

**UNIVERSIDAD DE SANTIAGO DE
COMPOSTELA**



ESCOLA TÉCNICA SUPERIOR DE ENXEÑARÍA

Gestión de un aparcamiento de la USC

Autores:

Aarón García Filgueira

Pablo García Fuentes

Pablo Guerrica-Echevarría Basterrechea

Jorge Lojo Abal

Adrián Martínez Balea

Profesor:

Roi Santos Mateos

Grao en Ingeniería Informática

Mayo 2023

Proyecto presentado en la Escola Técnica Superior de Enxeñaría de la Universidad de Santiago de Compostela para la materia de Bases de Datos II (G4012226).

Resumen

Aquí se recoge un ejemplo de lo que sería el desarrollo íntegro de un proyecto que tenga que gestionar cierto volumen de información. Ante una semántica que describe una problemática real, se modela una base de datos que la satisfaga. Dicho modelo lógico se implementa en PostgreSQL, y se desarrolla una aplicación en Java que se conecte y haga uso de la misma.

Índice general

I Estudio semántico-lógico del problema	XIII
1. Análisis Semántico	1
1.1. Semántica	1
2. Análisis conceptual	5
2.1. Modelo Entidad - Relación (MER)	5
2.1.1. Semántica no reflejada (SNR)	6
2.2. Diccionario de datos	8
3. Análisis lógico	13
3.1. Política de integridad referencial	14
3.2. Modelo Relacional (MR)	15
3.3. Normalización	16
II Implementación de la base de datos y diseño de la aplicación	19
4. Base de datos PostgreSQL	21
4.1. Tablas	21
4.1.1. Funciones	21
4.2. Vistas	22
4.3. Consultas	24
4.4. Transacciones	30
5. Aplicación en Java	41
5.1. Interfaz de usuario	41
6. Conclusiones	43

A. Creación de tablas	45
B. Interfaz gráfica	51

Índice de figuras

1.	Fases del proyecto en un diagrama de Gantt	XII
2.1.	Modelo entidad-relación	7
B.1.	Interfaz gráfica: inicio de sesión.	51
B.2.	Interfaz gráfica: menú principal.	52
B.3.	Interfaz gráfica: ingresar vehículo.	52
B.4.	Interfaz gráfica: ingresar vehículo. Seleccionar parking.	53
B.5.	Interfaz gráfica: ingresar vehículo. Seleccionar plaza.	53
B.6.	Interfaz gráfica: ingresar vehículo. Finalización.	54
B.7.	Interfaz gráfica: retirar vehículo.	54
B.8.	Interfaz gráfica: retirar vehículo. Información del aparcamiento. .	55
B.9.	Interfaz gráfica: retirar vehículo. Pago.	55
B.10.	Interfaz gráfica: retirar vehículo. Sin saldo.	56
B.11.	Interfaz gráfica: ver parking.	56
B.12.	Interfaz gráfica: ver parking. Representación de plazas.	57
B.13.	Interfaz gráfica: reservar.	57
B.14.	Interfaz gráfica: reservar. Seleccionar parking.	58
B.15.	Interfaz gráfica: reservar. Seleccionar horario.	58
B.16.	Interfaz gráfica: reservar. Seleccionar plaza.	59
B.17.	Interfaz gráfica: reservar. Pago.	59
B.18.	Interfaz gráfica: reservar. Sin saldo.	60
B.19.	Interfaz gráfica: mis reservar.	60
B.20.	Interfaz gráfica: mis reservar. Cancelación.	61
B.21.	Interfaz gráfica: ajustes.	61

Índice de cuadros

2.1. Diccionario de datos de las entidades.	8
2.2. Diccionario de datos. Miembros USC y vehículos.	9
2.3. Diccionario de datos. Aparcar y parkings.	10
2.4. Diccionario de datos. Plazas, asignar y reservar.	11

Introducción

A lo largo del siguiente documento se recogen las diferentes fases del proceso seguido para la elaboración del proyecto de la asignatura. En primer lugar se analiza a nivel lógico y semántico una problemática propuesta. Para el desarrollo de esta fase se toma como base uno de los trabajos proporcionados: “Gestión de un aparcamiento de la USC”. Partiendo de la semántica previa, se aporta un enfoque diferente que introduce funcionalidades nuevas con la necesidad de ser modeladas.

A continuación se propone un posible modelo conceptual (Modelo Entidad Relación, MER) que satisfaga la realidad expuesta en la semántica. También se anexa un diccionario que recoja los diferentes datos con los que se trabajará, exponiendo su tipo, características e información relevante de los mismos.

Para finalizar esta fase del desarrollo, es necesario traducir dicho modelo conceptual a un modelo lógico (Modelo Relacional, MR) para la posterior implementación de la base de datos. En este punto también se analizan las políticas de integridad referencial que nos permitan mantener la consistencia de nuestro sistema ante el borrado de datos.

La segunda parte del desarrollo de este trabajo se basa en la implementación de la base de datos diseñada en un entorno PostgreSQL. También se analizarán e implementarán una serie de vistas, consultas y transacciones de relevancia para el contexto del problema.

La segunda fase de la parte de implementación, consistirá en el desarrollo en Java de una aplicación que se conecte con la base de datos, mediante el controlador JDBC (Java DataBase Connectivity).

Este trabajo, por tanto, recoge la fase completa del desarrollo de un posible proyecto real. Implica tanto la fase inicial de modelado e implementación de una base de datos, como la construcción de un proyecto realista que nos permita hacer uso de la misma.

XII

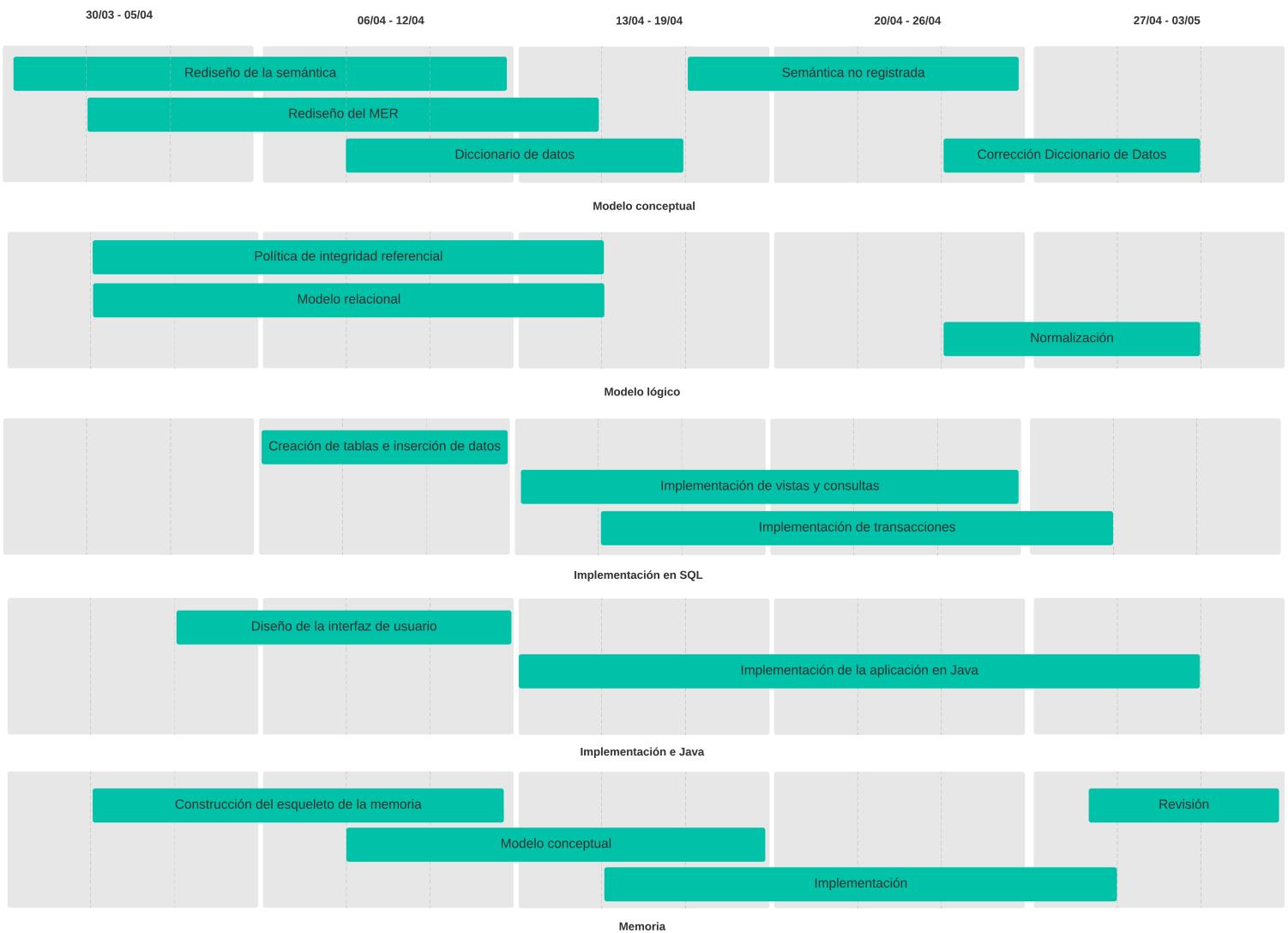


Figura 1: Fases del proyecto en un diagrama de Gantt

Parte I

Estudio semántico-lógico del problema

Capítulo 1

Análisis Semántico

El enfoque semántico del proyecto se realiza en base y como ampliación a uno de los trabajos propuestos: “Gestión de un aparcamiento de la USC”. Para un mayor aporte de funcionalidades a la hora del desarrollo en Java se incluyen entidades y relaciones que son necesarias en el contexto de una aplicación de registro y gestión de reservas para un parking.

Los principales cambios implementados son el añadir diferentes tipos de parkings, modificando la estructura de las plazas (se introducen en una jerarquía, como entidad débil de los parkings). También se añade a la semántica los PDI, para así poder tener en cuenta la nueva relación con las plazas: asignar. También se completó la idea del pago, mediante un saldo. Esto se traducirá a un modelo entidad relación diferente. Se eliminan las agregaciones, y se incluyen nuevas jerarquías para las plazas y los usuarios.

1.1. Semántica

La USC desea tener una Base de Datos para gestionar de manera más eficiente la información relativa a todos sus parkings. Para ello, expone las especificaciones recogidas en el siguiente apartado, que han de tenerse en cuenta para su diseño.

- ⁵ De los **miembros de la USC** que reservan una plaza para su vehículo en cualquier parking se desea conocer su DNI, nombre completo, teléfono de contacto, correo electrónico y contraseña. También se almacenará el saldo que tienen en su cuenta de la aplicación, que se actualiza de forma presencial en cualquier parking. Como todos pertenecen a la USC, se quiere almacenar la fecha de ingreso en esta institución. Aunque alguien de la USC abandone la universidad, se le seguirá “tratando” como si todavía lo estuviera, como agradecimiento por su labor. Las personas que no pertenezcan a la universidad podrán a su vez aparcar en el parking, pero sólo en el momento de llegada al mismo y sin capacidad
- ¹⁰

de reserva, de tal manera que no forman parte de los usuarios registrados. Así mismo, es necesario considerar al **Personal Docente e Investigador** de forma separada, pues este puede poseer una o más plazas asignadas únicamente para uso individual durante los períodos en los que desempeñe una de estas funciones en la USC.

Cada miembro puede poseer varios **vehículos**, de los cuales interesa su matrícula, el tipo de vehículo (coche o moto), marca y modelo. Un vehículo tiene un máximo de un único propietario, aunque puede haber vehículos no asociados a propietarios, aquellos aparcados por personal ajeno a la USC. Además, se desea almacenar por cada vehículo el número de infracciones que comete, *pues puede ser vetado en caso de alcanzar un máximo*. Por tanto, si un miembro de la USC llega al máximo de infracciones teniendo en cuenta todos sus vehículos, se restringe su capacidad de reserva de plazas.

La base de datos debe almacenar los diferentes **parkings** que pertenezcan a la universidad, de forma que interesa conocer su localización (en qué campus se encuentran), el centro al que se asocian, su capacidad y el número máximo de plazas que son privadas en ese parking.

En cada parking hay dos tipos de plazas: las destinadas a que **cualquier coche aparque en ellas** (vehículos no asociados a un miembro de la USC) y las destinadas a ser asignadas, o reservadas por los usuarios por un cierto período de tiempo (durante el cual solo ellos las podrán ocupar). Por motivos de seguridad/mantenimiento, no se podrá aparcar en una plaza privada de las que están disponibles para reservar ni, claro está, reservar una de las del tipo “aparcar”. Las **plazas para reservar/aparcar** se identifican por un código numérico único dentro de cada párking. Además, se exige que *solamente se permita reservar y aparcar en plazas vehículo del mismo tamaño* (los tamaños se consideran como de coche o moto) que la plaza reservada, ya que en otros parkings algunos usuarios ocupan varias plazas de motos con un coche.

Se necesita mantener un **registro detallado de la actividad del parking**: en qué plazas se aparcó/fueron reservadas, por quién (y con qué coche), la fecha de entrada y salida y el precio pagado son datos relevantes, así como mostrar las plazas asignadas a los PDI. Las reservas las realizan los usuarios de la USC mediante la aplicación, mientras que las plazas en las que se aparca son añadidas por los usuarios en el momento de llegada al párking, únicamente siendo necesario introducir la matrícula del coche aparcado y obteniendo un ticket que se usa a la salida. Además, existe un **rol de administrador** que se encarga de introducir las infracciones y asignar las plazas de los PDI. Cada miembro de la USC no puede tener reservada más de una plaza por vehículo a la vez y, por supuesto, una plaza no puede estar reservada por más de una persona a la vez.

Realizar maniobras imprudentes y otras infracciones son acciones almacenadas en la base de datos para cada vehículo. Si un vehículo tiene asociadas

III

IV

V

VI

VII

- ⁵⁵ 5 infracciones se considerará vetado del parking de forma indefinida. *Para los miembros de la universidad, su veto se aplicará si entre todos sus vehículos se suman 5 infracciones*, de forma que no podrá acceder a la aplicación y tendrá que acudir a la secretaría de la USC para eliminar el voto si así lo decide la universidad.

Capítulo 2

Análisis conceptual

Aquí se analiza exhaustivamente la semántica propuesta en el Capítulo 1. Primero se mostrará la solución propuesta para el modelado de la problemática, siguiendo el esquema del Modelo Entidad - Relación (MER).

Posteriormente, se tratarán los razones de cada detalle del diseño, para finalizar enumerando la semántica no reflejada en el modelo propuesto.

2.1. Modelo Entidad - Relación (MER)

Con el fin de entender de forma más visual la semántica, se resaltaron en negrita aquellas partes de las que se puede extraer una posible entidad. Además, se subrayaron los posibles datos a almacenar. Si se observa con detenimiento la Figura 2.1 (el MER), puede comprobarse como efectivamente los atributos de las distintas entidades coinciden con los elementos subrayados.

Tras el análisis de la semántica del problema, se pueden observar cuatro entidades diferentes: **MiembrosUSC** (y **PDI**) en el párrafo II línea 5 y 15; **Vehículos** en el párrafo III línea 19; **Parking** en el párrafo IV línea 27 y **Plazas (públicas y privadas)** en el párrafo V líneas 32 y 37.

La entidad **Parking** participa únicamente en la relación **Contener**, tal y como se observa en el párrafo IV línea 31. Se trata de una relación en la que un parking puede contener entre una o múltiples plazas, pero que una plaza solo puede estar en un aparcamiento.

En el párrafo V se habla con detenimiento de las plazas. En concreto se comenta que existen algunos aparcamientos destinados a que cualquier coche aparque en ellos (línea 32) y otros para reserva, en los que no se permite aparcar. De esta diferencia conceptual pueden clasificarse dos tipos de plazas: "públicas" (donde todos pueden aparcar) y "privadas" (donde personal de la USC puede reservar). La solución más adecuada para el modelado de esta parte es una jerarquía. De

esta forma una entidad padre (**Plazas**) es la que se relaciona con **Parking**, y las entidades hijas se relacionan de forma diferente con el resto del modelo. Se trata, por tanto, de una jerarquía total disjunta, ya que el conjunto de plazas públicas y privadas es todo el conjunto de plazas, así como una plaza no puede ser pública y privada al mismo tiempo. Por último, también se habla en la línea 38 que las plazas tienen un identificador numérico dentro de cada parking. Entonces, las plazas son entidades débiles del aparcamiento al que pertenecen, necesitándose el id del parking y el identificador numérico de la plaza para distinguir ésta únicamente.

La entidad **MiembrosUSC** se extrae del párrafo II, donde también se habla de los **PDI** (línea 15). Tal y como se expresa, los PDI son un tipo de miembro, por lo que se modela mediante una jerarquía parcial (no todos los miembros son PDI). Los **MiembrosUSC** se relacionan con Vehículos mediante **Poseer**, donde un miembro puede tener varios coches, y un coche puede estar asociado (o no, ya que puede haber usuarios no registrados) con un miembro.

A su vez los **Vehículos** se relacionan con las **Plazas Públicas (Aparcar)**, en el párrafo VI, línea 43). Un vehículo puede haber aparcado en múltiples plazas, así como en una plaza pueden haber aparcado múltiples vehículos. Como un vehículo puede aparcar en una plaza más de una vez, es necesario diferenciar las distintas relaciones. Para ello el atributo de la relación *fechaEntrada* nos sirve como clave primaria parcial de la relación (no se puede aparcar en múltiples plazas en el mismo instante).

En esa misma parte de la semántica se expone una relación análoga a **Aparcar: Reservar**. Esta es la que ocurre entre **MiembrosUSC** y **Plazas Privadas**. De esta forma se modela lo comentado en el párrafo V, diferenciando los tipos de plazas según si se puede aparcar (todo el público) o reservar (solo miembros).

La entidad hija de **MiembrosUSC**, **PDI**, se relaciona también con las plazas privadas (Asignar, tal y como se comenta en el párrafo II, línea 16). Un PDI puede haber tenido asignadas múltiples plazas, así como una plaza pudo haber sido asignada a múltiples PDI. Por esta razón se usa como clave primaria parcial de la relación *fechaInicio*.

2.1.1. Semántica no reflejada (SNR)

Es de interés la semántica no reflejada (SNR) en el MER propuesto. En el párrafo III en la línea 24 se especifica que los vehículos pueden ser vetados al alcanzar un máximo de infracciones. Posteriormente en la línea 57 del párrafo VII se especifica como para los miembros de la USC el voto se aplica cuando la suma de las infracciones de todos sus coches suma 5. Por otro lado, en el párrafo V línea 39 se especifica que cuando se aparcá y reserva una plaza, debe ser del mismo tipo que del coche.

2.1. MODELO ENTIDAD - RELACIÓN (MER)

7

SNR Cuando un coche realiza cierto número de infracciones se considera vetado, y no puede usar ningún parking de la USC. Para miembros, el voto sucede cuando se alcanzan 5 infracciones entre todos los vehículos de dicha persona.

SNR Cuando se aparcá en una plaza, o se reserva la misma, el tipo del coche debe ser el mismo que el de la plaza. Los tipos son 'coche' o 'moto'.

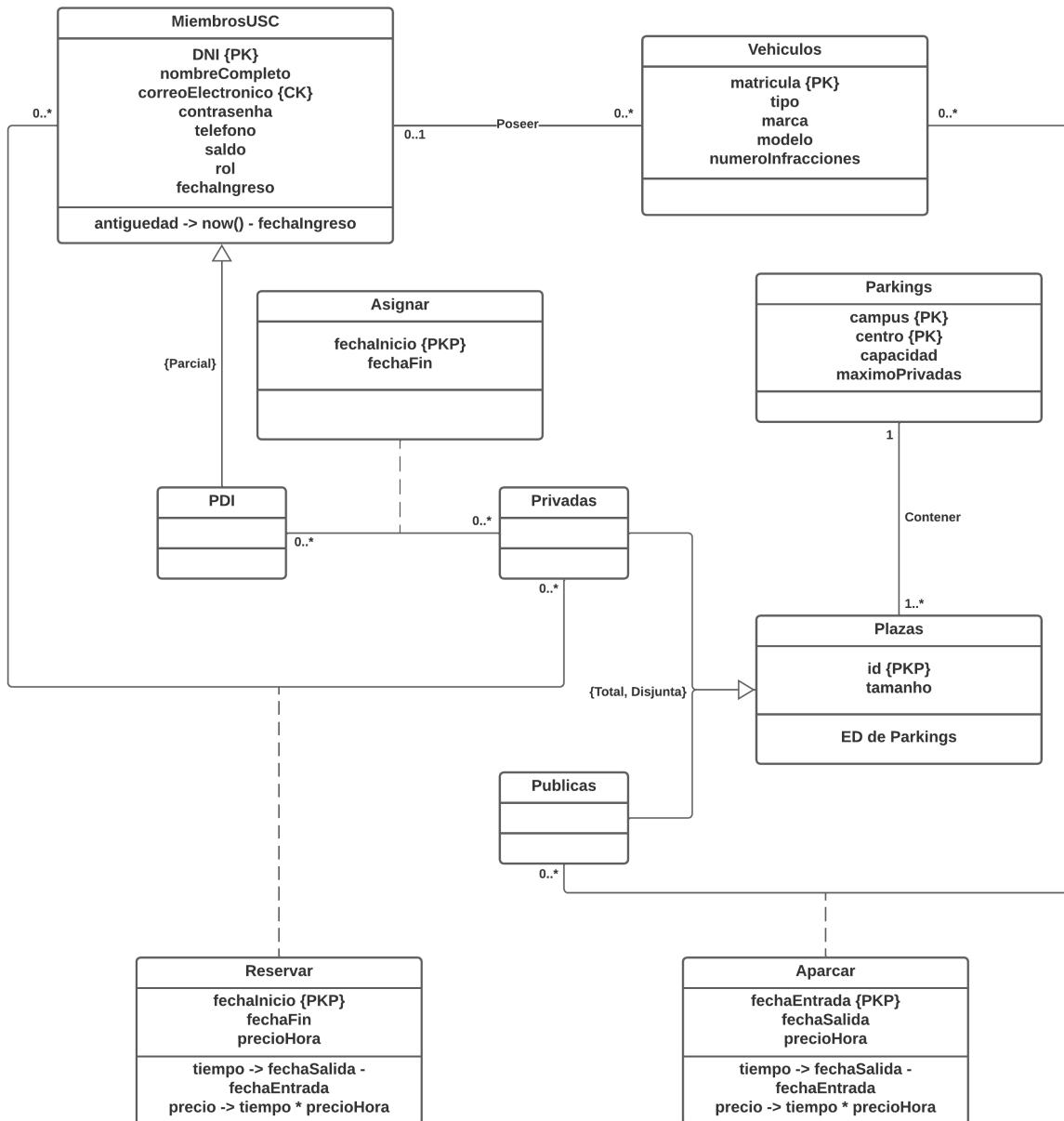


Figura 2.1: Modelo entidad-relación

2.2. Diccionario de datos

Cuadro 2.1: Diccionario de datos de las entidades.

ENTIDAD	DESCRIPCIÓN	INSTANCIAS PREVISTAS
Miembros USC	Empleados o asociados de la universidad.	Número de empleados de la universidad.
Vehículos	Vehículos que ingresan en uno de los parkings.	Número de vehículos que se registran al ingresar en un parking.
Parkings	Ubicación compuesta por múltiples plazas.	Número de parkings que están registrados en la universidad.
Plazas	Espacios dentro de un parking destinados para aparcar.	Número de plazas que se encuentran en cada parking.
Públicas	Tipo de plazas accesibles para todos los vehículos que entren en el parking.	Número de plazas públicas dentro de un parking.
Privadas	Tipo de plaza accesible sólo a través de una reserva o por asignación.	Número de plazas privadas que se encuentran en un parking.
PDI	Miembro de la USC al que, por su puesto, es posible asignarle una plaza.	Número de personal docente o investigador registrado en aplicación.

Cuadro 2.2: Diccionario de datos. Miembros USC y vehículos.

ENTIDAD O RELACIÓN	ATRIBUTOS	DESCRIPCIÓN	TIPOS DE DATOS	N	M	S	P
Miembros	DNI{PK}	DNI del miembro	carácter 9	No	No	No	No
	Nombre completo	Nombre y apellidos del trabajador	Carácter variable 50	No	No	No	No
	Rol	Tipo de acceso que tiene el usuario a la base de datos	Carácter variable 30	No	No	No	No
	Correo electrónico	Dirección del correo electrónico del trabajador	Carácter variable 60	No	No	No	No
	Contraseña	Contraseña usada por el miembro para iniciar sesión en la aplicación	carácter 128	No	No	No	No
	Saldo	Fondos disponibles del usuario para gestionar pagos en la aplicación	numeric (8, 2)	No	No	No	No
	Fecha de ingreso a la aplicación	Fecha de dada de alta del usuario en la aplicación	date	No	No	No	No
	Teléfono	Teléfono de contacto del usuario	int	No	No	No	No
	Antigüedad	Tiempo que lleva registrado en la aplicación	float	No	No	fechaActual - fechaIngreso	No
Vehículos	Matrícula {PK}	Identificador de los vehículos	carácter 7	No	No	No	No
	Tipo	Tipo de vehículo	Carácter 10	No	No	No	No
	Marca	Marca del vehículo	Carácter variable 30	No	No	No	No
	Modelo	Modelo del vehículo registrado	carácter variable 30	No	No	No	No
	Número infracciones	Número de infracciones cometidas por el vehículo	int	No	No	No	No

N: Admite NULL, M: Multivalorado, D: Derivado, P: Valor predeterminado

Cuadro 2.3: Diccionario de datos. Aparcar y parkings.

ENTIDAD O RELACIÓN	ATRIBUTOS	DESCRIPCIÓN	TIPOS DE DATOS	N	M	S	P
Aparcar	Fecha de entrada {PKP}	Fecha en la que entró el vehículo a la plaza	timestamp	No	No	No	No
	Fecha de salida	Fecha en la que el vehículo abandona la plaza	timestamp	No	No	No	No
	Tiempo	Tiempo transcurrido desde que un vehículo entra a la plaza hasta que sale	float	No	No	fechaSalida - fechaEntrada	No
	Precio por Hora	Coste por la estancia en el parking durante una hora	float	No	No	No	No
	Precio	Coste final para un vehículo en función del tiempo transcurrido en el interior del parking	float	No	No	tiempo * precioPorHora	No
Parkings	Campus {PK}	Campus de la USC en el que se encuentra el parking	caracter variable 20	No	No	No	No
	Centro {PK}	Centro/facultad a la que pertenece el parking	caracter variable 50	No	No	No	No
	Capacidad	Número de plazas totales en el parking	int	Sí	No	No	No
	Máximo Privadas	Número máximo de plazas que pueden establecerse como privadas	int	Sí	No	No	No

N: Admite NULL, M: Multivalorado, D: Derivado, P: Valor predeterminado

Cuadro 2.4: Diccionario de datos. Plazas, asignar y reservar.

ENTIDAD O RELACIÓN	ATRIBUTOS	DESCRIPCIÓN	TIPOS DE DATOS	N	M	S	P
Plazas	ID {PK}	Código identificador de la plaza	int	No	No	No	No
	Tamanho	Tamaño de plaza	Carácter variable 10	No	No	No	No
Asignar	Fecha de inicio {PK}	Fecha en la que se realiza la asignación de la plaza	Date	No	No	No	No
	Fecha de fin	Fecha en la que concluye la asignación de la plaza	Date	No	No	No	No
Reservar	Saldo	Fondos disponibles del usuario para gestionar pagos en la aplicación	numeric (8, 2)	No	No	No	No
	Fecha de ingreso a la aplicación	Fecha de dada de alta del usuario en la aplicación	date	No	No	No	No
Reservar	Teléfono	Teléfono de contacto del usuario	int	No	No	No	No
	Fecha de inicio {PK}	Fecha en la que se inicia la reserva	timestamp	No	No	No	No
Reservar	Fecha de fin	Fecha límite de la reserva	timestamp	No	No	No	No
	Precio por hora	Coste asociado a una hora de reserva	float (8, 2)	No	No	No	No
Reservar	Tiempo	Tiempo total reservado	date	No	No	fechaFin - fechaInicio	No
	Precio	Coste total por la reserva	float	No	No	tiempo * precioPorHora	No

N: Admite NULL, M: Multivalorado, D: Derivado, P: Valor predeterminado

Capítulo 3

Análisis lógico

Para la transformación del Modelo Conceptual (MER) al Modelo Lógico (MR) se deben seguir una serie de reglas, las cuales se explican a continuación. En primer lugar, todas las entidades fuertes constituyen una tabla con todos sus atributos en forma atómica (se descomponen los compuestos y en el caso de haber multivalorados se crea una tabla específica para cada uno de ellos). Además, la clave primaria de esta tabla será la misma clave primaria del MER para esa entidad en concreto. Por otra parte, las entidades débiles se generan de igual forma, a excepción de que se incluye una copia de la clave primaria de la entidad fuerte de la que depende.

Para transformar las relaciones se seguirán dos criterios dependiendo de las multiplicidades asociadas a las mismas. Primero, en las relaciones 1:N se añade una copia de la clave primaria de la entidad del lado 1 en la tabla del lado N. Además, los atributos de la relación se incluyen en la tabla del lado N de nuevo. En el caso de encontrarnos con relaciones N:N, se crea una tabla para la relación con copias de las claves primarias de las entidades relacionadas, así como la clave primaria parcial de la relación (formando la clave primaria de la nueva tabla). Además se añaden los atributos de la relación.

Por último, en el caso de las jerarquías se explica la forma de modificarlas en los casos concretos que aparecen en nuestro MER (jerarquía total disjunta y jerarquía parcial). En el primer caso, se crea una tabla para cada combinación superclase/subclase con todos los atributos de ambas (siendo la clave primaria la de la superclase). Por otro lado, en la jerarquía parcial, la superclase se entiende como una entidad fuerte y la subclase formará una tabla que contiene todos sus atributos. La clave primaria para esta tabla será una copia de la clave primaria de la superclase.

3.1. Política de integridad referencial

Cada vez que se trabaja con claves foráneas, se debe realizar un análisis detallado sobre cuáles deben ser las políticas de integridad referencial que se deben aplicar. En este proyecto se utilizarán tres de ellas:

- **No Action:** la restricción en caso de borrado, consiste en no permitir borrar una tupla si existe una clave primaria referenciada por alguna clave foránea. De forma similar, la restricción en caso de modificación consiste en no permitir modificar ningún atributo de una tupla si tiene una clave primaria referenciada por alguna clave foránea. Se utilizará en la base de datos para el borrado de datos, excepto en las tabla de miembros de la USC.
- **Cascade:** el borrado en cascada elimina las filas de referencia en la tabla secundaria cuando la fila referenciada se elimina en la tabla donde se encuentra la clave primaria. En el caso de la actualización, las filas de referencia se van a actualizar en la tabla secundaria cuando la fila referenciada se actualiza en la tabla principal que tiene una clave primaria. La actualización en cascada se utilizará en todas las tablas de la base de datos para garantizar la consistencia de los datos.
- **Set Default:** en la eliminación o actualización de una tupla de una tabla que tiene una clave externa en otra tabla, se establece en un valor predeterminado en ambas tablas. De esta manera, se asegura que no haya registros en la tabla de referencia que no tienen una correspondencia en la tabla principal, lo que podría comprometer la integridad de la base de datos. En la tabla de miembros de la USC se utiliza cuando un usuario quiere eliminar su cuenta. Por la ley de protección de datos se deberá borrar el DNI de la persona en cuestión, asignando al atributo DNI el valor 99999999R (DNI por defecto).

3.2. Modelo Relacional (MR)

MIEMBROUSC (dni, nombreCompleto, correoElectronico, contrasenha, telefono, saldo, rol, fechaIngreso).

CA: antiguedad → now() - fechaIngreso.

PK: dni.

CK: correoElectronico.

PDI (dniMiembroUS).

PK: dniMiembroUSC.

FK: dniMiembroUSC REFERENCES **MIEMBROSUSC** (dni).

VEHICULOS (matricula, tipo, marca, modelo, numeroInfracciones, dniMiembroUSC).

PK: matricula.

FK: dniMiembroUSC REFERENCES **MIEMBROUSC** (dni).

PARKING (campus, centro, capacidad, maximoPrivadas).

PK: campus, centro.

PLAZAPRIVADA (campusParking, centroParking, id, tamanho).

PK: campusParking, centroParking, id.

FK: campusParking, centroParking REFERENCES **PARKING** (campus, centro).

PLAZAPUBLICA (campusParking, centroParking, id, tamanho).

PK: campusParking, centroParking, id.

FK: campusParking, centroParking REFERENCES **PARKING** (campus, centro).

APARCAR (fechaEntrada, fechaSalida, precioHora, matriculaVehiculo, campusParking, centroParking, idPlaza).

CA: tiempo → fechaSalida - fechaEntrada.

CA: precio → tiempo - precioHora.

PK: fechaEntrada, matriculaVehiculo, campusParking, centroParking, idPlaza.

FK: matriculaVehiculo **REFERENCES VEHICULO** (matricula).

FK: campusParking, centroParking, idPlaza **REFERENCES PLAZA PÚBLICA** (campusParking, centroParking, id).

RESERVAR (fechaInicio, fechaFin, precioHora, dniMiembroUSC, campusParking, centroParking, idPlaza).

CA: tiempo → fechaSalida - fechaEntrada.

CA: precio → tiempo - precioHora.

PK: fechaInicio, dniMiembroUSC, campusParking, centroParking, idPlaza.

FK: dniMiembroUSC **REFERENCES MIEMBRO USC** (dni).

FK: campusParking, centroParking, idPlaza **REFERENCES PLAZA PRIVADA** (campusParking, centroParking, id).

ASIGNAR (fechaInicio, fechaFin, dniPDI, campusParking, centroParking, idPlaza).

PK: fechaInicio, dniPDI, campusParking, centroParking, idPlaza.

FK: dniPDI **REFERENCES PDI** (dniMiembroUSC).

FK: campusParking, centroParking, idPlaza **REFERENCES PLAZA PRIVADA** (campusParking, centroParking, id).

3.3. Normalización

Una vez elaborado el modelo relacional de nuestra base de datos podemos pasar a tratar la normalización de la misma. La normalización para las bases de datos relacionales es un proceso de diseño que busca alcanzar una organización lo más eficiente posible. Entre los múltiples beneficios de esta destacamos: la eliminación o reducción de redundancia de datos, la conservación de la integridad y la mejora de la eficiencia global de la base.

El proceso de normalización se basa en la aplicación de reglas específicas, conocidas como formas normales, que ayudan a estructurar las tablas y sus relaciones de manera lógica y coherente. En este caso, nuestro objetivo es alcanzar la forma normal de Boyce-Codd (FNBC), que garantiza la ausencia de redundancia y dependencias en nuestro modelo. Para lograrlo, es necesario cumplir con la 3^a forma normal y, además, garantizar la inexistencia de dependencias funcionales no triviales en los atributos que no formen parte de un conjunto de clave candidata.

Cuando iniciamos la normalización del modelo se ve a priori que está en una 1^a FN, ya que todos los atributos tienen valores atómicos y no hay grupos repetitivos de atributos. Para pasar a 2^a FN, todos los atributos que no actúen como clave deben depender completamente de las claves primarias, cosa que también sucede. Finalmente examinamos el modelo en busca de todas las dependencias transitivas (proceso en el que no entraremos aquí) para eliminarlas en caso de existir. Tras esta secuencia de pasos nos encontraremos en 3^a FN y seremos capaces de pasar a la FNBC.

Parte II

**Implementación de la base de
datos y diseño de la aplicación**

Capítulo 4

Base de datos PostgreSQL

A lo largo del siguiente capítulo se estudia la aplicación práctica del modelo lógico diseñado anteriormente. Para ello se crea una base de datos en PostgreSQL. Además, se implementan diversas vistas, consultas y transacciones que enriquezcan las funcionalidades de la base de datos, de cara a la posterior implementación en Java.

4.1. Tablas

En el script de la creación de las tablas, es relevante destacar la implementación de diversas funciones. Por un lado se implementan dos funciones que comprueben la validez de los datos. Se comprueba que se inserten DNI válidos (que coincida correctamente el dígito de control), así como matrículas en el formato español (cuatro dígitos y tres letras).

4.1.1. Funciones

Con el objetivo de incrementar el número de funcionalidades de la base y proporcionar a nuestro proyecto un mayor grado de realismo, hemos decidido desarrollar algunas funciones adicionales que complementan la creación de tablas.

Entre estas tres funciones, validar_dni y validar_matricula se fundamentan en restricciones adicionales elaboradas mediante el uso de expresiones regulares. Ambas aportan una capa adicional de seguridad al verificar que los datos proporcionados por un usuario sean adecuados, al menos en términos de formato. Si el proyecto continuara desarrollándose hacia niveles superiores de complejidad, ambas funciones podrían ser reconstruidas de tal manera que se aseguren de que los documentos ingresados, además de válidos, hayan sido emitidos oficialmente.

La función hash_password, por otro lado, se centra en la seguridad de la propia base de datos frente a amenazas de robo de información. De nuevo, se

trata de un añadido que nos permite vislumbrar parte del realismo de las bases de datos, aunque no llega a ser una solución completa. El cifrado de contraseñas con el algoritmo SHA-512 ilustra tanto los procedimientos aplicados en términos defensivos como el amplio abanico de herramientas disponibles para estructurar y perfeccionar los sistemas de datos en los que nos involucramos

4.2. Vistas

Además de las tablas, resulta de interés crear vistas que proporcionen conjuntos de datos para restringir información y facilitar la elaboración de consultas.

1. Vehiculos y sus respectivos propietarios.

Aquí se busca listar los vehículos que tiene asociado un usuario registrado. Esta vista puede resultar importante a la hora de implementar una aplicación, para que un usuario al logearse pueda ver los vehículos que tiene registrados.

```
create or replace view vehiculos_con_propietarios as
    select v.matricula, v.tipo, v.marca, v.modelo,
    → v.numeroInfracciones, m.dni,
        m.nombreCompleto, m.correoElectronico,
    → m.telefono
        from vehiculos v
            join miembrosusc m on v.dniMiembroUSC = m.dni;
```

2. Devuelve todas la asignaciones de la base de datos.

Mediante esta vista se pueden obtener todas las asignaciones de plazas que tienen los PDI. Mostrándose además la información del miembro, y de la relación de asignación. Puede ser interesante para conocer una vista detallada de todas las asignaciones actuales.

```
create or replace view asignaciones_pdi as
    select a.dniPDI, a.campusParking, a.centroParking,
    → a.idPlaza,
        a.fechaInicio, a.fechaFin,
    → m.nombreCompleto, m.correoElectronico,
        m.telefono
            from asignar a
                join pdi p on a.dniPDI = p.dniMiembroUSC
                    join miembrosusc m on p.dniMiembroUSC = m.dni;
```

3. Devuelve los vehiculos que hayan cometido infracciones.

A la hora de gestionar a la gente que está vetada de la base de datos, puede resultar interesante. De esta forma puede verse de forma descendente a los vehículos con más infracciones. Con esta información podrían aplicarse los vetos pertinentes.

```
create or replace view vehiculos_con_infracciones as
    select v.matricula, v.marca, v.modelo,
    ↵ v.numeroInfracciones, m.dni,
        m.nombreCompleto, m.correoElectronico,
    ↵ m.telefono
        from vehiculos v
        join miembrosusc m on v.dniMiembroUSC = m.dni
        where v.numeroInfracciones > 0
        order by v.numeroInfracciones desc;
```

4. Devuelve todas las plazas publicas y privadas con sus características.

Si se desea mostrar la estructura de un parking puede ser interesante esta consulta. Para obtener la información de todas las plazas que lo conforman. En un nivel más alto de abstracción podría usarse dicha información para realizar una idea esquematizada de lo que es el parking, o todo el sistema de aparcamiento de la USC.

```
create or replace view combined_plazas as (
    select pp.id as numero_plaza, 'Privada' as
    ↵ tipo_plaza,
        pp.tamanho as
    ↵ tamanho_plaza, pp.campusParking, pp.centroParking
        from plazasPrivadas pp
        union all
            select pb.id as numero_plaza, 'Publica' as
    ↵ tipo_plaza, pb.tamanho as tamanho_plaza,
            pb.campusParking, pb.centroParking
            from plazasPublicas pb
);
```

5. Ocupacion actual de plazas publicas. Esta vista resulta de las más interesantes. A la hora de gestionar un aparcamiento lo más importante es gestionar el nivel de ocupación del mismo. Mediante esta vista puede obtenerse esta información, que puede servir al administrador para gestionar restricciones de entrada al parking. También limitarse en la aplicación la capacidad aparcar, o seleccionar un parking.

```

create or replace view ocupacion_actual as (
    select matriculaVehiculo, campusParking,
    → centroParking, idPlaza
        from aparcar
    where fechaSalida is null
);

```

4.3. Consultas

A la hora de implementar las funcionalidades que requiere la aplicación, se necesitan una serie de consultas que permitan obtener y manejar los datos necesarios en cada casuística diseñada. Para esto se hace uso de las vistas anteriormente desarrolladas que facilitan el filtrado de datos.

1. Obtener los datos de registro introducidos por un usuario en la pantalla de registro.

Se usa el correo electrónico y contraseña en la tabla de usuarios para comprobar si el usuario existe. A mayores, si el usuario acumula 5 infracciones entre todos sus vehículos, no se permitirá su acceso en la interfaz de registro, de forma que la consulta devuelve este número. También se devuelve el rol para determinar qué tipo de acceso se obtiene.

```

select m.correoelectronico, m.rol,
→ sum(v.numeroinfacciones)
from miembrosusc m, vehiculos v
where m.dni = v.dnimiembrousc
      and m.correoelectronico =
→ 'josemaria.alonso@usc.es'
group by m.correoelectronico, m.rol ;

```

2. Obtener los datos asociados a todos los vehículos de un usuario concreto para su representación por pantalla.

```

select *
from vehiculos v
where v.dnimiembrousc = '97288336S'

```

3. Obtener la información asociada al usuario para su representación por pantalla.

```
select *
from miembrosusc m
where m.dni = '24451980J'
```

4. Mostrar los datos asociados a todas las plazas de un determinado parking: número de plaza, su ocupación, tipo (privada o pública) y tamaño. Para la realización de esta consulta se usaron vistas ya implementadas. Tal y como se comentó anteriormente, esto puede ser de interés a la hora de implementar una interfaz gráfica que muestre una visión simplificada del parking.

```
select cp.numero_plaza,
       case
           when oa.matriculaVehiculo is null then
               'Libre'
           else 'Ocupada'
       end as ocupacion,
       cp.tipo_plaza,
       cp.tamanho_plaza
  from parkings p
 join combined_plazas cp on p.campus = cp.campusParking and
    p.centro = cp.centroParking
 left join ocupacion_actual oa on cp.campusParking =
    oa.campusParking and
        cp.centroParking = oa.centroParking and
        cp.numero_plaza = oa.idPlaza
 where p.campus = 'Lugo' and p.centro = 'Escola Técnica de
    Enxeñaría';
```

5. Mostrar todas las reservas realizadas por un usuario en cada parking, así como las plazas asignadas. Esta consulta es de interés en el desarrollo de la aplicación, para poder mostrarle al usuario su información.

```
select r.fechaInicio, r.fechaFin, p.campus, p.centro
  from reservar r
 left join plazasPrivadas pp on r.campusParking =
    pp.campusParking and
        r.centroParking = pp.centroParking and r.idPlaza =
    pp.id
 left join plazasPublicas pb on r.campusParking =
    pb.campusParking and
```

```

        r.centroParking = pb.centroParking and r.idPlaza =
        ↵ pb.id
inner join parkings p on p.campus =
        ↵ coalesce(pp.campusParking, pb.campusParking)
            and p.centro = coalesce(pp.centroParking,
        ↵ pb.centroParking)
where r.dniMiembroUSC = '24451980J'
order by r.fechaInicio asc;

```

6. Obtener el numero de vehiculos en funcion de su tipo.

```

select tipo, COUNT(*) as cantidad
from vehiculos
group by tipo;

```

7. Obtener la cantidad de vehículos de cada marca y modelo que han cometido infracciones.

```

select marca, modelo, COUNT(*) as cantidad_infracciones
from vehiculos
where numeroInfracciones > 0
group by marca, modelo
order by modelo asc;

```

8. Obtener el numero de plazas publicas disponibles en un parking concreto.

```

select count(*) as cantidad
from plazasPublicas
where campusParking = 'Santiago Norte'
        and centroParking = 'Facultade de Economia'
        and id not in(select idPlaza
            from aparcar
            where campusParking = 'Santiago Norte' and
        ↵ centroParking = 'Facultade de Economia'
            and fechaSalida is null);

```

9. Obtener cual es el centro con el mayor numero de plazas privadas.

```

select p.campus, p.centro, count(*) as cantidadPrivadas
from parkings p
inner join plazasPrivadas pp on p.campus =
        ↵ pp.campusParking
            and p.centro = pp.centroParking

```

```

        group by p.campus, p.centro
having count(*) = (
    select max(cantidad)
    from (
        select count(*) as cantidad
        from plazasPrivadas
        group by campusParking, centroParking
    ) cantidad
);

```

10. Obtener aquellos parkings en los que un determinado PDI tenga una plaza asignada.

```

select distinct a. pp.campusParking, pp.centroParking
from plazasPrivadas pp
join asignar a on pp.campusParking = a.campusParking
    and pp.centroParking = a.centroParking and pp.id =
    a.idPlaza
where a.dniPDI = '38289664T';

```

11. Consulta que nos devuelve si hay algun vehiculo aparcado en una plaza que no le corresponde por no ser de su tipo (coche o moto).

```

select a.matriculaVehiculo, a.fechaEntrada,
    a.campusParking,
    a.centroParking, a.idPlaza, v.tipo as
    tipo_vehiculo, pp.tamanho as tamanho_plaza
from aparcar a
join plazasPublicas pp on (a.campusParking,
    a.centroParking, a.idPlaza) =
    (pp.campusParking, pp.centroParking, pp.id)
join vehiculos v on a.matriculaVehiculo = v.matricula
where (v.tipo = 'coche' and pp.tamanho = 'moto') or
    (v.tipo = 'moto' and pp.tamanho = 'coche');

```

12. Consultamos el numero de plazas de cada tipo que hay en cada centro.

```

select p.campus, p.centro, COUNT(*) as
    total_plazas_privadas,
    (select COUNT(*) from plazasPublicas where
    campusParking = p.campus and
    centroParking = p.centro) as total_plazas_publicas
from plazasPrivadas pp

```

```

join parkings p on pp.campusParking = p.campus and
→ pp.centroParking = p.centro
group by p.campus, p.centro;

```

13. Consulta que muestra el estado de una plaza determinada de un parkin concreto en una fecha elegida.

```

select
    case
        when a.matriculaVehiculo is not null then
→  'ocupada'
        when r.dniMiembroUSC is not null then
→  'reservada'
            when s.dniPDI is not null then 'asignada'
        else 'libre'
    end as estado_plaza
from
    (select campusParking, centroParking, id, tamanho
     from plazasPublicas
     where campusParking = 'Santiago Sur' and
→ centroParking = 'CIBUS' and id = 17
     union all
     select campusParking, centroParking, id, tamanho
     from plazasPrivadas
     where campusParking = 'Santiago Sur' and
→ centroParking = 'CIBUS' and id = 17) as p
    left join aparcar a on p.campusParking =
→ a.campusParking and p.centroParking = a.centroParking
        and p.id = a.idPlaza and a.fechaEntrada <=
→ '2023-10-30' and (a.fechaSalida is null or
→ a.fechaSalida >= '2023-10-30')
    left join reservar r on p.campusParking =
→ r.campusParking and p.centroParking = r.centroParking
        and p.id = r.idPlaza and r.fechaInicio <=
→ '2023-10-30' and r.fechaFin >= '2023-10-30'
    left join asignar s on p.campusParking =
→ s.campusParking and p.centroParking = s.centroParking
        and p.id = s.idPlaza and s.fechaInicio <=
→ '2023-10-30' and (s.fechafin is null or s.fechafin >=
→ '2023-10-30');

```

14. Consulta que devuelve todas las reservas activas de un usuario en un periodo

de tiempo determinado

```
select *
from reservar
where dniMiembroUSC = 'DNI_USUARIO' and fechaInicio <=
    current_timestamp
and fechaFin >= current_timestamp;
```

15. Consulta que devuelve el porcentaje de ocupación en cada parking.

```
with total_plazas as (
    select campusParking, centroParking, count(*) as
    total
        from (select * from plazasPublicas union all
    select * from plazasPrivadas) as pp
            group by campusParking, centroParking),
ocupadas as (
    select campusParking, centroParking, count(*) as
    ocupadas
        from aparcar
            where fechaSalida is null
                group by campusParking, centroParking)
select p.campus, p.centro,
    round(coalesce ((ocupadas.ocupadas * 100.0) /
    total_plazas.total, 0),2)
        as porcentaje_ocupacion
from parkings p
    left join total_plazas on p.campus =
    total_plazas.campusParking
        and p.centro = total_plazas.centroParking
    left join ocupadas on p.campus =
    ocupadas.campusParking
        and p.centro = ocupadas.centroParking
group by p.campus, p.centro, ocupadas.ocupadas,
    total_plazas.total
order by porcentaje_ocupacion desc;
```

16. Consulta que devuelve el tipo de una plaza determinada.

```
select
    case
        when exists (
            select * from plazasPrivadas
```

```

        where campusParking = 'CampusX'
        and centroParking = 'CentroY'
        and id = 123
    ) then 'Privada'
when exists (
    select * from plazasPublicas
    where campusParking = 'CampusX'
    and centroParking = 'CentroY'
    and id = 123
) then 'Pública'
else 'No encontrada'
end as TipoDePlaza

```

4.4. Transacciones

Las transacciones son elementos de mayor importancia en lo relativo al funcionamiento de la base de datos. A través de las mismas, se permite la gestión y modificación de los registros; garantizando la integridad de la información y preservando la consistencia de la base de datos. De aquí se extraen las principales funcionalidades que están presentes en la aplicación desarrollada.

En primer lugar se presentan las transacciones relacionadas con las funciones que puede realizar un miembro de la USC con el rol de administrador.

1. Introducir los datos de registro de un nuevo usuario en la base de datos.

Esta será una de las funciones principales del administrador, ya que se encarga de ir registrando a todos los miembros de la USC para que puedan utilizar el servicio de parking ofrecido por la universidad. Además el administrador puede asignar el rol de administrador a otro miembro, pudiendo haber varios administradores que realicen estas tareas de gestión.

En esta transacción no es necesario comprobar que el DNI o el correo electrónico estén repetidos en la base de datos. Esto se debe a que el DNI es la clave primaria y el correo una clave candidata, entonces si se intenta añadir un nuevo registro con valores repetidos para estos atributos, SQL revierte la transacción y te devuelve un error.

```

begin transaction;
    insert into miembrosusc (dni, nombreCompleto,
    → correoElectronico, contrasenha, telefono, saldo, rol,
    → fechaIngreso)

```

```

    values ('54654082X', 'José María Alonso Moral',
    ↵ 'josemaria.alonso@usc.es', hash_password('JMAM432'),
    ↵ '881816432', 45, 'usuario', '2008-05-23');
commit transaction;

```

2. Actualizar el número de infracciones de un vehículo.

Cuando un usuario comete una infracción en el parking, el administrador es el encargado de modificar el valor de este atributo de la tabla de vehículos. Además, debe comprobar que si la suma de todas las infracciones de los vehículos de un usuario existente es 5, este será vetado del sistema y no podrá volver a reservar plaza en el mismo.

En el caso de que un usuario vetado quiera aparcar en alguno de los parkings de la USC, se comprobará que haya un usuario asociado a ese vehículo y si está vetado no podrá acceder al aparcamiento. Esta comprobación se hará en la transacción correspondiente a registrar el aparcamiento de un vehículo en un parking. Además hacemos un savepoint después del primer update, para garantizar la integridad de la base de datos. En caso de error se puede revertir la transacción hasta ese punto.

```

begin transaction;
    -- Se actualiza el numero de infracciones de un
    ↵ vehiculo
        update vehiculos set numeroInfracciones =
    ↵ numeroInfracciones + 1 where matricula like '3457RGF';
        -- Guardamos los cambios sobre el atributo
    ↵ numeroInfracciones por si tenemos que revertir la
    ↵ transaccion
        savepoint save1;
        -- Se consulta el numero de infracciones totales del
    ↵ miembro de la USC en cuestion (utilizando una de las
    ↵ vistas)
        with totalInfracciones as (
            select sum(numeroInfracciones) as
    ↵ totalInfracciones
            from vehiculos_con_infracciones
            where dni like '3457RGF')
        -- Si el miembro tiene 5 infracciones totales se le
    ↵ veta del sistema asignando el rol correspondiente
        update miembrosusc set rol = 'vetado' where dni in (
            select dni
            from totalInfracciones

```

```
        where totalInfracciones = 5);
commit transaction;
```

3. Modificar el saldo de los usuarios.

Esta transacción se incluye ya que no hay un método de pago implementado en la aplicación. La única forma de aumentar el saldo de tu cuenta como usuario es proporcionarle al administrador (cuando aparcas) los fondos a ingresar en la misma

```
begin transaction;
    update miembrosusc set saldo = saldo + 100 where dni =
    ↵ '54654082X';
commit transaction;
```

4. Asignar una plaza a un profesor de la universidad.

Los profesores de la USC como son PDI (Personal Docente e Investigador), tienen asignada una plaza en alguno de los parkings de la universidad. Es tarea del administrador asignar esa plaza. A diferencia de los otros tipos de plazas estas se mantendrán asignadas por tiempo indefinido lo que simplifica la transacción.

```
begin transaction;
update miembrosusc set saldo = saldo + 100 where dni =
    ↵ '54654082X';
commit transaction;
```

5. Liberar la plaza asignada a un profesor de la universidad.

Cabe destacar que no se borra la tupla de la tabla, ya que nos interesa mantener la máxima cantidad de datos posible. Únicamente se tendrá que asignar al atributo fechaFin la fecha actual, que será cuando la plaza quedará liberada.

```
begin transaction;
    update asignar
    set fechafin = current_date;
    where dniPDI = '54654082X' and campusParking =
    ↵ 'Santiago Sur' and centroParking = 'ETSE' and idPlaza = 1;
commit transaction;
```

6. Registrar un aparcamiento de un vehículo en un parking.

Cuando cualquier persona aparcá en uno de los parkings de la USC, el administrador registra ese aparcamiento en la base de datos. No recoge datos del usuario del aparcamiento en cuestión, si no que registra al vehículo aparcado. De esta forma se puede asociar un coche con un miembro de la USC.

```

begin transaction;
    insert into aparcar (fechaEntrada, fechaSalida,
    ↵  precioHora, matriculaVehiculo, campusParking,
    ↵  centroParking, idPlaza)
        values ('2018-07-23 12:00:00', null, 1.47, '3457RGF',
    ↵  'Santiago Sur', 'CIBUS', 4);
commit transaction;

```

7. Registrar la salida de un vehículo de un parking.

El administrador además de registrar aparcamientos también debe anotar cuando un vehículo deja una plaza libre, cambiando el valor de fechaSalida en la tupla de la tabla aparcar por la fecha en ese momento.

```

begin transaction;
    update aparcar set fechaSalida = current_date where
    ↵  matriculavehiculo like '3457RGF';
commit transaction;

```

A continuación se muestran las transacciones de las funciones de los usuarios.

8. Ingresar un nuevo vehículo, con todos sus datos asociados.

Esta transacción representa una de las principales funcionalidades de un usuario en la aplicación. Permite añadir vehículos que se asocian a tu cuenta y con los que posteriormente podrás realizar reservas de plazas en los diferentes parkings de la USC.

```

begin transaction;
    insert into vehiculos (matricula, tipo, marca, modelo,
    ↵  numeroInfracciones, dniMiembroUSC)
        values ('3457RGF', 'coche', 'Audi', 'A3', 2,
    ↵  '54654082X');
commit transaction;

```

9. Retirar un vehículo.

De la misma forma que se pueden añadir vehículos a la aplicación también se pueden eliminar. Debido a la ley de protección de datos, cuando se retire un vehículo de la aplicación se actualizará el valor del atributo matrícula a una por defecto (9999 ZZZ). El resto de datos al no ser comprometidos se mantendrán una vez termine la transacción.

```

begin transaction;
    update vehiculos set matricula = default where
→  matricula = '3457RGF';
commit transaction;

```

10. Registrar una reserva de un vehículo en un parking.

Cuando un usuario tiene vehículos registrados en la aplicación podrá realizar una reserva. Cuando se introduzcan valores válidos que no generen inconsistencias en la base de datos (garantizado por las políticas de integridad referencial), se generará un ticket en base a estos atributos. Así se muestra de forma más realista la reserva de una plaza a través de la transacción.

```

begin transaction;
    insert into RESERVAR (fechaInicio, fechaFin,
→  precioHora, dniMiembroUSC, campusParking, centroParking,
→  idPlaza)
        values ('2012-04-09 12:00:00', '2012-04-15 12:00:00',
→  0.89, '54654082X', 'Santiago Sur', 'Facultade de Física',
→  8);
commit transaction;

```

11. Cancelar una reserva de un vehículo en un parking.

Además de reservar el usuario puede cancelar una reserva. Para realizar esta acción, el usuario debe tramitar la anulación de la reserva antes de la fecha de salida del vehículo de la misma. Si se cumple este requisito se le devolverá la mitad del coste de la reserva, ya que se cobran gastos de gestión. En caso de que no se cancele en tiempo, se deberá pagar el importe completo de la reserva. Por último, para indicar que la plaza queda libre, se asigna a la fecha de salida el día de cancelación. Como en la segunda transacción al haber dos updates añadimos un savepoint en medio para poder revertir hasta ese punto.

```

begin transaction;
    -- Se le devuelve la mitad del coste de la reserva al
→  cancelarla
    update miembrosusc set saldo = saldo +
        (select (extract(epoch FROM age(fechaFin, fechaInicio)))
→  / 3600) * precioHora * 0.5
    from reservar
    where fechaInicio = '2022-09-09 20:00:00'
        and dnimiembrousc = '54654082X'
        and campusParking = 'Santiago Sur'

```

```

        and centroParking = 'Facultade de Física'
        and idPlaza = 10
        and extract(epoch from age(fechaFin,
→ current_date)) > 0)
        where dni = '54654082X';

        -- Guardamos los cambios en el saldo por si hay que
→ revertir la transaccion
        savepoint save1;

        -- Actualizamos fechaFin ala fecha en la que se ejecuta
→ la transaccion, de esta forma indicamos que la plaza queda
→ libre
        update reservar set fechaFin = current_date
        where fechaInicio = '2022-09-09 20:00:00'
            and dñimiembrousc like '54654082X'
            and campusparking like 'Santiago Sur'
            and centroparking like 'Facultade de Física'
            and idplaza = 10
            and extract(epoch from age(fechaFin,
→ current_date)) > 0;
commit transaction;

```

12. Modificar los datos de la cuenta.

Los miembros de la USC, podrán realizar cambios sobre algunos de sus datos. Concretamente se podrá modificar el nombre, la contraseña y el teléfono. Para mayor seguridad, en la transacción se debe introducir la contraseña actual antes de modificarla por una nueva. Así introduces una capa más de seguridad en las contraseñas, a mayores del hasheo de las mismas al introducirlas en los registros de la base de datos.

```

begin transaction;
    update miembrosusc
        set nombreCompleto = 'Josema Alonso Moral', contrasenha
→ = hash_password('JAM275'), telefono = '881815439'
        where dni like '54654082X' and contrasenha like
→ hash_password('JMAM432');
commit transaction;

```

13. Eliminar la cuenta.

De la misma forma que ocurría con los vehículos un usuario puede borrar su cuenta. Cuando esta transacción se lleva a cabo únicamente se actualiza el DNI

a un valor por defecto (99999999R), debido a la ley de protección de datos de nuevo. El resto de información de la tupla en cambio permanecerá en la base de datos.

```
begin transaction;
    update miembrosusc set dni = default where dni like
    ↵  '54654082X';
commit transaction;
```

Por último, se presentan transacciones adicionales, que no se implementarán en la aplicación pero que son necesarias para completar la información de la base de datos.

14. Agregar una nueva plaza pública a un parking.

Se necesita una transacción para agregar plazas públicas o privadas a los diferentes parkings de la USC. No se implementará en la aplicación pero se representa la transacción igualmente. En este caso se crea una plaza pública, si se quisiera crear una plaza privada solo habría que cambiar la tabla indicada después del INSERT INTO por plazasprivadas.

```
begin transaction;
    insert into plazaspublicas (campusParking,
    ↵  centroParking, id, tamanho)
        values ('Santiago Sur', 'ETSE', 1, 'coche');
commit transaction;
```

15. Agregar un nuevo parking al conjunto de aparcamientos de la USC.

Para la generación de parkings de la universidad, se necesita otra transacción que no va a ser representada en la aplicación pero que, debe añadirse a la base de datos para completar información.

```
begin transaction;
    insert into parkings (campus, centro, capacidad,
    ↵  maximoPrivadas)
        values ('Santiago Sur', 'ETSE', 22, 0);
commit transaction;
```

16. Eliminar una plaza pública de un parking.

En cuanto al borrado de plazas estamos en la misma situación, no se implementa pero se debe incluir como transacción. Para modificar la eliminación de plazas públicas a plazas privadas deberemos cambiar la tabla plazaspublicas indicada después del DELETE FROM por plazasprivadas. En este caso borraremos la tupla entera, sin mantener ninguna información de las plazas ya que no resulta de interés.

```

begin transaction;
    delete from plazaspublicas
        where campusParking = 'Santiago Sur' and centroParking
    ↵ = 'ETSE' and id = 1;
commit transaction;

```

17. Eliminar un parking del conjunto de aparcamientos de la USC.

Al igual que creamos parkings con una transacción, deberemos representar la eliminación de un parking con otra transacción. De la misma forma que en las plazas, se borra el registro entero asociado al parking, ya que no contiene información tan relevante como la de otras tablas.

```

begin transaction;
    delete from parkings
        where campus = 'Santiago Sur' and centro = 'ETSE';
commit transaction;

```

18. Aumentar en 10€ el saldo de los miembros con una antigüedad superior a 10 años sin infracciones en un vehículo.

Como agradecimiento por no tener infracciones en por lo menos uno de sus vehículos, se premia con 10€ a todos los miembros de la USC que lleven más de 10 años en la institución. Esta transacción no se implementa en Java pero tiene su interés en SQL.

```

begin transaction;
    -- Se actualiza el saldo de los miembros de la USC
    update miembrosusc
        set saldo = saldo + 10

    -- Subconsulta para filtrar los miembros que tienen al
    ↵ menos un vehículo sin infracciones
    from (
        select dni
            from vehiculos_con_propietarios
            where numeroInfracciones = 0
    ) as v

    -- Se obtienen los miembros con una antigüedad superior
    ↵ a 10 años
    where miembrosusc.dni = v.dni

```

```

        and date_part('year', age(current_date, fechaIngreso))
→ > 10;
commit transaction;

```

19. Modificar el rol de un miembro de la USC.

Nos puede interesar en algún momento dado asignar un rol de administrador por ejemplo a otro miembro de la USC. Esta funcionalidad se contempla en la transacción.

```

begin transaction;
    update miembrosusc set rol = 'administrador' where dni
→ like '54654082X';
commit transaction;

```

20. Poner una multa a un miembro de la universidad.

Otra transacción que se puede contemplar en la base de datos es poner una multa a los usuarios del parking en caso de cometer una infracción. Esto se refleja como una reducción del saldo de los mismos, en vez de apuntar el número de infracciones. En el caso de los miembros de la USC, se les reduce el dinero de la cuenta. Si un miembro está en números negativos se le asigna el rol de moroso (no podrá entrar en la aplicación hasta que vuelva a estar en saldo positivo). En el caso de que el usuario no sea miembro de la universidad, se le pasará la multa a la policía (sabiendo la matrícula del vehículo). Después del primer update añadimos un savepoint, para que en caso de revertir la transacción se guarden esos cambios.

```

begin transaction;
    -- Se reduce el saldo del propietario en función de la
→ multa
    update miembrosusc
    set saldo = saldo - 100.0
    where dni in (
        select dni
        from vehiculos_con_propietarios
        where matricula = '3457RGF');

    -- Guardamos los cambios sobre el atributo saldo por si
→ tenemos que revertir la transaccion
    savepoint save1;

    -- Se cambia el rol del miembro de la universidad si el
→ saldo es negativo

```

```
update miembrosusc set rol = 'moroso' where saldo < 0
← and dni in (
    select dni
    from vehiculos_con_propietarios
    where matricula = '3457RGF');
commit transaction;
```


Capítulo 5

Aplicación en Java

Mediante Java y el JDBC se implementó una aplicación que se conecte con la base de datos. Es en este capítulo en el que se estudiará dicha implementación, así como el proceso de diseño de las ventanas y la interfaz de usuario de la misma.

5.1. Interfaz de usuario

En primer lugar el usuario deberá iniciar sesión en la aplicación (figura B.1). Para ello deberá proporcionar un nombre de usuario así como contraseña. De aquí se derivan dos casuísticas: el usuario ha sido vetado o a introducido mal sus credenciales. En ambas situaciones se plantea un mensaje de error, impidiendo el acceso a la propia aplicación.

Si inicia sesión correctamente podrá acceder al menú principal de la aplicación (figura B.2), desde el cual podrá acceder a las diferentes funcionalidades. En total se plantean ocho: ingresar vehículo, retirar vehículo, ver parking, reservar, mis reservas, configuración y salir .

La primera funcionalidad permite al usuario ingresar un vehículo (figura B.3) que ha aparcado. Para ello se le muestra un menú con los diferentes coches que tiene registrados. Una vez selecciona uno que no esté ya aparcado, pasa a la siguiente ventana (figura B.4). Ahora, mediante un menú desplegable podrá seleccionar el Campus en el que se sitúa, y el respectivo parking. Una vez realizado esto, se mostrará una representación esquemática de las propias plazas del parking (figura B.5). De esta forma los usuarios pueden ver a simple vista las plazas ya ocupadas, reservadas, asignadas, o libres. Si realiza todos los pasos correctamente, el sistema generará un "ticket" en formato pdf, con toda la información del coche recién aparcado (figura B.6).

La segunda funcionalidad es la análoga a la anterior. Se trata de la retirada de un vehículo aparcado. También se mostrarán los vehículos del usuario (figura B.7). Una vez seleccionado uno, se pasará a otro menú que resuma la información

de cuándo se ha aparcado, cuanto tiempo, y el precio a pagar (figura B.8). Se debe aceptar dicha información, y en función del saldo actual de la cuenta, se confirmará el pago con éxito (figuras B.9, B.10).

La aplicación permite también ver el estado actual de un parking, tal y como se mostraba en el ingreso de un vehículo cuando se seleccionaba una plaza (figura B.11). De esta forma el usuario puede conocer el estado del parking para decidir si ir a aparcar en él, o en otro (figura B.12).

Como todos los usuarios de la aplicación son miembros de la USC, también se implementa la funcionalidad de las reservas (figura B.13). Es muy similar a la de ingreso de un vehículo (figura B.14). En este caso, antes de la selección de una plaza (figura B.16), el usuario deberá confirmar las fechas de inicio y fin de la reserva (figura B.15). Se realiza la misma gestión de pagos (figuras B.17, ??).

De la mano del apartado comentado ahora, surge la pestaña de consulta de las reservas de un usuario (figura B.19). Donde se le mostrará para qué coche hizo la reserva, así como la información temporal de la misma. Se permite la cancelación de una reserva desde aquí, seleccionando el coche en cuestión (figura B.20).

Por último también se incluye un apartado de ajustes (figura B.21), donde los miembros de la USC puedan modificar sus datos, así como registrar más vehículos en la base de datos.

Capítulo 6

Conclusiones

Durante la realización de este proyecto pudimos tener una visión más completa del desarrollo real de una base de datos y su posterior uso mediante una conexión.

La implementación de una aplicación en Java que se conecte a la base de datos modelada mediante el JDBC brinda la funcionalidad completa para la resolución de la problemática propuesta. El desarrollo de esta aplicación nos permite tener una perspectiva más cercana a la realidad sobre los contenidos dados en esta materia.

Una vez expuesto todo lo trabajado, se puede afirmar que ante la problemática propuesta, la USC dispondrá de una base de datos y una aplicación que solucione la gestión de sus parkings.

Apéndice A

Creación de tablas

Aquí se muestran las declaraciones de tablas.

```
-- Script de creacion de tablas Proyecto BDII:

----- FUNCIONES -----

-- Funcion para llevar a cabo la comprobacion de la validez de
-- un DNI
create or replace function is_valid_dni(dni char(9))
    returns boolean as $$
declare
    dni_num integer;
    dni_letter char(1);
    letter char(1);
    letters char(23) = 'TRWAGMYFPDXBNJZSQVHLCKE';
    pos integer;
begin
    -- Extrae el número y la letra del DNI
    dni_num = substring(dni, 1, 8)::integer;
    dni_letter = substring(dni, 9, 1);
    -- Calcula la letra correcta para el DNI
    pos = dni_num % 23;
    letter = substring(letters, pos + 1, 1);
    -- Comprueba si la letra del DNI es correcta
    if dni_letter != letter then
        return false;
    end if;
    -- Se confirma la validez del DNI
    return true;
```

```

        end;
$$ language plpgsql;

-- Funcion que valida el formato español de las matriculas
create or replace function validar_matricula(matricula text)
returns boolean as $$

begin
    -- Comprueba la estructura de la matricula, 4
    -- numeros seguidos de 3 letras consonantes
    if matricula ~
    '^[0-9]{4}[BCDFGHJKLMNPQRSTVWXYZ]{3}$' then
        return true;
    else
        -- Si no es correcta devuelve false
        return false;
    end if;
end;
$$ language plpgsql;

-- Funcion que hashea las contrasenhas para guardarlas en la
-- base de datos
create or replace function hash_password(password text) returns
text as $$

begin
    return encode(digest(password, 'sha512'), 'hex'); --
    -- Equivalente a 128 char (hexadecimal)
end;
$$ language plpgsql;

```

----- TABLAS -----

```

-- Tabla de miembros USC:
create table miembrosusc(
    dni char(9) not null default '99999999R',
    nombreCompleto varchar(50) not null,
    correoElectronico varchar(60) not null,
    contrasenha char(128) not null,
    telefono integer not null,
    saldo numeric(8,2) not null,
    rol varchar(30) not null,

```

```

fechaIngreso date not null,
constraint dni_es_valido check (is_valid_dni(dni)),
primary key(dni),
unique (correoElectronico, telefono)
);

-- Tabla de PDI
create table pdi(
    dniMiembroUSC char(9) not null,
    primary key(dniMiembroUSC),
    foreign key (dniMiembroUSC) references miembrosusc(dni)
    ↳ on delete set default on update cascade
);

-- Tabla de vehiculos registrados
create table vehiculos(
    matricula char(7) not null default '9999ZZZ',
    tipo varchar(10) not null,
    marca varchar(30) not null,
    modelo varchar(30) not null,
    numeroInfracciones integer not null,
    dniMiembroUSC char(9) not null,
    constraint matricula_valida check
    ↳ (validar_matricula(matricula)),
    constraint tipo_vehiculo check (tipo like 'coche' or
    ↳ tipo like 'moto'),
    constraint dni_es_valido check
    ↳ (is_valid_dni(dniMiembroUSC)),
    primary key(matricula),
    foreign key (dniMiembroUSC) references miembrosusc(dni)
    ↳ on delete set default on update cascade
);

-- Tabla de parkings
create table parkings(
    campus varchar(20) not null,
    centro varchar(50) not null,
    capacidad integer not null,
    maximoPrivadas integer not null,
    primary key (campus, centro)
);

```

```

-- Tabla de plazas privadas
create table plazasPrivadas(
    campusParking varchar(20) not null,
    centroParking varchar(50) not null,
    id integer not null,
    tamanho varchar(10),
    constraint validar_tamanho check (tamanho like 'coche'
    → or tamanho like 'moto'),
    primary key (campusParking, centroParking, id),
    foreign key (campusParking, centroParking) references
    → parkings(campus, centro) on delete no action on update
    → cascade
);

-- Tabla de plazas publicas
create table plazasPublicas(
    campusParking varchar(20) not null,
    centroParking varchar(50) not null,
    id integer not null,
    tamanho varchar(10) not null,
    constraint validar_tamanho check (tamanho like 'coche'
    → or tamanho like 'moto'),
    primary key (campusParking, centroParking, id),
    foreign key (campusParking, centroParking) references
    → parkings(campus, centro) on delete no action on update
    → cascade
);

-- Tabla de la relacion aparcar
create table aparcar(
    fechaEntrada timestamp not null,
    fechaSalida timestamp,
    precioHora numeric(6,2) not null,
    matriculaVehiculo char(7) not null,
    campusParking varchar(20) not null,
    centroParking varchar(50) not null,
    idPlaza integer not null,
    primary key(fechaEntrada, matriculaVehiculo,
    → campusParking, centroParking, idPlaza),
    foreign key (matriculaVehiculo) references
    → vehiculos(matricula) on delete set default on update
    → cascade,

```

```

        foreign key (campusParking, centroParking, idPlaza)
    ↵ references plazasPublicas(campusParking,
    ↵ centroParking, id) on delete no action on update cascade
);

-- Tabla de la relacion reservar
create table reservar(
    fechaInicio timestamp not null,
    fechaFin timestamp not null,
    precioHora numeric(6,2) not null,
    dniMiembroUSC char(9) not null,
    campusParking varchar(20) not null,
    centroParking varchar(50) not null,
    idPlaza integer not null,
    primary key (fechaInicio, dniMiembroUSC, campusParking,
    ↵ centroParking, idPlaza),
    foreign key (dniMiembroUSC) references miembrosusc(dni)
    ↵ on delete set default on update cascade,
    foreign key (campusParking, centroParking, idPlaza)
    ↵ references plazasPrivadas(campusParking, centroParking, id)
    ↵ on delete no action on update cascade
);

-- Tabla de la relacion asignar
create table asignar(
    fechaInicio date not null,
    fechaFin date,
    dniPDI char(9) not null,
    campusParking varchar(20) not null,
    centroParking varchar(50) not null,
    idPlaza integer not null,
    primary key (fechaInicio, dniPDI, campusParking,
    ↵ centroParking, idPlaza),
    foreign key (dniPDI) references pdi(dniMiembroUSC) on
    ↵ delete set default on update cascade,
    foreign key (campusParking, centroParking, idPlaza)
    ↵ references plazasPrivadas(campusParking, centroParking, id)
    ↵ on delete no action on update cascade
);

```


Apéndice B

Interfaz gráfica

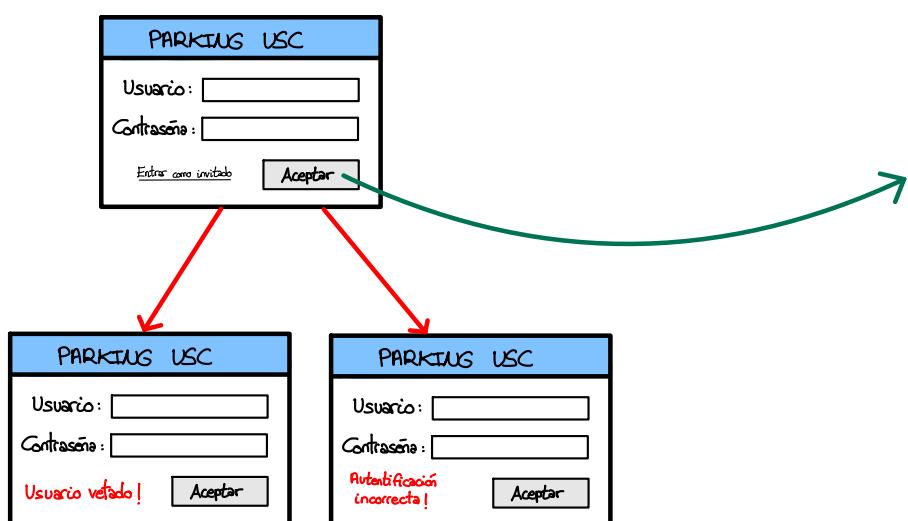


Figura B.1: Interfaz gráfica: inicio de sesión.

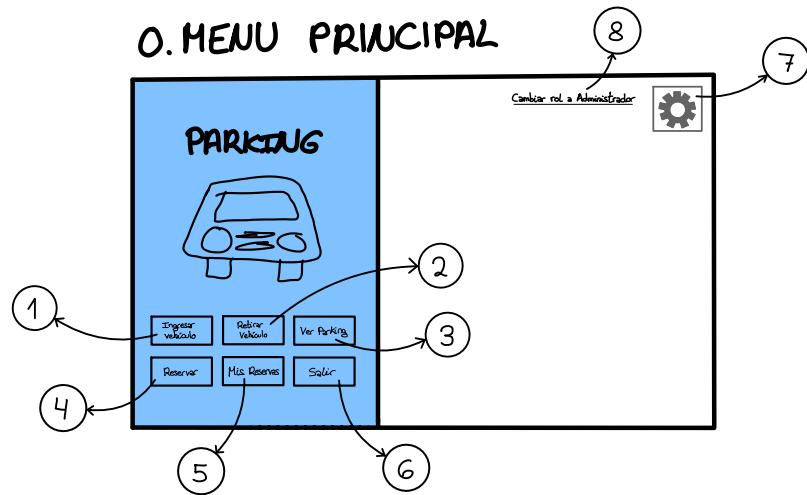


Figura B.2: Interfaz gráfica: menú principal.

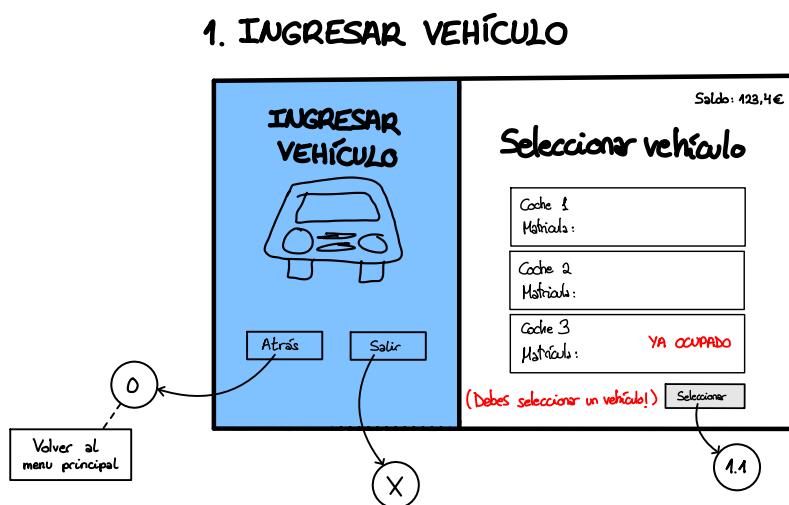


Figura B.3: Interfaz gráfica: ingresar vehículo.

1.1. INGRESAR VEHÍCULO (con vehículo ya seleccionado)



Figura B.4: Interfaz gráfica: ingresar vehículo. Seleccionar parking.

1.1.1. INGRESAR VEHÍCULO (Con vehículo y parking ya seleccionados)

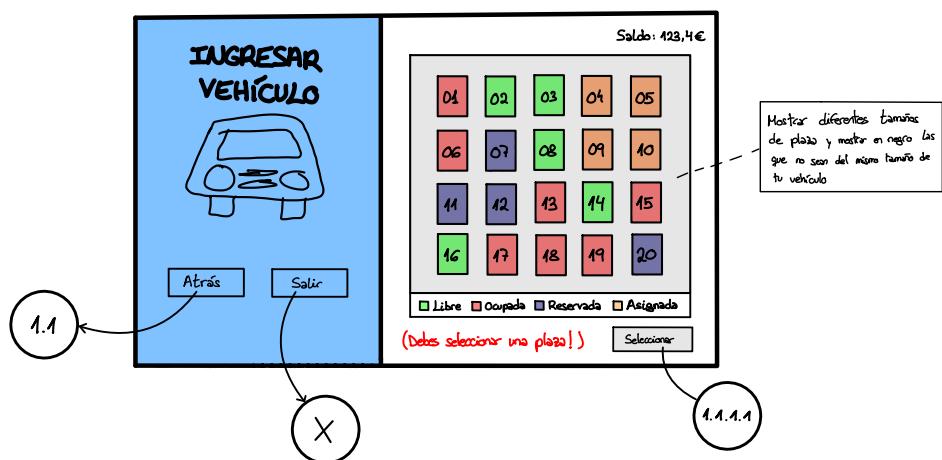


Figura B.5: Interfaz gráfica: ingresar vehículo. Seleccionar plaza.

1.1.1.1. INGRESAR VEHÍCULO (Con vehículo y plaza seleccionadas)

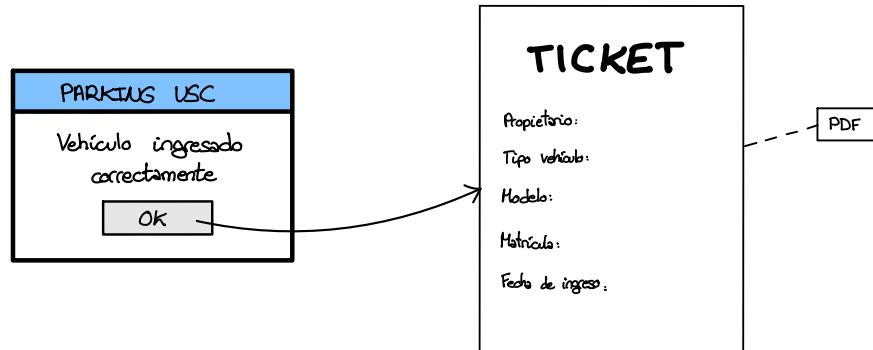


Figura B.6: Interfaz gráfica: ingresar vehículo. Finalización.

2. RETIRAR VEHÍCULO

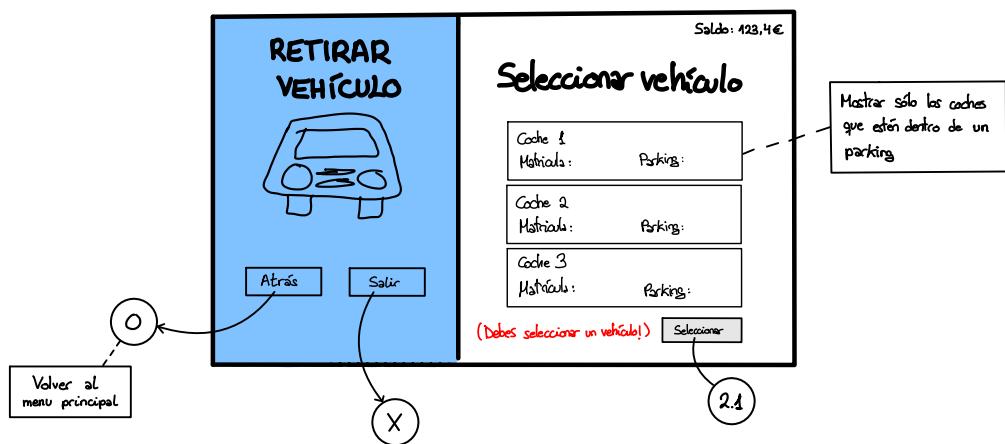


Figura B.7: Interfaz gráfica: retirar vehículo.

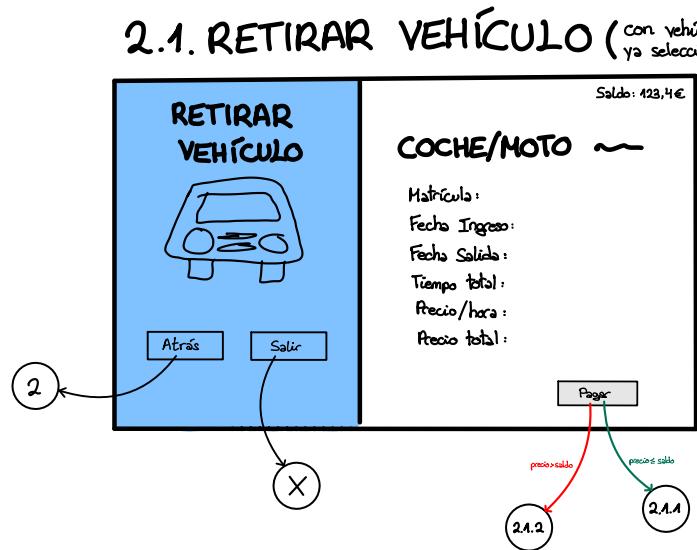


Figura B.8: Interfaz gráfica: retirar vehículo. Información del aparcamiento.

2.1.1. RETIRAR VEHÍCULO (queriendo pagar)

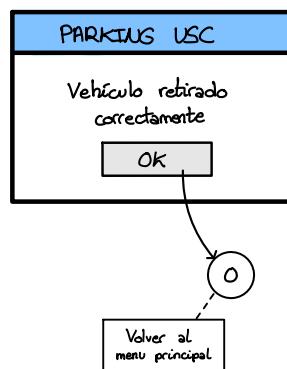


Figura B.9: Interfaz gráfica: retirar vehículo. Pago.

2.1.2. RETIRAR VEHÍCULO (^{queriendo pagar})



Figura B.10: Interfaz gráfica: retirar vehículo. Sin saldo.

3. VER PARKING

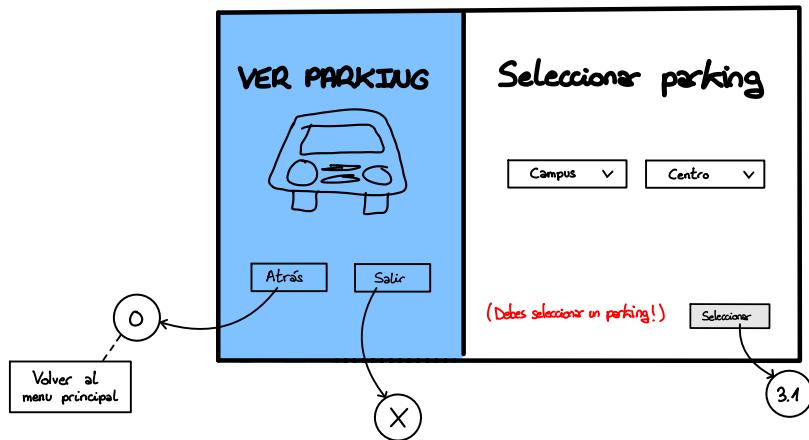


Figura B.11: Interfaz gráfica: ver parking.

3.1. VER PARKING (con parking ya seleccionado)

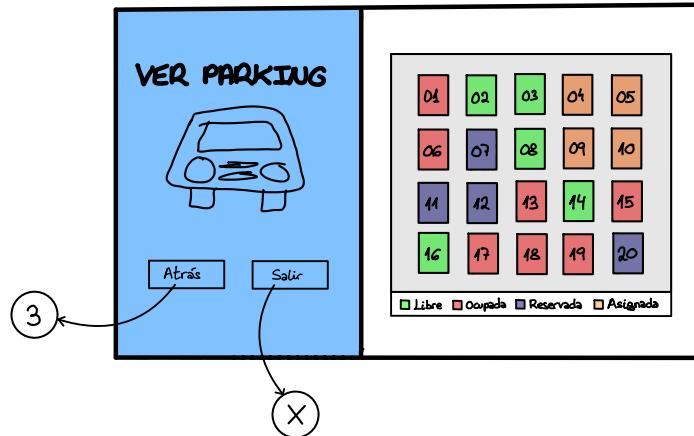


Figura B.12: Interfaz gráfica: ver parking. Representación de plazas.

4. RESERVAR

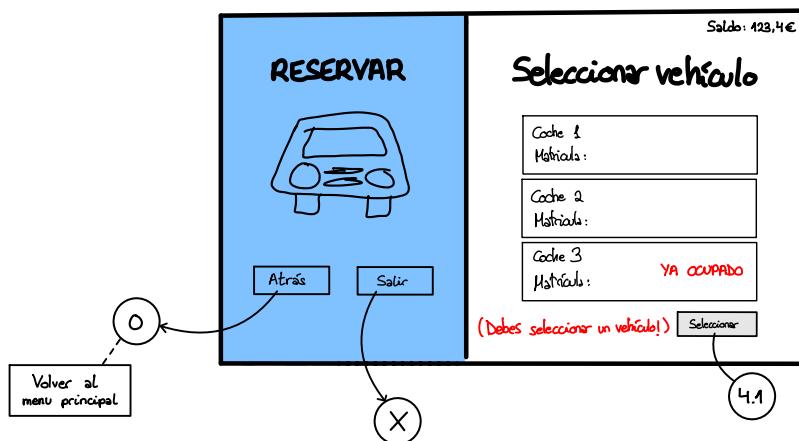


Figura B.13: Interfaz gráfica: reservar.

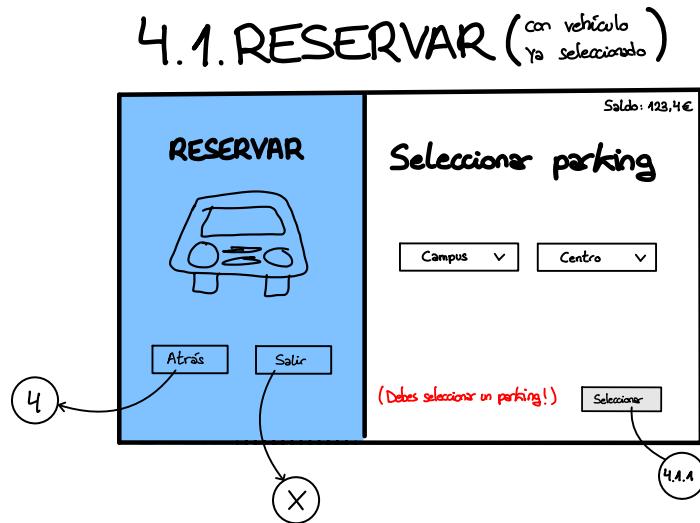


Figura B.14: Interfaz gráfica: reservar. Seleccionar parking.



Figura B.15: Interfaz gráfica: reservar. Seleccionar horario.

4.1.1.1. RESERVAR (con vehículo, parking y hora ya seleccionados)

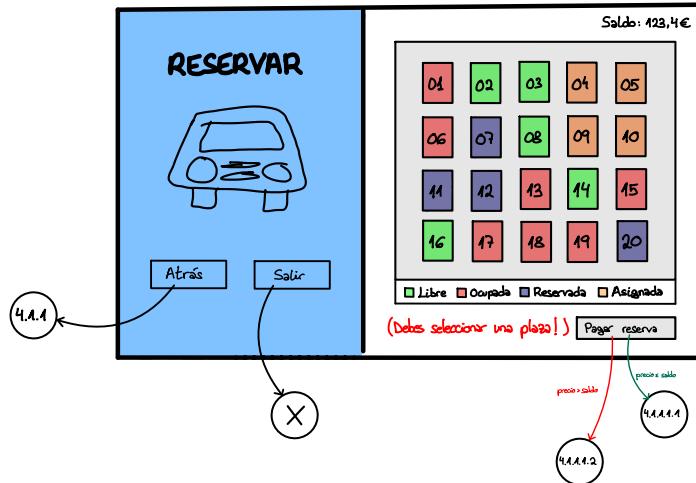


Figura B.16: Interfaz gráfica: reservar. Seleccionar plaza.

4.1.1.1.1. RESERVAR (queriendo pagar)

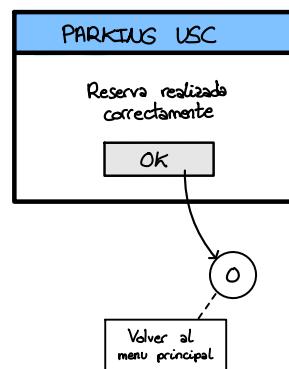


Figura B.17: Interfaz gráfica: reservar. Pago.

4.1.1.1.2. RESERVAR (^{queriendo pagar})

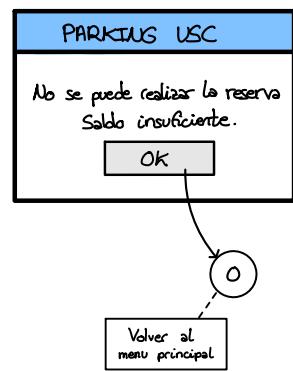


Figura B.18: Interfaz gráfica: reservar. Sin saldo.

5. MIS RESERVAS

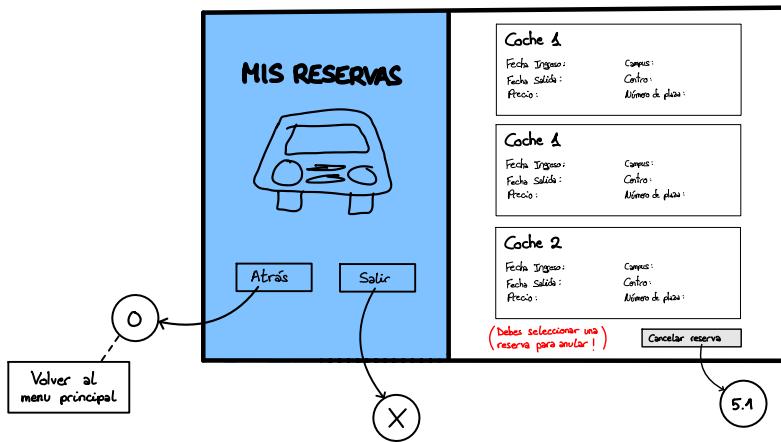


Figura B.19: Interfaz gráfica: mis reservas.

5.1. MIS RESERVAS

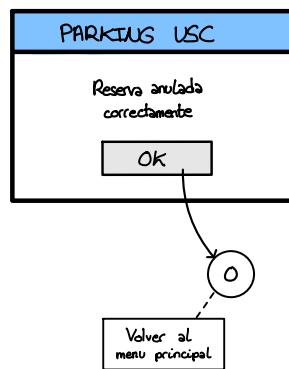


Figura B.20: Interfaz gráfica: mis reservas. Cancelación.

7. AJUSTES

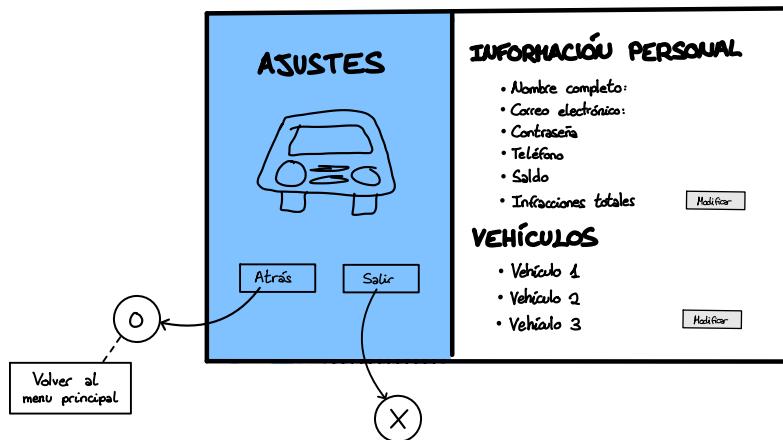


Figura B.21: Interfaz gráfica: ajustes.