Sistemas de información

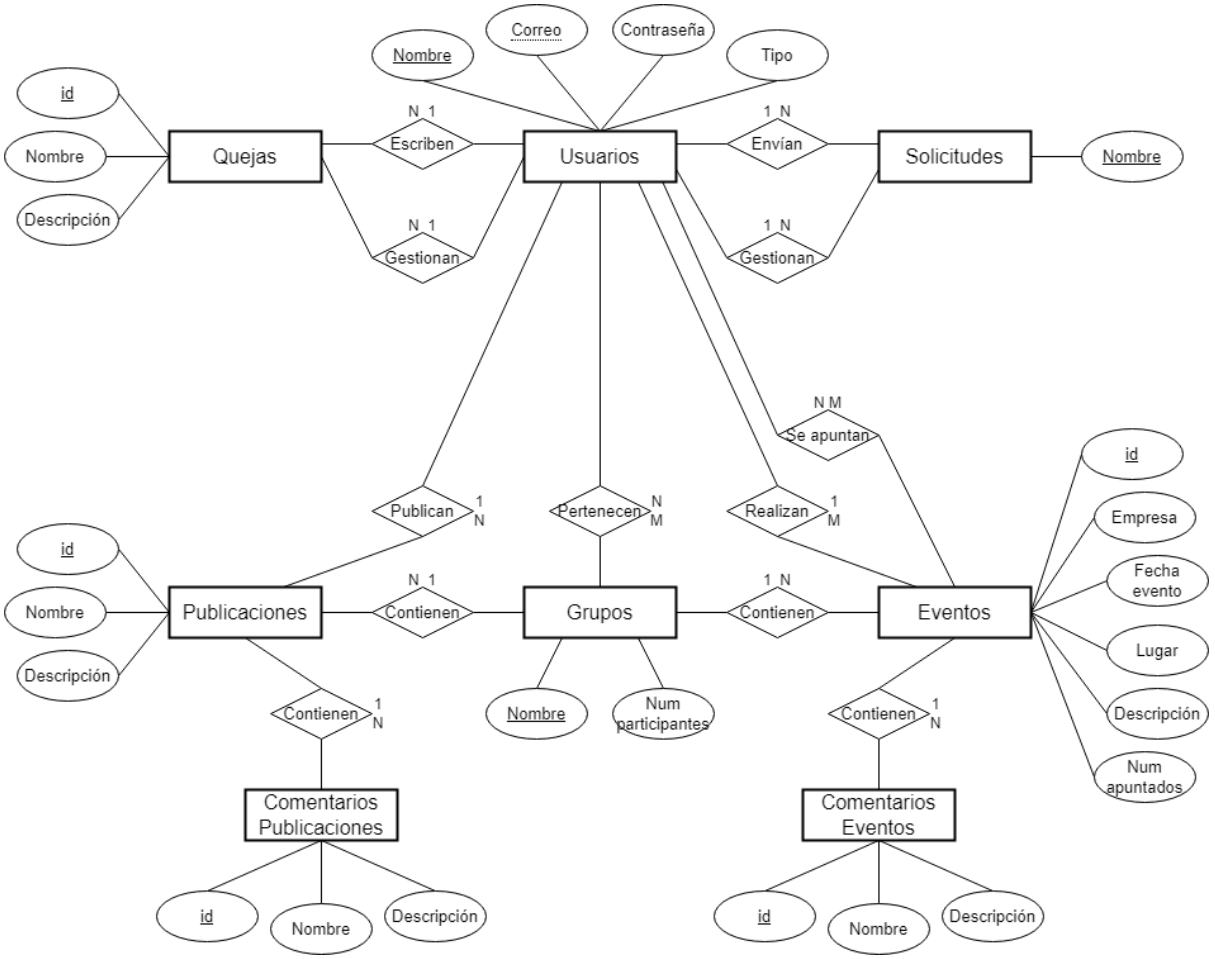
Diseño del modelo de datos

Álvaro Seral Gracia - 819425
Cristian Andrei Selivanov Dobrisan - 816456
Dorian Boleslaw Wozniak - 817570

Índice

Modelo Entidad-Relación

A continuación se muestra el diseño del modelo Entidad-Relación utilizado en la base de datos requerida para almacenar los datos de la aplicación web.



Además, se deben tener en cuenta las siguientes restricciones:

- Solo los Usuarios de tipo Normal pueden enviar Solicitudes para convertirse en Usuarios de tipo Especial, y solo los Usuarios de tipo Administrador pueden gestionar dichas Solicitudes.
- Solo los Usuarios de tipo Normal o Especial pueden escribir Quejas, y solo los Usuarios de tipo Administrador pueden gestionar dichas Quejas.
- Solo los Usuarios de tipo Especial o Administrador pueden realizar Eventos.
- Solo los Usuarios de tipo Normal o Especial pueden publicar Publicaciones o añadir Comentarios tanto de Publicaciones como de Eventos.
- Solo los Usuarios de tipo Normal o Especial pueden apuntarse a Eventos.
- El número de personas apuntadas (Num_apuntados) a un Evento nunca puede ser < 0 .
- El número de participantes de un grupo (Num_participantes) nunca puede ser < 0 .

Modelo Relacional

A partir del modelo Entidad-Relación planteado se ha desarrollado el siguiente esquema relacional. Las tablas Grupos_Usuarios y Apuntados son el resultado de evaluar las relaciones N:M del esquema E-R anterior.

Usuarios (
	Nombre : cadena;
	Correo : cadena, único, no nulo;
	Contrasenya : cadena, no nulo;
	Tipo : cadena("Normal", "Especial", "Administrador"), no nulo);

Grupos (
	Nombre : cadena;
	Num_Participantes : natural, no nulo);

Publicaciones (
	id : entero *con auto-incremento*;
	Nombre : cadena, no nulo, clave ajena de **Usuarios**;
	Descripcion : cadena, no nulo;
	Grupo : cadena, no nulo, clave ajena de **Grupos**);

Eventos (
	id : entero;
	Empresa : cadena, no nulo;
	Fecha_Evento : fecha, no nulo;
	Lugar : cadena, no nulo;
	Descripcion : cadena, no nulo;
	Num_Apuntados : natural, no nulo;
	Grupo : cadena, no nulo, clave ajena de **Grupos**;
	Nombre : cadena, no nulo, clave ajena de **Usuarios**);

Comentarios_Publi (
	id : entero;
	Nombre : cadena, no nulo, clave ajena de **Usuarios**;
	Descripcion : cadena, no nulo;
	Publicacion : entero, no nulo, clave ajena de **Publicaciones**);

Comentarios_Event (
	id : entero;
	Nombre : cadena, no nulo, clave ajena de **Usuarios**;
	Descripcion : cadena, no nulo;
	Evento : entero, no nulo, clave ajena de **Eventos**);

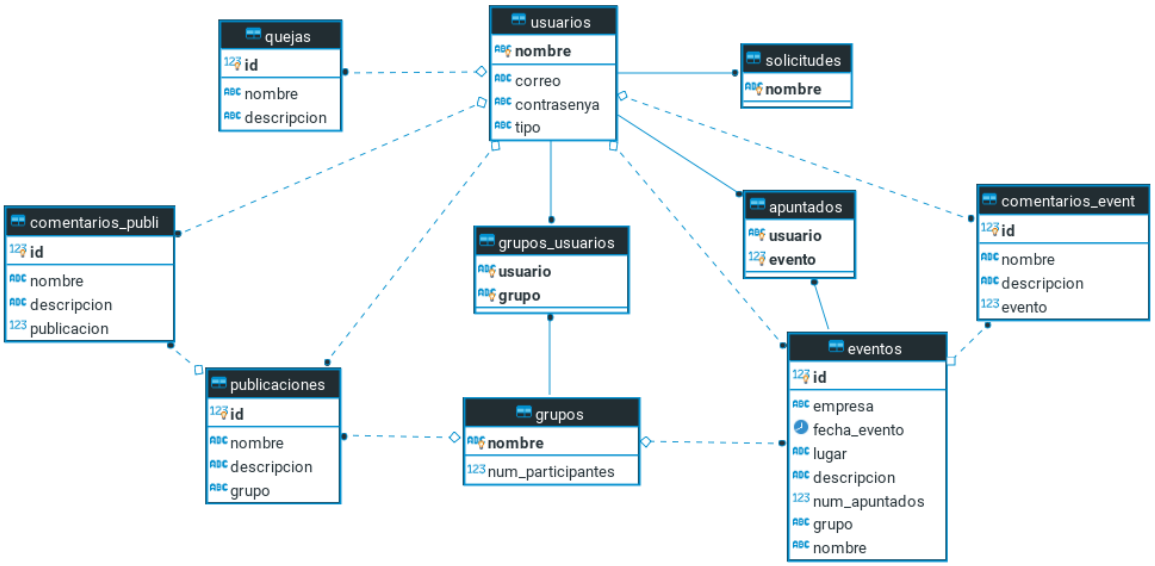
Quejas (
	id : entero;
	Nombre : cadena, no nulo, clave ajena de **Usuarios**;
	Descripcion : cadena, no nulo);

Solicitudes (
	Nombre : cadena, clave ajena de **Usuarios**);

Grupos_Usuarios (
	Usuario : cadena, clave ajena de **Usuarios**;
	Grupo : cadena, clave ajena de **Grupos**);

Apuntados (
	Usuario : cadena, clave ajena de **Usuarios**;
	Evento : cadena, clave ajena de **Eventos**);

Para observar una idea más visual del modelo Relacional se presenta un esquema con los atributos de las tablas, destacando las claves primarias y las relaciones de claves ajenas entre las distintas tablas.



Diseño de las clases implementadas en la capa de persistencia de datos

Los *"DAO"* que componen la capa de persistencia de datos permiten abstraer y encapsular la comunicación con la base de datos, de manera que la capa de lógica de la aplicación no necesita en ningún momento conocer cómo está implementada la base por dentro (tablas) ni a cuál se conecta.

Siguiendo dicho modelo, se ha decidido distribuir los *"DAO"* para cada una de las funcionalidades que nuestro sistema puede llegar a ofrecer, de tal forma que los desarrolladores que implementan la parte lógica de la aplicación sepan intuitivamente donde han de hacer las llamadas a los métodos de los propios "DAO" para obtener la información que deseen de las distintas entidades.

Un ejemplo de funcionalidad sería el sistema de *"Quejas"*, donde los administradores pueden consultar todas las quejas presentadas por los distintos usuarios con el método *"obtenerQuejas(...)"*, los usuarios pueden enviar dichas quejas mediante *"presentarQueja(...)"*, etc.

A continuación se presentan las cabeceras de todos los métodos utilizados en los *"DAO"* diseñados para las distintas funcionalidades que el sistema ofrece:

Funcionalidad de la página de inicio *"InicioDAO"* :

```
// Devuelve true si el nombre de usuario está almacenado en la base de datos
public static boolean existeNombre(UsuariosVO usuario_) {}

// Devuelve true si el correo está almacenado en la base de datos
public static boolean existeCorreo(UsuariosVO usuario_) {}

// Devuelve true si la contraseña es equivalente a la almacenada en la base de datos
para ese usuario
public static boolean iniciarSesion(UsuariosVO usuario_) {}

// Devuelve un vector con todos los grupos existentes
public static ArrayList<GruposVO> mostrarGrupos() {}

// Introduce en la base de datos un nuevo usuario y crea las relaciones necesarias con
los grupos a los que el nuevo usuario se ha unido
public static void crearCuenta(UsuariosVO usuario_, ArrayList<GruposVO> grupos_) {}
```

Funcionalidad relacionada con el sistema de grupos *"GruposDAO"* :

```
// Devuelve un vector de booleanos donde cada elemento vale true si el usuario en
cuestión pertenece al grupo correspondiente, siendo la posición relativa de cada grupo
en el vector resultante la misma que la del método "obtenerGrupos"
public static ArrayList<Boolean> obtenerApuntados(UsuariosVO usuario_) {}

// Devuelve un vector con todos los grupos existentes
public static ArrayList<GruposVO> obtenerGrupos(UsuariosVO usuario_) {}

// Devuelve un vector con todos los grupos a los que pertenece el usuario en cuestión
public static ArrayList<GruposUsuariosVO> obtenerGruposUsuario(UsuariosVO usuario_) {}

// Introduce en la base de datos una nueva relación entre el usuario y el grupo al que
se ha unido
```

```
public static void entrarAGrupo(GruposUsuariosVO grupoUsuario_) {}

// Elimina de la base de datos la relación entre el usuario y el grupo del que ha salido
public static void salirDeGrupo(GruposUsuariosVO grupoUsuario_) {}
```

Funcionalidad relacionada con el sistema de publicaciones *"PublicacionesDAO"*:

```
// Devuelve un vector con todas las publicaciones de un grupo concreto ordenadas
cronológicamente
public static ArrayList<PublicacionesVO> obtenerPublicacionesCrono(GruposVO grupo_) {}

// Introduce en la base de datos una nueva publicación
public static void publicarPublicacion(PublicacionesVO publicacion_) {}

// Devuelve un vector con todos los comentarios asociados a una publicación concreta y
ordenados cronológicamente
public static ArrayList<ComentariosPubliVO> obtenerComentarios(PublicacionesVO
publicacion_) {}

// Introduce en la base de datos un nuevo comentario de publicación
public static void publicarComentario(ComentariosPubliVO comentario_) {}
```

Funcionalidad relacionada con el sistema de eventos *"EventosDAO"*:

```
// Introduce en la base de datos una nueva relación entre el usuario y el evento al que
ha apuntado
public static void apuntarseEvento(ApuntadosVO apuntado_) {}

// Devuelve un vector de booleanos donde cada elemento vale true si el usuario en
cuestión se ha apuntado al evento correspondiente de un grupo concreto, siento la
posición relativa de cada evento en el vector resultante la misma que la del método
"obtenerEventosCrono"
public static ArrayList<Boolean> obtenerApuntadosCrono(GruposVO grupo_, UsuariosVO
usuario_) {}

// Devuelve un vector de booleanos donde cada elemento vale true si el usuario en
cuestión se ha apuntado al evento correspondiente de un grupo concreto, siento la
posición relativa de cada evento en el vector resultante la misma que la del método
"obtenerEventosPorApuntados"
public static ArrayList<Boolean> obtenerApuntadosPorApuntados(GruposVO grupo_,
UsuariosVO usuario_) {}

// Devuelve un vector con todos los eventos de un grupo concreto ordenados
cronológicamente
public static ArrayList<EventosVO> obtenerEventosCrono(GruposVO grupo_, UsuariosVO
usuario_) {}

// Devuelve un vector con todos los eventos de un grupo concreto ordenados según el
número de personas apuntadas en cada evento
public static ArrayList<EventosVO> obtenerEventosPorApuntados(GruposVO grupo_,
UsuariosVO usuario_) {}

// Introduce en la base de datos un nuevo evento
public static void publicarEvento(EventosVO evento_) {}

// Devuelve un vector con todos los comentarios asociados a un evento concreto y
ordenados cronológicamente
public static ArrayList<ComentariosEventVO> obtenerComentarios(EventosVO evento_) {}

// Introduce en la base de datos un nuevo comentario de evento
```

```
public static void publicarComentario(ComentariosEventVO comentario_) {}
```

Funcionalidad relacionada con el sistema de solicitudes *"SolicitudesEspecialesDAO"* :

```
// Devuelve true si el nombre de usuario está almacenado en la base de datos. Además, si
el rol de dicho usuario es "Especial", se actualiza a "Normal"
public static boolean quitarPrivilegios(UsuariosVO usuario_) {}

// Dado un usuario cuyo rol es "Normal", se actualiza dicho rol a "Especial".
Posteriormente, se elimina la solicitud de la tabla correspondiente
public static void darPrivilegios(UsuariosVO usuario_) {}

// Dado un usuario cuyo rol es "Normal", se elimina la solicitud de la tabla
correspondiente (sin cambiar su tipo de rol)
public void negarSolicitud(UsuariosVO usuario_) {}

// Devuelve un vector con todas las solicitudes pendientes de gestionar
public ArrayList<SolicitudesVO> obtenerSolicitudes() {}

// Introduce en la base de datos una nueva solicitud de usuario especial
public void solicitarPrivilegios(UsuariosVO usuario_) {
```

Funcionalidad relacionada con sistema de quejas *"QuejasDAO"* :

```
// Devuelve un vector con todas las quejas pendientes de gestionar
public static ArrayList<QuejasVO> obtenerQuejas() {}

// Introduce una nueva queja en la base de datos
public static void presentarQueja(QuejasVO queja_) {}

// Se elimina la queja en cuestión de la tabla correspondiente
public static void eliminarQueja(QuejasVO queja_) {}
```

Funcionalidad para deshabilitar cuenta *"EliminarCuentaDAO"* :

```
// Se elimina la cuenta de un usuario de la base de datos
public static void eliminarUsuario(UsuariosVO usuario_) {}
```

Funcionalidad para el sistema de estadísticas de la aplicación "AdministradoresDAO" :

```
// Devuelve un vector con todos los eventos de un grupo concreto ordenados
cronológicamente
public static ArrayList<EventosVO> verEventosCrono(GruposVO grupo_, UsuariosVO usuario_)
{}

// Devuelve un vector con todos los eventos de un grupo concreto ordenados según el
número de personas apuntadas en cada evento
public static ArrayList<EventosVO> verEventosPorApuntados(GruposVO grupo_, UsuariosVO
usuario_) {}

// Devuelve el número total de usuarios (cuentas) registrados en la base de datos
public static int verTotalUsuarios() {}

// Devuelve el número de total de participantes correspondientes a un grupo concreto
public static int verTotalParticipantes(GruposVO grupo_) {}

// Devuelve el evento de un grupo concreto con el mayor número de usuarios apuntados
desde siempre
public static EventosVO verMejorEventoSiempre(GruposVO grupo_) {}
```

```
// Devuelve el evento de un grupo concreto con el mayor número de usuarios apuntados
durante el año actual
public static EventosVO verMejorEventoAnyo(GruposVO grupo_) {}

// Devuelve el evento de un grupo concreto con el mayor número de usuarios apuntados
durante el mes actual
public static EventosVO verMejorEventoMes(GruposVO grupo_) {}
```

Metodología

Planificación y herramientas utilizadas

En una primera instancia se realizó el boceto en papel del modelo Entidad-Relación de la base de datos, para posteriormente refinarlo y pasarlo a limpio a través de la herramienta web *draw.io.*

Posteriormente, se tradujo el anterior esquema Entidad-Relación en un modelo Relacional que estableció las tablas necesarias que requeriría la base de datos. Dicho modelo se realizó mediante sentencias escritas en un documento de texto.

Para almacenar los consecuentes ficheros .java que se implementaran se creó un nuevo proyecto java donde se añadió el *Driver JDBC PostgresArchivo* proporcionado por los profesores.

A través de la interfaz de eclipse, más concretamente mediante la perspectiva DBeaver, se añadieron y configuraron todas las tablas de la base de datos, con sus respectivas columnas, restricciones y claves ajenas. El script entregado *crearTablas.sql* es el código generado por la interfaz automáticamente para la creación de dichas tablas. Además, a través de esta perspectiva también se creó el rol "usuario" con su respectiva contraseña, de modo que esté no tuviese permisos para crear tablas, crear nuevos roles, etc. si no únicamente para consultar las tablas existentes o insertar y eliminar filas de dichas tablas.

El siguiente punto que se realizó fue la creación del PoolConnectionManager con la configuración necesaria de los ficheros server.xml y context.xml relativos al servidor Tomcat. Esto se llevó a cabo en la perspectiva de java de eclipse.

A través de la misma perspectiva, se implementaron todos los ficheros VO necesarios para todas las tablas. Consecuentemente, también se crearon los patrones DAO requeridos para la aplicación web. De estos últimos se probaron las sentencias sql más complejas a través de la terminal para asegurar el correcto resultado de dichas consultas.

Principales dificultades

A la hora de implementar los distintos DAO surgió el problema de cómo diseñar la distribución de las clases (por funcionalidades, por tablas, por derechos de usuario, etc.). Al final se decidió distribuirlo por funcionalidades para facilitar a un observador externo que no conozca la estructura de la base de datos la comprensión visual de lo que hay en cada clase.

También surgieron problemas a la hora de realizar pruebas básicas con un DAO de ejemplo para comprobar su correcta estructura. Se resolvió conectándose, únicamente para la prueba de ejemplo, a través del *ConnectionManager* (no el *PoolConnectionManager*) y ejecutándolo desde un proyecto java como *java application*.

Se requirió informarse sobre postgresql para aprender a utilizar correctamente el tipo *date* utilizado para *fecha_evento* en la base de datos.

Distribución de trabajo y tareas

	Álvaro	Cristian	Dorian	**Total**
Modelo Entidad-Relación y Modelo Relacional	4	3	2	**9**
Creación de la base de datos	5	3	2	**10**
Patrones VO y DAO	9	9	4	**22**
Pruebas	2	2	1	**5**
Total	**20**	**17**	**9**	**46**

Bibliografía

Referencias utilizadas para la comprensión e implementación de los ficheros VO y DAO, al igual que la configuración del PoolConnectionManager:

- https://moodle.unizar.es/add/pluginfile.php/7072995/mod_resource/content/2/PatronDAO.pdf
- Código de ejemplo proporcionado por los profesores a través de la plataforma moodle

Referencias empleadas para utilizar el campo *fecha_evento* correctamente dentro de las consultas empleadas en los DAO:

- https://dba.stackexchange.com/
- https://stackoverflow.com/
- https://database.guide/