Contents

1.0 - Part I

This sections outlines at a high level the universe of discourse (business scenario/organisation) for which a relational database is being developed, as well as justifying the need for a relational database in this situation and describing the user architecture of the database system (how users will interact with the system).

1.1 - Universe of Discourse

For this assignment, the proposed organisation is a hypothetical website (name pending) which allows users to compare offerings for hotel rooms in Australian cities and lodge bookings. It is an open platform to which any hotel owner can register and post information about their rooms and availabilities, increasing their exposure to customers.

For hotel owners, they will be capable of:

- Posting information about their hotel including contact information and the address and city it is located in
- Post the associated details of their hotel including a textual description of the hotel, the hotel's star rating and the facilities and services which the hotel offers (Wifi, a pool, whether or not they allow pets, breakfast and so on)
- Post details of the rooms available in their hotel, including the number of beds, the type of room (penthouse etc.) and whether it has a kitchen or television
- Post details about the availability of their rooms (for what periods they are available and at what nightly rate). This allows for hotel owners to specify different rates for certain periods (eg. higher rates over christmas holidays)
- Viewing the details of bookings made by customers for their hotel, as well as generating data based on these bookings (such as how many guests will be staying over a specified period and the expected gross income calculated from the nightly rates of the rooms booked)
- View reviews left about their hotels by customers
- Viewing the contact details of customers who have made bookings
- Creating coupon codes which customers can enter when making a booking for discounts or other benefits

Customers will be able to search for available hotel rooms in a given city over a given date range, allowing them to further refine their search based on various criteria about the hotel's facilities.

Customers will be capable of:

- Entering their own contact details, including their name, phone number, an email address and their home address
- Viewing the available rooms in a particular city over a specified period
- Viewing the available rooms in a particular city over a specified period, further filtered based on the hotels facilities (do they offer breakfast, do they have a gym etc.)
- Viewing the available rooms for a particular hotel
- Leaving a review for a hotel
- Make a booking for an available hotel room (optionally using a coupon they may have)

- Cancel a booking they have made
- View the status of their bookings
- View the details for a particular hotel

1.2 - Justification for Database

This section outlines why a relational database is the most suitable solution to building the aforementioned service along various criteria

To provide concurrent, distributed access to large amounts of data

Potentially thousands of users may be accessing the proposed website simultaneously, all searching through the database for hotel offerings, making bookings and changing their details. A DBMS using a relational database in a client-server architecture provides a powerful tool for customers (and hotel owners) to interface with the system simultaneously while still maintaining the integrity of the data

To provide a uniform, logical model for representing data

By providing a suitable relational data model of information about hotel rooms and availabilities, customers are able to easily find hotel rooms based on arbitrary criteria - a user may just want to find any hotel rooms in a city on a given date range, or they may want to find specifically a 5* hotel with a gym, pool and wifi but which doesn't allow pets. A relational data model is powerful in that it can handle these just as easily as each other.

To provide a powerful, uniform language for querying and updating data

By providing a uniform language for querying the data in the database, accessing the database can be extended to a variety of platforms easily (the same database can function via a web browser but also a mobile application, written in totally different languages).

To allow powerful optimisations for efficient query evaluation

Since many users may be querying the system simultaneously it is very important that queries run on the data are as efficient as possible and take little time to execute. It also ensure that users feel the service is responsive and receive results in a timely manner. The query optimiser of a typical dbms is a powerful tool for accomplishing this.

To ensure data integrity within single applications

The constraint checking offered by a relational database ensures that users are not able to enter erroneous data, even if by mistake (eg. users cannot make a booking for a date that has already passed, users must post a valid phone number, hotel owners cannot accidentally overlap availability periods causing a conflict). This helps to ensure the integrity and validity of the data

To ensure data integrity across multiple concurrent applications

Similarly, a dbms with proper concurrency control and transactional logging prevents the database state from being altered erroneously in between operations and ensures that customers see correct data (for instance multiple customers may try to book a room at the same time and proper

transactional logging ensures that the room will not accidentally be double booked. Similarly it ensure that the booked state of the room is properly reflected to successive users)

Potential for operational data to be the source of analytics

From the information in the database alone, hotel owners can calculate useful information such as the number of guests and expected income over a specified date period based on the bookings made. There is also the potential for more advanced analysis of data - for example:
-during which periods were the highest volume of bookings?
-which rate ranges attract the most companies?
-which were the most popular destinations?

1.3 - Proposed Implementation

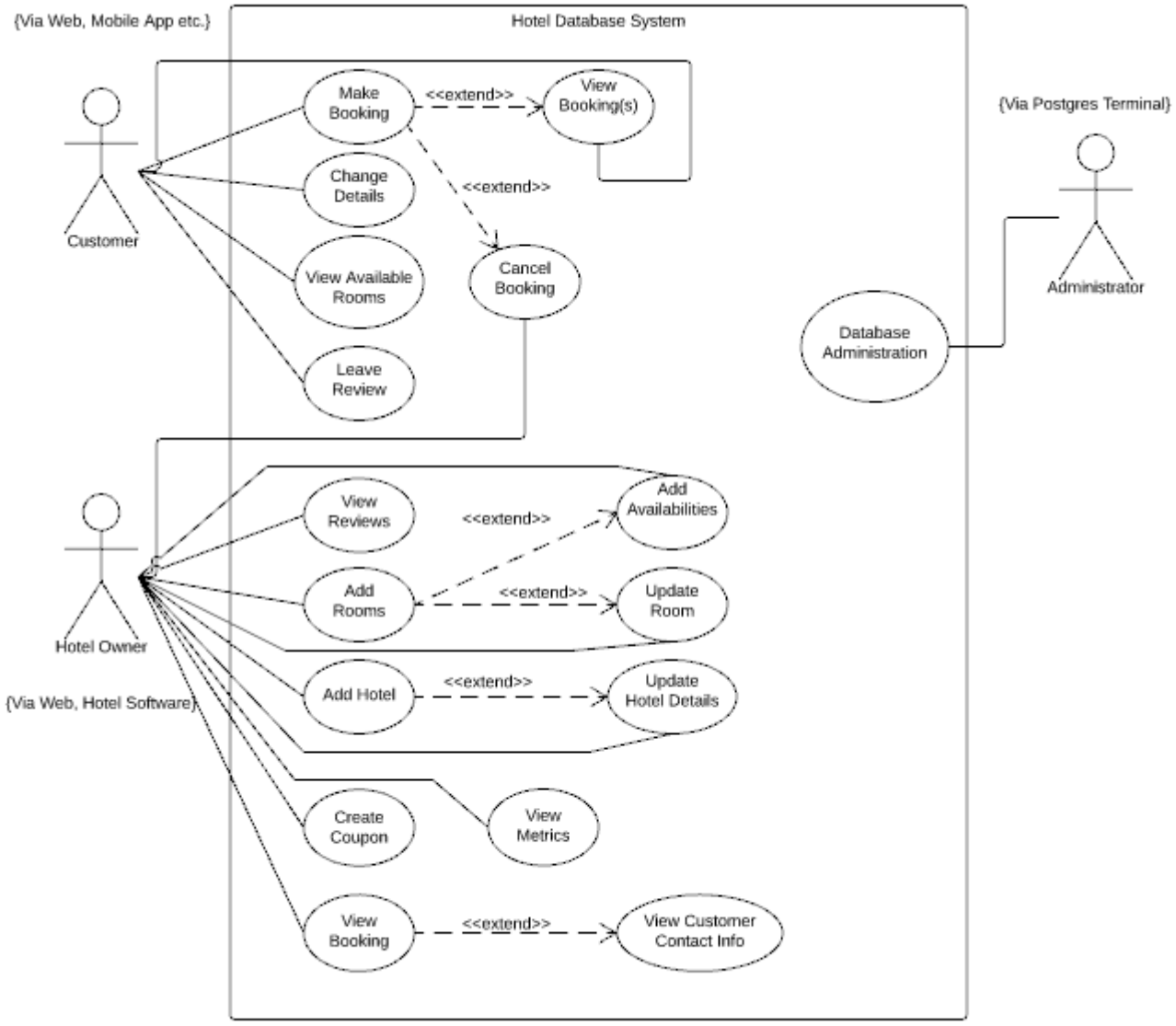
The proposed relational database is likely to be implemented as a PostgreSQL server accessed by users indirectly through some outward facing application (eg. a mobile app, computer software or website).

PostgreSQL (Postgres or PG for short) is a powerful open-source DBMS (Database Management System) which provides a suite of tools for creating a database and managing concurrent access to that database, as well as providing an outward facing API for other software to interact with the database. Postgres provides a rich suite of features which will be useful in implementing the described relational database system including notably

- Majority SQL support for the SQL:2011 standard
- Full support for joins, primary keys and foreign key referential integrity constraints as well as type checking and domain constraints
- Support for a large range of data types, including dates
- Fully transactional concurrency control and ACID compliant
- Full serializability
- Support for recursive, dynamic and materialised views
- Support for a wide range of indexes, including B+ Trees and Hash indexes
- Extensible procedural language for enforcing advanced business rules with functionality such as cursors, triggers and stored functions
- Programming interfaces for a wide variety of languages including C/C++, Java (JDBC), Python and LUA
- Cross platform OS compatibility (including Linux, Windows and OS X)
- A sophisticated query optimiser
- Write-ahead transactional logging for fault tolerance and recovery from crashing

Postgres is a widely used, enterprise grade DBMS that easily scales to large, web-facing applications with many concurrent users. Based on this, it is a well-suited choice for the proposed system

A UML use case diagram for the three different user roles is shown below

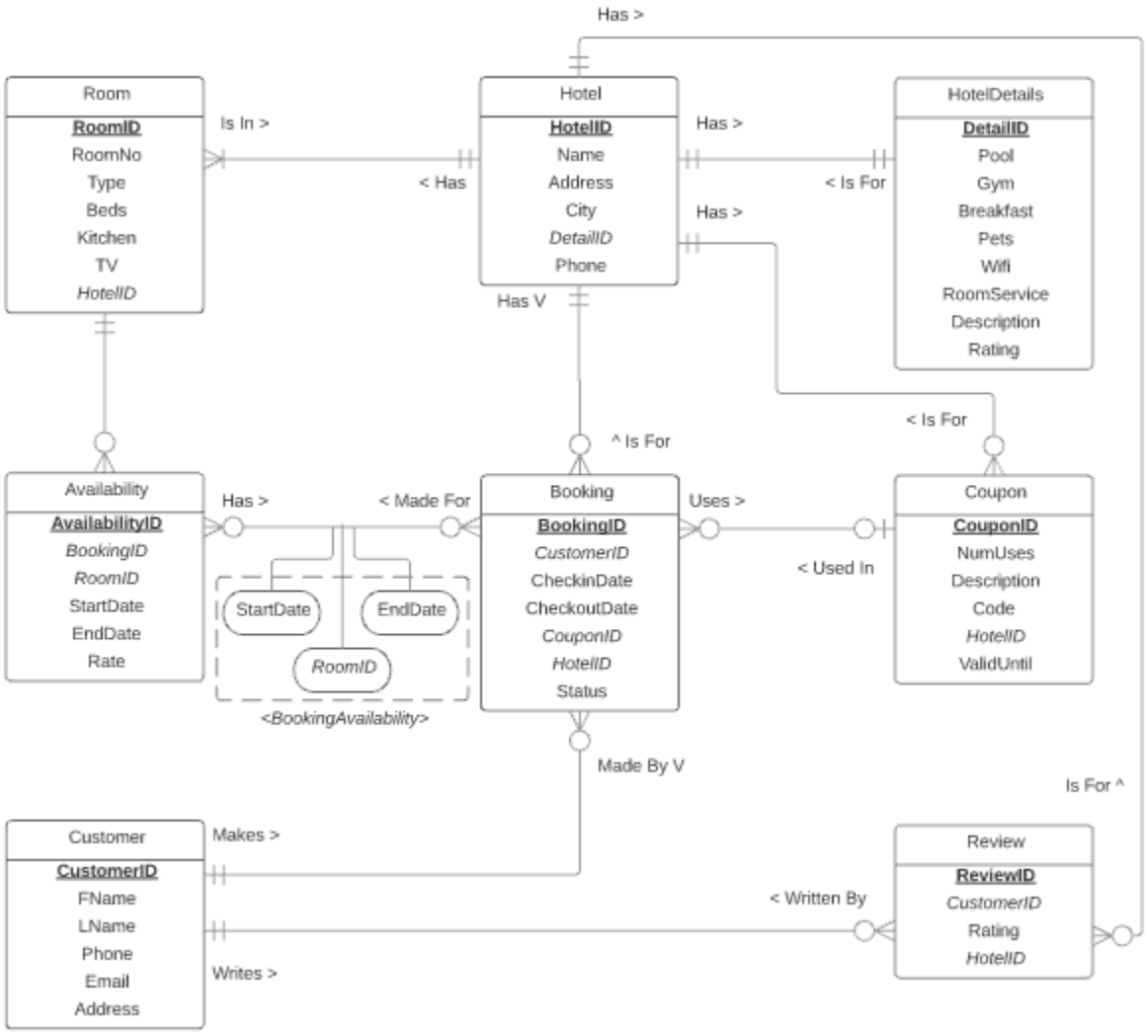


Notably, the system doesn't handle payments or store any sort of payment information. In practice, this would be done by some external payment system (and when payment is accepted, the corresponding booking status set to reflect that it has been paid).

2.0 - Part II

2.1 - Proposed Entity-Relationship Model

The proposed ER model for the hotel bookings website is shown below. In this Diagram, Primary Keys are represented as **bold and underlined** and foreign keys (with the same name as the primary key of another entity's primary key) are *italicised*.



A brief description of the entities in the above diagram

- Hotel - A hotel with several rooms which a user may book. Has basic information about the hotel like its address, the city it is located in and contact details
- HotelDetails - Advanced Details about a given hotel that a customer can use to select a hotel (does it have a pool, does it have wifi, does it allow pets etc.)
- Customer - Details about a customer making hotel bookings
- Review - a review for a particular hotel left by a customer
- Room - details about a particular room at a given hotel (what type is the room, how many beds does it have, does it have a kitchen etc.)
- Availability - details about when a room is made available *by the hotel*. Includes the start and end date of the availability, and the nightly rate of the room during that time

- Booking - details about a booking made by a customer (including the check in and checkout dates and which coupon was used with the booking, if any and the status of the booking ('pending', 'paid', 'cancelled' etc.))
- Coupon - a coupon for a specific hotel. Stores details about the coupon code the user has to enter, the date the coupon is valid until, how many times a coupon can be used and a description of the coupon's benefit (eg. "complimentary breakfast", "free wifi" etc.)

2.2 - Queries, Constraints, User Roles

This section outlines in greater detail the anticipated functions and constraints of the database

2.2.1 - User Roles

There are three main user roles for the proposed database, with only 2 of interest (Hotel Owner and Customer)

Administrator:
Anyone responsible for the administration and maintenance of the database. In practical terms, anybody performing this role will have access to the database itself and interact with it as a superuser via a terminal or similar, so no special consideration or permissions are required

Hotel Owner:

The owner of a hotel who manages details about their hotel and room availabilities in the database system. The hotel owner should be able to manage hotel details, room availability and view details about bookings which have been made

Customer:

The customer searching for and booking hotel rooms. The customer should be able to enter their contact details and search available hotel rooms based on a variety of criteria (such as those specified in the hotel details: 'does it have a pool', 'does it have wifi' etc.). Customers should also be able to make a booking for available hotel rooms and leave reviews for a hotel

2.2.2 - Typical Queries

This section gives an idea of the typical queries expected to be run by a given user role

Hotel Owner

- View all bookings (for a given hotel) over a given period
- View all bookings for a particular room over a given period
- Calculate the number of customers for the hotel over a given period (i.e the number of people with bookings)
- Calculate the expected income over a given period, based on room rates and bookings
- View the contact information of the customer who made a given booking
- View customer reviews

Customer:

- View all hotels in a particular city
- View all available rooms in a particular city within a specified period (date range)
- View all available rooms in a particular city, filtered by some criteria ('does it have a gym', 'does it offer breakfast' etc.) within a specified period (date range)
- View information about bookings they have made
- View the contact information for a given hotel
- View customer reviews for a given hotel

2.2.3 - Functions

This section outlines the functions of the two main user roles (functions which will modify the state of the database). Transactional operations are those which are likely to require multiple read/writes and need to be encapsulated in an atomic 'transaction' to ensure data integrity

Hotel Owners

Non-Transactional Operations:

- Update hotel information (contact number, etc.)
- Update hotel details (availability of wifi, etc.)
- Update room details (number of beds etc.)
- Add new rooms
- Add new coupons

Transactional Operations:

- Add a room availability (this will require comparison to existing availabilities to ensure no overlapping before inserting the availability into the database)
- Cancelling a booking (this will finding which "bookingavailability" entries correspond to the given entry and deleting them, as well as setting the status field of the booking record)
- Updating or deleting an availability - this is a complex, cascading delete. If an availability is removed, then all "bookingavailability" entries corresponding to that availability must be deleted and all corresponding booking statuses changed (i.e if a room is made unavailable for a period, all bookings for that period should be automatically cancelled).

  Similarly, if an availability is updated (i.e made to finish at an earlier date), the "bookingavailability" entries must be examined to determine which bookings are no longer valid (have a start or end date which no longer falls inside an available period) and automatically cancelled.

  This must be encapsulated as a singular transaction otherwise data anomalies could become prevalent, causing conflicting bookings, lost booking information and so on.

Customers:

Non-transactional Operations:

- Updating customer details (contact number, address etc.)
- Leaving a review
- Updating a review

Transactional Operations:

- Making a booking (this requires a booking entry to be made as well as *at least* one entry in the bookingavailability table, as well as searching availabilities and potentially coupons)
- Cancelling a booking (this requires a lookup of the bookingavailability table to delete all the entries corresponding to the booking, as well as changing the booking status field)

2.2.4 - Constraints and Business Rules

A list of the constraints and business rules imposed on the system. This is split into 2 categories - basic constraints which can be modelled in the data definition language (SQL) and are represented in the ER diagram, and complex constraints which require advanced logic or handling to ensure integrity

Basic Constraints:

- A customer can enter their details without having to make a booking (a customer record can exist with no corresponding bookings)
- A hotel entry cannot exist without at least one corresponding room (to enter a hotel in the system it should have at least one room)
- A hotel entry must have exactly one set of hoteldetails
- A room may not have any availabilities (a room can be entered into the system even if it is not made available to book)
- A booking may not have a corresponding coupon (the couponID field of that booking is NULL i.e the customer did not use a coupon when making the booking)
- A booking may correspond to *multiple* availabilities (a customer may book a date range which crosses two or more availability periods)
- An availability period may have several corresponding bookings (several different people may make sequential bookings for a room in a single, larger availability period)
- An availability period can exist without any bookings (a room can be made available to book even if nobody has booked it)
- A booking may exist without a corresponding bookingavailability entry (and hence availability) - that is, information about a cancelled booking is retained
- A customer can only leave one review per hotel

Advanced Constraints

- A booking cannot be made using a coupon if the sum of (non-cancelled) bookings using the same coupon is equal to the number of uses of the coupon, or if the system date is past the coupon's validity date

That is, a new booking entry cannot use a coupon if it would exceed the number of uses for that coupon or the coupon has expired. Cancelled bookings are exempted as even though a record of the booking exists the coupon was not 'used' in this case

- A Booking cannot be made for a given room over a given date range if there exists a bookingavailability entry (i.e non-cancelled Booking) for that room which overlaps the given date range. In other terms - a booking cannot be made if it overlaps with an existing booking

  An exception exists if the entry in question if the entry corresponds to the *same* booking as the one being made. This occurs if the period of the customer's booking overlaps two or more availability periods.

  For example, if a room is available from the 10th to the 21st and the 22nd to the 29th, and a customer wants to book that room from the 18th to the 25th, the following steps (roughly) take place to ensure integrity

  -Get all availabilities for the desired room
  -Check the customer's check in date and find the availability it falls within (if it does not fall within one then the room is unavailable and the booking isn't made)
  -If it is, check all bookingavailability entries which overlap that date range. If there are none, then there is no overlapping booking
  -perform the same checks as the above for the customer's desired checkout date (except checking for all booking. If the checkout date is within a different availability period, then a second bookingavailability entry will be made referencing the different availability period for the same dates

  This means it is possible (and valid) for entries like the following to occur in the above scenario

BookingID	AvailabilityID	StartDate	EndDate	RoomID
1	2	18/05/2016	25/05/2016	101
1	3	18/05/2016	25/05/2016	101
2	3	26/05/2016	29/05/2016	101
3	2	17/05/2016	20/05/2016	102

  Note that the start and end dates refer to the checkin and checkout dates of the booking and not the period of the availability itself

- An Availability for a given room cannot overlap with another availability for that room. Allowing this would cause the possibility for dates where a room has two differing rates which is not allowed

- A booking cannot be made which starts before the current system date
- An availability period cannot be defined which starts before the current system date
- A coupon cannot be made which expires before the current system date
- Hotel owners may only manipulate the information for hotels which they own
- Customers may only manipulate and view their own bookings

2.2 Justification of Design

This section outlines some of the key design decisions for the proposed ER model (in section 2.1)

Entities and Attributes:

- Customer: this entity just encapsulates a single customer and all the attributes are just details about that customer - their name, address etc.
- Hotel: this entity just encapsulates a hotel and it's attributes are just basic details about the hotel - the city it is in, the name, the address etc. It also contains a foreign key referencing a hoteldetails entity
- Hoteldetails: this entity encapsulates various information about a hotels facilities. The attributes are intended to be read as a boolean statement ('wifi' = does this hotel offer wifi? = true or false). It also includes the star rating of the hotel (rating) and a string containing a textual description of the hotel (description).

  These details were included in a separate entity as a way of intending to normalise the database. By separating this information, it doesn't slow queries where it is not useful (eg. the customer wants the address of phone number of a hotel). In this way, these details can be loaded only when necessary (i.e a customer is searching based on the criteria encapsulated by this entity)
- Room: stores information about a particular room at a given hotel. Its attributes include information about the type of room and the facilities offered (does the room have a kitchen or a television).

  Storing rooms as their own entity is useful because rooms within a hotel may vary. A hotel may offer rooms with a differing number of beds or different facilities and in turns those rooms will have different rates. Similarly, rooms will have their own individual schedule of availability which needs to be represented (an individual room may be undergoing maintenance or renovation). Representing rooms as a standalone entity allows these facts to be adequately represented in the database
- Coupon: this entity encapsulates information about a coupon, including the number of times it can be used, the date which it expires and the code the user needs to enter to use the coupon, as well as a description of what the coupon does.

  The reason for just giving the coupon an arbitrary description is that a coupon may have a very arbitrary effect that does not correspond to any information stored in the database. For instance a coupon may be for a free breakfast or free wifi. A coupon may have hypothetically endless 'benefits' so it is not practicable to design the database to store information which is functionally dependent on coupon use

- Review: this entity just encapsulates a review left for a hotel, including the review text, the customerID of the customer who left the review and the hotel the review corresponds to, as well as an overall rating (such as 0-5)

  This allows for the running of typical queries one would expect for the system (eg. a user wants to see all the review for a hotel, or an average of review scores, or a hotel owner will want to see details of bad reviews [below a certain rating])
- Availability: This entity encapsulates information about the availability of a specific hotel room. Specifically, the periods for which it is available and the nightly rate during that period. Availability periods specify a start date and end date, which can be used to determine which bookings fall within this availability period.

  It is assumed that if, for a given date, there is no corresponding availability period that the corresponding room is unavailable for that day. This information can be used when attempting to make a booking (if there is no corresponding availability is found, the room is not available and the booking cannot be made and the database is not altered)

  In general, an availability period could be considered a weak entity (it has no identifying information, and only has meaning in the context of a room and should not be stored otherwise) but in this case it is also given a uniquely identifying availabilityID which can be referenced by bookings

- Booking: this entity encapsulates information about a booking made by a customer. It includes the customerID of the customer who made the booking and the check-in and checkout dates.

  A couponID field is included which references a coupon that the user used, *if* they used a coupon when making the booking. This field can also be used to determine across many bookings how many times a coupon has been used (and determine if making a booking would 'overuse' a coupon)

  Similarly the hotel ID is included. While this information can be functionally determined from the availability (therefore by extension the room and then hotel), storing this information in the booking itself is useful as it saves several joins when a hotel owner wants to query all the bookings within a given date range. In this way storing the information redundantly can save a lot of overhead

  Lastly it includes a descriptive status, which is just a string corresponding to the status of the order (i.e 'pending', 'paid', 'cancelled')

Relationships

- Hotel-Hoteldetails - this is just a total participation relationship as it just represents information which was 'split off' from the hotel entity itself for normalisation. Because of this, every hotel should have exactly one hoteldetails and vice versa
- Hotel-Review - a hotel may have zero or more reviews about it. A hotel may still exist in the system without any corresponding reviews
- Hotel-Room - a hotel may have many rooms, but should have at least one room. It is not meaningful for a hotel to be entered into the system if it is not offering at least one room which may be available to customers at some time
- Hotel-Coupon - a hotel owner may enter many coupons for that hotel which are valid simultaneously. Similarly, a hotel may not have any coupons at all
- Hotel-Booking - This relationship is made possible via the hotelID attribute stored as a field in the booking entity. The reason is it allows for hotel owners to query all the bookings for their hotel without having to involve the room and availability entities to determine which bookings correspond (which has a massive overhead)
- Customer-Review - A customer may leave many reviews, as they may stay at multiple different hotels
- Customer-Booking - A customer may make many bookings, including booking the same room at the same hotel across different dates, or booking two different rooms simultaneously (i.e booking multiple rooms under the same name)
- Room-availability - A room may not have any periods for which it is available, but it is still meaningful to maintain information about the room's existence. Similarly, a room may have many availability periods with different rates and such, essentially forming an availability 'schedule'
- Booking-coupon - A booking may optionally use a coupon, but it is not necessary. A coupon may be used by many bookings as coupons can be multiple use. In the case a booking does use a coupon, only one coupon can be used per booking
- Booking-Availability

  A complicated many-many relationship intended to encapsulate some complicated business rules about hotel rooms

  - A hotel room may have differing rates at differing periods. In this way, rates must correspond to a period and cannot be stored as an attribute of the room
  - A user's booking of a room may not directly correspond to an availability period Because of this:
  - A booking may cross several availability periods and;
  - A given availability period may have multiple sequential bookings within the period
  - Information about a booking is retained even if an availability period is removed or adjusted (so users can see they made a booking which was cancelled)

  Because of this it becomes a many to many relationship with no mandatory participation (though in practical terms this is only a transient condition). It is meant to encapsulate the semantic idea that an availability period may have no bookings corresponding to that period, and similarly a booking record may still exist on its own even if no longer corresponds to an

existing availability period (i.e an availability period was deleted and the booking was updated to reflect that it was cancelled)

The many to many relationship necessitates that an additional table (referred to as "bookingavailability") is made which references as foreign keys an availabilityID and corresponding bookingID.

This is not enough on its own to enforce the described business rules, however, so some additional fields are included. The start date and end date are functionally determined by the check-in and checkout dates (respectively) of the corresponding booking. In this way, entries in this table represent "the days for which a given room is occupied (booked)" vs the availability entity which represents "the days for which a given room is available to be booked"

Including this date information from the booking is important, as it allows new bookings for a room to be checked against these dates to determine whether or not the booking clashes with a time which the room will be occupied.

Storing this information redundantly in the relational table itself, along with the roomID (which can be functionally determined from the availabilityID) saves the overhead of having to join the booking and availability and room tables to determine the same information when checking for booking validity.

Importantly, having the relational table in this manner allows for a single bookingID to correspond to several availabilityIDs, representing the fact that a booking may cross several availability periods. This information can be used then to calculate accurately the rates for a booking given only the booking ID (for each entry in bookingavailability, compare the booking dates to the availability period dates and determine how many days of the booking fall within that period, then multiply that by the nightly rate, and sum the result of this for all bookingavailability entries for the given booking)

The relationship was structured in this way because it was determined as the only way to retain important semantic information about the universe of discourse