LoCHDB

Gestor de la base de datos de LoCH

Por: Antonio Bañón Serrano

Índice

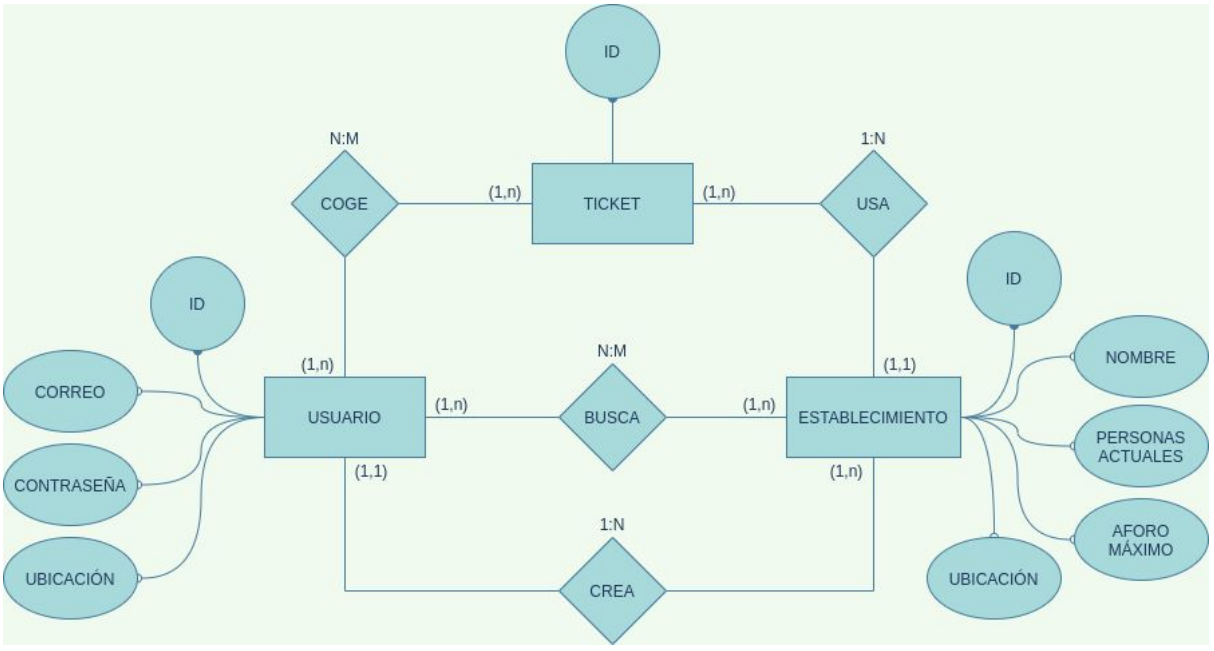
Introducción

Esta es la documentación del programa encargado de gestionar la base de datos de la aplicación LoCH.
LoCH es una aplicación que muestra la cantidad de personas que se encuentran en un mismo establecimiento en tiempo real.

Planificación

Entidad relacional

LoCH es una aplicación de control de aforo: muestra al usuario cuantas personas se encuentran en un establecimiento al mismo tiempo. Por tanto, la aplicación maneja dos entidades: el usuario y el establecimiento.
El usuario puede tanto buscar un establecimiento, para ver su información, como para crear un nuevo establecimiento -- añadir su negocio a la base de datos --.
Además, el usuario puede reservar plaza para entrar a un establecimiento. Eso se gestiona a través de un ticket virtual que el usuario puede pedir y que ofrece prioridad a la hora de ocupar el establecimiento.
Una vez explicado el funcionamiento de la base de datos, entendemos que la entidad relación se estructura así:



Entidades y atributos

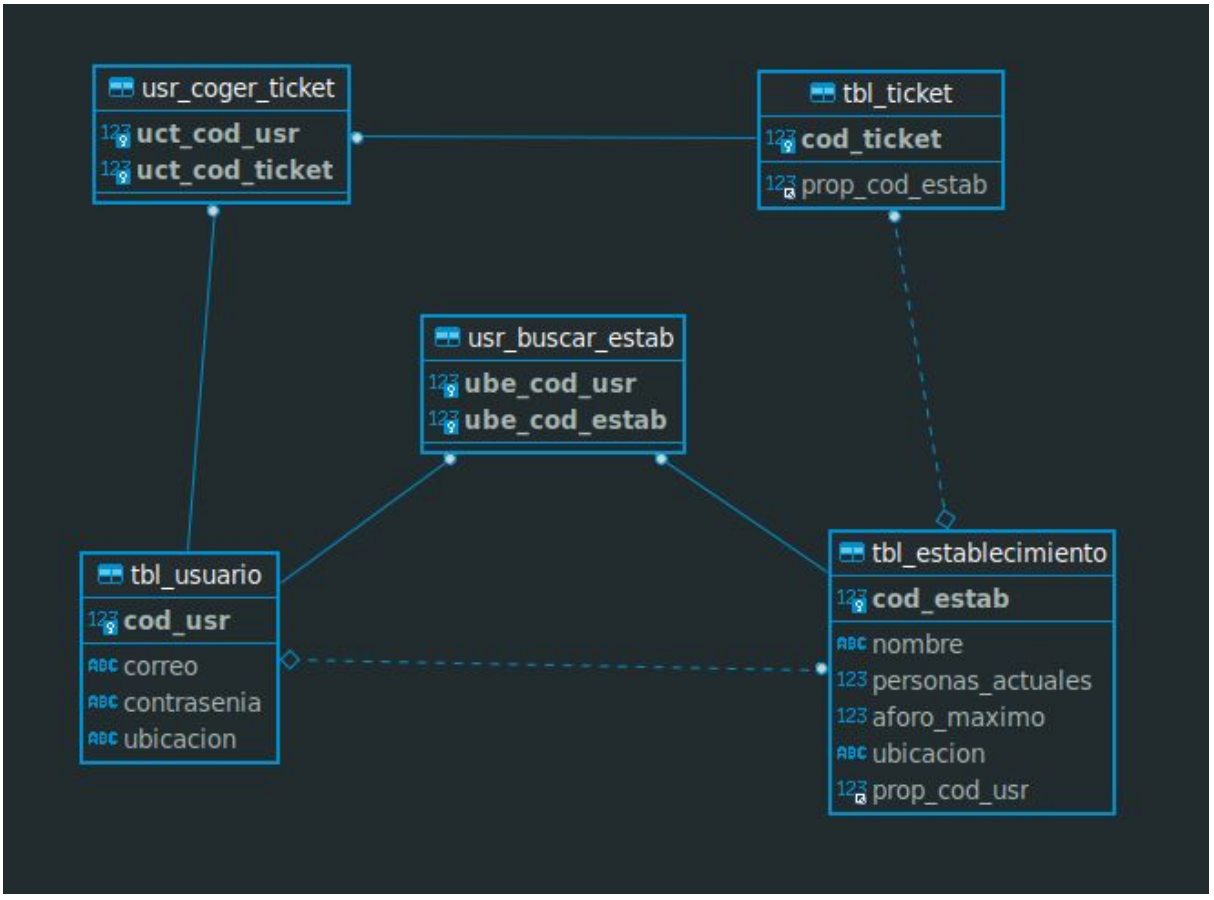
- Usuario
    - ID
    - Correo electrónico
    - Contraseña
    - Ubicación
- Establecimiento
    - ID
    - Nombre
    - Personas actuales
    - Aforo máximo
    - Ubicación
- Ticket
    - ID
    - ID del Establecimiento

Relaciones

Usuario (1,n)	Busca N:M	Establecimiento (1,n)
Usuario (1,1)	Crea 1:N	Establecimiento (1,n)
Usuario	Coge	Ticket
Ticket	Usa	Establecimiento

Representación UML de la Entidad Relacional

Una vez hecho el paso a tablas de la entidad relacional, el UML queda de la siguiente forma:



Implementación

Base de datos

Una vez creada la entidad relacional y el paso a tablas, queda crear las bases de datos. Para este proyecto he elegido tres tipos de gestores distintos: MariaDB, ObjectDB y MongoDB. El usuario encargado tiene libertad de elegir que tipo de base de datos utilizar antes de usar el programa.
Tanto MariaDB como MongoDB se encuentran alojadas en el gestor de contenedores Docker mientras que ObjectDB está embebida dentro de la aplicación.

MariaDB

Es muy útil para manejar tablas con muchos datos. El mapeado de las tablas está hecho con Hibernate.
Las credenciales de MariaDB en Docker son:

- **Server**: localhost:3306
- **Database**: lochdb
- **Username**: root
- **Password**: secret

Usuario

```
CREATE TABLE tbl_usuario (
	cod_usr INT NOT NULL,
	correo VARCHAR(30) NOT NULL,
	contrasenia VARCHAR(20) NOT NULL,
	ubicacion VARCHAR(50) NOT NULL,

	CONSTRAINT pk_usr PRIMARY KEY (cod_usr)
);
```

Establecimiento

```
CREATE TABLE tbl_establecimiento (
	cod_estab INT NOT NULL,
	nombre VARCHAR(20) NOT NULL,
	personas_actuales INT NOT NULL,
	aforo_maximo INT NOT NULL,
	ubicacion VARCHAR(50) NOT NULL,
	prop_cod_usr INT NOT NULL,

	CONSTRAINT pk_estab PRIMARY KEY (cod_estab),
	CONSTRAINT fk_prop_usr FOREIGN KEY (prop_cod_usr) REFERENCES
tbl_usuario(cod_usr) ON DELETE CASCADE
);
```

Ticket

```
CREATE TABLE tbl_ticket (
	cod_ticket INT NOT NULL,
	prop_cod_estab INT,

	CONSTRAINT pk_ticket PRIMARY KEY (cod_ticket),
	CONSTRAINT fk_prop_estab FOREIGN KEY (prop_cod_estab) REFERENCES
tbl_establecimiento(cod_estab) ON DELETE CASCADE
);
```

Y las sentencias para la creación de las tablas que relacionan Usuario--Establecimiento y Usuario--Ticket son

```
CREATE TABLE usr_buscar_estab (
	ube_cod_usr INT NOT NULL,
	ube_cod_estab INT NOT NULL,

	CONSTRAINT pk_usr_estab PRIMARY KEY (ube_cod_usr, ube_cod_estab),
	CONSTRAINT fk_buscar_usr FOREIGN KEY (ube_cod_usr) REFERENCES
tbl_usuario(cod_usr),
	CONSTRAINT fk_buscar_estab FOREIGN KEY (ube_cod_estab) REFERENCES
tbl_establecimiento(cod_estab)
);
```

y

```
CREATE TABLE usr_coger_ticket (
	uct_cod_usr INT NOT NULL,
	uct_cod_ticket INT NOT NULL,

	CONSTRAINT pk_usr_ticket PRIMARY KEY (uct_cod_usr,
uct_cod_ticket),
	CONSTRAINT fk_coger_usr FOREIGN KEY (uct_cod_usr) REFERENCES
tbl_usuario(cod_usr),
	CONSTRAINT fk_coger_ticket FOREIGN KEY (uct_cod_ticket) REFERENCES
tbl_ticket(cod_ticket)
);
```

respectivamente.

ObjectDB

Eficiente para trabajar con pocos datos. Es recomendable usar esta base de datos para hacer pruebas del manejo de datos. Al ser embebido, no se requiere ningún tipo de sentencia SQL pero sí que es necesario definir las tablas con las etiquetas de Java.

Usuario

```
@Entity
public class Usuario implements Serializable {
	@Id
	private int id;
	private String correo;
	private String contrasenia;
	private String ubicacion;

	@ManyToMany (targetEntity=com.ad_proyecto.bbdd_loch.Ticket.class)
	private Set<Ticket> tickets;

	@OneToMany
(targetEntity=com.ad_proyecto.bbdd_loch.Establecimiento.class,
		cascade = {CascadeType.PERSIST, CascadeType.REFRESH},
		mappedBy="usuario")
	private Set<Establecimiento> otm_estabs;

	@ManyToMany
(targetEntity=com.ad_proyecto.bbdd_loch.Establecimiento.class)
	private Set<Establecimiento> mtm_estabs;
	...
}
```

Ticket

```
@Entity
public class Ticket implements Serializable
	@Id
	private int id;
	private transient int id_estab;
	@ManyToOne (optional=false)
	@JoinColumn (name="establecimiento_id", nullable=false,
updatable=false)
	private Establecimiento estab
	@ManyToMany (targetEntity=com.ad_proyecto.bbdd_loch.Usuario.class,
mappedBy="ticket")
	private Set<Usuario> usuarios;
	...
}
```

Establecimiento

```
public class Establecimiento implements Serializable {
	@Id
	private int id;
	private String nombre;
	private int personasActuales;
	private int aforoMaximo;
	private String ubicacion;

	@OneToMany (targetEntity=com.ad_proyecto.bbdd_loch.Ticket.class,
		cascade = {CascadeType.PERSIST, CascadeType.REFRESH},
		mappedBy="establecimiento")
	private Set<Ticket> tickets;

	@ManyToOne (optional=true)
	@JoinColumn (name="usuario_id", nullable=true, updatable=false)
	private Usuario usuario;

	@ManyToMany (targetEntity=com.ad_proyecto.bbdd_loch.Usuario.class,
mappedBy="establecimiento")
	private Set<Usuario> usuarios;
	...
}
```

MongoDB

Este gestor de base de datos, a diferencia de los dos anteriores, es NoSQL lo que permite mayor libertad a la hora de manejar los datos insertados. Es, junto con el anterior, recomendado para ser usado para hacer pruebas de manejo de datos de la aplicación.
Al igual que MariaDB, se usa a través de Docker y las credenciales son:

- **Server**: localhost:27017
- **Database**: LoCHDB
- **Username**: root
- **Password**: secret

DAO

Métodos para la conexión y desconexión con la base de datos:

```
void crearConexion() throws DAOConnectionException;
void cerrarConexion() throws DAOConnectionException;
```

Consultas de usuario, establecimiento y ticket:

```
Usuario consultar(Usuario usuario) throws DAOUsuarioException;
Establecimiento consultar(Establecimiento establecimiento) throws
DAOEstablecimientoException;
Ticket consultar(Ticket ticket) throws DAOTicketException;
```

Inserción de usuario, establecimiento y ticket:

```
void insertar(Usuario usuario) throws DAOInsertarException;
void insertar(Establecimiento establecimiento) throws
DAOInsertarException;
void insertar(Ticket ticket) throws DAOInsertarException;
```

Modificación de datos:

```
void modificar(Usuario usuario) throws DAOModificarException;
void modificar(Establecimiento establecimiento) throws
DAOModificarException;
void modificar(Ticket ticket) throws DAOModificarException;
```

Eliminación:

```
void eliminar(Usuario usuario) throws DAOEliminarException;
void eliminar(Establecimiento establecimiento) throws
DAOEliminarException;
void eliminar(Ticket ticket) throws DAOEliminarException;
```

Métodos para consultas masivas de un mismo tipo de datos:

```
List<Usuario> consultarTodosLosUsuarios() throws DAOUsuarioException;
List<Establecimiento> consultarTodosLosEstablecimientos() throws
DAOEstablecimientoException;
List<Ticket> consultarTodosLosTickets() throws DAOTicketException;
```

Métodos para eliminaciones masivas de datos (todos los datos de una tabla):

```
void eliminarTodosLosUsuarios() throws DAOUsuarioException;
void eliminarTodosLosEstablecimientos() throws
DAOEstablecimientoException;
void eliminarTodosLosTickets() throws DAOTicketException;
```

Métodos que se encargan de comprobar si determinado objeto existe en la base de datos:

```
boolean existe(Usuario usuario);
boolean existe(Establecimiento establecimiento);
boolean existe(Ticket ticket);
```

Estructura del código

El código está estructurado usando un patrón de diseño llamado **Factory**. Cada vez que arranca la aplicación, pide al usuario elija que gestor de base de datos usar durante la ejecución del programa y tras darle a **Aceptar** el objeto encargado de comunicarse con el **modelo** será del tipo del gestor de base de datos elegido. En el código se refleja de la siguiente manera:

```
// Creación del objeto DAO.
private DAO peticion = null;
...
// Definición del objeto DAO según la base de datos elegida.
switch (elegirbbdd.getOpcion()) {
	case 1: // MariaDB
	peticion = new DAOImplementacionHibernate();
	break;

	case 2: // ObjectDB
	peticion = new DAOImplementacionObjectDB();
	break;

	case 3: // MongoDB
	peticion = new DAOImplementacionMongoDB();
	break;
}

try {
	peticion.crearConexion();
} catch (DAOConnectionException ex) {
	...
}
```

De esta forma, un solo código puede ser usado por los tres tipos de implementación del DAO.

Interfaz gráfica

En LoCHDB hay tres tipos de ventanas:

- Elegir gestor de base de datos.
- Visualizar los datos de las tablas.
- Añadir / Modificar los datos de un registro.

Elegir gestor de base de datos

Al arrancar la aplicación, se muestra un JFrame cuyo propósito es dar a elegir al usuario con qué gestor de base de datos trabajar durante la sesión. La interfaz es la siguiente:

Visualizar los datos de las tablas

Tras elegir el gestor de la base de datos, se mostrará un JFrame donde se pueden visualizar los datos de una de las tres tablas disponibles. Las tablas son cambiadas a través del menú desplegable. Si seleccionas una fila, puedes realizar una de las tres funciones: **añadir**, **modificar**, **eliminar**. Si no hay ninguna fila seleccionada, no ocurrirá nada.

Además, en la esquina superior izquierda, se aloja un menú desplegable llamado **Archivo** cuyo subitem **Salir** permite salir de la aplicación.

Añadir / Modificar

Con el fin de ahorrar recursos y tiempo, hay una sola ventana tanto para añadir un nuevo registro como para modificarlo. Nótese que a la hora de modificar, la ID permanecerá bloqueada para evitar conflictos con otras claves primarias si esta se llegase a modificar.
Cada tabla tiene su propia ventana para añadir / modificar sus propios datos. Las ventanas son las siguientes: