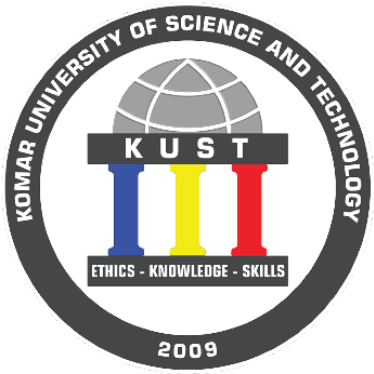


Komar University of Science and Technology

College of Engineering

Subject: Term Report (Final Assessment)– Spring 2020

Department Name: Computer Science

Report Title: The Important Procedures to Design and Create a Database

Course Code: CPE3330

Course Name: Fundamentals of Database Systems

Student ID: F180292

Student Name: Abdulrahman Tawffeq Jalal

Submission Date: 26-Jun-2020

Contents

Introduction

The DBMS (abbreviation of “Database Management System”) is one of the most important fields related to computer science. Any program or application needs to have a database as a back-end to be effective and useful. Otherwise, the system will be fixed. There are several DBMS that can be chosen by the programmer like MySQL, SQL Server, Microsoft Access, and others. Choosing any specific DBMS depends on the vision of the DBA “Database Administrator”. The DBA will choose the suitable DBMS according to the requirements, type of the application, number of users, etc. In this report, we will explain about the steps that the Database Designer will follow to design any database. Then, we will show how the DBA will change everything in the design to an actual database using a specific type of Database Management System.

Steps of Database Creation

User Requirements

Gathering or collecting the requirements and the information from the person who wants the Database. The Database Designer is the person who will listen to the people they ask for creating a database. He will gather all the requirements and identify the problem.

DBMS: A software (tool or package) that allows you to create the database itself. Then, you can manipulate or maintain it. You can also create a form for it, a report, or query.

The DBMS will be very useful to solve many problems:

1. DBMS will help to reduce, control, or prevent redundancy (repetition). Because you have to keep any table in the database as lower size as possible by having another table using foreign key.
2. By reducing repetition using relations between tables, we can update anything inside the Database easily (not one by one) and saving time.
3. If there is a lot of information put in one place with a database (digitalised one), it gives you the ability to answer any question. Instead of going through all records physically by yourself one by one manually which takes time. However, by couples of lines of code using SQL, the question will be answered quickly.

Conceptual Design

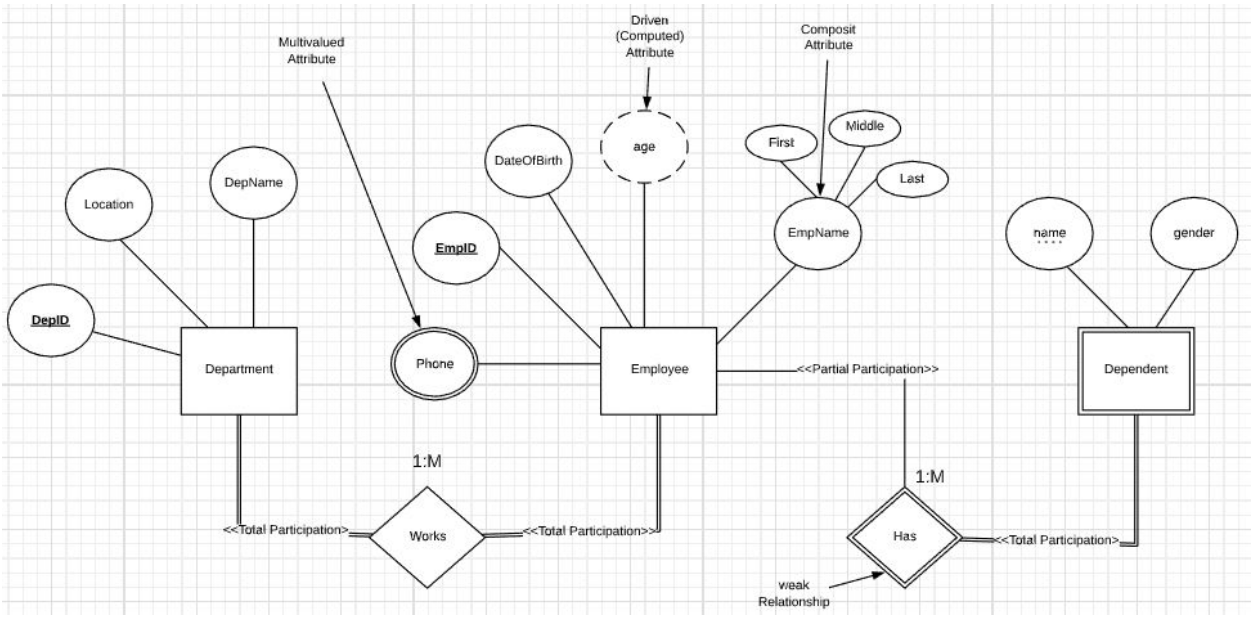
Here, the Database designer will decide how many entities are needed and what are the relationships between them. He will determine the attributes in the tables and that will be according to the requirements that he collected before.

In this level, the Database Designer will use something called entity relationship (ER) diagram.

ER Diagram: It is a type of a flowchart that illustrates how entities related to each other within a system.

Example:

This ER diagram is created according to a specific information collected from the person who wants the database.



Total Participation:

- Every department must have employees
- Every employee must related to a department
- Dependent must be related to a specific employee

Partial Participation:

- It is possible to have an employee without dependent

Composite Attribute:

- It can be divided into several values and each value will be an attribute later on

Driven Attribute:

- Like the age which can be computed from the date of birth. So, it will not be an attribute later on.

Multivalued Attribute:

- Like the phone which is possible to have an employee with multiple phone numbers. So, this attribute will be a separate table later on.

Note: The weak entity like the dependent is an entity without a primary key. It only has the discriminator with dotted underline.

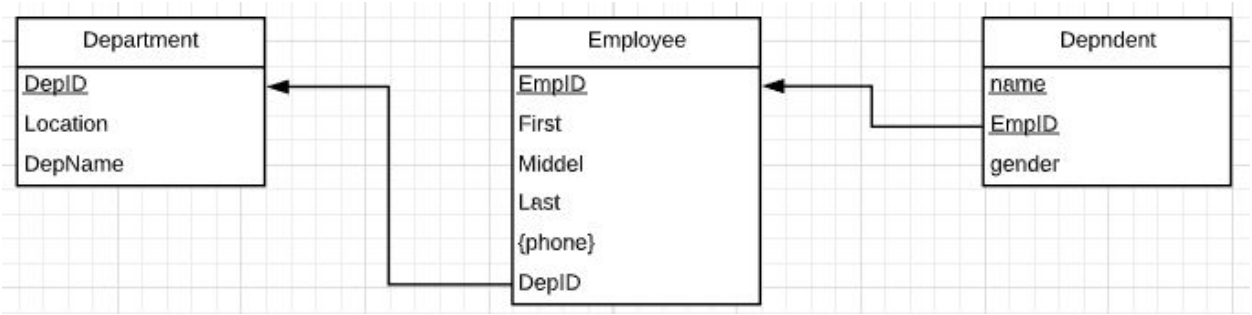
Logical Design

In this level, the entities in the ER diagram will be converted to tables with no more details.

We will add the relationships between the tables by depending on the primary key:

- In 1:M Relationship, the primary key in the table of side 1 will be as foreign key in the other table of side M. In case M:1 it will be vice versa.
- In M:N Relationship, we will have the Relationship itself as an intermediary table with the primary keys of the two tables.

Now, we will change the ER diagram of the previous example to logical design:



Note: The multivalued Attribute will be inside two curly brackets.

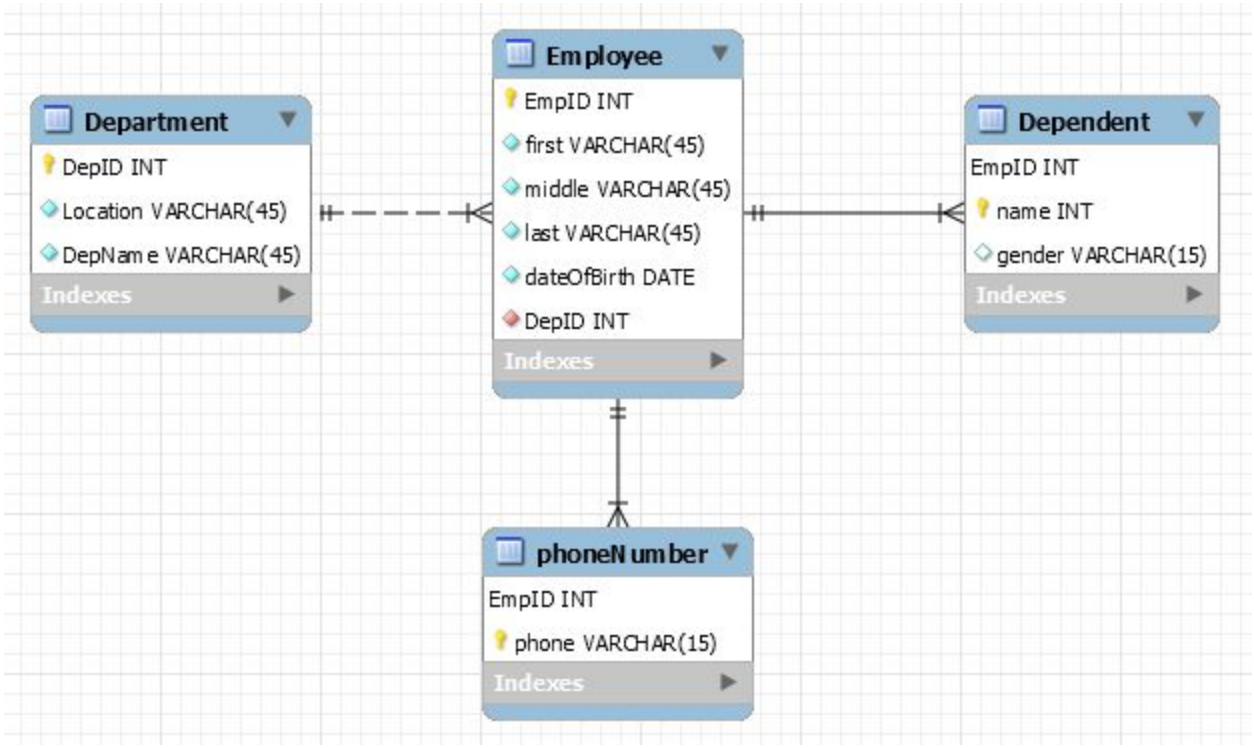
Note: If there are any weak entities, relationships, or attributes, we need to make it strong. That's why the Database Designer made the discriminator as primary key with the presence of the EmpID as primary key and foreign key at the same time. In this case, it became strong and each row is unique for now.

Note: The DepID is a primary key in the Department table and it is as foreign key in the Employee table. That's how the Database Designer changes the 1:M Relationship in the ER diagram to a relationship in the logical design.

Physical Design

In this level, the "Database Designer" will be in the last step before the implementation. He will draw everything including the data types, length of data types, participation, etc. All of that depends on the requirements that he took at the first step. The Database Designer will change the terminology as well to let everything clear to the DBA (Database Administrator).

Now, we will change the previous logical design into physical design:



In this level, the Database Designer will exclude any multivalued attribute (An employee may have more than one phone number) to a separated table. We notice that in the phone number table, both EmpID and phone are primary keys and this case is called “composite”. Both of them will identify the row uniquely.

Implementation

This is the last step of the database creation steps. It is related to computer work and the DBA (The person who is responsible for creating the actual database) will start creating the database. He will computerize everything in the design.

One of the DBA's responsibilities is to choose the suitable DBMS for the project, and that will depend on specific reasons.

The reasons behind choosing MySql DBMS

The most important features of MySQL database systems are speed and reliability, which explains why they are frequently used by developers, administrators, and users around the world.
Here are some reasons behind choosing MySql DBMS:

1- Security in MySQL:
Security is a strong point in MySQL, as it comes with a complex access control system and an authorization system to prevent unauthorized users from accessing the database. This system is implemented in the form of five layers of powers hierarchically, as MySQL administrators can protect access to sensitive data. It can also allow users to perform operations on specific databases or only specific fields.

2- Ease of use MySQL:
Ease of use is an important point that MySQL has focused on. The MySQL development team has taken on the task of facilitating the use, management and improvement of MySQL performance. The main interface of MySQL is a simple inline interface, which consists of two graphical interfaces which are MySQL Control Center and MySQL Administrator, which were developed by MySQL AB to use and manage MySQL.

3- Support various programming languages:
MySQL provides a programming interface for various programming languages to enable you to write database applications in the language of your choice. It supports PHP, Java, c ++, Perl, Python, Tcl and others to give developers maximum freedom in designing applications that rely on MySQL.

Normalisation

The process that allows you to improve the tables in terms of design in order to prevent two things:

- data redundancy.
- data integrity.

First Normal Form

1. Every single cell must have one value
2. Every data in one column must be in same data type
3. Need to have a primary key
4. Repeated group should be removed

Second Normal Form

1. The relation should be in first normal form.
2. Every non-key attribute must depend on the primary key. In other words, partial dependency should be removed.

Third Normal Form

1. The relation should be in second normal form
2. Transitive dependency should be removed

Functional dependency: If columns in the table depend on the primary key, it means that it has the functions of dependency on the primary key.

A ---> B (B depends on A)

Partial dependency: When the relation has more than one primary key and there is an attribute depends only on one of those primary keys.

Total dependency: When the relation has more than one primary key and there is an attribute depends on all primary keys.

Transitive dependency: When there is an attribute depends on another attribute which depends on the primary key.

Ex: A--->B---->C ,C depends on another attribute (B) that depends on the primary key A

Examples

Example 1

Student(sID, sLast, address, city, dateOfBirth, courseID, courseName, grade, facultyID, facultyLast)

1st NF: Tell courseID everything is fixed with no changes, after that there will be changes in the values. So, repeated should be transferred to another separated table in this step. In the new table we should use the primary key from the original table with another suitable primary key from the attributes of the new table.

Student(sID, sLast, address, city, dateOfBirth)

Course&Grade(sID,courseID, courseName, grade, facultyID, facultyLast)

2nd NF: We should remove the partial dependency to a new separated table.

Student(sID, sLast, address, city, dateOfBirth)

no partial dependency in the student table. However, the courseName, facultyID, facultyLast depend only on one primary key which is the courseID. So, we should give them a table with their primary key.

Grade(sID,courseID, grade)

Course(courseID, courseName, facultyID, facultyLast)

3rd NF: We should remove the transitive dependency if any. facultyLast attribute depends on facultyID and facultyID depends on the primary key "courseID". We will choose a suitable primary key for the new table and put it as foreign key in the course table to keep the relationship.

Student(sID, sLast, address, city, dateOfBirth)

Grade(sID, courseID, grade)

Faculty (facultyID, facultyLast)

Course(courseID, courseName, facultyID)

Now, we have 4 tables as a result of the normalisation steps. They are ready to insert, update, and delete with no problems or deletions for other data.

Table Before Normalization

sID	sLast	Address	City	DOB	cID	cName	Grade	fID	fLast
300	Khalid	Sarchnar	Sulaimani	2000	200	COA	+A	102	Mustafa
300	Khalid	Sarchnar	Sulaimani	2000	201	FOD	+B	101	Muhammed
300	Khalid	Sarchnar	Sulaimani	2000	203	OOP	+C	100	Ali
301	Omar	Kazi	Sulaimani	2001	203	OOP	+B	100	Ali
301	Omar	Kazi	Sulaimani	2001	200	COA	+A	102	Mustafa
302	Talal	Mamustaian	Sulaimani	1999	201	FOD	+B	101	Muhammed
302	Talal	Mamustaian	Sulaimani	1999	203	OOP	-B	100	Ali

Tables After Normalization

cID	cName	fID
200	COA	102
201	FOD	101
203	OOP	100

sID	cID	Grade
300	200	+A
300	201	+B
300	203	+C
301	203	+B
301	200	+A
302	201	+B
302	203	-B

fID	flast
100	Ali
101	Muhammed
102	Mustafa

sID	sLast	Address	City	DOB
300	Khalid	Sarchnar	Sulaimani	2000
301	Omar	Kazi	Sulaimani	2001
302	Talal	Mamustaian	Sulaimani	1999

Example 2

Bill (bNO, accountantID, accountantName, customerNo, customerName, address, date, itemNo, itemName, price, quantity)

1stNF: The attributes from bNO till date are fixed. After that if the customer buys several items in the same bill, the values will change from itemNo attribute. So, we should separate that to remove the repeated group. The new table will have two primary keys the same as the previous example.

Bill (bNO, accountantID, accountantName, customerNo, customerName, address, date)

Item (bNO, itemNo, itemName, price, quantity)

2ndNF: Removing the partial dependency in the item table. Only the quantity depends on both of the primary keys.

Bill (bNO, accountantID, accountantName, customerNo, customerName, address, date)

Item_quantity (bNO, itemNo, quantity)

Item (itemNo, itemName, price)

3rd NF: Removing transitive dependency.

bNO ----> accountantID ----> accountantName

bNO ----> customerID ----> customerName, address

Those two will be in new tables with a suitable primary key which will be as foreign key in their previous table "Bill" to keep the relationships.

Bill (bNO, accountantID, customerNo, date)

Item_quantity (bNO, itemNo, quantity)

Item (itemNo, itemName, price)

accountant (accountantID, accountantName)

customer (customerNo, customerName, address)

Hotels Database

Introduction

Database for hotel reservation system. Hotels reservation full system in Kurdistan, this database will store the data about hotels. This database stores multiple users accounts and each user has the ability to add multiple hotels and each hotel will have multiple rooms with options. This Database will present the backend of the website and the main structure of the website, it will be used for storing, Processing and fetching out all the required information from and into the website.

What can any type of system's user do

Users have the ability to:

- Add a hotel and specify the hotel information
- Add Rooms to the registered hotel
- Add Rates
- Add Rates Types
- Add Supported payment methods.

Guest will have the ability to:

- Search and View the list of hotels and the services that they provide
- View the available rooms and prices for each hotel.
- Reserve a room or multiple rooms.
- See the hotels pictures and the amenities that they provide.
- Book and pay either online or through one of the available payment systems, or pay at a hotel according to the hotel policy.
- View hotels locations according to their coordinates through google map.

It have the ability to store multiple companies accounts each company will have same options as the guests permissions plus:

- 5% discount on each reservation.
- Free cancelation service

Admin has the ability to:

- View all the registered Users, Guests, Companies accounts in the database.
- Add, Remove, Active, deactivate users, guests and companies accounts.
- Add, Remove, modify
    - ➢ Facilities options
    - ➢ Amenities Options

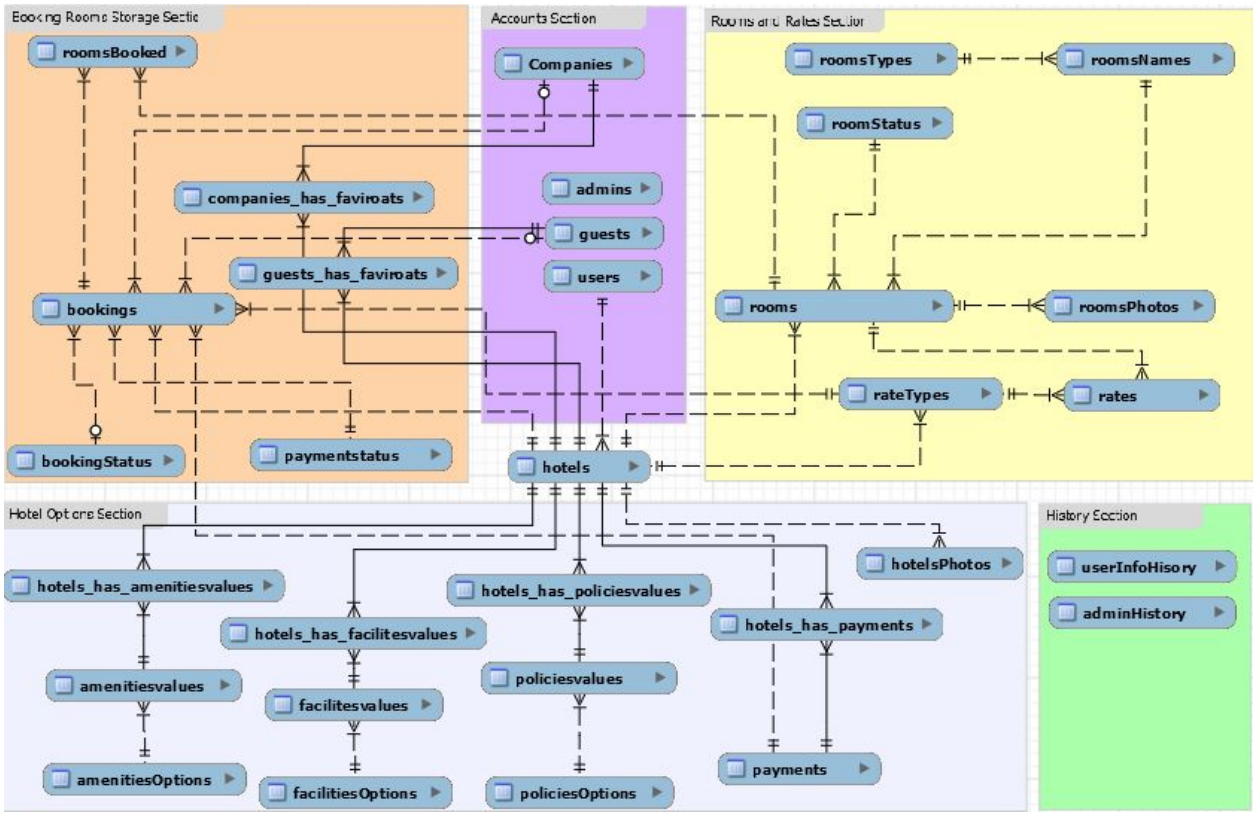
- ➢ Available Payment Options
- ➢ Rooms types and rooms names.

Diagram

ER Diagram to Database schema

The diagram consists of several sections. Each section holds several tables to handle specific tasks.

1. Guest and Users Accounts Section: this section stores the users and guests registration information, users are representing the hotels owners and they will be able to have multiple hotels in the hotels table.
2. Booking rooms Storage Section: this section is used for storing booked rooms for the booking status and specify the booked rooms types.
3. Rooms and Rates Section: Here we store each hotel room's types and specify their kind, storing the state of the rooms as well the prices and rates.
4. Hotel Options Section: For storing each hotel amenities, facilities and policies.
5. Tables History Section: For storing the changes that occurred on the Accounts information table.



Tables

Accounts Section Tables

- **Admin**:to store admins accounts they are able to control the whole database.
- **Users** : to store hotels owners accounts and their information.
- **Guests**: to store guests accounts and their information.
- **Companies**: to store registered companies accounts for reserving for groups.

Table: companies

Columns:

cCompanyID	int(11) AI PK
cOwnerFullName	varchar(45)
cUserName	varchar(45)
cPassword	varchar(45)
cAddress	varchar(45)
cAddress2	varchar(45)
cCity	varchar(45)
cState	varchar(45)
cZipCode	varchar(45)
cPhoneNumber	varchar(45)
cEmail	varchar(45)
cStatus	int(11)
cCreatingDate	timestamp

Table: guests

Columns:

gGuestID	int(11) AI PK
gFullName	varchar(45)
gUserName	varchar(45)
gPassword	varchar(45)
gAddress	varchar(45)
gAddress2	varchar(45)
gCity	varchar(45)
gState	varchar(45)
gZipCode	varchar(45)
gCountry	varchar(45)
gPhoneNumber	varchar(45)
gEmail	varchar(45)
gGender	varchar(45)
gStatus	int(11)
gCreatingDate	timestamp

Table: users

Columns:

uUserID	int(11) AI PK
uUserName	varchar(45)
uUserEmail	varchar(45)
uUserPassword	varchar(45)
uFullName	varchar(255)
uUserPhoneNumber	varchar(45)
uStatus	int(11)
uCreationDate	timestamp

Table: admins

Columns:

aAdminID	int(11) AI PK
aFullName	varchar(45)
aUserName	varchar(45)
aPassword	varchar(45)
aEmail	varchar(45)
aCreatingDate	timestamp

Hotels table

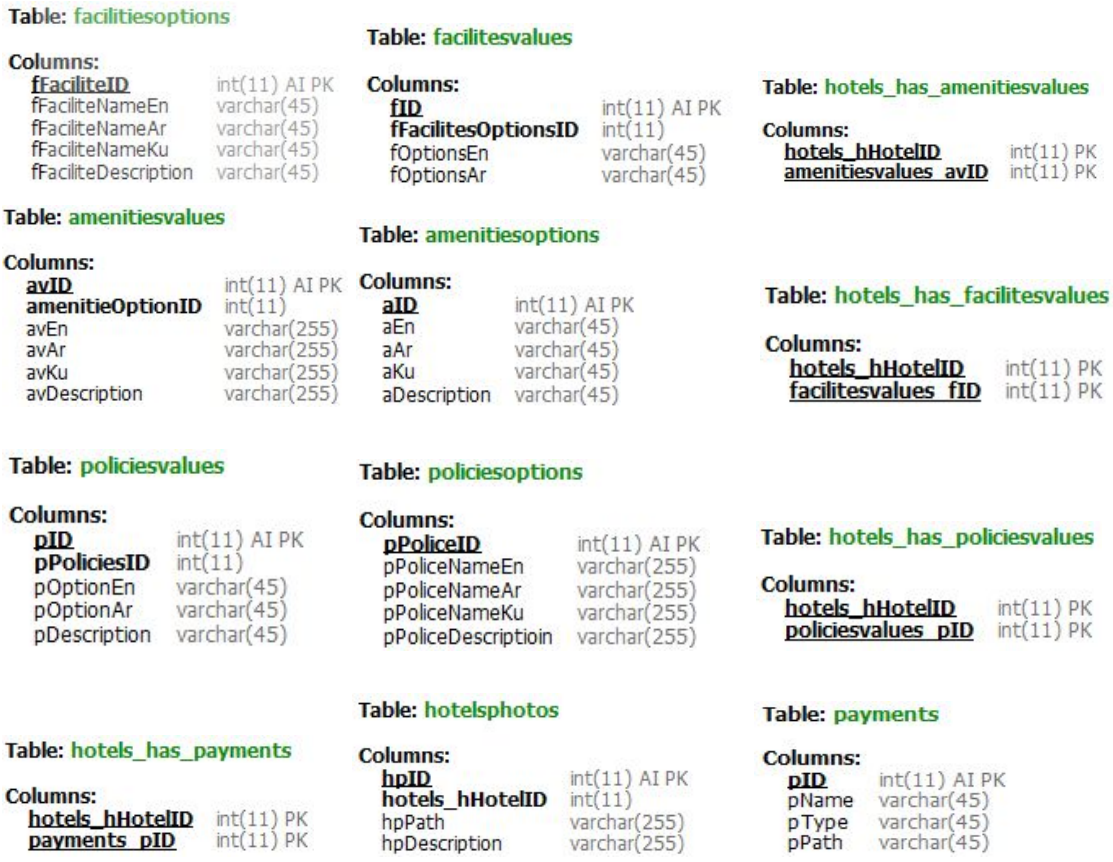
to store the hotels which are added by the users and full information.

Columns:

hHotelID	int(11) AI PK
hUserID	int(11)
hHotelName	varchar(45)
hHotelRating	varchar(45)
hAddress2	varchar(45)
hCity	varchar(45)
hState	varchar(45)
hZipCode	varchar(45)
hMainPhoneNumber	varchar(45)
hNormalNumber	varchar(45)
hCompanyMailAddress	varchar(45)
hWebsiteAddress	varchar(45)
hLogoPath	varchar(45)
hActive	int(11)
hAddress	varchar(45)
hMapsLocation	varchar(255)
hCreatingDate	timestamp

Hotel Options Section Tables

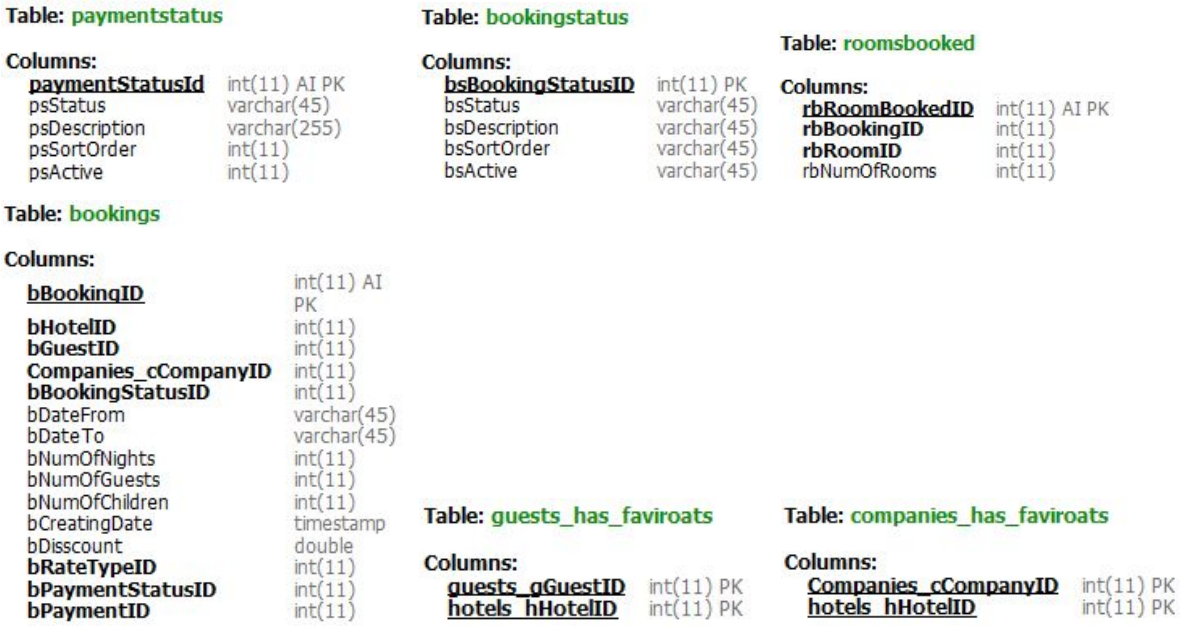
- **AmenitiesOptions**: for storing the standard amenities types options that hotels will use.
- **amenitiesValues**: Stores a list of amenities for each amenity type in amenities options.
- **hotels_has_amenitiesValues**: for soring the amenities for each hotel.
- **FacilitiesOptions**: for storing the standard facilities options which hotels are going to use.
- **facilitesValues**: Stores a list of facilities for each facility type in facilitiesOptions Table.
- **hotels_has_facilitesValues**: for storing the facilities for each hotel.
- **PoliciesOptions**: to store the standard Policies to be assigned to each hotel.
- **policiesValues**: to store the values of each policy type.
- **hotels_has_policiesValues**:for string each hotel policies.
- **Payments**: for storing the supported payments in Iraq that hotels are going to use for getting paid from guests.
- **hotels_has_payments**: for storing the supported payment methods for each hotel.
- **hotelsPhotos**:for storing each hotel photo's paths.



Booking Rooms & Storage Section Tables

- **Bookings**: to store all bookings which have been done by guests and companies.
- **RoomsBooked**: declare which rooms have been reserved along with hotels (For storing both booked hotel ID and the room ID.
- **BookingStatus**: to store the statues types for the booking.
- **Paymentstatus**: for storing the status of the payment of the customer if it's paid, not paid, fail to read credit card etc.
- **Companies_has_faviroats**: to store the favorite hotels added by companies.
- **guests_has_faviroats** : to store the favorite hotels added by guests.

Note: when new bookings have been added the payment status by default is '1' which stands for not paid.



Rooms and Rates Section tables

- **Rooms**: for holding and storing all the rooms added by each hotel.
- **RoomsPhotos**: for storing rooms photos paths.
- **RoomsTypes**: for storing the standard rooms types.
- **RoomsNames**: for storing the rooms kinds for each room type.
- **RoomsStatus**: to store the statues types determined by admin.
- **Rates**: for storing available rates for each room.
- **RateTypes**: for storing the standard rates determined by the users.

Table: rates

Columns:

rRateID	int(11) AI PK
rRoomsID	int(11)
rRateTypeID	int(11)
rRate	double
rDate	date
rARoomsNum	int(11)
rAddingDate	timestamp

Table: ratetypes

Columns:

rtRateTypeID	int(11) PK
hotels_hHotelID	int(11)
rtRateType	varchar(45)
rtDescription	varchar(45)
rtSortOrder	varchar(45)
rtActive	int(11)

Table: roomstatus

Columns:

rsRoomStatusID	int(11) PK
rsRoomStatus	varchar(45)
rsDescription	varchar(45)
rsSortOrder	varchar(45)
rsActive	varchar(45)

Table: roomsnames

Columns:

rnRoomNameID	int(11) PK
rnRoomNameEn	varchar(255)
rnRoomNameDescription	varchar(255)
rnRoomNameAr	varchar(255)
rnRoomNameKu	varchar(255)
rnRoomTypeID	int(11)

Table: roomstypes

Columns:

rtRoomTypeID	int(11) PK
rtRoomTypeEn	varchar(45)
rRoomTypeAr	varchar(45)
rRoomTypeKu	varchar(45)
rtDescription	varchar(45)
rtSortOrder	varchar(45)
trActive	varchar(45)

Table: roomsphotos

Columns:

rpPhotoID	int(11) AI PK
rpRoomsID	int(11)
rpPhotoPath	varchar(45)
rpPhotoDescription	varchar(45)

Table: rooms

Columns:

rRoomID	int(11) AI PK
rHotelID	int(11)
rRoomStatusID	int(11)
rRoomNameID	int(11)
rMaxPas	int(11)
rFloor	int(11)
rRoomsNumbers	int(11)
rDescription	varchar(45)
rCustomName	varchar(45)
rSmokingPolicy	int(11)
rRoomSize	double
rLowestPrice	double
rOfferLower	int(11)
rDisscount	double
rCreatingDate	timestamp

History Section tables

- **UserInfoHistory**: for storing the history of changes happened on the user account.
- **AdminHistory**: for storing the history of the operations that have been made by admins.

Table: userinfohisory

Columns:

tnID	int(11) AI PK
tnUserName	varchar(45)
tnPassword	varchar(45)
tnEmail	varchar(45)
tnFullName	varchar(45)
tnPhoneNumber	varchar(45)
tnState	varchar(45)
tnAtTime	varchar(45)

Table: adminhistory

Columns:

ahID	int(11) AI PK
ahFullName	varchar(45)
ahUserName	varchar(45)
ahPassword	varchar(45)
ahEmail	varchar(45)
ahAtTime	varchar(45)

Relationships

Many to Many relationships ---->Amenities Relationship with hotel:

aminites Options Table

aID	aEn
1	most_requested
2	Room amenities

aminiteies list Table

avID	amenitieOptionID	avEn
1	1	Air conditioning
2	1	Bath
3	1	Spa Bath
4	1	Flat-screen TV
5	1	Electric kettle
6	1	Balcony
7	1	View
8	1	Terrace
9	2	Clothes rack
10	2	Drying rack for clothing
11	2	Fold-up bed
12	2	Sofa bed
13	2	Air conditioning
14	2	Wardrobe or closet
15	2	Carpeted
16	2	Dressing Room
17	2	Extra Long Beds (> 2 metres)
18	2	Fan
19	2	Fireplace

hotels_has_amenitiesvalues

hotels_hHotelID	amenitiesvalues_avID
1	3

<-- Many- to-Many -->

HotelsTable

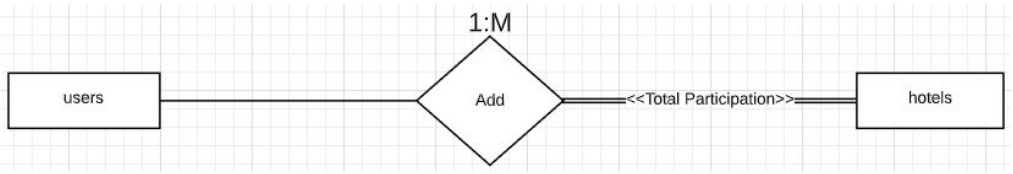
Hotel ID	Hotel Name
1	Avenue
2	Titanic

HotelID (1) => aminiteyOption(1) => aminiteyValue(3)

Avenue > Most Requisted >Spa Bath

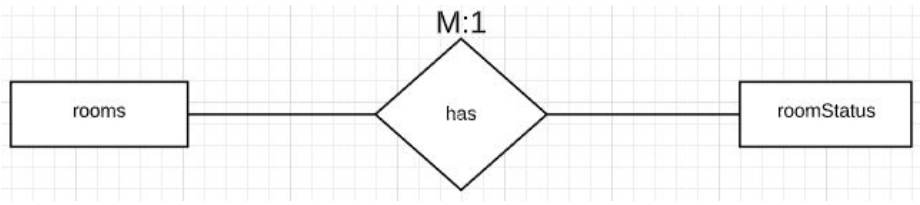
One to Many relationship-----> Users with the hotels

The user “the owner of the hotel” can add many hotels, and the hotel or hotels will be related to a specific user. Each hotel must be related to an owner. That’s why it is total participation.



Many to One relationship-----> rooms with roomStatus

Each room will have one status like “closed, available, or reserved” and the status like “reserved” may be related to many rooms.



Note: The other relationships are declared in the diagram.

Queries

Query will be executed by the Admin

1- Get a Report of all guests names, their booking ID, types of rooms, Dates of bookings, Number of nights they stay and number of passengers:

```
SELECT g.gFullName as 'Guest full Name', b.bBookingID as 'Booking ID', rt.rtRoomTypeEn as 'Room Type', rn.rnRoomNameEn as 'Room Kind'
, b.bDateFrom as 'From Date', b.bDateTo as 'To Date', b.bNumOfNights as 'Number of nights to stay',
b.bNumOfGuests as 'Number of adults', b.bNumOfChildren as 'Number of children' FROM bookings b
JOIN roomsbooked rb ON b.bBookingID = rb.rbBookingID
JOIN rooms r ON rb.rbRoomID = r.rRoomID
JOIN roomsnames rn ON r.rRoomNameID = rn.rnRoomNameID
JOIN roomstypes rt ON rt.rtRoomTypeID = rn.rnRoomTypeID
JOIN guests g ON g.gGuestID = b.bGuestID
WHERE b.bHotelID = 1
AND b.bDateFrom >= '2020-05-19'
AND b.bDateFrom <= '2020-05-31';
```

Explanation:
Here we used the inner join between the tables that have common attributes. To do so, we have to rename the tables to differentiate the common attribute to which table is related. Then, specific attributes will be selected from several tables according to specific conditions after the "where" clause.

2- To show the number of hotels in each city:

```
  SELECT
    `hotels`.`hCity` AS `City`,
   COUNT(`hotels`.`hCity`) AS `Number of hotels`
  FROM
    `hotels`
  GROUP BY `hotels`.`hCity`
```

Explanation:

Here we selected the cities from the hotels table to be listed all one by one. Then, we used the built in function "count()" to count all the cities listed before. Then, by using "group by", we specified the number counted for each city. According to the number of times each city is repeated, it will be the number of hotels in it.

City
Sulaymaniyah
Sulaymaniyah
Sulaymaniyah
Sulaymaniyah
Sulaymaniyah
Sulaymaniyah
Sulaymaniyah
Erbil
Erbil
Erbil
Erbil
Erbil

City	Number of hotels
Sulaymaniyah	12

City	Number of hotels
Erbil	5
Sulaymaniyah	7

Query will be executed by the Users

1- To view the list of the hotels this user have:

Using SQL:

```
SELECT * FROM `m&a_hotels`.hotels WHERE hUserID= 1;
```

Algebra**:**

σ{hUserID = 1}(hotels)

Explanation:

Here we used the "*" to show all the attributes related to specific hotels according to the user ID.

2- To view a list of the rooms types for the hotel that have been created :

Using SQL:

```
SELECT rt.rtRoomTypeEn, rn.rnRoomNameEn FROM `m&a_hotels`.rooms `r`
JOIN roomsnames `rn` ON r.rRoomNameID = rn.rnRoomNameID
JOIN roomstypes `rt` ON rn.rnRoomTypeID = rt.rtRoomTypeID
WHERE r.rHotelID = "1";
```

Using Algebra:

π rt.rtRoomTypeEn, rn.rnRoomNameEn σ r.rHotelID = 1 ρ r rooms ⋈ r.rRoomNameID = rn.rnRoomNameID ρ rn roomsnames ⋈ rn.rnRoomNameID = rt.rtRoomTypeID ρ rt roomstypes

Explanation:

Here we depend on the rooms table to specify the ID of the hotel that we want to view its room types and names. We used the inner join between rooms and roomsnames tables because there is a common attribute between them. Then, we can get the actual room name. We did the same between roomstype and roomsnames tables to get the actual room type. All according to a specific condition using the "where" clause to get these info of a specific hotel only.

rtRoomTypeEn	rnRoomNameEn
Single	Budget Single Room
Single	Single Room with Sea View
Double	Budget Double Room
Double	Deluxe Double Room (1 adult + 2 children)

Queries which will be executed by Guests & Companies

1- View all hotels and their rooms in specific city eg.sulaymaniyah:

Using SQL:

SELECT h.hHotelName,rt.rtRoomTypeEn,rn.rnRoomNameEn, r.rCustomName, rs.rRate, rs.rDate FROM `m&a_hotels`.hotels `h` JOIN rooms `r` ON h.hHotelID = r.rHotelID JOIN roomsnames `rn` ON r.rRoomNameID = rn.rnRoomNameID JOIN roomstypes `rt` ON rn.rnRoomTypeID = rt.rtRoomTypeID JOIN rates `rs` ON r.rRoomID = rs.rRoomsID WHERE h.hCity LIKE "Sulaymaniyah"

Using Algebra:

π h.hHotelName,rt.rtRoomTypeEn,rn.rnRoomNameEn, r.rCustomName, rs.rRate, rs.rDate σ h.hCity LIKE 'Sulaymaniyah' ρ h hotels ⋈ h.hHotelID = r.rHotelID ρ r rooms ⋈ r.rRoomNameID = rn.rnRoomNameID ρ rn roomsnames ⋈ rn.rnRoomTypeID = rt.rtRoomTypeID ρ rt roomstypes ⋈ r.rRoomID = rs.rRoomsID ρ rs rates

hHotelName	rtRoomTypeEn	rnRoomNameEn	rCustomName	rRate	rDate
Avenue	Single	Deluxe Single Room	custom Name	46	2020-05-27
Avenue	Single	Deluxe Single Room	custom Name	45	2020-05-28

Explanation:
We used the same concept, inner join with the tables that have common attributes and by giving a short name for each table, to relate the column to its table because we have common attributes with the same names. All of that according to the condition using the “Where clause” and by using the LIKE keyword to ignore the case.

2- Reserving a room:

Using SQL:

Using procedure

```
-- adding bookings

-- Using addReservation procedure

--  (hotelID INT, RoomID INT, guestID INT, rateTypeID INT, dateFrom DATE, dateTo DATE,numOfRooms INT,psNum INT,ChildrenNum INT, bPaymentID INT)

CALL addReservation ('1','1','3','1','2020-06-9','2020-06-11', '1', '2', '1', '1');

CALL addReservation ('2','2','4','1','2020-06-9','2020-06-11', '1', '2', '1', '1');

CALL addReservation ('2','3','5','1','2020-06-9','2020-06-11', '1', '2', '1', '2');

CALL addReservation ('2','4','4','1','2020-06-9','2020-06-11', '1', '2', '1', '2');
```

Explanation:
Here we used the procedure which is done in the procedures section

3- View bookings for the current guest (guest ID:1):

Using SQL:

```
SELECT b.bBookingID as 'Booking ID', h.hHotelName as 'Hotel Name', g.gFullName as 'Guest Name', b.bDateFrom as 'From Date',
b.bDateTo as 'To Date', b.bNumOfNights as 'Number of Nights', rn.rnRoomNameEn as 'Room Type', rb.rbRate as 'Booking Price',
bs.bsStatus as 'Booking Status' FROM bookings b
JOIN roomsbooked rb ON b.bBookingID = rb.rbBookingID
JOIN rooms r ON r.rRoomID = rb.rbRoomID
JOIN roomsnames rn ON r.rRoomNameID = rn.rnRoomNameID
JOIN rates rs ON rs.rRoomsID = r.rRoomID
JOIN hotels h ON b.bHotelID = h.hHotelID
JOIN guests g ON b.bGuestID = g.gGuestID
JOIN bookingstatus bs ON bs.bsBookingStatusID = b.bBookingStatusID
WHERE b.bGuestID = 1;
```

Eg. Output:

Booking ID	Hotel Name	Guest Name	From Date	To Date	Number of Nights	Room Type	Booking Price	Booking Status
1	Avenue	Mohammed salahadin Hazim	2020-05-25	2020-05-28	3	Deluxe Single Room	50	Pending
1	Avenue	Mohammed salahadin Hazim	2020-05-25	2020-05-28	3	Deluxe Single Room	50	Pending

Functions

1- Find numbers of bookings for a specific hotel:

```
-- Get Number of bookings for a specific hotel using it's ID
CREATE FUNCTION get_hotel_bookings(hotelID int)
RETURNS INT deterministic
return (SELECT COUNT(b.bHotelID) FROM bookings b WHERE b.bHotelID = hotelID);
```

Triggers

1.When inserting new row to the admin,user,guest and companies accounts change the user letters from Uppercase to lowercase

```
-- for Change users to lowercase
DROP TRIGGER IF EXISTS userToUpper;
CREATE TRIGGER userToUpper
BEFORE INSERT ON users
FOR EACH ROW
SET NEW.uUserName =lower(new.uUserName);

-- for Change Admins to lowercase
DROP TRIGGER IF EXISTS adminToUpper;
CREATE TRIGGER adminToUpper
BEFORE INSERT ON admins
FOR EACH ROW
SET NEW.aUserName =lower(new.aUserName);


-- for Change guests to Lowercase
DROP TRIGGER IF EXISTS guestToUpper;
CREATE TRIGGER guestToUpper
BEFORE INSERT ON guests
FOR EACH ROW
SET NEW.gUserName =lower(new.gUserName);
```

2. Insert the number of nights on each booking according to the reservations dates:

```
DROP TRIGGER IF EXISTS insertNights;
CREATE TRIGGER insertNights
AFTER INSERT ON bookings
FOR EACH ROW
SET NEW.bNumOfNights = DATEDIFF(NEW.bDateTo, NEW.bDateFrom);
```

Procedures

1- Procedure for decreasing from the number of rooms each time we add new reservation:

```
-- To decrase the number of rooms each time a customer reserves
DROP PROCEDURE IF EXISTS decreserooms;
DELIMITER $$
CREATE PROCEDURE decreserooms(IN dateStart DATE, IN dateEnd DATE,IN roomID INT, IN
numOfRooms INT)
BEGIN
        WHILE dateStart <= dateEnd DO
          UPDATE `m&a_hotels`.`rates` SET `rARoomsNum` = `rARoomsNum` - numOfRooms
          WHERE (`rRoomsID` = roomID and `rDate` = dateStart);
          SET dateStart = date_add(dateStart, INTERVAL 1 DAY);
         END WHILE;
END$$
DELIMITER ;
-- CALL decreserooms('2021-01-01','2022-12-31',1, 5);
-- will be called from inside addReservation procedure
```

2- Procedure for inserting new reservations to the database

Note (the booking status is pending by default)

```
        DROP PROCEDURE IF EXISTS addReservation;
        -- Procedure for inserting new reservations to the database Note (the booking status is pending by
        default
        DELIMITER $$
        CREATE PROCEDURE addReservation (IN hotelID INT,IN RoomID INT, IN guestID INT,IN
        rateTypeID INT, IN dateFrom DATE, IN dateTo DATE,IN numOfRooms INT, IN psNum INT,
        IN ChildrenNum INT, ,bPaymentID INT)
        BEGIN
                DECLARE bookingID INT;
                INSERT INTO `m&a_hotels`.`bookings` (`bHotelID`, `bGuestID`,`bRateTypeID`,
                `bDateFrom`, `bDateTo`,  `bNumOfGuests`, `bNumOfChildren`, `bPaymentID`)
                VALUES (hotelID, guestID,rateTypeID, dateFrom, dateTo, psNum, ChildrenNum,
                bPaymentID);
```

```
        -- to get the booking id from the bookings table to insert it to the roomsBooked table
        SET bookingID = LAST_INSERT_ID();
        INSERT INTO `m&a_hotels`.`roomsbooked` (`rbBookingID`, `rbRoomID`,
        `rbNumOfRooms`) VALUES (bookingID, RoomID, numOfRooms);
        -- to decrase the reserved rooms from the
        CALL decreserooms(dateFrom,dateTo,RoomID, numOfRooms);
END$$
DELIMITER ;

-- For testing the procedure:
--  CALL addReservation ('1','1','1','1','2020-05-29','2020-06-01', '2', '3', '1');
```

Views

1- Create a view to view the list of hotels who have available rooms from now and in the future:

```
CREATE OR REPLACE VIEW available_hotels AS
SELECT * FROM hotels h
JOIN rooms r ON h.hHotelID = r.rHotelID
JOIN rates rs ON r.rRoomID = rs.rRoomsID
WHERE rs.rARoomsNum > 0 AND rs.rDate >= now()
```

Note: in order for this view to work you have to make sure you have bookings in rates table for feature dates.

The outcome of this database is fully managed data and it is:

- Easy to fetch and select using the join techniques.
- Takes less storage to store huge data.
- Takes less time to execute the queries.
- Effective multiple tasks in one command.
- Manageable easily.
- Flexible data input and output.
- Easy maintenance.

Looking forward to updating it and adding some features to it by giving the guest the ability to search for specific things related to the hotels. As well as, putting some constraints on the errors that may the user make through using the service, like inputting wrong dates in the fields of the “date from” and “date to”.

Conclusion

In conclusion, by following all these steps and rules by the designer, the database will be designed logically. Then, the database administrator will implement it correctly to get an excellent database as a result. However, it will not be the last step for the Designer and DBA because after use from the user, there is a possibility to get an error that should be resolved by the DBA. Or, they may ask again for other different requirements that need the designer to make analysis and design to be implemented by the DBA again.

References

1. https://www.w3schools.com
2. https://stackoverflow.com
3. Ramez Elmasri and Shamkant B Navathe. 2015. Fundamentals of database systems.
4. https://www.youtube.com/channel/UCs9QB-ccbNQDWvIQIu_RWlQ
5. https://www.youtube.com/user/alxs1aa