Report Deliverable 1

CSI 2132 - Databases 1

Winter 2025

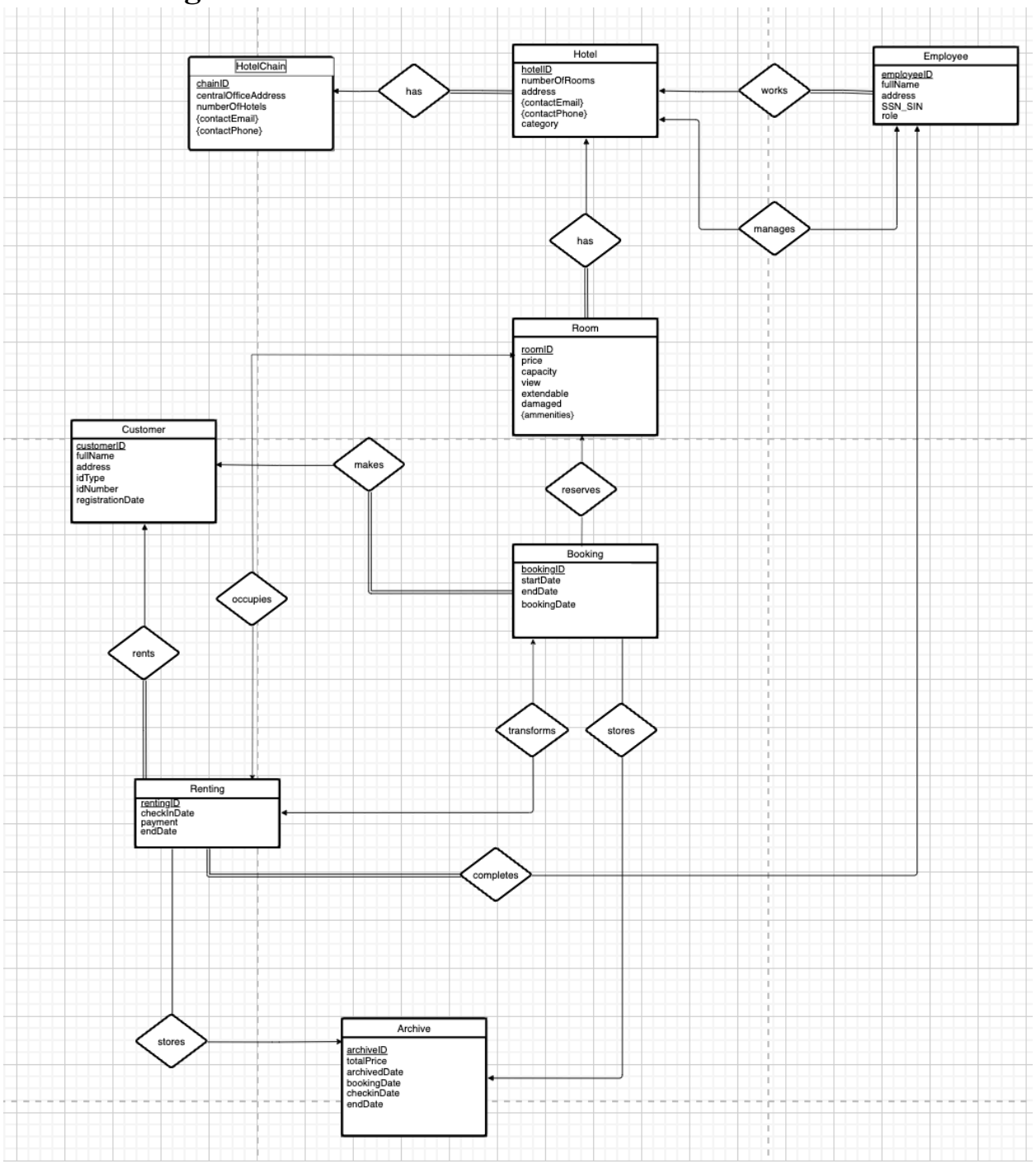
University of Ottawa

Professor: Verena Kantere

Nicolas Maalouly #300272916
Vincent Préseault #300170802

Submission Date: 02/14/2025

1.a : ER Diagram

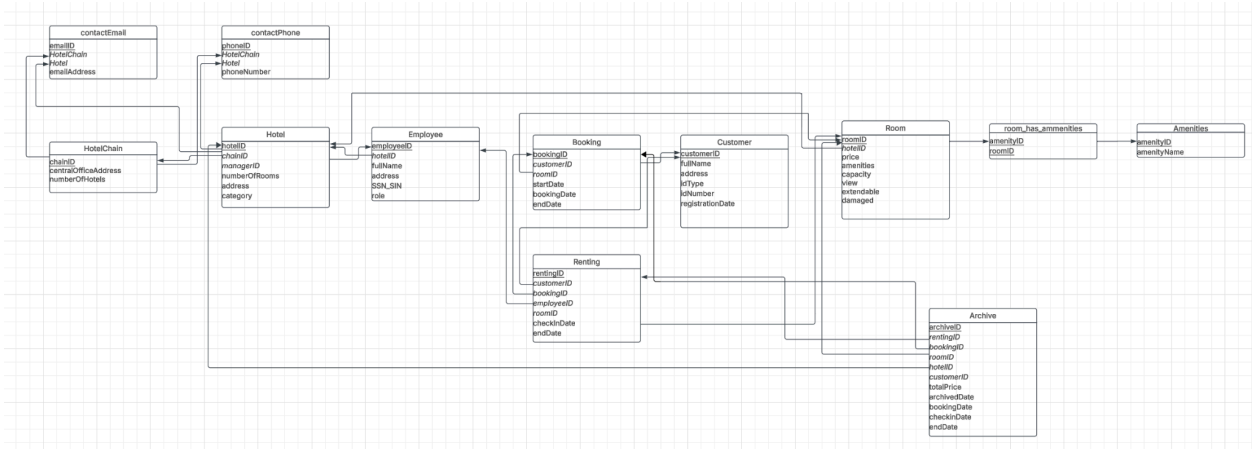


Link to view our ER Diagram on Draw.io :
https://drive.google.com/file/d/1UN1DW5hSYqDC3sEJDvknpwbXQrB0m63S/view?usp=sharing

Justification :

Our ER Diagram represents the hotel management system as described in the project description. We've identified the key relationships between entities such as *HotelChain*, *Hotel*, *Room*, *Customer*, *Booking*, *Renting*, and *Archive*, defining their attributes and relationships. To describe further, a *HotelChain* manages one to many *Hotels*, both having one to many contact emails and phone numbers. Each *Hotel* can also be categorized by a rating which has a value between 1-5. *Customers* can make *Bookings* for *Rooms*, which are later transformed into *Renting* records. Once a *Renting* is completed they are stored as an *Archive*. *Employees* work at *Hotels* and manage different aspects of the system such as completing the *Rentings* and checking-in *Customers*. Furthermore, each *Room* is connected to an *Amenities* identity that stores all *Amenities* for that specific *Room*. The relationships ensure data consistency within our database by defining interactions, such as *Customers* reserving *Rooms*, *Employees* managing *Hotels*, and *Bookings* transitioning into *Renting* records. This structured approach will allow us to transform our diagrams and constraints efficiently to a website for hotels that captures the idea of managing hotel operations while maintaining integrity across reservations, room availability, and customer interactions.

1.b : Relational Diagram



Link to view the diagram on Ludichart :
https://lucid.app/lucidchart/92705ff9-4a7f-4810-9007-e4d2f1e6aeec/edit?viewport_loc=-388%2C-915%2C4049%2C1955%2C0_0&invitationId=inv_b1fcc307-b97f-4856-99b6-e4a74ad2376d

Justification :

Our relational diagram represents the database structure for the hotel management system by modifying what was captured in our ER diagram. The *HotelChain* table stores its own attributes while linking to multiple *Hotels*. The *Hotels* contain details such as category, number of rooms,

and contact information, each managed by an Manager (*Employee)*. *Employees* are assigned specific *Hotels* and handle *Bookings*, which links to *Customers*. A *Booking* transitions into *Renting*, thus indirectly associating a *Customer* with a *Room*.*Rentings* store details such as check-in date and which employee initiated the renting. Per our user defined constraints at the end of this report, we've decided for the sake of efficiency within our database that when a customer goes to Book/Rent, it can only be associated with a single *Room*. The *Room* table includes attributes like price, capacity, and condition, with a many-to-many relationship to *Amenities* via the *room_has_amenities* table. Completed transactions are archived in the *Archive* table, maintaining historical records of *Bookings* and *Rentings*. The model ensures proper exchange of data while covering key relationships such as employee roles, customer interactions, and room assignments.

1.c : Integrity Constraints

1. Primary Keys

Here are our primary keys that are unique identifier to ensure data consistency throughout our diagrams:

- **HotelChain**: chainID - Allows for managing them centrally without having overlap.
- **Hotel**: hotelID - Allows for management of hotels without conflicts with attributes such as number of rooms, category etc.
- **Employee**: employeeID - Ensures each employee is distinguished by their roles compared to other employees.
- **Customer**: customerID - Links customers to their bookings and transactions which ensures proper validation and check in for arrival.
- **Room**: roomID - Differentiates the rooms for amenities, type and pricing.
- **Amenities**: amenityID - Track amenities to ensure they're in the right rooms.
- **Booking**: bookingID - Assures bookings are associated with a customer and a room to avoid duplications and double booking.
- **Renting**: rentingID - Links the renting to an employee and the customer that checked-in for efficient tracking and record history.
- **Archive**: archiveID - Distinguishes historical records for all previous years ensuring proper tracking of previous transactions.
- **room_has_amenities**: Composite primary key (roomID, amenityID) - This composite primary key exists in our sequential diagram since room and amenities are connected by a many to many relationship. This means we will have a table (room_has_amenities) to link Rooms with their Amenities.

2. Referential Integrity Constraints

- **Hotel**: chainID → references HotelChain(chainID) - Each hotel needs to be associated with a parent chain.
- **Hotel**: managerID → references Employee(employeeID) - Each hotel has an employee whose role is manager and manages everything in the hotel and supervises the other employees.

- **Employee**: hotelID → references Hotel(hotelID) - Each employee is associated with a hotelID since they must work at a hotel.
- **Booking**: customerID → references Customer(customerID) - Each booking has a customer that placed it.
- **Renting**: employeeID → references Employee(employeeID) - Each renting has an employee that handled the check-in.
- **Renting**: customerID → references Customer(customerID) - Each renting has a customer that is currently staying at the hotel.
- **Renting**: bookingID → references Booking(bookingID) - In the case that the customer reserved rooms ahead of time, that renting entity will have an associated booking.
- **Renting**: roomID → references Room(roomID) - Each renting room has a room that is occupied and cannot be booked by other customers.
- **room_has_amenities**: roomID → references Room(roomID) - Each room has certain amenities.
- **room_has_amenities**: amenityID → references Amenities(amenityID) - Each amenity is associated with a room.
- **Archive**: rentingID → references Renting(rentingID) - The archive holds old renting IDs to keep track of previous sales.
- **Archive**: bookingID → references Booking(bookingID) - The archive holds old booking IDs to keep track of which rooms have been booked before.

3. Domain and Attribute Constraints

- **Non-null constraints**:
    - Primary key attributes cannot be null - since each entity with a primary key needs to be uniquely identified.
    - Essential attributes such as hotelName, customerName, roomPrice, employeeName must be not null - since these are attributes that customer or employees would need to see when processing bookings and rentings.
- **Data types**:
    - Price in Room should be a number including decimals, ensuring valid monetary values.
    - Email attributes should follow an email format (VARCHAR(255) CHECK (email LIKE '%_@_%._%')). Ex: name@gmail.com
    - ContactPhone should follow a phone number format (VARCHAR(15) CHECK (contactPhone ~ '^[0-9-]+$')). Ex: 123-456-7890
    - Capacity in Room should be a positive integer capacity > 0.
    - The category of the hotel should only allow values from 1 to 5.
    - The view attribute in a Room is a string with value of “Mountain” or “Sea”
    - The damaged attribute in a Room is a boolean
    - The extendable attribute in a Room is a boolean

4. User-Defined Constraints

Since the description said to be inventive, here are our choices for user-defined constraints:

- **Employee Role Restriction**: The employees role should only allow valid roles Manager, Receptionist, Cleaner, Maintenance.
- **Booking/Renting:** Each Booking and Renting can only be associated with one room. If the Customer is creating a Booking/Renting for a group size that exceeds the capacity of every room in a hotel, they will have to create separate Bookings/Rentings themselves.
- **Booking Date Logic**:
    - startDate < endDate ensures that bookings have a valid duration.
    - Renting.checkInDate >= Booking.startDate ensures that check-in happens on or after the booking start date.
- **Extendable Rooms**:
    - If a room is extendable the capacity of the room can be equal to Room.capacity or Room.capacity + 1.
- **Prevent Overlapping Bookings**:
    - Each roomID can only be booked once in a date range.
- **Archive Pricing Consistency**:
    - Check for totalPrice >= 0, ensures that the final archived booking price is not negative.