckOutTime and the EndDate in the stay record.

Then, we release any dedicated staff that may have been serving the room. This is to ensure a clean database, and to keep other queries from considering these staff members as unavailable to serve other rooms. Note that this means that we cannot answer questions about which staff members were ***dedicated*** to which rooms in the past. We know of no reason to keep track of such information. On the other hand, we can answer questions about which staff members actually ***provided*** services to which rooms in the past.

Releasing a room happens at check-out time, as does generating an itemized receipt and a bill for the customer. When an itemized receipt and bill is generated, the stay record will also have AmountOwed updated. The query for calculating AmountOwed uses EndDate to calculate the cost of the room, based on how many nights the customer stayed in the room. Therefore, when these queries are organized into an application, it will be important to release the room prior to generating the itemized receipt and bill.

```
UPDATE Stays
SET CheckOutTime = CURTIME(), EndDate = CURDATE()
WHERE ID = 8;

UPDATE Rooms
SET DCStaff = NULL, DRSStaff = NULL
WHERE
    RoomNum = (SELECT RoomNum FROM Stays WHERE ID = 8)
    AND HotelID = (SELECT HotelID FROM Stays WHERE ID = 8);
```

```
MariaDB [smscoggi]> SELECT * FROM Stays WHERE ID = 8;
+----+------------+-------------+---------+---------+--------------+-----------+--------------+------------+-------------+-------------+----------+------------------+-----------------------------------+
| ID | StartDate  | CheckInTime | RoomNum | HotelID | CustomerSSN  | NumGuests | CheckOutTime | EndDate    | AmountOwed  | PaymentMethod | CardType | CardNumber       | BillingAddress                    |
+----+------------+-------------+---------+---------+--------------+-----------+--------------+------------+-------------+-------------+----------+------------------+-----------------------------------+
|  8 | 2018-02-14 | 09:00:00    |       4 |       8 |   888091545 |         2 | 18:26:48     | 2018-03-14 |        NULL | CARD          | VISA     | 4400178498564512 | 34 Hall Ave. Cranberry Twp, PA 16066 |
+----+------------+-------------+---------+---------+--------------+-----------+--------------+------------+-------------+-------------+----------+------------------+-----------------------------------+
1 row in set (0.00 sec)

MariaDB [smscoggi]> SELECT * FROM Rooms WHERE RoomNum = 4 AND HotelID = 8;
+---------+---------+--------------------+--------+-------------+----------+---------+
| RoomNum | HotelID | Category           | MaxOcc | NightlyRate | DRSStaff | DCStaff |
+---------+---------+--------------------+--------+-------------+----------+---------+
|       4 |       8 | PRESIDENTIAL_SUITE |      4 |         450 |     NULL |    NULL |
+---------+---------+--------------------+--------+-------------+----------+---------+
1 row in set (0.00 sec)
```

1.2. Maintaining Service Records

1.2.1. "*For each customer stay, enter/update service records for services such as phone bills, dry cleaning, gyms, room service, and special requests.*"

1.2.1.1. Enter a new service record

In this example, we enter a new service record associated with stay ID 8, where the customer visits the gym, assisted by staff member ID 55 (Jude Cole).

As noted in our project assumptions:

- All services must be provided by a staff member with a job title matching the service provided. There are only two exceptions to this rule. First, a manager may provide any type of service. Second, phone service or a special request may be provided by any staff member
- All services must be provided by a staff member assigned to the hotel where the customer's stay is taking place.
- The only standard services offered by the hotel are phone bills, dry cleaning, gyms, room service, and catering. All other services are considered "special requests"
- A staff member dedicated to the service of one room may not provide a service to another room.

These project assumptions are all enforced by the query below. Any violations of these assumptions will result in the new service record not being added to the database. In our example, none of these assumptions are violated, and so the new service record is successfully added.

```
INSERT INTO Provided (StayID, StaffID, ServiceName)
SELECT Stays.ID, Staff.ID, ServiceTypes.Name
    FROM Stays, Staff, ServiceTypes WHERE
        Stays.ID = 8 AND
        Staff.ID = 55 AND
        ServiceTypes.Name = "Gym" AND (
            Staff.JobTitle = ServiceTypes.Name OR
            ServiceTypes.Name = "Phone" OR
            ServiceTypes.Name = "Special Request" OR
            Staff.JobTitle = "Manager"
        ) AND
        Staff.HotelID = Stays.HotelID AND
    NOT EXISTS (
        SELECT * FROM Rooms WHERE
            RoomNum <> Stays.RoomNum AND (
                DRSStaff = Staff.ID OR
                DCStaff = Staff.ID
            )
    );
```

```
MariaDB [smscoggi]> SELECT * FROM Provided;
+----+--------+---------+--------------+
| ID | StayID | StaffID | ServiceName  |
+----+--------+---------+--------------+
|  1 |      1 |       7 | Gym          |
|  2 |      1 |       7 | Gym          |
|  3 |      1 |       5 | Catering     |
|  4 |      2 |      11 | Room Service |
|  5 |      3 |      19 | Dry Cleaning |
|  6 |      4 |      26 | Gym          |
|  7 |      5 |      32 | Dry Cleaning |
|  8 |      6 |      38 | Room Service |
|  9 |      7 |      48 | Gym          |
| 10 |      8 |      54 | Dry Cleaning |
| 11 |      8 |      55 | Gym          |
+----+--------+---------+--------------+
11 rows in set (0.00 sec)
```

1.2.1.2. Update a service record

In this example, we update all important aspects of a service record: stay ID, staff ID, and service type name. Note that this comprehensive query can, if needed, be broken down into 3 simpler queries, one for each of these aspects. Alternatively, this query could be used as-is, and the 2 aspects which we not not wish to change could be provided to the query as-is.

In this example, we change the service record with ID 4, to change the stay ID to 4, the staff ID to 24, and the service type to Catering.

As noted in our project assumptions:

- All services must be provided by a staff member with a job title matching the service provided. There are only two exceptions to this rule. First, a manager may provide any type of service. Second, phone service or a special request may be provided by any staff member
- All services must be provided by a staff member assigned to the hotel where the customer's stay is taking place
- The only standard services offered by the hotel are phone bills, dry cleaning, gyms, room service, and catering. All other services are considered "special requests"
- A staff member dedicated to the service of one room may not provide a service to another room.

These project assumptions are all enforced by the query below. Any violations of these assumptions will result in the service record not being updated. In our example, none of these assumptions are violated, and so the service record is successfully updated.

Note: There are several places in the query below where the string "Catering" is used. As usual, where a value is in **large bold** font, this should be interpreted as a value that would changed based on the example. In our example, we wish to change the service type to catering. In some other example, we might wish to change service type to "Gym", or "Phone", etc.

```
UPDATE Provided SET StayID = 4, StaffID = 24, ServiceName = "Catering" WHERE
    ID = 4 AND (
    4 IN (SELECT ID FROM Stays) AND
    Provided.StaffID IN (SELECT ID FROM Staff) AND
    "Catering" IN (SELECT Name FROM ServiceTypes) AND (
        "Catering" = (SELECT JobTitle FROM Staff WHERE ID = 24) OR
        "Catering" = "Phone" OR
        "Catering" = "Special Request" OR
        "Manager" = (SELECT JobTitle FROM Staff WHERE ID = 24)
    ) AND
    (SELECT HotelID FROM Stays WHERE ID = 4) = (SELECT HotelID FROM Staff WHERE ID = 24) AND
    NOT EXISTS (
        SELECT * FROM Rooms WHERE
            RoomNum <> (SELECT RoomNum FROM Stays WHERE ID = 4) AND (
                DRSStaff = 24 OR
                DCStaff = 24
            )
    )
);
```

```
MariaDB [smscoggi]> SELECT * FROM Provided;
+----+--------+---------+--------------+
| ID | StayID | StaffID | ServiceName  |
+----+--------+---------+--------------+
|  1 |      1 |       7 | Gym          |
|  2 |      1 |       7 | Gym          |
|  3 |      1 |       5 | Catering     |
|  4 |      4 |      24 | Catering     |
|  5 |      3 |      19 | Dry Cleaning |
|  6 |      4 |      26 | Gym          |
|  7 |      5 |      32 | Dry Cleaning |
|  8 |      6 |      38 | Room Service |
|  9 |      7 |      48 | Gym          |
| 10 |      8 |      54 | Dry Cleaning |
+----+--------+---------+--------------+
10 rows in set (0.00 sec)
```

1.2.1.3. Add a new service type

This potentially desired task is mentioned only for completeness, so that we may note that it is not supported by the Wolf Inns system.

As noted in our project assumptions, the only standard services offered by the hotel are phone bills, dry cleaning, gyms, room service, and catering. All other services are considered “special requests”. Additionally, all services must be provided by a staff member with a job title matching the service provided. There are only two exceptions to this rule. First, a manager may provide any type of service. Second, phone service or a special request may be provided by any staff member.

Supporting a feature to add a new service type would introduce complexity into the Wolf Inns database system design which is not merited by the required tasks and operations.

1.2.1.4. Update the cost of a service type

Per our project assumptions, “While the Wolf Inns system supports updating most aspects of a record, the primary key of any record can never be changed - this is not supported by the Wolf Inns system.” Thus a service type's name may not be updated. In this example, we update the cost of a visit to the gym to $20.

```
UPDATE ServiceTypes SET Cost = 20 WHERE Name = "Gym";
```

```
MariaDB [smscoggi]> SELECT * FROM ServiceTypes;
+-----------------+------+
| Name            | Cost |
+-----------------+------+
| Phone           |   25 |
| Dry Cleaning    |   20 |
| Gym             |   20 |
| Room Service    |   25 |
| Catering        |   50 |
| Special Request |   40 |
+-----------------+------+
6 rows in set (0.00 sec)
```

1.3. Maintaining billing accounts

1.3.1. "*Generate/maintain billing accounts for each customer stay. When generating bills, take into account that customers that pay with the hotel credit card get a 5% discount.*"

Here, we will discuss only the total amount owed, taking into account the possible 5% discount. In a section below, we discuss the itemized receipt.

In this example, we produce the total amount owed for the first stay recorded in our database. In this stay, the guest stayed for 8 nights at a rate of $150 per night, got catering once at a cost of $50, and went to the gym twice at a rate of $35 per visit. This customer did not use the hotel credit card, and so did not get the 5% discount. Thus the total amount owed is $1,320.00.

```
SELECT IF(
   (
      SELECT CardType
      FROM Stays
      WHERE ID = 1
   ) = 'HOTEL',
   SUM(TotalCost) * 0.95,
   SUM(TotalCost)
) AS AmountOwed
FROM ([Itemized Receipt query shown in next section]) AS ItemizedReceipt;
```

```
+-------------+
| AmountOwed  |
+-------------+
|        1320 |
+-------------+
1 row in set (0.01 sec)
```

When the total amount owed is generated (at check-out time), it should be updated in the Stays relation. The reason for this is that some elements that influence total amount owed (e.g. a room's nightly rate) may change in the future - but we will still need to run accurate reports on actual revenue generated by hotels, given the room rates etc. that were applicable at the time.

```
UPDATE Stays SET AmountOwed = 1320 WHERE ID = 1;
```

```
MariaDB [smscoggi]> SELECT ID, AmountOwed FROM Stays WHERE ID = 1;
+----+------------+
| ID | AmountOwed |
+----+------------+
|  1 |       1320 |
+----+------------+
1 row in set (0.01 sec)
```

1.3.2. "*At check-out time, return the total amount owed by the customer, as well as an itemized receipt.*"

Here, we will discuss only the itemized receipt. In a previous section, we discussed the total amount owed, taking into account the possible 5% discount.

In this example, we produce an itemized receipt for the first stay recorded in our database. This produces the union of two relations. First, a relation showing costs incurred based on nights stayed in a particular room. Second, a relation showing costs incurred by getting extra services (catering, etc).

```
(
    SELECT 'NIGHT' AS Item, Nights AS Qty, NightlyRate AS ItemCost, Nights * NightlyRate AS TotalCost
    FROM (
        SELECT DATEDIFF(EndDate, StartDate) AS Nights, NightlyRate
        FROM (
            SELECT StartDate, EndDate, NightlyRate
            FROM Rooms NATURAL JOIN (
                SELECT StartDate, EndDate, RoomNum, HotelID
                FROM Stays
                WHERE ID = 1
            ) AS A
       ) AS B
    ) AS C
)
UNION
(
    SELECT Name AS Item, COUNT(*) AS Qty, Cost AS ItemCost, COUNT(*) * Cost AS TotalCost
    FROM (
        ServiceTypes NATURAL JOIN (
            SELECT StayID, ServiceName AS Name
            FROM Provided
            WHERE StayID = 1
        ) AS ServicesProvided
    ) GROUP BY Item
);
```

```
+----------+------+----------+-----------+
| Item     | Qty  | ItemCost | TotalCost |
+----------+------+----------+-----------+
| NIGHT    |    8 |      150 |      1200 |
| Gym      |    2 |       35 |        70 |
| Catering |    1 |       50 |        50 |
+----------+------+----------+-----------+
3 rows in set (0.01 sec)
```

1.4. Reports

1.4.1. "*Report occupancy by hotel, room type, date range, and city.*"

1.4.1.1. Report occupancy by hotel

Per our project assumptions, reports of occupancy are expected to report occupancy by room, and not by guest (e.g. "4 rooms occupied" rather than "6 guests staying"). For the convenience of the user, we have decided to report occupancy with some context (total rooms, available rooms).

```
SELECT
    HotelID,
    count(*) AS TotalRooms,
    COALESCE(SUM(OccupiedFlag),0) AS OccupiedRooms,
    count(*) - COALESCE(SUM(OccupiedFlag),0) AS AvailableRooms
FROM (
    Rooms NATURAL LEFT OUTER JOIN
    (
        SELECT RoomNum, HotelID, 1 AS OccupiedFlag
        FROM Stays AS X
        WHERE CheckOutTime IS NULL OR EndDate IS NULL
    ) AS Y)
GROUP BY HotelID;
```

```
+---------+------------+---------------+----------------+
| HotelID | TotalRooms | OccupiedRooms | AvailableRooms |
+---------+------------+---------------+----------------+
|       1 |          3 |             0 |              3 |
|       2 |          3 |             0 |              3 |
|       3 |          3 |             0 |              3 |
|       4 |          2 |             0 |              2 |
|       5 |          3 |             0 |              3 |
|       6 |          2 |             0 |              2 |
|       7 |          2 |             1 |              1 |
|       8 |          4 |             1 |              3 |
+---------+------------+---------------+----------------+
8 rows in set (0.00 sec)
```

1.4.1.2. Report occupancy by room type

Per our project assumptions, reports of occupancy are expected to report occupancy by room, and not by guest (e.g. “4 rooms occupied” rather than “6 guests staying”).

```
SELECT Category AS RoomType, count(*) AS TotalRooms, SUM(OccupiedFlag) AS Occupancy, count(*) -
SUM(OccupiedFlag) AS Availability
FROM (
    SELECT RoomNum, HotelID, Category, IF(
        EXISTS(
            SELECT *
            FROM Stays
            WHERE Stays.RoomNum = Rooms.RoomNum AND Stays.HotelID = Rooms.HotelID AND (
                CheckOutTime IS NULL OR
                EndDate IS NULL
            )
        ),
        1,
        0
    ) AS OccupiedFlag
    FROM Rooms
) AS X
GROUP BY Category;
```

```
+--------------------+------------+-----------+--------------+
| RoomType           | TotalRooms | Occupancy | Availability |
+--------------------+------------+-----------+--------------+
| DELUXE             |          5 |         0 |            5 |
| ECONOMY            |          7 |         0 |            7 |
| EXECUTIVE_SUITE    |          6 |         1 |            5 |
| PRESIDENTIAL_SUITE |          4 |         1 |            3 |
+--------------------+------------+-----------+--------------+
4 rows in set (0.02 sec)
```

1.4.1.3. Report occupancy by date range

Because the Wolf Inns system supports adding new rooms, and deleting existing rooms, and because it does not maintain historical snapshots of what rooms existed in the system at all times in the past, there are some things that we do not have the ability to accurately report. We cannot report percentage of rooms occupied during some date range in the past, likewise we cannot report how many rooms were available (not occupied) during some date range in the past.

We can however report how many rooms in the Wolf Inns hotel chain were occupied at some point during a given date range.

In this example, we report how many rooms in the Wolf Inns hotel chain were occupied at some point during February 2018. There are 2 rooms which were assigned and then released within February 2018. There are 2 additional rooms which were assigned in or before February 2018, and which still have not been released. So these can also be considered as rooms which were occupied in February 2018.

Note that there is a protection designed into the query to keep records being returned in the case that the given date range has a start that takes place in the future - we cannot make any claims about such date ranges.

```
SELECT SUM(OccupiedFlag) AS RoomsOccupied
FROM (
    SELECT RoomNum, HotelID, IF (count(*) > 0, 1, 0) AS OccupiedFlag
    FROM Stays WHERE
        StartDate <= "2018-02-28" AND
        CURDATE() >= "2018-02-01" AND
        (EndDate >= "2018-02-01" OR EndDate IS NULL)
    GROUP BY RoomNum, HotelID
) AS X;
```

```
+---------------+
| RoomsOccupied |
+---------------+
|             4 |
+---------------+
1 row in set (0.01 sec)
```

1.4.1.4. Report occupancy by city

Per our project assumptions, reports of occupancy are expected to report occupancy by room, and not by guest (e.g. “4 rooms occupied” rather than “6 guests staying”).

```
SELECT
    City,
    count(*) AS TotalRooms,
    COALESCE(SUM(OccupiedFlag),0) AS OccupiedRooms,
    count(*) - COALESCE(SUM(OccupiedFlag),0) AS AvailableRooms
FROM (
    Rooms NATURAL LEFT OUTER JOIN
    (
        SELECT RoomNum, HotelID, 1 AS OccupiedFlag
        FROM Stays AS X
        WHERE CheckOutTime IS NULL OR EndDate IS NULL
    ) AS Y NATURAL LEFT OUTER JOIN
    (SELECT ID AS HotelID, City FROM Hotels) AS Z
)
GROUP BY City;
```

```
+----------+------------+---------------+----------------+
| City     | TotalRooms | OccupiedRooms | AvailableRooms |
+----------+------------+---------------+----------------+
| New York |         12 |             1 |             11 |
| Raleigh  |         10 |             1 |              9 |
+----------+------------+---------------+----------------+
2 rows in set (0.00 sec)
```

1.4.2. "*Report total occupancy and percentage of rooms occupied, return information on staff grouped by their role.*"

1.4.2.1. Report total occupancy

Per our project assumptions, reports of occupancy are expected to report occupancy by room, and not by guest (e.g. "4 rooms occupied" rather than "6 guests staying"). We have 22 rooms, 2 of which are occupied.

```
SELECT count(*) AS TotalOccupancy
FROM Stays
WHERE
    CheckOutTime IS NULL OR
    EndDate IS NULL;
```

```
+-----------------+
| TotalOccupancy |
+-----------------+
|               2 |
+-----------------+
1 row in set (0.00 sec)
```

1.4.2.2. Report percentage of rooms occupied

We have 22 rooms, 2 of which are occupied.

```
SELECT (SUM(OccupiedFlag) / count(*)) * 100 AS PercentOccupied
FROM (
    Rooms NATURAL LEFT OUTER JOIN
    (
        SELECT RoomNum, HotelID, 1 AS OccupiedFlag
        FROM Stays AS X
        WHERE CheckOutTime IS NULL OR EndDate IS NULL
    ) AS Y);
```

```
+------------------+
| PercentOccupied |
+------------------+
|           9.0909 |
+------------------+
1 row in set (0.00 sec)
```

1.4.2.3. Return information on staff grouped by their role

```
SELECT * FROM Staff ORDER BY JobTitle;
```

```
MariaDB [smscoggi]> SELECT * FROM Staff ORDER BY JobTitle;
+----+------------------+------------+--------------------------+-----+------------+-----------------------------------------------------+---------+
| ID | Name             | DOB        | JobTitle                 | Dep | PhoneNum   | Address                                             | HotelID |
+----+------------------+------------+--------------------------+-----+------------+-----------------------------------------------------+---------+
|  5 | Michael Day      | 1983-02-25 | Catering                 | B   | 3294931245 | 4 Goldfield Rd. Honolulu, HI 96815                  |       1 |
|  8 | Nicholas Read    | 1981-01-14 | Catering                 | B   | 2564132017 | 236 Pumpkin Hill Court Leesburg, VA 20175           |       1 |
| 12 | Caitlin Cole     | 1989-08-15 | Catering                 | B   | 4997845612 | 7 Ivy Ave. Traverse City, MI 49684                  |       2 |
| 18 | Crystal Barr     | 1989-04-06 | Catering                 | B   | 4591247845 | 9094 6th Ave. Macomb, MI 48042                      |       3 |
| 24 | Alessandro Beck  | 1981-06-12 | Catering                 | A   | 5774812452 | 682 Glen Ridge St. Leesburg, VA 20175               |       4 |
| 30 | Ryan Parker      | 1972-08-13 | Catering                 | B   | 1183024152 | 157 State Dr. Attleboro, MA 02703                   |       5 |
| 31 | Glen Elliott     | 1971-09-14 | Catering                 | B   | 6502134785 | 9775 Clinton Dr. Thornton, CO 80241                 |       5 |
| 39 | Carmen Lee       | 1976-01-04 | Catering                 | A   | 9885234562 | 120 Longbranch Drive Port Richey, FL 34668          |       6 |
| 45 | Grace Francis    | 1974-04-09 | Catering                 | B   | 3425612345 | 282 Old York Court Mechanicsburg, PA 17050          |       7 |
| 53 | Jamie Young      | 1986-06-05 | Catering                 | A   | 9892145214 | 8111 Birch Hill Avenue Ravenna, OH 44266            |       8 |
|  6 | David Adams      | 1985-01-17 | Dry Cleaning             | C   | 9194153214 | 44 Shirley Ave. West Chicago, IL 60185              |       1 |
| 13 | Victoria Medina  | 1989-02-04 | Dry Cleaning             | C   | 1341702154 | 8221 Trenton St. Jamestown, NY 14701                |       2 |
| 19 | Dayana Tyson     | 1980-05-05 | Dry Cleaning             | B   | 4072134587 | 837 W. 10th St. Jonesboro, GA 30236                 |       3 |
| 25 | Emily Petty      | 1984-08-19 | Dry Cleaning             | A   | 5772812456 | 7604 Courtland St. Easley, SC 29640                 |       4 |
| 32 | Ash Harrison     | 1977-02-15 | Dry Cleaning             | C   | 9192451365 | 9924 Jefferson Ave. Plainfield, NJ 07060            |       5 |
| 40 | Mell Tran        | 1979-06-05 | Dry Cleaning             | A   | 9162451245 | 32 Pearl St. Peoria, IL 61604                       |       6 |
| 46 | Macy Fuller      | 1975-05-02 | Dry Cleaning             | C   | 4665127845 | 57 Shadow Brook St. Hudson, NH 03051                |       7 |
| 54 | Jackie Miller    | 1978-08-06 | Dry Cleaning             | A   | 9795486234 | 9895 Redwood Court Glenview, IL 60025               |       8 |
|  2 | Katelyn Weeks    | 1970-04-20 | Front Desk Representative | B   | 6926641058 | 123 6th St. Melbourne, FL 32904                     |       1 |
| 10 | Oliver Lucas     | 1961-05-11 | Front Desk Representative | A   | 2519881245 | 7 Edgefield St. Augusta, GA 30906                   |       2 |
| 16 | Willow Roberts   | 1987-02-08 | Front Desk Representative | A   | 5535531245 | 7868 N. Lees Creek Street Chandler, AZ 85224        |       3 |
| 22 | Bobby Matthews   | 1982-02-14 | Front Desk Representative | C   | 5771812456 | 25 W. Dogwood Lane Bemidji, MN 56601                |       4 |
| 28 | Billy Lopez      | 1982-05-11 | Front Desk Representative | B   | 5124562123 | 99 Miles Road Danbury, CT 06810                     |       5 |
| 36 | Gabe Howard      | 1987-03-01 | Front Desk Representative | A   | 6501421523 | 914 Edgefield Dr. Hartselle, AL 35640               |       6 |
| 43 | Macy Fuller      | 1972-02-07 | Front Desk Representative | B   | 7485612345 | 676 Myers Street Baldwin, NY 11510                  |       7 |
| 50 | Kiran West       | 1985-02-02 | Front Desk Representative | C   | 9623154125 | 68 Smith Dr. Lexington, NC 27292                    |       8 |
|  7 | Ishaan Goodman   | 1993-04-19 | Gym                      | A   | 5203201425 | 514 S. Magnolia St. Orlando, FL 32806               |       1 |
| 14 | Will Rollins     | 1982-07-06 | Gym                      | C   | 7071264587 | 346 Beacon Lane Quakertown, PA 18951                |       2 |
| 20 | Tommy Perry      | 1979-06-04 | Gym                      | B   | 5774812456 | 785 Bohemia Street Jupiter, FL 33458                |       3 |
| 26 | Rudy Cole        | 1972-01-09 | Gym                      | A   | 5774812856 | 37 Marconi Drive Owensboro, KY 42301                |       4 |
| 33 | Leslie Little    | 1979-12-16 | Gym                      | C   | 9192014512 | 7371 Pin Oak St. Dalton, GA 30721                   |       5 |
| 34 | Mason West       | 1970-10-17 | Gym                      | C   | 6501231245 | 798 W. Valley Farms Lane Saint Petersburg, FL 33702 |       5 |
| 41 | Leslie Cook      | 1970-10-08 | Gym                      | B   | 6501245126 | 59 W. High Ridge Street Iowa City, IA 52240         |       6 |
| 47 | Cory Hoover      | 1976-06-12 | Gym                      | C   | 9252210735 | 892 Roosevelt Street Ithaca, NY 14850               |       7 |
| 48 | Sam Graham       | 1977-07-25 | Gym                      | C   | 7226251245 | 262 Bayberry St. Dorchester, MA 02125               |       7 |
| 55 | Jude Cole        | 1979-03-07 | Gym                      | A   | 9195642251 | 8512 Cambridge Ave. Lake In The Hills, IL 60156     |       8 |
|  1 | Zoe Holmes       | 1980-10-02 | Manager                  | A   | 8141113134 | 123 6th St. Melbourne, FL 32904                     |       1 |
|  9 | Dominic Mitchell | 1971-03-13 | Manager                  | A   | 2922497845 | 7005 South Franklin St. Somerset, NJ 08873          |       2 |
| 15 | Masen Shepard    | 1983-01-09 | Manager                  | A   | 8995412364 | 3 Fulton Ave. Bountiful, UT 84010                   |       3 |
| 21 | Joshua Burke     | 1972-01-10 | Manager                  | C   | 1245214521 | 8947 Briarwood St. Baldwin, NY 11510                |       4 |
| 27 | Blair Ball       | 1981-01-10 | Manager                  | A   | 8854124568 | 551 New Saddle Ave. Cape Coral, FL 33904            |       5 |
| 35 | Riley Dawson     | 1975-01-09 | Manager                  | C   | 1183021245 | 898 Ocean Court Hilliard, OH 43026                  |       6 |
| 42 | Rory Burke       | 1971-01-05 | Manager                  | B   | 7702653764 | 9273 Ridge Drive Winter Springs, FL 32708           |       7 |
| 49 | Charlie Adams    | 1981-01-01 | Manager                  | C   | 6084254152 | 9716 Glen Creek Dr. Newark, NJ 07103                |       8 |
|  3 | Abby Huffman     | 1990-12-14 | Room Service             | C   | 6738742135 | 71 Pilgrim Avenue Chevy Chase, MD 20815             |       1 |
|  4 | Oliver Gibson    | 1985-05-12 | Room Service             | A   | 1515218329 | 70 Bowman St. South Windsor, CT 06074               |       1 |
| 11 | Molly Thomas     | 1987-07-10 | Room Service             | B   | 5425871245 | 541 S. Holly Street Norcross, GA 30092              |       2 |
| 17 | Maddison Davies  | 1981-03-07 | Room Service             | A   | 6784561245 | 61 New Road Ithaca, NY 14850                        |       3 |
| 23 | Pedro Cohen      | 1983-04-24 | Room Service             | C   | 8774812456 | 9708 Brickyard Ave. Elyria, OH 44035                |       4 |
| 29 | Lee Ward         | 1983-06-12 | Room Service             | B   | 9209124562 | 959 S. Tailwater St. Ridgewood, NJ 07450            |       5 |
| 37 | Jessie Nielsen   | 1982-06-02 | Room Service             | A   | 7574124587 | 7973 Edgewood Road Gallatin, TN 37066               |       6 |
| 38 | Gabe Carlson     | 1983-08-03 | Room Service             | A   | 5771245865 | 339 Pine Lane Tampa, FL 33604                       |       6 |
| 44 | Megan Lloyd      | 1973-03-01 | Room Service             | B   | 7221452315 | 849 George Lane Park Ridge, IL 60068                |       7 |
| 51 | Franky John      | 1986-03-03 | Room Service             | A   | 8748544152 | 6 Shirley Road Fairborn, OH 45324                   |       8 |
| 52 | Charlie Bell     | 1985-04-04 | Room Service             | A   | 9845124562 | 66 Elm Street Jupiter, FL 33458                     |       8 |
+----+------------------+------------+--------------------------+-----+------------+-----------------------------------------------------+---------+
55 rows in set (0.00 sec)
```

1.4.3. "*For each customer stay, return information on all the staff members serving the customer during the stay.*"

Per our project assumption, "for each customer stay" can best be translated to "for a given customer stay", rather than "for every single customer stay". In this example, we produce a report on all the staff members serving the customer during the stay with ID 1.

Note: no special consideration is needed for staff dedicated to presidential suites, in this query. If those staff members actually provide a service, then they will be represented in this query. If they do not, then they have not actually served the customer during the stay - rather, they were simply on standby to do so if needed.

Note: To extend this query to also list date of birth, phone number, etc. is trivial. The information projected out in the query below is what the project team felt was the most useful information about staff members in this context.

```
SELECT ServiceName, StaffID, Name, JobTitle, Dep
FROM Provided, Staff
WHERE Provided.StaffID = Staff.ID AND StayID = 1;
```

```
+-------------+---------+-----------------+----------+-----+
| ServiceName | StaffID | Name            | JobTitle | Dep |
+-------------+---------+-----------------+----------+-----+
| Gym         |       7 | Ishaan Goodman | Gym      | A   |
| Gym         |       7 | Ishaan Goodman | Gym      | A   |
| Catering    |       5 | Michael Day    | Catering | B   |
+-------------+---------+-----------------+----------+-----+
3 rows in set (0.00 sec)
```

1.4.4. "*Generate revenue earned by a given hotel during a given date range.*"

In this example, we generate revenue earned by the hotel with ID of 1, during January 2018. We use the EndDate attribute of Stays, because it is at the end of the guest's stay that the hotel actually receives money.

The response accounts for the 1 stay that took place in hotel ID 1 during January 2018, where the guest stayed for 8 nights at a rate of $150 per night, got catering once at a cost of $50, and went to the gym twice at a rate of $35 per visit. This customer did not use the hotel credit card, and so did not get the 5% discount.

```
SELECT SUM(AmountOwed) AS Revenue
FROM Stays
WHERE (
   HotelID = 1 AND
   EndDate >= '2018-01-01'
   AND EndDate <= '2018-01-31'
);
```

```
+---------+
| Revenue |
+---------+
|    1320 |
+---------+
1 row in set (0.01 sec)
```

2. Execution Plans

2.1. Query 1

2.1.1. SQL query

In this example, we check to see if we have any available presidential suites accommodating at least 4 people for less than $500 per night. We see that we do have 1 such room available. This is copied from “Check if room(s) and room type requested are available”.

```
SELECT RoomNum, HotelID, MaxOcc, NightlyRate FROM Rooms WHERE
    Category = "PRESIDENTIAL_SUITE" AND
    MaxOcc >= 4 AND NightlyRate < 500 AND
    NOT EXISTS (SELECT * FROM Stays WHERE
        Stays.HotelID = Rooms.HotelID AND
        Stays.RoomNum = Rooms.RoomNum AND
        (CheckOutTime IS NULL OR EndDate IS NULL)
    ) AND (
        Category <> "PRESIDENTIAL_SUITE" OR (
            EXISTS (SELECT * FROM Staff WHERE
                Staff.JobTitle = "Room Service" AND
                Staff.HotelID = Rooms.HotelID AND
                Staff.ID NOT IN (SELECT DRSStaff FROM Rooms WHERE DRSStaff IS NOT NULL)
            ) AND
            EXISTS (SELECT * FROM Staff WHERE
                Staff.JobTitle = "Catering" AND
                Staff.HotelID = Rooms.HotelID AND
                Staff.ID NOT IN (SELECT DCStaff FROM Rooms WHERE DCStaff IS NOT NULL)
            )
        )
    );
```

```
+---------+---------+--------------------+--------+-------------+
| RoomNum | HotelID | Category           | MaxOcc | NightlyRate |
+---------+---------+--------------------+--------+-------------+
|       2 |       1 | PRESIDENTIAL_SUITE |      4 |         450 |
+---------+---------+--------------------+--------+-------------+
1 row in set (0.00 sec)
```

2.1.2. Execution plan without index

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	PRIMARY	Rooms	ALL	NULL	NULL	NULL	NULL	22	Using where
5	DEPENDENT SUBQUERY	Staff	ref	FK_STAFFHID	FK_STAFFHID	5	smscoggi.Rooms.HotelID	3	Using where
6	MATERIALIZED	Rooms	range	FK_ROOMDCID	FK_ROOMDCID	5	NULL	1	Using where; Using index
3	DEPENDENT SUBQUERY	Staff	ref	FK_STAFFHID	FK_STAFFHID	5	smscoggi.Rooms.HotelID	3	Using where
4	MATERIALIZED	Rooms	range	FK_ROOMDRSID	FK_ROOMDRSID	5	NULL	1	Using where; Using index
2	DEPENDENT SUBQUERY	Stays	ALL	FK_STAYRID	NULL	NULL	NULL	8	Using where

6 rows in set (0.00 sec)

We can see from “type” of “ALL” that a full table scan is performed on Rooms and also on Stays.

2.1.3. Index creation statement

```
create index CatIndex ON Rooms(Category);
```

2.1.4. Execution plan with index

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	PRIMARY	Rooms	ref	CatIndex	CatIndex	227	const	4	Using index condition; Using where
5	DEPENDENT SUBQUERY	Staff	ref	FK_STAFFHID	FK_STAFFHID	5	smscoggi.Rooms.HotelID	3	Using where
6	MATERIALIZED	Rooms	range	FK_ROOMDCID	FK_ROOMDCID	5	NULL	1	Using where; Using index
3	DEPENDENT SUBQUERY	Staff	ref	FK_STAFFHID	FK_STAFFHID	5	smscoggi.Rooms.HotelID	3	Using where
4	MATERIALIZED	Rooms	range	FK_ROOMDRSID	FK_ROOMDRSID	5	NULL	1	Using where; Using index
2	DEPENDENT SUBQUERY	Stays	ALL	FK_STAYRID	NULL	NULL	NULL	8	Using where

6 rows in set (0.00 sec)

We can see that after adding the index, there is no longer a need to perform a full table scan on Rooms.

2.2. Query 2

2.2.1. SQL query

In this example, we generate revenue earned by the hotel with ID of 1, during January 2018. We use the EndDate attribute of Stays, because it is at the end of the guest's stay that the hotel actually receives money. This is copied from "Generate revenue earned by a given hotel during a given date range."

```
SELECT SUM(AmountOwed) AS Revenue
FROM Stays
WHERE (
   HotelID = 1 AND
   EndDate >= '2018-01-01'
   AND EndDate <= '2018-01-31'
);
```

```
+---------+
| Revenue |
+---------+
|    1320 |
+---------+
1 row in set (0.01 sec)
```

2.2.2. Execution plan without index

```
+------+-------------+-------+------+---------------+------+---------+------+------+-------------+
| id   | select_type | table | type | possible_keys | key  | key_len | ref  | rows | Extra       |
+------+-------------+-------+------+---------------+------+---------+------+------+-------------+
|    1 | SIMPLE      | Stays | ALL  | NULL          | NULL | NULL    | NULL |    8 | Using where |
+------+-------------+-------+------+---------------+------+---------+------+------+-------------+
1 row in set (0.00 sec)
```

We can see from "type" of "ALL" that a full table scan is performed on Stays.

2.2.3. Index creation statement

```
create index EndDateIndex ON Stays(EndDate);
```

2.2.4. Execution plan with index

```
+------+-------------+-------+-------+---------------+--------------+---------+------+------+-----------------------------------+
| id   | select_type | table | type  | possible_keys | key          | key_len | ref  | rows | Extra                             |
+------+-------------+-------+-------+---------------+--------------+---------+------+------+-----------------------------------+
|    1 | SIMPLE      | Stays | range | EndDateIndex  | EndDateIndex | 4       | NULL |    1 | Using index condition; Using where |
+------+-------------+-------+-------+---------------+--------------+---------+------+------+-----------------------------------+
1 row in set (0.34 sec)
```

We can see that after adding the index, there is no longer a need to perform a full table scan on Stays.

3. Query Correctness

3.1. Query 1

3.1.1. Specification

"For each customer stay, return information on all the staff members serving the customer during the stay." Per our project assumption, "for each customer stay" can best be translated to "for a given customer stay", rather than "for every single customer stay". In this example, we produce a report on all the staff members serving the customer during the stay with ID 1.

3.1.2. SQL query

```
SELECT ServiceName, StaffID, Name, JobTitle, Dep
FROM Provided, Staff
WHERE Provided.StaffID = Staff.ID AND StayID = 1;
```

3.1.3. Relational Algebra Expression

$$\pi_{\text{ServiceName, StaffID, Name, JobTitle, Dep}} (\text{Provided} \bowtie_{\text{Provided.StaffID = Staff.ID ^ StayID = 1}} \text{Staff})$$

3.1.4. Correctness Proof

Suppose p is any tuple in the Provided relation, and s is any tuple in the Staff relation. We find all pairs p,s such that StayID is 1 and the staff member noted in p matches the staff member noted in s. This relation essentially augments all tuples p which are related to the stay of interest, with additional information about the staff member providing the service. The final step is to project out only the attributes of interest for the report: the name of the service, the ID of the staff member, the name of the staff member, the job title of the staff member, and the department of the staff member. Thus we have information reported for all staff members serving the customer during the stay of interest, as noted in the specification.

3.2. Query 2

3.2.1. Specification

Report percentage of rooms occupied.

3.2.2. SQL query

```
SELECT (SUM(OccupiedFlag) / count(*)) * 100 AS PercentOccupied
FROM (
    Rooms NATURAL LEFT OUTER JOIN
    (
        SELECT RoomNum, HotelID, 1 AS OccupiedFlag
        FROM Stays AS X
        WHERE CheckOutTime IS NULL OR EndDate IS NULL
    ) AS Y);
```

```
+-----------------+
| PercentOccupied |
+-----------------+
|          9.0909 |
+-----------------+
1 row in set (0.00 sec)
```

3.2.3. Relational Algebra Expression

$$\gamma_{(\text{SUM}(\text{OccupiedFlag})/\text{COUNT}(*))*100 \rightarrow PercentOccupied}(\text{Rooms} \bowtie \rho_{S(RoomNum,HotelID,OccupiedFlag)} (\pi_{RoomNum,HotelID,1} (\sigma_{CheckOutTime=NULL \vee EndDate=NULL}(\text{Stays}))))$$

3.2.4. Correctness Proof

Suppose r is any tuple in the Rooms relation, and s is any tuple in the Stays relation. We first derive a new relation, each tuple in which is related to one of the s in the original Stays relation. Only those s which represent a currently occupied room are kept, and those are stripped down to just the room number, hotel ID, and a simple flag with value 1, to be used in the next step. Next, we perform a natural left outer join between Rooms and our derived relation. This essentially augments all tuples r with a flag that is set to 1 if r represents a room that is currently occupied. The final step is to count the occupied rooms, and the total rooms, and output the ratio as a percentage. Thus we have the percentage of rooms occupied, as noted in the specification.