Opciones obligatorias

La BD debe constar de al menos tres tablas relacionadas entre sí.

- En el documento **clases.pdf** se puede observar que la aplicación dispone de 4 tablas relacionadas.
- En el proyecto, en el paquete **dam.samuel.modelo**, se encuentran las clases **Desarrollo**, **Empresa**, **Juego** y **Valoración** que implementan la interfaz Serializable para que sus objetos puedan ser persistidos.

Deben existir relaciones 1:N (al menos 1)

- En el documento **clases.pdf** se muestra la cardinalidad de las relaciones entre las tablas. Ahí se muestra como todas las tablas **Empresa** y **Juego** poseen relaciones de **uno a muchos** con otras tablas.

Debe existir algún campo autonumérico.

- Todas las tablas poseen el campo **id** respectivo a su clase, el cual es un **número** entero **auto incrementado**.

Tipo enumerado en alguna tabla

- En la clase Juego existe el tipo **enum EstiloJuego**, que en la base de datos se guarda como **VARCHAR estilo**.

Validaciones con Hibernate Validator en todos los datos.

- En las clases situadas en **dam.samuel.modelo** se puede observar cómo se usa la validación de Hibernate en los campos de datos persistidos.

Deben poder realizarse operaciones de inserción en todas las tablas de la BD (al menos 3)

- En el paquete **dam.samuel.dao** se encuentran las clases DAO de cada clase persistida y todos heredan de **GenericDAO**, el cual permite realizar la acción de inserción del objeto en la base de datos.

Deben poder realizarse operaciones de consultas de todas las tablas de la BD (al menos 3).

- En el paquete **dam.samuel.dao** se encuentran las clases DAO de cada clase persistida y todas tienen al menos un método de consulta para su clase de objetos.

Deben poder realizarse operaciones de consultas donde intervengan varias tablas.

- En el paquete **dam.samuel.dao** se encuentra la clase **ValoracionDAO** que posee un método de consulta de objetos Valoracion que pertenezcan a un objeto Juego con id determinado. Se usa la cláusula **INNER JOIN** entre las tablas **valoracion** y **juego**.

Deben poder realizarse operaciones de modificación de al menos una tabla de la BD

- El paquete **dam.samuel.vista** contiene el fichero **ControladorDetallesJuego** con el método **guardar()** (línea 180) que permite al administrador actualizar los datos del

juego seleccionado usando el método **actualizar(T entidad)** situado en el fichero **GenericDAO** en dam.samuel.DAO.

Deben poder realizarse operaciones de baja de al menos una tabla de la BD.

- El paquete **dam.samuel.vista** contiene el fichero **ControladorDetallesJuego** con el método **borrar()** (línea 139) que permite al administrador eliminar una valoración sobre el juego seleccionado hecha por un usuario a través del método **borrar(T entidad)** situado en el fichero **GenericDAO** en dam.samuel.DAO.

Utilización del patrón DAO.

- En las opciones anteriormente explicadas se ha mostrado el uso del patrón DAO en la aplicación para todas las clases que serán persistidas.

Gestión de transacciones

- El fichero **GenericDAO** usa la gestión de transacciones para sus tres métodos (save(Object arg0), update(Object arg0) y delete(Object arg0)).

Opciones no obligatorias implementadas

Existencia de una relación N:M implementada como dos relaciones y 1:N.

- Las clases **Empresa** y **Juego** poseen una relación **N:M** de la que ha surgido **Desarrolla** como una doble relación **1:N** con las dos anteriores.

Tipo fecha en alguna tabla.

- La clase **Juego** posee el tipo de dato **DATE** para el atributo **publicación** el cual almacena la fecha en la que el juego fue o será publicado.

Métodos java de validación. Se indicará en qué lugar del código se encuentran.

- No se dispone de métodos exclusivos de validación de datos, aunque en los métodos de creación y actualización de objetos se comprueba algunos datos recibidos desde la interfaz gráfica. Ejemplo: en **dam.samuel.vista.ControladorValorarJuego** en el método **valorar()** (línea 83) se comprueba que el usuario haya seleccionado un tipo de voto, positivo o negativo, para evitar publicar una valoración con recomendación nula.

Existencia de claves alternativas (únicas).

- En la tabla **desarrolla** se ha creado una clave única **idDesarrolla** para cada relación entre Empresa y Juego.

Complejidad de la aplicación, de las consultas y operaciones realizadas.

- La aplicación permite realizar diferentes acciones tanto a usuarios como al administrador, dividiendo cada acción en ventanas diferentes. La interfaz de usuario se ha creado para dar simplicidad de uso, pero para ello se ha aplicado un complejo **sistema de gestión de ventanas** y manejo de funciones ordenado por paquetes.

- Las **consultas** a la base de datos son **variadas**. Se realizan consultas para obtener un único objeto, una lista de objetos que cumplan una condición, listas ordenadas por un atributo, objetos que estén relacionado con otros objetos en otras tablas, guardado y borrado de objetos relacionados con otras tablas, así como la actualización de datos de un objeto que concreto.
- Cabe destacar la complejidad que supone **convertir** un **dato** de tipo LocalDate **obtenido del componente** DatePicker al tipo Date que usa Hibernate **para almacenar** fechas **en la base de datos**.

Calidad de software y estilo de programación.

- Se ha tenido especial cuidado en el **orden** de los diferentes ficheros **según su función**, haciendo uso del **sistema MVC** ordenados por paquetes en el proyecto.
- En el mismo sentido se ha hecho todo lo posible por poner en práctica el **Clean Code**, o código limpio, a la hora de programar las clases. Se han estructurado las diferentes funciones a realizar en métodos independientes para hacer uso de la **reutilización de código** y de una **estructuración clara**.

Calidad de la documentación.

- Además de la documentación del código para JavaDoc, se han **comentado** aquellas **líneas** en las que las funciones a realizar deben seguir un orden o se supone de difícil lectura o **comprensión de lo que se está haciendo**.

Consultas realizadas contra la base de datos

Como el proyecto trata sobre la herramienta Hibernate y para una mayor práctica de ella, se ha tomado la decisión de que todas las consultas se hicieran en HQL.

Consultas **genéricas** para todas las clases:

- save(Object arg0)
- update(Object arg0)
- delete(Object arg0

Consultas para la clase **Desarrolla**:

- createQuery("SELECT d.juego FROM dam.samuel.modelo.Desarrolla d WHERE d.empresa.idEmpresa = " + empresa.getIdEmpresa());

Consultas para la clase **Empresa**:

- get(Empresa.class, id);
- createQuery("SELECT e FROM dam.samuel.modelo.Empresa e ORDER BY nombre");
- createQuery("SELECT e FROM dam.samuel.modelo.Empresa e WHERE nombre = '" + nombre + "'").uniqueResult();

Consultas para la clase **Juego**:

- get(Juego.class, id);
- createQuery("SELECT j FROM dam.samuel.modelo.Juego j ORDER BY nombre");
- createQuery("SELECT j FROM dam.samuel.modelo.Juego j WHERE estilo = '" + estilo.toString() + "' ORDER BY nombre");

Consulta para la clase **Valoracion**:

- createQuery("SELECT v FROM Valoracion v INNER JOIN v.juego j WHERE j.idJuego = " + juego.getIdJuego());

Opciones de Cascade configuradas

En todas las relaciones 1:N se ha configurado Cascade para todas las opciones ya que se ha considerado que si una empresa deja de existir, no puede seguir vendiendo los juegos que ha desarrollado, y si se eliminan los juegos no tiene sentido conservar las valoraciones de los usuarios sobre ese juego.

En una situación real, si una empresa cierra por cualquier motivo, sus datos siguen persistiendo e incluso sus juegos pueden ser vendidos por terceras compañías con las que se lleguen a acuerdos o compren los derechos de venta. Si un juego deja de venderse sus datos también siguen persistiendo como datos históricos para análisis futuros de la empresa. Así mismo, las valoraciones sobre un juego que ya no está en venta se siguen almacenando para el mismo propósito de análisis futuros de calidad o estudios de mercado.

Debido a que este proyecto no podía tener una estructura muy compleja, las opciones de Cascade configuradas se han intentado hacer con lógica y también ha influido los requisitos exigidos, en los que se piden acciones complejas sobre la base de datos y para ello era necesario la configuración en cascada.

Dificultades encontradas en el desarrollo

La principal dificultad es la falta de tiempo para el desarrollo en profundidad de una aplicación “decente”. A esto se le suma el poco estudio sobre la herramienta Hibernate, la cual ofrece una gran cantidad de opciones, configuraciones, y acciones que permiten trabajar y tratar los datos a persistir sin necesidad de conocer la base de datos.

En un primer momento no se ha entendido el funcionamiento de Hibernate, por lo que las consultas eran casi con el formato SQL, provocando confusiones entre los elementos a buscar (objetos y no tablas, atributos y no campos). Además, al realizar consultas de cualquier tipo, Hibernate realiza otras consultas por su cuenta para poder realizar la que le hemos pedido, pero por desconocer esto, se han producidos consultas duplicadas y, sobre

todo, contrarias unas con otras provocando continuos errores y excepciones en la ejecución del programa.

Decisiones de optimización y decisiones de desarrollo

La estructura de la aplicación final ha sido modificada en varias ocasiones para satisfacer los requerimientos mínimos exigidos evitando crear una compleja aplicación que no habría dado tiempo a entregar dentro de plazo.

Al principio se pretendía crear un sistema de registro y logueo de usuarios pero tal objetivo estaba lejos de ser cumplido dentro de plazo. Además que habría supuesto una dedicación de tiempo en un aspecto que no era relevante para el proyecto. Por lo tanto, sólo se controla el acceso en modo administrador.

La apertura y cierre de sesiones de Hibernate ha supuesto un auténtico quebradero de cabeza, habiendo sido necesario probar varias situaciones (una única sesión durante todo el uso de la aplicación, sesiones individuales por cada acción del usuario, sesiones localizadas por agrupación de funciones de una ventana, etc). Finalmente se ha optado por usar una única sesión mientras un usuario está logueado (o el administrador) y el cierre de esta si se vuelve al Login.

Todo el proceso de desarrollo ha sido gestionado por el sistema de control de versiones de Git, estando el proyecto almacenado en GitHub, el cual que ofrece la posibilidad de ver y revisar las diferentes versiones y los comentarios de los commits realizados así como los cambios efectuados en el proyecto y cómo ha ido evolucionando.

Enlace: https://github.com/SamuelReyesAlvarez/ProyectoHibernateJFX