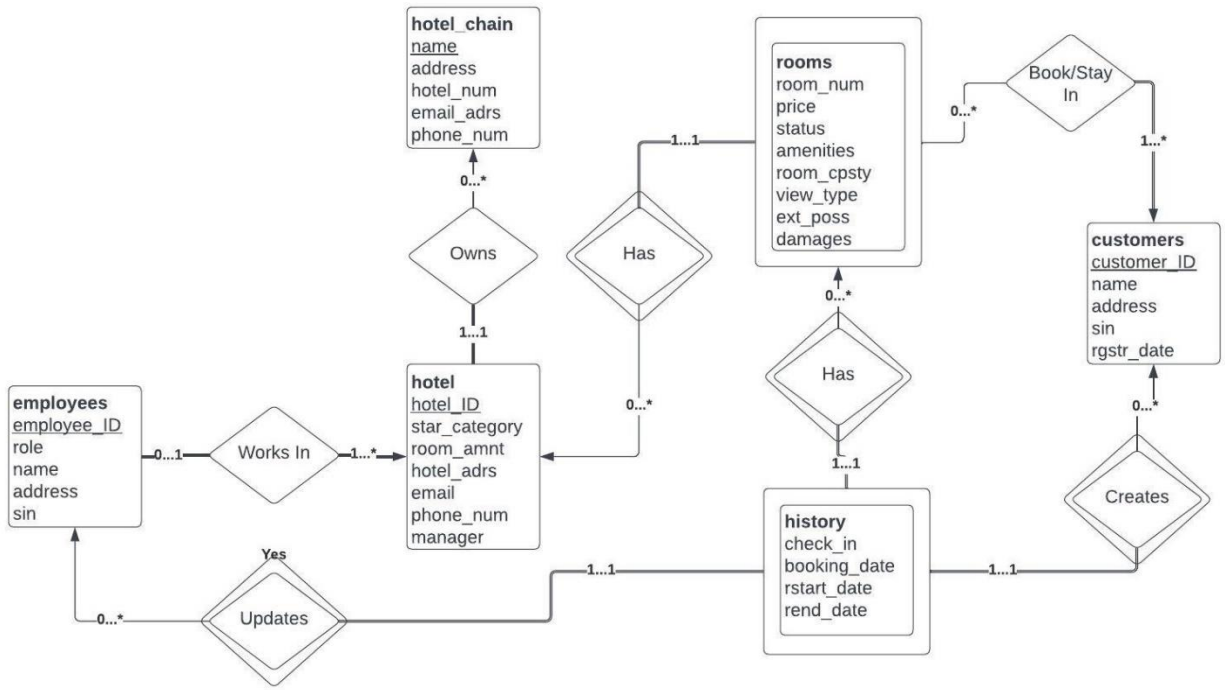
Course Project

Course CSI2132	Databases I
Academic Year	2022-23
Instructor	Wail Mardini
Group Members	Sara Rigotti 300226854
	Arine Karapetyan 300258951
	Samih Karroum 300188957
Submission Date	12.04.2023

DBMS

E-R Diagram



Justification of relationships between tables:

Rooms-History:

Rooms and History is a one-to-many relationship where the arrow points to rooms. The relationship is “has” because each room has its own “history” such as damages, previous rentees, etc. Due to the history belonging to individual rooms, history would not exist without the rooms entity, making history the weak entity in this relationship. Furthermore, the arrow points towards rooms because each room can have a new instance of history added to it as well

as having it all stored for said room, justifying the “one to many” relationship. The “1..1” association fro history makes sense because only one instance of history for a specific record is created and ultimately held by the rooms entity. The rooms having a “0..*” association makes sense because a room can be brand new, containing no previous history, comparatively an old room can have damages, many rentees, etc, making the number of “history” virtually infinite.

Customers-rooms:

Customers and rooms is a one-to-many relationship with the arrow pointing towards customers due to the customer being able to book as many rooms as possible, justifying their respective placements in the relationship. The relationship is “Book/Stay in” because the customer books the room either through the application or by going to the hotel in person. The notation "1...*" from customers to the relationship "book/stay in" means that each customer can book at least one hotel room but can also book multiple hotel rooms. This makes sense in a hotel booking system where a customer may book multiple rooms for a group or family. The notation "0..*" from rooms to the relationship "book/stay in" means that a hotel room may or may not be booked by any customer. This is because some hotel rooms may be vacant or unoccupied, while others may be booked by customers.

History-Customers:

History to customers is a one-to-many relationship with the arrow pointing towards customers due to the customers ability to create an endless amount of “history” (booking information). The relationship is “create” because it is the customer who ends up creating the information. It is a weak entity relationship because the information would never have existed without a customer checking in and prompting the employee to enter the information into history, making history the weak entity. The association from customers to the relationship is "0...*," which indicates that a customer may or may not create any booking history. This is because not all customers may have made bookings. The association from history of bookings to the relationship is "1..1," which indicates that each booking history must be created by a single customer, and it cannot be created without a customer. This is because a booking history is a record of a customer's booking activity, and it is uniquely identified by the combination of the customer entity and its own attributes.

Hotel-Rooms:

Hotel to rooms is a one-to-many relationship where “rooms” is the weak entity due to it not being possible for them to exist without the hotel. The "has" relationship indicates that a hotel can have many rooms, but a room can belong to only one hotel, justifying why the arrow points towards hotel. This is a one-to-many relationship, which can be represented in the ER diagram. The association from hotel to the relationship is "0...*," which indicates that a hotel may or may not have any rooms. This is because a hotel may have just opened or may have been fully booked, leaving no available rooms. The association from rooms to the relationship is "1..1," which indicates that a room must belong to a hotel, and it cannot belong to more than one

hotel. Additionally, the rooms entity is a weak entity, which means it cannot exist without the Hotel entity. This is because the room entity's existence depends on the hotel entity; it is uniquely identified by the combination of the hotel entity and its own attributes.

History-Employees:

This is a weak entity relationship with the arrow pointing towards employees. The relationship between them is updates because it is the employees that will update the information, therefore the history entity set will not exist without employees, making it a weak entity relationship. The relationship is “updates” because the employee updates the history of the hotel booking. The association from employees to the relationship is "0...*," which indicates that an employee may or may not update the information in the history of bookings entity. This is because not all employees have the authority or permission to update the information in the entity. The association from history of bookings to the relationship is "1…1," which indicates that the history of bookings entity must have at least one employee who updates its information, and it can be updated by only one employee at a time.

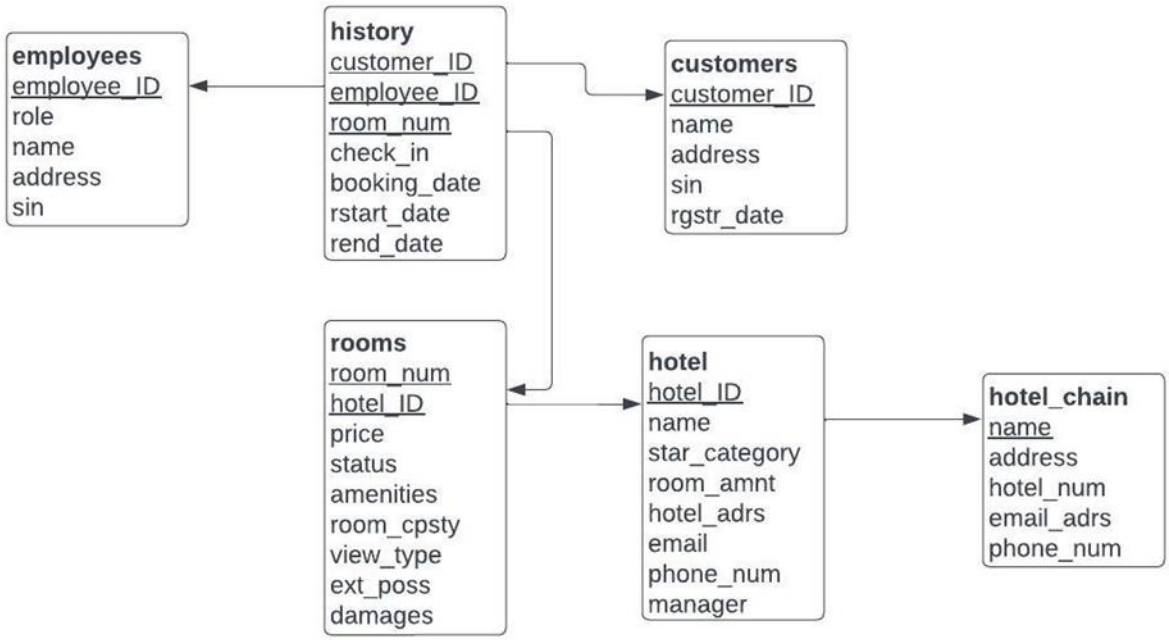
Hotel-Employees:

The arrow points towards hotel due to the relationship between the entity sets being “one to many”, this is because the employee would work for the hotel. The "works in" relationship indicates that an employee can work in only one hotel, but a hotel can have many employees. This is a one-to-many relationship, which can be represented in the ER diagram as follows. The association from employees to the relationship is "0...1," which indicates that an employee may or may not be associated with a hotel. This is because an employee may be unemployed or may have resigned from the hotel. The association from hotel to the relationship is "1…*," which indicates that a hotel must have at least one employee, but it can have many employees. Attributes are not shared between the entity sets,

Hotel_chain-Hotel:

This is a one-to-many relationship because the hotel chain can have multiple hotels belonging to it. Hotel belongs to hotel chain due to the hotel being an asset of the company, hence why the relationship between the entities is “Owns”. The association from hotel_chain to hotel is 0..* because a hotel chain can have 0 to a virtually infinite number of hotels belonging to it. The association from hotel-to-hotel chain is 1 to 1 because the hotel itself is a single entity and can only belong to a singular hotel chain. The attributes of the entity sets do not have relations to each other even though phone_num are the same attributes, the reason being that the hotel chain number is separate from that of the specific hotels number.

Relational Database Schema



Justification:

The above diagram outlines all attributes, including shared ones. The history entity contains the customer_ID of the customer who created it, the employee_ID of the employee who updates it, and the room_num of the room booked. Each room has the hotel_ID of the hotel it's located within, and each hotel has the name of the corresponding hotel_chain.

Relations (With Constraints)

```
CREATE TABLE hotel_chain(
   name VARCHAR(255) PRIMARY KEY NOT NULL,
   address VARCHAR(255) NOT NULL,
   hotel_num INT NOT NULL,
   email_adrs VARCHAR (255),
   phone_num VARCHAR (20)
);

CREATE TABLE hotel(
   hotel_ID INT PRIMARY KEY NOT NULL,
   hotel_chain_name VARCHAR(255) NOT NULL,
   name VARCHAR(50) NOT NULL,
   star_category NUMERIC(1,0) NOT NULL,
   room_amnt INT NOT NULL,
   hotel_adrs VARCHAR(255) NOT NULL,
```

```
    email VARCHAR(255),
    phone_num VARCHAR (20),
    manager VARCHAR(25) NOT NULL,
    FOREIGN KEY (hotel_chain_name) REFERENCES hotel_chain(name)
            ON DELETE CASCADE
            ON UPDATE CASCADE
);

CREATE TABLE rooms(
    room_num INT NOT NULL,
    hotel_ID INT NOT NULL,
    price NUMERIC(10,3) NOT NULL,
    amenities VARCHAR(255),
    room_cpsty INT NOT NULL,
    view_type VARCHAR (255),
    ext_poss VARCHAR(255) NOT NULL,
    damages VARCHAR (255),
    r_status VARCHAR(20) NOT NULL,
    FOREIGN KEY (hotel_ID) references hotel(hotel_ID)
            ON DELETE CASCADE
);

CREATE TABLE customers (
    customer_ID SERIAL PRIMARY KEY NOT NULL,
    first_name VARCHAR(20) NOT NULL,
    last_name VARCHAR(20) NOT NULL,
    address VARCHAR(255),
    sin INT NOT NULL,
    rgstr_date VARCHAR(20) NOT NULL
);

CREATE TABLE employees (
    employee_ID SERIAL PRIMARY KEY NOT NULL,
    first_name VARCHAR(20) NOT NULL,
    last_name VARCHAR(20) NOT NULL,
    address VARCHAR(255),
    sin INT NOT NULL,
    role VARCHAR(50) NOT NULL
);

CREATE TABLE history (
    customer_ID SERIAL NOT NULL ,
    employee_ID SERIAL,
    room_num INT NOT NULL,
```

```
    check_in BOOLEAN,
    booking_date VARCHAR(10),
    rstart_date VARCHAR(10),
    rend_date VARCHAR(10),
    FOREIGN KEY (customer_ID) references customers (customer_ID)
        ON DELETE NO ACTION,
    FOREIGN KEY (employee_ID) references employees (employee_ID)
        ON DELETE SET NULL
);
```

Justification:
Hotel.chain:
It holds 5 attributes, of which name is the primary key. Name and address always need to have a value in them. Thus they can't have their value be equal to null. The hotel number is an integer, and the rest are strings with different sizes.

Hotel:
The primary key is Hotel_ID, and the foreign key is name that references hotel chain. The entity holds 8 attributes. All attributes except for the email and phone number aren't allowed to be null.

Rooms:
The rooms entity has 9 attributes. Since it is a weak entity, it doesn't have a primary key. However, it does shave a foreign key hotel_ID that references hotel.
The hotel's price is numeric, with a precision of 13, from which 3 digits are counted after the decimal. Every attribute in this entity, except for amenities, view_type, and damages, aren't allowed to be null.

Customers:
This entity has 5 attributes, from which Customer_ID is the primary key, and sin is an integer. Everything can't be null except for the address. The customers entity does not have a foreign key.

Employees:
The entity has 5 attributes, and just like in customers; it doesn't have a foreign key, and only the address is allowed to be not null. The primary key is empoyee_ID.

History:
The History entity has 6 attributes, such that only customer_ID and room_num aren't allowed to be null. There are 3 foreign keys (customer_ID, employee_ID, room num), thanks to which entity history stores their information based on availability and their history.

Programming Languages

- Python
- Javascript
- PostgreSQL
- CSS
- HTML

DDLS

- Hotel_chain relation:
  - CREATE TABLE hotel_chain ().
- Hotel relation:
  - CREATE TABLE hotel ().
  - FOREIGN KEY (hotel_chain_name) REFERENCES hotel_chain(name)
- Rooms relation:
  - CREATE TABLE rooms ().
  - FOREIGN KEY (hotel_ID) references hotel (hotel_ID)
- Customers relation:
  - CREATE TABLE customers ().
  - DROP TABLE customers
- Employees relation:
  - CREATE TABLE employees ().
- History relation:
  - CREATE TABLE history ().
  - FOREIGN KEY (customer_ID) references customers (customer_ID)
  - FOREIGN KEY (employee_ID) references employees (employee_ID)
- Indexes:
  - create index room_status on rooms(r_status)
  - create index customerID_index on customers(customer_ID)
  - create index employeeID_index on employees(employee_ID)
  - create index hotelarea_index on hotel(hotel_adrs)
  - create index star_index on hotel(star_category)
- Views:
  - create view rooms_avail as select count (*) from rooms where status = ‘Available’ group by hotel_adrs in (select hotel_adrs from hotel);
  - create view capacity as select room_cpsty from rooms group by hotel_ID;

Queries

```
/* See available rooms with various criteria */
select *
        from rooms
        where status = 'Available';

/* See available rooms with various criteria */
```

```
select *
	from rooms
	where r_status = 'Available';

select *
	from rooms
	where r_status = 'Available'
	and hotel_ID = 1; /* can alter hotel_ID to any room attribute (ie amenities, capacity,
view_type) */

select *
	from rooms
	where r_status = 'Available'
	and price between 1000 and 2000; /* search for room with a min and max price
parameter */

/* search by hotel chain */
select *
	from rooms
	where r_status = 'Available'
	and hotel_ID in (select hotel_ID
			from hotel
			where hotel_chain_name = 'Montage Hotels & Resorts.');

/* search by hotel category */
select *
	from rooms
	where r_status = 'Available'
	and hotel_ID in (select hotel_ID
			from hotel
			where star_category between 3 and 5);

/* search by hotel total number of rooms */
select *
	from rooms
	where r_status = 'Available'
	and hotel_ID in (select hotel_ID
			from hotel
			where room_amnt between 10 and 100);

/* With implementation of DELETE CASCADE on hotel and rooms, if we delete from relation
hotel_chain: */
DELETE FROM hotel_chain
WHERE name = 'Westin Hotels & Resorts.';
```

```
/* all tuples in hotel with the corresponding foreign key hotel_chain_name will be deleted. */
SELECT *
FROM hotel;
/* Now the above query shows information on only 4 hotel chains */
/* Also, because rooms is dependent on hotel, all tuples in rooms with corresponding foreign
key hotel_ID are deleted. */
SELECT *
FROM rooms;
/* Now the above query shows information only for the hotels in the remaining 4 chains. */

/* Alternatively, say a hotel chain shuts down one of its hotels: */
DELETE FROM hotel
WHERE hotel_ID = '1';
/* With the delete cascade integrity constraint, all rooms tuples with hotel_ID '1' are also
deleted */
SELECT *
FROM rooms;
/* Above query now shows all rooms from all hotels except hotel with hotel_ID = '1' */

/* Say a hotel chain decides to rebrand and change their name (the primary key) */
UPDATE hotel_chain
SET name = 'Paradise Living.'
WHERE name = 'The Luxury Collection Hotels.';
/* The UPDATE CASCADE integrity constraint on hotel causes the tuples to update with the new
hotel_chain_name */
SELECT hotel_chain_name
FROM hotel;
/* Now the above query no longer contains instances of 'The Luxury Collection Hotels.' - they
have been replaced with 'Paradise Living.' */

/* Say an employee leaves the hotel/hotel chain and is deleted from the system */
DELETE FROM employees
WHERE last_name = 'PERVOT';
/* The SET TO NULL constraint ensures that the history of the room is kept
   even when the employee is removed */
SELECT employee_ID
FROM history;
/* Now the above query no longer contains the ID of the deleted employee, but keeps the rest
*/
```

Triggers

```
/* When a new room is add to a hotel, damages are automatically set to 'none'. and the status
is set to 'available' */
```

```
CREATE TRIGGER new_room
before INSERT
ON rooms
FOR EACH ROW
SET new.damages = 'none' and new.r_status = 'available'; /* keyword new refers to rows being affected */

/* In the event a user attempts to delete a room directly room the room relation (without the deletion of the corresponding hotel
 the trigger halts the actions and insteads alters the status (in case the room is merely out of commision and should be hidden from queries) */
 CREATE TRIGGER rooms_insteadOfDelete
 ON rooms
 INSTEAD OF DELETE
 AS
 UPDATE rooms
 SET r_status = 'Unavailable'
 WHERE r_status = old.stat
```

Installation Instructions

1. Download and Install ReactJs software application.
2. Download the program code zip file.
3. Extract files and save it as a folder on your computer.
4. Open the program file with vscode (text editor).
5. Within application terminal install
    a. pg (to read the SQL from the database) and
    b. express (to utilize the SQL code).
6. Unpack modules and code files on the terminal side bar.
7. Run the program.