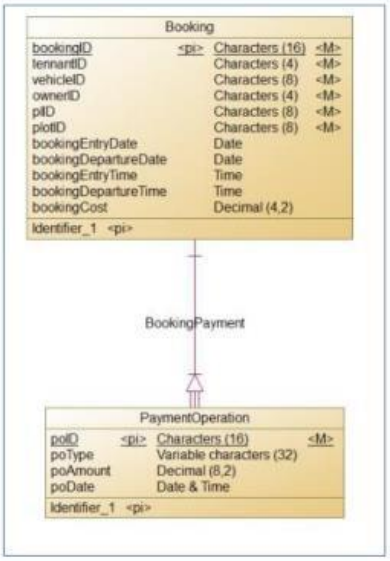
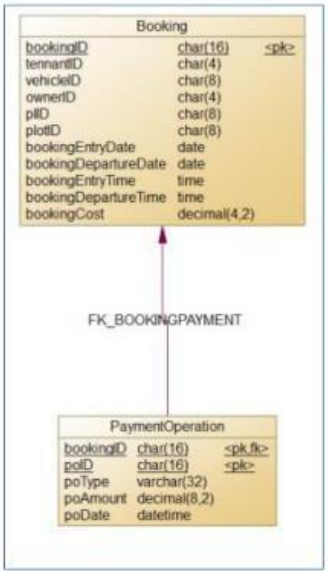
Database documentation

Management Module

Conceptual Model



Physical Model



The database used in the development program for a mobile application to search for parking spaces is made up of two main entities: "Reservation" and "Payment Operation" (PaymentOperation). A detailed explanation of the structure and relationship between these entities is provided below.

Booking Entity

The "Booking" entity represents the information related to the reservations made by the parking lot tenants. The attributes of this entity are the following:

- BookingID (Character) (Required): It is a unique identifier that represents each reservation in the database.
- TeenantID (Character): It is an identifier that refers to the tenant associated with the reservation.
- OwnerID (Character): It is an identifier that refers to the owner of the reserved parking space.
- VehicleID (Character): It is an identifier that refers to the vehicle associated with the reservation.
- plID (Character): It is an identifier that refers to the location of the reserved parking space.
- PlotID (Character): It is an identifier that refers to a specific area or terrain where the parking lot is located.
- BookingEntryDate (Date): Represents the booking entry date.
- BookingDepartureDate (Date): Represents the departure date of the reservation.
- BookingEntryTime (Hour): Represents the booking entry time.
- BookingDepartureTime (Hour): Represents the departure time of the reservation.
  BookingCost (Decimal): It is the cost associated with the reservation.

PaymentOperation Entity:

The "PaymentOperation" entity records information related to payments made by lessees to confirm and complete a reservation. The attributes of this entity are the following:

- PoID (Character) (Required): It is a unique identifier that represents each payment transaction in the database.
- PoType (Variable Character): Indicates the type of payment transaction made, which can be cash, credit card, bank transfer, etc.
- PoAmount (Decimal): It is the amount of the payment made by the lessee.
- PoDate (Date and time): Represents the date and time in which the payment operation was carried out.

Relationship between entities:

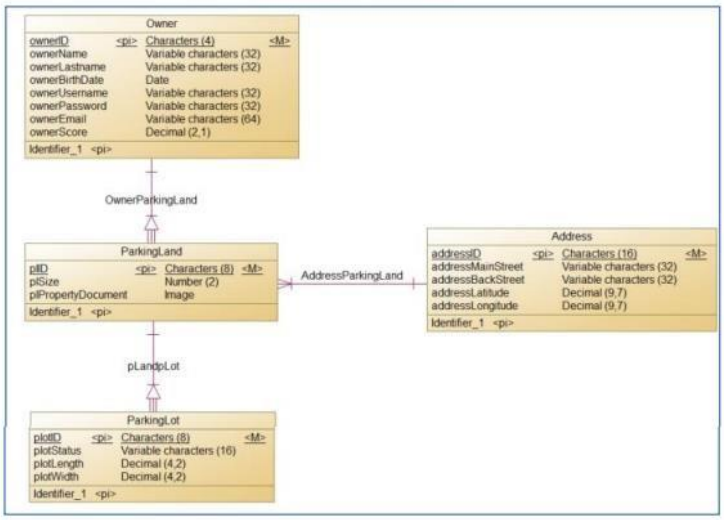
There is a dependency relationship between the entities "Reservation" and "Payment operation". This means that a reservation can have multiple payments associated with it, but one payment is specifically related to a reservation. In other words, each reservation has one or more payment transactions associated with it, but a payment transaction is linked to only one reservation.

This database structure allows for accurate tracking of reservations made by renters and the corresponding payments associated with each reservation. Provides detailed information on renters, parking lot owners, vehicles, locations, and check-in and check-out dates/times for each reservation. In addition, it records the details of the payment operations, including the type of payment and the amount.

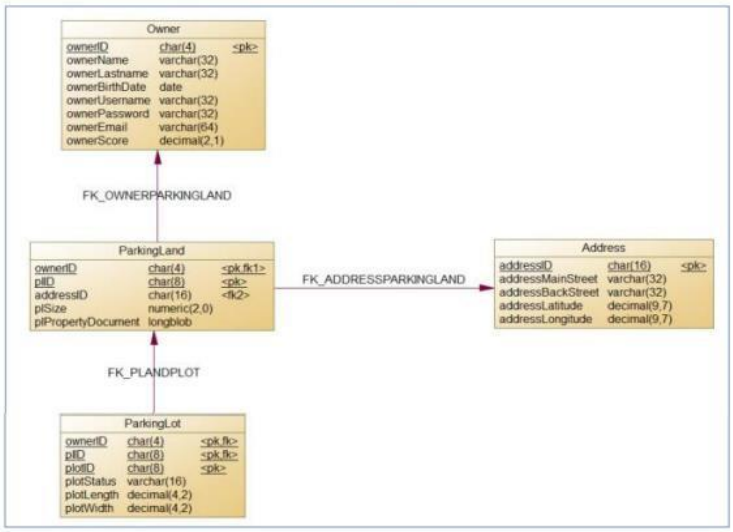
This data is essential for the functionality of the mobile application, as it allows users to search for available parking spaces, make reservations and complete the necessary payments efficiently. The database guarantees a reliable and complete record of all transactions related to reservations and payments in the system.

Owner Module

Conceptual Model



Physical Model



The following documentation details the structure of the database related to the parking lot owner in the parking lot search application project. Tables, attributes, primary keys, and relationships between entities are included.

ADDRESS table:

The "ADDRESS" table stores information related to the address of a parking lot. Its attributes are the following:

- ADDRESSID (Character) (Not null): It is a unique identifier that represents each address in the database.
- ADDRESSMAINSTREET (Character String): Represents the name of the main street associated with the address.

- ADDRESSBACKSTREET (Character String): Represents the name of the secondary street associated with the address.
- ADDRESSLATITUDE (Decimal): Stores the geographic latitude of the address.
- ADDRESSLONGITUDE (Decimal): Stores the geographic longitude of the address.
- PK_ADDRESS: It is a primary key that guarantees the uniqueness of each record in the "ADDRESS" table.

OWNER table:

The "OWNER" table stores the information related to the owner of the parking lot. Its attributes are the following:

- OWNERID (Character) (Not null): It is a unique identifier that represents each owner in the database.
- OWNERNAME (Character string): Stores the name of the owner of the parking lot.
- OWNERLASTNAME (Character string): Stores the last name of the parking lot owner.
- OWNERBIRTHDATE (Date): Represents the date of birth of the owner of the parking lot.
- OWNERUSERNAME (Character string): Stores the user name of the owner to access the system.
- OWNERPASSWORD (Character string): Stores the owner's password to access the system.
- OWNEREMAIL (Character string): Stores the email address of the owner of the parking lot.
- OWNERSCORE (Decimal): Represents a score assigned to the owner of the parking lot.
- PK_OWNER: It is a primary key that guarantees the uniqueness of each record in the "OWNER" table.

PARKINGLAND table:

The "PARKINGLAND" table stores information about the land or properties related to a parking lot owner. Its attributes are the following:

- OWNERID (Character) (Not null): It is an identifier that refers to the owner of the parking lot associated with the land.
- PLID (Character) (Not null): It is a unique identifier that represents each terrain in the database.
- ADDRESSID (Character) (Not null): Is an identifier that refers to the address associated with the terrain.
- PLSIZE (Numeric): Indicates the size of the lot or property of the parking lot.
- PLPROPERTYDOCUMENT (Character): Stores a document related to land ownership.
- PK_PARKINGLAND: It is a primary key that guarantees the uniqueness of each record in the "PARKINGLAND" table.

PARKINGLOT table:

The "PARKINGLOT" table stores information about parking spaces within a parking lot. Its attributes are the following:

- OWNERID (Character) (Not null): It is an identifier that refers to the owner of the parking lot associated with the parking space.
- PLID (Character) (Not Null): Is an identifier that refers to the parking lot associated with the parking space.
- PLOTID (Character) (Not Null): This is a unique identifier that represents each parking space in the database.
- PLOTSTATUS (Character string): Indicates the status of the parking space, such as "available", "occupied", etc.
- PLOTLENGTH (Decimal): Represents the length of the parking space.
- PLOTWIDTH (Decimal): Represents the width of the parking space.
- PK_PARKINGLOT: It is a primary key that guarantees the uniqueness of each record in the "PARKINGLOT" table.

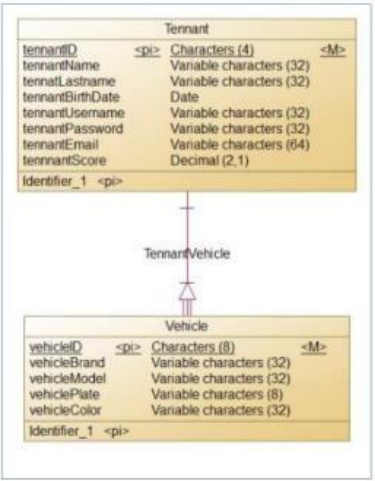
Relationships between entities:

- The "PARKINGLAND" table has a foreign key relationship with the "ADDRESS" table through the ADDRESSID attribute. This means that each PARKINGLAND is associated with a unique address in the "ADDRESS" table.
- The "PARKINGLAND" table has a foreign key relationship with the "OWNER" table through the OWNERID attribute. This means that each PARKINGLAND is associated with a unique owner in the "OWNER" table.
- The "PARKINGLOT" table has a foreign key relationship with the "PARKINGLAND" table through the OWNERID and PLID attributes. This means that each PLOT is associated with a unique PARKINGLAND in the "PARKINGLAND" table.

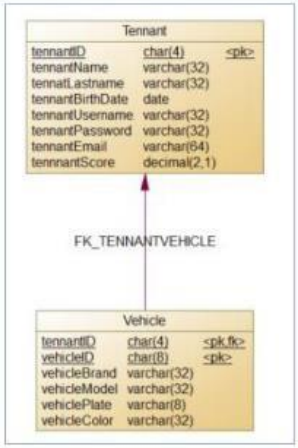
These structures and relationships in the database make it possible to keep track of the parking lot owners, the associated addresses, the parking lots or properties, and the parking spaces within each lot. This makes it easy to manage and search for available parking spaces in the mobile app, providing accurate information about the owners, locations and details of the parking spaces.

Tennant Module

Conceptual Model



Physical Model



Below is the detailed documentation of the database related to the tenants in the parking search application project. Tables, attributes, primary keys, and relationships between entities are included.

TENANT table:

The "TENNANT" table stores the information related to the parking lot tenants. Its attributes are the following:

- TENNANTID (Character) (Not Null): This is a unique identifier that represents each tenant in the database.
- TENNANTNAME (Character string): Stores the name of the tenant.
- TENNATLASTNAME (Character String): Stores the last name of the tenant.
- TENNANTBIRTHDATE (Date): Represents the date of birth of the tenant.
- TENNANTUSERNAME (Character string): Stores the user name of the tenant to access the system.
- TENNANTPASSWORD (Character string): Stores the tenant's password to access the system.
- TENNANTEMAIL (String): Stores the email address of the tenant.
- TENNNANTSCORE (Decimal): Represents a score assigned to the tenant.
- PK_TENNANT: It is a primary key that guarantees the uniqueness of each record in the "TENNANT" table.

VEHICLE table:

The "VEHICLE" table stores information related to vehicles owned by lessees. Its attributes are the following:

- TENNANTID (Character) (Not null): It is an identifier that refers to the lessee who owns the vehicle.

- VEHICLEID (Character) (Not null): It is a unique identifier that represents each vehicle in the database.
- VEHICLEBRAND (Character string): Stores the brand of the vehicle.
- VEHICLEMODEL (Character string): Stores the vehicle model.
- VEHICLEPLATE (Character string): Stores the vehicle plate.
- VEHICLECOLOR (Character string): Stores the color of the vehicle.
- PK_VEHICLE: It is a primary key that guarantees the uniqueness of each record in the "VEHICLE" table.

Relationships between entities:

The "VEHICLE" table has a foreign key relationship with the "TENNANT" table through the TENNANTID attribute. This means that each VEHICLE is associated with a single lessee in the "TENNANT" table.

These structures and relationships in the database make it possible to store the information of the parking lot tenants, including their personal data and the vehicles they own. This facilitates the management and association of tenants with their respective vehicles in the parking search mobile application.

Database Arquitecture

Product description: Develop a mobile application that allows users to search for and find a parking area, close to their location, and generate a structure, both of the system and of the database, that is as efficient as possible to store all the user's information, in a safe way, and that allows to provide satisfaction to the user, and financial support to the tenant.

functional requirements

- **Data storage:** The database will store the user's personal information, such as his name, surname, email, address, etc. As well as data that will be saved throughout its use time, such as your visited places, your places marked as favorites, as well as the scores to different areas.
- **Data validation:** The database must validate the existence of unique users, so that no new user can enter with already existing data.
- **Registry of owners:** The database will store all the information provided by the people who offer a parking area, such as the type of area, the security options, the price, the existing space, etc.
- **Parking request management:** The database will store the parking search requests, and will link them with the existing data and available parking areas to later show the user the areas available for use.
- **Queries and generalization of reports**: Queries can be made to the database to generate statistical reports on the use of the application and user satisfaction.

non-functional requirements

- **Performance**
  The system and the database must be able to handle a good performance, even when there is a large amount of data at the same time, and minimizing the waiting times of the users and be optimal for them.
- **Scalability**
  The database must allow the entry of new users, be they tenants, or people looking for an area, and should not affect the performance of the system and have the ability to increase its storage more easily and efficiently.
- **Security**
  Security measures must be implemented, such as encryption techniques or good security measures so that the data stored in the database cannot be violated or modified by an attacker. In this sense, access to the database must be allowed only for authorized personnel.
- **Availability**
  The database must have backup and data recovery measures for fortuitous cases such as failures or errors, allowing the continuity of the system at all times for users.

- **Usability**
  Navigation through the application and interactions with the database must be simple and clear, minimizing the learning curve and facilitating the adoption of the system by users.
- **Normative compliance:**
  The system and the database must comply with the applicable regulations and standards regarding the protection of personal data, privacy and information security.