UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Máster Online en Ingeniería Informática

TFM del Máster Online en Ingeniería Informática

Gestión web de bases de datos relacionales

Presentado por José Ignacio Huidobro del Arco

en Universidad de Burgos – junio de 2019

Tutora: María Belén Vaquerizo García

Índice general

Índice de figuras

Índice de tablas

Apéndice A

Plan de Proyecto Software

1.1. Introducción

Ámbito y recursos

Se comunica al alumno el ámbito y recursos con los que se debe plantear el desarrollo de software. Tras analizar los conocimientos previos del mismo y sus necesidades formativas, se llega a la conclusión de que se desarrollará una aplicación para entorno web, desarrollada en lenguaje de programación Java, con Netbeans IDE y software de servidor web Apache Tomcat. Se utilizará la plataforma Struts2 y Maven como apoyo al desarrollo.

Costes y planificación temporal

[1] El análisis de los costos es el proceso de identificación de los recursos necesarios para llevar a cabo el proyecto. La evaluación del costo determina la calidad y cantidad de recursos necesarios en términos de dinero, esfuerzo, capacidad, conocimientos y tiempo, que en ocasiones no son estimados o como sucede en otros casos, se valora que el costo es tan bajo que no es necesario realizar el análisis.

Los modelos de cálculo del costo de desarrollo de software se basan en un conjunto de variables, estas se denominan factores como el costo, el esfuerzo o el tamaño, cada modelo basa sus estimaciones en un conjunto propio de factores.

El desarrollo de software es una actividad muy compleja obteniendo como resultado un producto intangible que depende principalmente del esfuerzo intelectual y creatividad de personas que lo realizan. Los errores humanos están presentes en todas las etapas de un proyecto de este tipo y puede llegar a ser muy costosa su corrección.

En todo proyecto existen tres variables relacionadas, el llamado “triángulo de hierro”: el alcance que refleja los requisitos o tareas para realizar, el tiempo o planificación muestra cuánto durará el proyecto, el costo o recursos que analiza cuánto dinero, personas, recursos se dedicará al proyecto. Para mantener unos objetivos de calidad determinados, cualquier modificación en una de las tres variables implica la modificación de alguna(s) de las otras dos.

Figura 1: Triángulo de hierro

Si se reducen las personas que se dedican al proyecto, dada una calidad determinada,será necesario reducir el alcance del proyecto y/o aumentar su fecha de entrega. Si se reduce la fecha en la cual se debe entregar el proyecto, dada una calidad determinada, será necesario reducir el alcance delproyecto y/o aumentar los recursos que se dedicarán a él. Si se aumenta el alcance del proyecto, dada una calidad determinada, será necesario aumentar la fecha de entrega del proyecto y/o aumentar los recursos que se dedicarán a él.

1.2. Planificación temporal

Para establecer una planificación temporal adecuada, comenzamos indicando las tareas que debemos realizar e indicaremos entre ellas cuales son los hitos que debemos controlar para llevar a cabo la ejecución en plazo del proyecto.

Las tareas para realizar, típicas en cualquier creación de software, en orden de ejecución, son:

- Análisis y especificaciones
- Revisión de los requisitos
- Diseño arquitectónico y de datos
- Revisión del diseño preliminar
- Inspección del diseño
- Codificación
- Inspección del código
- Prueba de unidad
- Prueba de integración
- Prueba de validación

Entre las siguientes tareas es importante establecer puntos de control o hitos, para controlar la correcta evolución del proyecto:

- Revisión de los requisitos – Diseño arquitectónico y de datos
- Revisión del diseño preliminar – Inspección del diseño
- Inspección del código – Prueba de unidad
- Prueba de unidad – Prueba de integración
- Prueba de integración – Prueba de validación

En cuanto al diseño de las funcionalidades y tareas que se han realizado en la confección del proyecto, junto con su fecha de inicio y finalización, así como la duración de estas:

Nombre de la tarea	Fecha de inicio	Fecha final	Duración (dias)
Estado del arte	10/11/2018	13/11/2018	3
Configuración inicial de la aplicación	13/11/2018	17/11/2018	4
Funcionalidad de login	27/11/2018	11/12/2018	14
Funcionalidad de DatabaseMetaData	11/12/2018	18/12/2018	7
Documentación de estándares utilizados	21/11/2018	27/11/2018	6
Filtrado y ordenación de resultsets	18/12/2018	26/12/2018	8
Presentación de tablas y vistas	18/12/2018	26/12/2018	8
Modificación de datos de tablas y vistas	27/12/2018	03/01/2019	7
Hojas de estilo	26/12/2018	27/12/2018	1

Nombre de la tarea	Fecha de inicio	Fecha final	Duración (dias)
Diseño de gráficos y documentación	27/12/2018	28/01/2019	32
Presentación pública	28/01/2019	04/02/2019	7
Generación y empaquetado de documentación	04/02/2019	05/02/2019	1

Tabla 1: Tareas realizadas

Que presentado en forma de gráfico de Gantt quedaría:

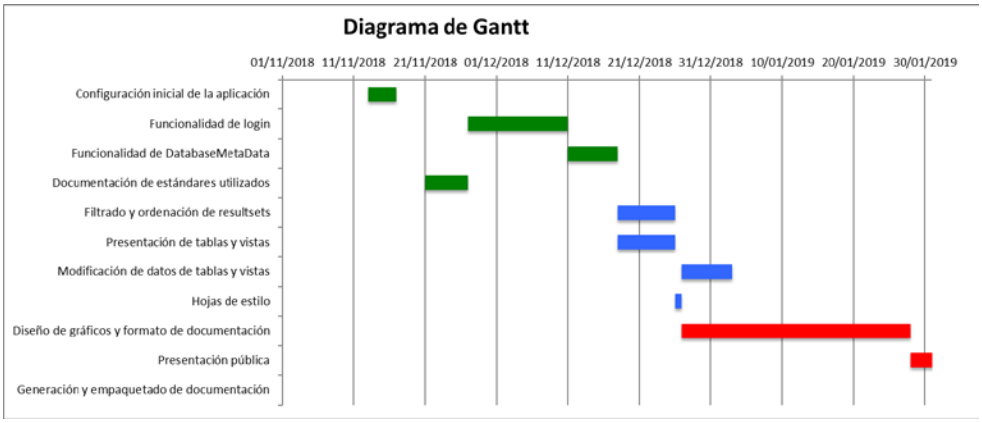

Figura 2: Diagrama de Gantt

1.3. Estudio de viabilidad

[2] Un proyecto siempre tiene como objetivo un beneficio, pero no necesariamente económico, si bien los demás beneficios (de servicio, de imagen, etc.) suelen tener a largo plazo repercusiones económicas. Siempre que se considere la posibilidad de abordar un proyecto, se deben tener en cuenta los recursos de los que se disponen o se puede disponer.

En la práctica, la rentabilidad no es valorada suficientemente, ni siquiera en las grandes empresas. En ocasiones no se tiene en cuenta el esfuerzo necesario para obtener un objetivo. Otro elemento que influye negativamente en la rentabilidad es la dispersión de jefaturas que da lugar a redundancias en pedir nuevos trabajos.

Criterios de viabilidad

Durante la ingeniería de producto, centramos nuestra atención en cuatro áreas principales.

Viabilidad operativa

Los criterios de viabilidad operativa miden la urgencia del problema durante las primeras fases del proyecto, o la aceptabilidad de la solución en fases posteriores. Deberemos tener en cuenta dos aspectos:

- Merece la pena resolver el problema
- Qué puede opinar el usuario final

Respendiendo a estas dos cuestiones, en primer lugar pensamos que sí merece la pena resolver este problema. El acceso genérico a bases de datos es, en sí mismo, una cuestión útil para todos aquellos usuario que suelan trabajar con gestores diversos de bases de datos. El usuario pensamos que tendrá una opinión favorable sobre esta aplicación.

Viabilidad técnica

La viabilidad técnica sólo puede evaluarse después de terminar las fases en las que se tengan que resolver cuestiones técnicas. En la actualidad, apenas nada es imposible desde el punto de vista técnico. En consecuencia, la viabilidad técnica analiza lo que será práctico o razonable de implantar.

Las consideraciones que están asociadas normalmente con la visibilidad técnica son:

- Riesgo de desarrollo
- Disponibilidad de recursos
- Tecnología

Teniendo en cuentas estas consideraciones asociadas al proyecto, llegamos a la conclusión de que la viabilidad técnica es posible, por supuesto, y razonable desde el punto de vista práctico.

Viabilidad de fechas

Aunque tengamos los conocimientos técnicos necesarios, ¿son razonables los plazos del proyecto? Este proyecto arranca ya con plazos fijados y obligatorios. Y aunque puede ser preferible entregar un sistema que funcione correctamente con dos meses de retraso, en nuestro caso es perentorio cumplir los plazos de entrega por la misma naturaleza de un trabajo fin de máster que debe ser evaluado en una fecha concreta.

Viabilidad económica

Aunque puede que no sea necesaria en un proyecto educativo, se detalla a continuación un análisis de coste que puede ser comparado con el beneficio que puede aportar el proyecto en el presente y, sobre todo, en el futuro.

A continuación se explican las métricas seleccionadas, estas tienen un rango de valor del 0 al 100 con el objetivo de hacerlas medibles, cuantificables para garantizar un cálculo más efectivo y objetivo.

1. **Experiencia con la tecnología utilizada: 90**. este factor tiene gran influencia en la productividad del equipo, reconociendo la importancia de conocer nuevas y potentes herramientas de diseño, gestores de base de datos, interfaces gráficas, redes, framework, etc. Contar con un grupo de desarrollo con una experiencia considerable garantiza que el producto esté listo en el tiempo estimado

Características	Bajo (0-39)	Medio (40-79)	Alto (80-100)
Experiencia con la tecnología utilizada	Poca	Básica	Considerable

Tabla 2: Métrica experiencia con la tecnología utilizada

2. **Complejidad del modelo de datos: 15**. el modelo de datos permite describir los elementos de la realidad que intervienen en un problema dado y la forma en que se relacionan esos elementos entre sí. Contar con un modelo de datos complejo obliga a los desarrolladores a emplear todo su intelecto para razonar, diseñar, modelar y programar la solución, dándole respuesta a los requerimientos iniciales.

Nivel de complejidad	Bajo (0-39)	Medio (40-79)	Alto (80-100)
Complejidad del modelo de datos	1 a 40 clases	41 a 60 clases	Más de 60 clases

Tabla 3: Métrica de complejidad del modelo de datos

3. **Facilidades administrativas: 90**. aquí se incluye el transporte necesario para realizar gestión comercial, económica o propiamente del proyecto, facilidades para obtener herramientas de trabajo como son dispositivos de almacenamiento, discos duros, papel, impresoras, teléfono, laptop, cuentas de internet entre otras. Las áreas de apoyo juegan un papel fundamental en la obtención de un producto acabado en tiempo, por lo cual su trabajo es necesario para mantener tanto la comunicación con el cliente como el trabajo dentro del equipo.

Características	Bajo (0-39)	Medio (40-79)	Alto (80-100)
Disposición, habilidad y gestión para cumplir objetivos	Poca	Básica	Considerable
Facilidad para obtener herramientas de trabajo	Poca	Básica	Considerable

Tabla 4: Métrica de facilidades administrativas

4. **Experiencia con el lenguaje de programación: 90**. Tener un dominio del lenguaje de programación permite expresar procesos que pueden ser llevados a cabo por computadoras. Poseer un equipo con dominio pleno del lenguaje de programación es una ventaja considerable pues permite que el tiempo de desarrollo sea menor, que los requerimientos sean programados correctamente, que no existan errores, le aporta brillo al código y seguridad al equipo. La capacitación es menor pues se considera que los conocimientos probados son los ideales para llevar a cabo la tarea.

Características	Bajo (0-39)	Medio (40-79)	Alto (80-100)
Experiencia con el lenguaje de programación	Poca	Media	Considerable

Tabla 5: Métrica experiencia con el lenguaje de programación

5. **Cronograma de desarrollo requerido: 80**. Este factor mide las restricciones en los plazos de tiempo impuestos al equipo de trabajo. Los valores se definen dada la extensión o aceleración de los plazos establecidos. Acelerar los plazos produce más esfuerzo en las últimas etapas del desarrollo, en las que se acumulan más temas a determinar por la escasez de tiempo para resolverlos. Un cronograma que mantenga lo congeniado en el contrato aporta calidad al proyecto creado.

Características	Bajo (0-39)	Medio (40-79)	Alto (80-100)
Cronograma de desarrollo requerido	Poca	Básica	Considerable

Tabla 6: Métrica cronograma de desarrollo requerido

Complejidad del proyecto: 36.5. Este análisis indica la complejidad del proyecto. Una vez se cuente con la complejidad de proyecto se prosigue a calcular el tiempo estimado por la tarifa horaria, el valor que se obtiene se le aplica entonces el porciento que representa la suma de las métricas que indican la complejidad del proyecto. Esta suma puede ser de hasta el 50% teniendo en cuenta que fueron seleccionadas 5 métricas, dándole un valor agregado al producto considerable.

Características	Bajo (0-19)	Medio (20-39)	Alto (40-50)
Complejidad del proyecto	Baja	Media	Considerable

Tabla 7: Complejidad del proyecto

Teniendo en cuenta que le proyecto se declara en una asignatura que abarca 6 créditos ETCS y que cada crédito se define como 25 horas de trabajo, nos da un total de

150 horas

Si consederamos un precio hora de programador junior de 30 euros, el coste total sería de

4.500 euros

Teniendo en cuenta la complejidad del proyecto (media), habría que aplicar un porcentaje del 36.5% por lo que el proyecto final tendría un coste de

6.142,50 euros

Consideramos despreciable, por no incluir detalles menores, el coste del equipamiento, software utilizado y consumos realizados.

En cuanto al análisis del beneficio económico, es sumamente difícil tasar el valor comercial de una aplicación como la realizada en este TFM. Muchas aplicaciones del mercado son puestas a disposición del público como software libre, constituyendo un apoyo en el aprendizaje de técnicas de programación y resolución de problemas concretos de un gran valor inmaterial: la transmisión del conocimiento.

Teniendo en cuenta, además, el ámbito educativo en el que se ha desarrollado el trabajo y sumándole el hecho de que todas las herramientas y tecnologías utilizadas son de acceso gratuito me veo en la obligación moral de asignarle una valoración económica de

0 euros

A pesar de lo cual considero que el TFM desarrollado tiene una gran viabilidad económica.

Viabilidad legal

El desarrollo del sofware se ha planteado mediante la utilización de herramientas de desarrollo de software libre, basado en software de libre distribución a nivel educativo y empresarial, por lo que la viabilidad legal del proyecto es total.

En concreto se han utilizado los siguientes tipos de licencia para las distintas técnicas y herramientas:

Componente	Licencia	Dirección
Apache Tomcat	Apache License 2.0	https://www.apache.org/licenses/LICENSE-2.0

Apache Struts 2	Apache License 2.0	https://www.apache.org/licenses/LICENSE-2.0
Netbeans IDE	Common Development and Distribution License (CDDL)	https://netbeans.org/about/legal/license.html
	GNU General Public License 2	https://netbeans.org/bugzilla/docs/en/html/gfdl.html
JDK y JavaDoc	GNU License	https://openjdk.java.net/legal/gplv2+ce.html

Tabla 8: Licencias de software

Apéndice B

Especificación de Requisitos

2.1. Introducción

A continuación realizaremos una descripción completa del comportamiento del sistema que se va a desarrollar. Se incluyen un conjunto de casos de uso que describe todas las iteracciones que tendrán los usuarios con el software. También se indicarán, si fuera necesario, los requisitos que imponen restricciones en el diseño o la implementación.

2.2. Objetivos generales

El objetivo es el desarrollo de un Sistema Software para la gestión vía Internet, de una base de datos relacional remota. A través de este sistema se permitirá administrar varios tipos de bases de datos (SQLServer, MySQL y Oracle), donde, através de mecanismos JDBC, se obtendrá la información necesaria para la realización de las operaciones básicas de mantenimiento que decida el usuario, tales como inserciones, borrados, modificaciones, consultas, etc. El sistema aportará al usuario una interfaz sencilla e intuitiva, y con una ayuda interactiva en línea para ayudarle en el manejo de la aplicación de un modo sencillo y en la comprensión de las operaciones que desee realizar.

2.3. Catálogo de requisitos

Objetivos

El presente documento tiene como objetivo puntualizar los requerimientos funcionales y no funcionales del sistema que se va a implementar. Los requerimientos son sustraídos de las necesidades del usuario para un mejor desempeño durante sus procesos, estudio o investigación.

Requisitos funcionales

En esta sección se detalla los requisitos funcionales para la implementación del proyecto. Así mismo, se definirá su prioridad y su relevancia para su implementación en el sistema. A continuación se detalla los requerimientos del sistema por módulo.

Nro	Funcionalidades	Prioridad	Exigible
Módulo de conexión			
RC01	El sistema deberá establecer conexión con base de datos relacional mediante interface JDBC.	1	E
RC02	Para establecer la conexión con la base de datos se proporcionará URL, nombre de usuario de base de datos y contraseña.	1	E
RC03	La conexión se mantendrá abierta de manera indefinida, hasta que la sesión http del usuario sea cerrada por el servidor web o la máquina virtual java subyacente.	1	E
RC04	Una conexión cerrada podrá ser reabierta de manera automática para lo que el sistema debe almacenar bajo el adecuado sistema de seguridad, la URL, el nombre de usuario y la clave.	1	E
RC05	Ante cualquier petición de una página del sistema, se comprobará si el usuario dispone de una conexión correctamente establecida. Si la conexión no estuviera establecida, el sistema deberá redirigir la petición a la pantalla de login.	1	E

Nro	Funcionalidades	Prioridad	Exigible
Módulo de Metadatos de Conexión y base de datos			
RM01	Mostrar al usuario la colección de información simple obtenida de la conexión.	2	E
RM02	Mostrar al usuario a modo de menú los distintos métodos proporcionados por los metadatos de la base de datos que se organizan en forma de ResultSet.	1	E
RM03	Mostrar al usuario a modo de menú los distintos métodos, agrupados en categorías configurables, proporcionados por los metadatos de la base de datos que devuelven información en formato simple (sin parámetros de ejecución y que devuelvan un tipo de dato simple).	1	E
RM04	Proporcionar al usuario un sistema que permita introducir los parámetros de ejecución de los métodos complejos de la base de datos.	1	E
RM05	Proporcionar al usuario un sistema que permita guardar en sesión los valores de los parámetros de ejecución establecidos en una llamada.	2	E
RM06	Presentar los datos del ResultSet obtenido en forma de tabla.	1	E
RM07	Permitir al usuario ordenar las tablas presentadas por cualquiera de las columnas que la componen, tanto en orden ascendente como descendente, de forma local.	2	E
Modelo de tabla			
RT01	Permitir visualizar cualquier tabla de la base de datos, de una manera práctica y sencilla.	1	E
RT02	Permitir visualizar los registros y campos de cualquier tabla de la base de datos.	1	E
RT03	Permitir al usuario la inserción de un criterio de filtrado de registros por cada campo de la tabla.	1	E
RT04	Permitir al usuario la eliminación de un registro de la tabla.	1	E
RT05	Permitir al usuario la modificación de un registro de la tabla.	1	E

Nro	Funcionalidades	Prioridad	Exigible
RT06	Permitir al usuario la inserción de un registro en la tabla.	1	E

Requisitos no funcionales

En esta sección se detalla los requisitos no funcionales referentes al proyecto. Ellos describen las necesidades tecnológicas, rendimiento y software necesarios para el uso de la aplicación. A continuación se describen los requerimientos no funcionales del sistema.

Nro	Descripción	Prioridad	Exigible
RN01	El software se desarrollará en una plataforma Web J2EE (Java 2 Enterprise Edition) según el patrón arquitectónico MVC (Modelo Vista Controlador).	1	E
RN02	El lenguaje de programación de desarrollo del proyecto será Java Web.	1	E
RN03	La aplicación deberá ser probada al menos para las bases de datos Oracle, MySQL y SQLServer	1	E
RN04	El sistema operativo que soportará la plataforma es Windows	1	E
RN05	El sistema deberá ser manejado mediante un navegador o browser.	1	E
RN06	Los navegadores comprobados para el manejo del sistema son: internet explorer (IE) versión 11, mozilla versión 64 y Chrome versión 71.0	1	E
RN07	La información estará protegida contra los accesos a usuarios no autorizadores.	2	E
RN08	Con el objeto de no demorar en exceso las respuestas del servidor ante alguna consultas, se establece un límite de 1000 registros en aquellas consultas a la base de datos. Si se excede dicho límite, el sistema solo presentará los primeros 1000 registros. Dicho límite es fácilmente modificable por parte del usuario.	2	E

Nro	Descripción	Prioridad	Exigible
RN09	Sólo se permitirá realizar operaciones de modificación o eliminación de registros en tablas con Primery Key	1	E

2.4. Especificación de requisitos

[1]

Introducción

Propósito

El propósito del capítulo es recopilar todos los requerimientos de la aplicación a desarrollar y está dirigido al desarrollador para una correcta implementación de la aplicación y al lector en general para la mejor comprensión del uso y funcionamiento de la aplicación.

Ámbito de Sistema

Se asigna el nombre de MyDatabaseJC como nombre de la aplicación. JC son los acrónimos de Java Connection. Como se ha explicado en los objetivos, la aplicación permite conectarse a cualquier aplicación de la que exista un driver JDBC y realizar sobre ella consultas generales y modificaciones de datos en tablas.

Los beneficios de la aplicación pueden ser la:

- Utilización generalista o comercial, sin necesidad de instalación de software propio del fabricante, para empresas con limitados recursos económicos y que no dispongan de amplios conocimientos en entornos de base de datos.
- Utilización con fines docentes, como base de estudio del estándar JDBC así como Java Reflection y Struts por sus amplios y variados ejemplos de utilización.

Descripción General

En esta sección se describen todos aquellos factores que afectan al producto y a sus requisitos. No se describen los requisitos, sino su contexto.

Perspectiva del Proyecto

El proyecto puede ser considerado como un producto independiente, que no forma parte de otro proyecto superior.

En el documento Memoria se ha descrito con profundidad los productos comerciales y sus requisitos, que pueden considerarse similares en cuanto al desarrollo conceptual de la aplicación.

Funciones del Proyecto

Las funciones del producto deberán dar cabida a los objetivos planteados y descritos tanto en este documento como en la memoria del proyecto. Podremos agrupar las funciones en las siguientes grandes categorías:

- Establecimiento de conexión JDBC con base de datos relacional
- Acceso a los metadatos de conexión y base de datos
- Acceso a los datos de la base de datos
- Modificación, eliminación o inserción de los datos de la base de datos.

Características de los Usuarios

El proyecto se ha diseñado para la utilización por parte de cualquier usuario que tenga unos conocimientos mínimos de las bases de datos relacionales y comprenda los conceptos de conexión, base de datos y tablas.

Suposiciones y Dependencias

Como ya se ha indicado en los documentos anexos, la aplicación funciona correctamente sobre plataformas Java 1.8. Se ha detectado que su funcionamiento no es adecuado, y por lo tanto no funcionará, en plataformas Java 1.7.

Requisitos Futuros

Existen una gran cantidad de cuestiones que pueden mejorarse de la aplicación y de funciones que pueden ser ampliadas. Entre las que podemos destacar, como ya se indicó en el apartado “Líneas de trabajo futuras” del capítulo “Conclusiones y Líneas de trabajo futuras” de la memoria:

- Posibilidad de que un mismo navegador, una única sesión, mantenga más de una conexión a bases de datos, incluso diferentes.
- Permitir modificar metadatos de la conexión y de la base de datos.
- Permitir la ejecución de comandos SQL de tipo DDL.
- Permitir la ejecución de comandos de definición de usuarios y de seguridad de la base de datos.
- Permitir navegar entre tablas atendiendo a campos que intervienen en índices de tipo Foreing Key.
- Permitir modificar, eliminar o insertar registros en una tabla a partir de una hoja de datos representada en forma de tabla HTML.
- Permitir trabajar en campos de tipo LOB.
- Permitir al usuario modificar, de forma permanente, los parámetros globales de configuración de la aplicación.
- Permitir realizar transacciones consistentes.
- Permitir filtros de búsqueda incrementales.
- Permitir nuevos gestores de base de datos.
- Permitir la modificación y eliminación de registros en tablas sin Primery Key
- Permitir la exportación e importación de datos en formato Excel o CSV.

Requisitos Específicos

Esta sección contiene los requisitos a un nivel de detalle suficiente como para permitir a los diseñadores diseñar un sistema que satisfaga estos requisitos, y que permita planificar y realizar las pruebas que demuestren si el sistema satisface, o no, los requisitos. Todo requisito aquí especificado describirá los comportamientos externos del sistema, perceptibles por parte de los usuarios, operadores y otros sistemas.

Los requisitos poserán las siguientes características:

Corrección: La Especificación de Requisitos Software (ERS) es correcta si y sólo si todo requisito que figura aquí (y que será implementado en el sistema) refleja alguna necesidad real. La corrección de la ERS implica que el sistema implementado será el sistema deseado.

No ambiguos: Cada requisito tiene una sola interpretación. En el caso de utilizar términos que, habitualmente, poseen más de una interpretación, se definirán con precisión en el glosario.

Completos: Todos los requisitos relevantes han sido incluidos en la ERS.

Consistentes: Los requisitos no pueden ser contradictorios. Un conjunto de requisitos contradictorio no es implementable.

Clasificados: Normalmente, no todos los requisitos son igual de importantes. Los requisitos pueden clasificarse por importancia (esenciales, condicionales u opcionales) o por estabilidad (cambios que se espera que afecten al requisito).

Verificables: La ERS es verificable si y sólo si todos sus requisitos son verificables. Un requisito es verificable (testeable) si existe un proceso finito y no costoso para demostrar que el sistema cumple con el requisito. Un requisito ambiguo no es, en general, verificable.

Modificables: La ERS es modificable si y sólo si se encuentra estructurada de forma que los cambios a los requisitos pueden realizarse de forma fácil, completa y consistente.

Trazables: La ERS es trazable si se conoce el origen de cada requisito y se facilita la referencia de cada requisito a los componentes del diseño y de la implementación.

Identificación	RC01
Nombre	Establecimiento de conexión.

Identificación	RC01
Características	El sistema deberá establecer conexión con base de datos relacional mediante interface JDBC.
Descripción	Mediante la clase DriverManager del paquete JDBC, se establecerá la conexión con la base de datos pasándole como parámetros url de conexión, usuario y contraseña.
Requerimiento no funcional	RN01 RN02 RN03
Prioridad	1

Tabla 9: Requerimiento establecimiento de conexión

Identificación	RC02
Nombre	Datos de conexión.
Características	Para establecer la conexión con la base de datos se proporcionará URL, nombre de usuario de base de datos y contraseña.
Descripción	Aunque el método getConnection de la clase DriverManager dispone de varias versiones, se utilizará el que necesita url, usuario y clave como datos necesarios para realizar la conexión.
Requerimiento no funcional	RN01 RN02 RN07
Prioridad	1

Tabla 10: Requerimiento Datos de conexión

Identificación	RC03
Nombre	Duración de la conexión.
Características	La conexión se mantendrá abierta de manera indefinida, hasta que la sesión http del usuario sea cerrada por el servidor web o la máquina virtual java subyacente.
Descripción	La aplicación no deberá cerrar la conexión tras la ejecución de comandos de base de datos. La aplicación deberá cerrar la conexión cuando se ejecute el método finalize de la clase en la que esta se encuentra, fruto del proceso de recolección de basura de la máquina virtual java.

Requerimiento no funcional	RN01 RN02 RN07
Prioridad	1

Tabla 11: Requerimiento Duración de conexión

Identificación	RC04
Nombre	Almacenamiento de datos de conexión.
Características	Una conexión cerrada podrá ser reabierta de manera automática para lo que el sistema debe almacenar bajo el adecuado sistema de seguridad, la URL, el nombre de usuario y la clave.
Descripción	Se utilizará una clase ConnectionImpl que, además de la conexión java, mantendrá los datos de la conexión, incluyendo la contraseña del usuario. La instanciación de la clase se realiza en el momento de crearse la conexión y será mantenida para cada usuario en el espacio de memoria de sesión mantenido por el servidor web.
Requerimiento no funcional	RN01 RN02 RN07
Prioridad	1

Tabla 12: Requerimiento Almacenamiento de datos de conexión

Identificación	RC05
Nombre	Control de acceso.
Características	Ante cualquier petición de una página del sistema, se comprobará si el usuario dispone de una conexión correctamente establecida. Si la conexión no estuviera establecida, el sistema deberá redirigir la petición a la pantalla de login.
Descripción	Utilizando el método validate de las clases Action de struts2 comprobaremos que existe sesión de usuario, que tiene una conexión y que es válida y no está cerrada. En caso contrario se deberá redirigir a la pantalla de login.

Requerimiento no funcional	RN01 RN02 RN07
Prioridad	1

Tabla 13: Requerimiento Control de aceso

Identificación	RM01
Nombre	Información de conexión.
Características	Mostrar al usuario la colección de información simple obtenida de la conexión.
Descripción	Se utilizarán los métodos de la clase Connection y se mostrará en pantalla al usuario la información de aquellos que devuelvan información sencilla mediante métodos que no requieran parámetros.
Requerimiento no funcional	RN05
Prioridad	2

Tabla 14: Requerimiento Información de conexión

Identificación	RM02
Nombre	Menú del usuario.
Características	Mostrar al usuario a modo de menú los distintos métodos proporcionados por los metadatos de la base de datos que se organizan en forma de ResultSet.
Descripción	Se crea una clase DatabaseMetaDataImpl que implementa el interface DatabaseMetaData y todos sus métodos. Un proceso que obtiene la lista de métodos de esta clase, filtra los que devuelven ResultSet y los presenta en una página web.
Requerimiento no funcional	RN05
Prioridad	1

Tabla 15: Requerimiento Menú del usuario

Identificación	RM03
Nombre	Menú de información.
Características	Mostrar al usuario a modo de menú los distintos métodos, agrupados en categorías

	configurables, proporcionados por los metadatos de la base de datos que devuelven información en formato simple (sin parámetros de ejecución y que devuelvan un tipo de dato simple).
Descripción	En la clase DatabaseMetaDataImpl se incorporan métodos que devuelven una lista de propiedades y valores.
Requerimiento no funcional	RN05
Prioridad	1

Tabla 16: Requerimiento Menú de información

Identificación	RM04
Nombre	Parámetros de ejecución de métodos.
Características	Proporcionar al usuario un sistema que permita introducir los parámetros de ejecución de los métodos complejos de la base de datos.
Descripción	Para llevar a cabo este requerimiento, debe utilizarse la opción getParameters de java Reflection. Este método requiere Java 1.8. Además, se utlizará el método getParameterTypes para obtener el tipo de datos al que pertenecen los parámetros.
Requerimiento no funcional	RN05
Prioridad	1

Tabla 17: Requerimiento Parámetros de ejecución de métodos

Identificación	RM05
Nombre	Parámetros en sesión.
Características	Proporcionar al usuario un sistema que permita guardar en sesión los valores de los parámetros de ejecución establecidos en una llamada.
Descripción	Cuando el usuario envía a la aplicación los parámetros de ejecución de un método de DatabaseMetaData, el sistema, previamente a su utilización, los deberá

	almacenar en la sesión del usuario en el servidor web.
Requerimiento no funcional	RN07 RN05
Prioridad	1

Tabla 18: Requerimiento Parámetros en sesión

Identificación	RM06
Nombre	Presentación del resultset
Características	Presentar los datos del ResultSet obtenido en forma de tabla
Descripción	El programa dispondrá de un sistema que permita visualizar un resultset en forma de tabla html
Requerimiento no funcional	RN05
Prioridad	1

Tabla 19: Requerimiento Presentación del resultset

Identificación	RM07
Nombre	Ordenación de tablas
Características	Permitir al usuario ordenar las tablas presentadas por cualquiera de las columnas que la componen, tanto en orden ascdente como descendente, de forma local
Descripción	El procedimiento de ordenación debe realizarse en el navegador del cliente, no se precisa una nueva request al servidor. Se utilizará la biblioteca jquery para realizar la operación.
Requerimiento no funcional	RN05 RN06
Prioridad	1

Tabla 20: Requerimiento Ordenación de tablas

Identificación	RT01
Nombre	Visualizar tablas de la base de datos
Características	Permitir visualizar cualquier tabla de la base de datos, de una manera práctica y sencilla

Descripción	Mediante llamada al método getTables de la clase DatabaseMetaData con los parámetros adecuados el usuario podrá elegir la tabla que consultar
Requerimiento no funcional	RN05
Prioridad	1

Tabla 21: Requerimiento Visualizar tablas de la base de datos

Identificación	RT02
Nombre	Visualizar tabla
Características	Permitir visualizar los registros y campos de cualquier tabla de la base de datos
Descripción	El usuario enviará una sentencia SELECT * FROM tabla y se visualizará todos los datos y todos los campos de la tabla consultada
Requerimiento no funcional	RN05
Prioridad	1

Tabla 22: Requerimiento Visualizar tabla

Identificación	RT03
Nombre	Filtrado de registros en consulta
Características	Permitir al usuario la inserción de un criterio de filtrado de registros por cada campo de la tabla
Descripción	Se debe permitir al usuario introducir valores que permitan filtrar los registros por todos los campos que sean posible
Requerimiento no funcional	RN08
Prioridad	1

Tabla 23: Requerimiento Filtrado de reegistros en consulta

Identificación	RT04
Nombre	Eliminación de un registro
Características	Permitir al usuario la eliminación de un registro de la tabla

Descripción	Se debe permitir la eliminación de un registro de una tabla en la que exista una primary key
Requerimiento no funcional	RT07
Prioridad	1

Tabla 24: Requerimiento Eliminación de un registro

Identificación	RT05
Nombre	Modificación de un registro
Características	Permitir al usuario la modificación de un registro de la tabla
Descripción	Se debe permitir al usuario la modificación de un registro de la tabla que disponga de una primary key
Requerimiento no funcional	RT07
Prioridad	1

Tabla 25: Requerimiento Modificación de un registro

Identificación	RT06
Nombre	Inserción de un registro
Características	Permitir al usuario la inserción de un registro en la tabla
Descripción	Permitir al usuario introducir un registro en la tabla
Requerimiento no funcional	Ninguno
Prioridad	1

Tabla 26: Requerimiento Inserción de un registro

Identificación	RT07
Nombre	Modificación solo con PK
Características	Sólo se permitirá realizar operaciones de modificación o eliminación de registros en tablas con Primary Key
Descripción	Sólo se permitirá realizar operaciones de modificación o eliminación de registros en tablas con Primary Key
Requerimiento no funcional	
Prioridad	1

Tabla 27: Requisito Modificación solo con PK

Nro	Corrección	No ambiguo	Completo	Consistente	Clasificado	Verificable	Modificable
RC01	Sí	Sí	Sí	Sí	Esencial	Sí	Sí
RC02	Sí	Sí	Sí	Sí	Esencial	Sí	Sí
RC03	Sí	Sí	Sí	Sí	Opcional	Sí	Sí
RC04	Sí	Sí	Sí	Sí	Opcional	Sí	Sí
RC05	Sí	Sí	Sí	Sí	Esencial	Sí	Sí
RM01	Sí	Sí	Sí	Sí	Opcional	Sí	Sí
RM02	Sí	Sí	Sí	Sí	Esencial	Sí	Sí
RM03	Sí	Sí	Sí	Sí	Esencial	Sí	Sí
RM04	Sí	Sí	Sí	Sí	Esencial	Sí	Sí
RM05	Sí	Sí	Sí	Sí	Esencial	Sí	Sí
RM06	Sí	Sí	Sí	Sí	Esencial	Sí	Sí
RM07	Sí	Sí	Sí	Sí	Opcional	Sí	Sí
RT01	Sí	Sí	Sí	Sí	Esencial	Sí	Sí
RT02	Sí	Sí	Sí	Sí	Esencial	Sí	Sí
RT03	Sí	Sí	Sí	Sí	Esencial	Sí	Sí
RT04	Sí	Sí	Sí	Sí	Esencial	Sí	Sí
RT05	Sí	Sí	Sí	Sí	Esencial	Sí	Sí
RT06	Sí	Sí	Sí	Sí	Esencial	Sí	Sí
RT07	Sí	Sí	Sí	Sí	Esencial	Sí	Sí

Tabla 28: Definición de requisitos

Requisitos no funcionales

Esta sección contiene los requisitos a un nivel de detalle suficiente como para permitir a los diseñadores diseñar un sistema que satisfaga estos requisitos, y que permita planificar y realizar las pruebas que demuestren si el sistema satisface, o no, los requisitos. Todo requisito aquí especificado describirá los comportamientos externos del sistema, perceptibles por parte de los usuarios, operadores y otros sistemas.

Identificación	RN01
Nombre	Entorno de explotación
Características	El software se desarrollará en una plataforma Web J2EE (Java 2 Enterprise Edition) según el patrón arquitectónico MVC (Modelo Vista Controlador).
Descripción	En concreto, el desarrollo se realizará en un servidor Apache Tomcat con una plataforma struts2
Requerimiento no funcional	
Prioridad	1

Tabla 29: Requisito Entorno de explotación

Identificación	RN02
Nombre	Lenguaje de programación
Características	El lenguaje de programación de desarrollo del proyecto será Java Web.
Descripción	El lenguaje de programación de desarrollo del proyecto será Java Web. Versión 1.8
Requerimiento no funcional	
Prioridad	1

Tabla 30: Requisito Lenguaje de programación

Identificación	RN03
Nombre	Pruebas en bases de datos

Identificación	RN03
Características	La aplicación deberá ser probada al menos para las bases de datos Oracle, MySQL y SQLServer
Descripción	Versiones 11 para Oracle, SQLSERVER 2012 express y MySQL ultima versión deben ser realizadas
Requerimiento no funcional	
Prioridad	1

Tabla 31: Requisito Pruebas en bases de datos

Identificación	RN04
Nombre	Sistema operativo
Características	El sistema operativo que soportará la plataforma es Windows
Descripción	El sistema operativo que soportará la plataforma es Windows
Requerimiento no funcional	
Prioridad	1

Tabla 32: Requisito Sistema operativo

Identificación	RN05
Nombre	Cliente navegador
Características	El sistema deberá ser manejado mediante un navegador o browser.
Descripción	El sistema deberá ser manejado mediante un navegador o browser.
Requerimiento no funcional	
Prioridad	1

Tabla 33: requisito Cliente navegador

Identificación	RN06
Nombre	Pruebas de navegación
Características	Los navegadores comprobados para el manejo del sistema son: internet explorer (IE) versión 11, mozilla versión 64 y Chrome versión 71.0
Descripción	Los navegadores comprobados para el manejo del sistema son: internet explorer

	(IE) versión 11, mozilla versión 64 y Chrome versión 71.0
Requerimiento no funcional	
Prioridad	1

Tabla 34: Requisito Pruebas de navegación

Identificación	RN07
Nombre	Protección de acceso
Características	La información estará protegida contra los accesos a usuarios no autorizadores.
Descripción	Toda petición realizada deberá ser redirigida a protocolo seguro HTTPS. Tras cada petición, la aplicación debe comprobar que el usuario esté correctamente autorizado
Requerimiento no funcional	
Prioridad	1

Tabla 35: Requisito Protección de acceso

Identificación	RN08
Nombre	Límite de response
Características	Con el objeto de no demorar en exceso las respuestas del servidor ante alguna consulta, se establece un límite de 1000 registros en aquellas consultas a la base de datos. Si se excede dicho límite, se presentan los primeros 1000 registros con la posibilidad de que el usuario lo amplíe.
Descripción	
Requerimiento no funcional	
Prioridad	1

Tabla 36: Requisito Límite de response

Interfaces Externas

Se describirán los requisitos que afecten a la interfaz de usuario, interfaz con otros sistemas (hardware y software) e interfaces de comunicaciones.

Funciones

Por jerarquía funcional, se desglosan en las siguientes categorías:

Funciones de conexión. Todas aquellas funcionalidades relacionadas con el mantenimiento y consecución de la conexión con la base de datos

Funciones de Metadatos de Conexión y base de datos. Funcionalidades que exploran los metadatos de la conexión o de la propia base de datos e informan al usuario de las posibilidades que tiene en cuanto a la organización de la propia base de datos

Funciones de tabla. El usuario no solo puede explorar la base de datos, también puede acceder a los datos almacenados en sus tablas y modificarlos, borrarlos o insertar nuevos registros en las tablas.

Requisitos de Rendimiento

No se ha realizado un estudio de requisitos de rendimiento ni de capacidad requerida por el sistema para almacenar la información del usuario.

Atributos del Sistema

Fiabilidad. La fiabilidad de la aplicación se basa en la fiabilidad del entorno de ejecución en la que se desarrolla, en la fiabilidad de la base de datos que se consulta y en la fiabilidad de las comunicaciones entre ambos sistema.

Mantenibilidad. El sistema no requiere mantenibilidad ninguna y, manteniendose las condiciones iniciales de ejecución, se mantendrá el funcionamiento de la aplicación

Portabilidad. Al tratarse de una aplicación Java, en teoría, es directamente exportable a cualquier servidor web que esté dotado de la posibilidad de ejecutar aplicaciones web basadas en java y en los estándares JSP, independientemente del sistema operativo subyacente.

Seguridad. El acceso por parte del usuario a la base de datos, y todo el conjunto de la aplicación, se realiza mediante un mapeo automático y obligatorio de todas las request desde el protocolo http hacia el protocolo https. Mediante URL, usuario y contraseña convenientemente codificado, se accederá a la base de datos. Tanto la conexión como la información de conexión son almacenadas en la sesión del usuario, por lo que toda comunicación entre cliente y aplicación estará protegida por el protocolo https.

Apéndice C

Especificación de diseño

3.1. Introducción

El análisis de los requisitos da lugar a la especificación software donde se concretan necesidades y restricciones con las que debe trabajar el software. Se intenta obtener uno o varios modelos que detallen el comportamiento deseado del sistema.

Trataremos de ofrecer una visión de alto nivel sin descender a explicar detalles concretos del mismo.

La técnica de modelado utilizado es la de aproximaciones sucesivas en la que tomamos como partida el modelo de un sistema similar y luego mediante la experiencia y el conocimiento del problema se van proponiendo modelos intermedios.

3.2. Diseño de datos

A continuación se detallan las distintas clases creadas para manejar los datos de la aplicación, y se indican las funciones precisas para trabajar con ellos.

Paquete es.ubu.alu.mydatabasejc

Menu.java

Contiene los datos necesarios para manejar una opción del menú.

```
public class Menu implements Comparable<Menu>{
```

```
  private String action;
  private String metodo;
  private String parametros;
  private String i18n;
```

PropiedadValor

Se utiliza para presentar al usuario información en forma propiedad/valor.

```
public class PropiedadValor {

  private String propiedad;
  private Object valor;

  public PropiedadValor(
       String propiedad, Object valor);
  ...
}
```

ValoresPorDefecto

Esta clase contiene una serie de valores de configuración de la aplicación de uso general que pueden ser modificados por el usuario en el futuro.

```
public class ValoresPorDefecto {
  public static int numMaxRecords = 1000;
  public static String autor =
       "José Ignacio Huidobro del Arco";
  public static String email = "jhd1001@ubu.alu.es";
  public static String titulo = "MyDatabaseJC";
  public static String subtitulo =
       "Trabajo Fin de Máster";
  public static String estudio =
       "Máster Universitario Online en Ingeniería Informática";
  public static String descripcionTrabajo =
       " Gestión web de bases de datos relacionales ";
}
```

Paquete es.ubu.alu.mydatabasejc.annotations

MetaDataInfoCategorias

Define la anotación para categorizar los métodos de DatabaseMetaDataImpl.

```
@Target(ElementType.METHOD)
@Retention(RetentionPolicy.RUNTIME)
@Documented
```

```
public @interface MetaDataInfoCategorias {
  String categoria() default "Básica";
}
```

MetaDataLink

Define la anotación para marcar los métodos de DatabaseMetaDataImpl que deben proporcionar la información para establecer un link hacia otra pantalla y cuales han de ser los parámetros para enviar en la request de ese link.

```
@Target(ElementType.METHOD)
@Retention(RetentionPolicy.RUNTIME)
@Documented
public @interface MetaDataLink {
  // devuelve la acción struts que debe ejecutarse
  String action();
  // devuelve el namespace en el que debe ejecutarse la acción
  String namespace();
  // devuelve los parametros que deben incluirse en el link, en
formato array String
  String[] parametros();
  // marca el número de la columna en la que debe situarse el link
  int columnNumber();
}
```

Paquete es.ubu.alu.mydatabasejc.exceptions

ConnectionException

```
/**
 * Excepciones producidas en la conexión con la base
 * de datos
 * @author jhuidobro
 */
public class ConnectionException extends Throwable {
  final static Logger logger = (Logger) LogManager
       .getLogger(ConnectionException.class);
  ConnectionImpl connectionImpl;

  public ConnectionException(
       Throwable cause, String driver);

  public ConnectionException();

  public ConnectionException(
       String msg, ConnectionImpl connectionImpl);

  public ConnectionException(
       Throwable cause, ConnectionImpl connectionImpl);
```

DatabaseMetaDataException

```
public class DatabaseMetaDataException
    extends Throwable {
  final static Logger logger = (Logger) LogManager
```

```
      .getLogger(DatabaseMetaDataException.class);

  public DatabaseMetaDataException();

  public DatabaseMetaDataException(
      String msg, String metodo);
```

MenuException

```
public class MenuException extends Throwable {
  final static Logger logger = (Logger) LogManager
      .getLogger(MenuException.class);
  ConnectionImpl connectionImpl;

  public MenuException(
      Throwable cause, String action, String metodo);

  public MenuException();
```

ResultSetException

```
public class ResultSetException extends Throwable {
  final static Logger logger = (Logger) LogManager
      .getLogger(ResultSetException.class);

  public ResultSetException();

  public ResultSetException(
      String msg, ResultSet resultSet) throws SQLException;
```

SQLCommandException

```
public class SQLCommandException extends Throwable
  private SQLCommand sqlCommand;
  final static Logger logger = (Logger) LogManager
      .getLogger(SQLCommandException.class);

  public SQLCommandException(
      SQLCommand aThis, int operacion);

  public SQLCommandException(
      SQLCommand aThis, SQLException ex);

  public SQLCommandException(
      String TABLE_SCHEM, String TABLE_NAME,
      String localizedMessage);
```

TablasActionException

```
public class TablasActionException extends Throwable {
  final static Logger logger = (Logger) LogManager
      .getLogger(TablasActionException.class);
  ConnectionImpl connectionImpl;
```

```
  public TablasActionException(String message);

  public TablasActionException(
       String message, Throwable cause);

  public TablasActionException(Throwable cause);
```

Paquete es.ubu.alu.mydatabasejc.jdbc

ConnectionInfo

```
/**
 * Define los métodos informativos de la conexión
 * @author jhuidobro
 */
public class ConnectionInfo {
  protected Connection connection;

  public boolean isReadOnly() throws SQLException;
  public String getSchema() throws SQLException;
  public boolean getAutoCommit() throws SQLException;
  public String getCatalog() throws SQLException;
  public Properties getClientInfo() throws SQLException;
  public int getHoldability() throws SQLException;
  public int getNetworkTimeout() throws SQLException;
  public int getTransactionIsolation() throws SQLException;
  public ConnectionInfo(Connection connection);
}
```

ConnectionImpl

```
/**
 * Establece y mantiene la información de una conexión
 * con la base de datos
 * @author jhuidobro
 */
public class ConnectionImpl {
  protected Connection connection;
  private String url;
  private String usuario;
  private String password;
  private String driver;

  /**
   * Cierra la conexión si estuviera establecida
   */
  public void close();

  /**
   * Instancia un objeto de la clase ConnectionImpl
   * @param url URL de conexión a la base de datos
   * @param usuario Usuario de la base de datos
   * @param password Contraseña del usuario
   * @throws ConnectionException Cualquier error
   * producido en el proceso
   * de conexión devuelve una excepción que es pasada
```

```
   * previamente por el log del sistema
   */
  public ConnectionImpl(String url, String usuario, String password)
throws ConnectionException;

  /**
   * Comprueba si la conexión a la base de datos está
   * o no cerrada
   * @return true si la conexión está cerrada, false
   * en caso contrario
   */
  public boolean isClosed();

  /**
   * Comprueba si la conexión con la base de datos es
   * válida
   * @param timeout Ver el mismo parámetro de
   * java.sql.Connection.isValid
   * @return true si la conexión es una conexión
   * válida, false en caso contrario
   */
  public boolean isValid(int timeout);

  /**
   * Cuando el objeto es recolectado para su
   * destrucción, se cierra
   * la conexión con la base de datos, si es que
   * existía alguna.
   * @throws Throwable
   */
  @Override
  protected void finalize() throws Throwable;
  }
```

DatabaseMetaDataImpl

```
/**
 * Implementa los métodos de DatabaseMetaData y se añaden
 * los métodos necesarios para agrupar los que devuelven
 * información en formato simple bajo anotaciones
 * MetaDataInfoCategorias
 * @author jhuidobro
 */
public class DatabaseMetaDataImpl
    implements DatabaseMetaData {

  protected DatabaseMetaData metadata;

  private Method[] getMetodos(String categoria);

  public DatabaseMetaDataImpl(DatabaseMetaData metadata);

  public List<List> getBasicInfo();
  public List<List> getConnInfo();
  public List<List> getDataBaseInfo();
```

```
  public List<List> getDataInfo();
  public List<List> getUserInfo();
  public List<List> getSQLInfo();
  private List<List> getInfo(Method[] methods);

  @Override
  @MetaDataInfoCategorias(categoria = "Usuario")
  public boolean allProceduresAreCallable() throws SQLException;
       ...

  @Override
  public String getURL() throws SQLException;
       ...

  @Override
  @MetaDataLink(action = "consulta",
       namespace = "/tablas",
       parametros =
          {"TABLE_CAT","TABLE_SCHEM","TABLE_NAME","TABLE_TYPE"},
       columnNumber = 3)
  public ResultSet getTables(
       String catalog, String schemaPattern,
       String tableNamePattern, String[] types)
       throws SQLException;
```

SQLCommand

```
/**
 * Clase que permite al usuario ejecutar comandos sql
 * contra la base de datos
 * @author jhuidobro
 */
public class SQLCommand {

  public static int OPERACION_UPDATE = 1;
  public static int OPERACION_DELETE = 2;
  public static int OPERACION_INSERT = 4;
  public static int OPERACION_WHERE = 8;
  public static int ISAUTOINCREMENT = 1;
  public static int ISCASESENSITIVE = 2;
  public static int ISCURRENCY = 4;
  public static int ISDEFINITELYWRITABLE = 8;
  public static int ISNULLABLE = 16;
  public static int ISREADONLY = 32;
  public static int ISSEARCHABLE = 64;
  public static int ISSIGNED = 128;
  public static int ISWRITABLE = 256;
  public static int ISALL = -1;
  public ConnectionImpl connectionImpl;
  public String cadenaWhere;
  public List<Object> listaParametrosWhere;
  public String cadenaSet;
  public List<Object> listaParametrosSet;
  public String sql;
  public String cadenaInsert;
  public String cadenaInsert2;
  public List<Object> listaParametrosInsert;

  public static Object getValor(int tipo, String valor);
```

```
public int executeUpdate(String esquema, String tabla,
     Map<String, Integer[]> mapa, String[] campos,
     String[] valores) throws SQLCommandException;

private ResultSet executeQuery(
     String TABLE_SCHEM, String TABLE_NAME,
     int TYPE_SCROLL, int CONCUR)
     throws SQLCommandException;

public ResultSet executeQuery(
     String TABLE_SCHEM, String TABLE_NAME,
     Set<String> arrayParametros,
     Map<String, Object> sesion)
     throws SQLCommandException;

public void setSQLPartList(
     String TABLE_SCHEM, String TABLE_NAME,
     Set<String> arrayParametros,
     Map<String, Object> sesion)
     Throws SQLCommandException;

public Map<String, Integer> getMap(
     Map<String, Integer[]> mapa, int tipo);

public Map<String, Integer[]> getMap(
     String TABLE_SCHEM, String TABLE_NAME);

public ResultSet executeQuery(
     String TABLE_SCHEM, String TABLE_NAME,
     Map<String, Integer[]> mapa,
     String[] pkArgumentos, String[] pkValores,
     int TYPE_SCROLL, int CONCUR)
     throws SQLCommandException;

private void setSQLPartList(
     Map<String, Integer[]> mapa,
     String[] pkArgumentos, String[] pkValores,
     int tipo)
     thows SQLommandException;

/**
 * Siempre será update
 *
 * @param pkArgumentos
 * @param pkValores
 * @param campos
 * @param valores
 */
private void setSQLPartList(
     Map<String, Integer[]> mapa,
     String[] pkArgumentos, String[] pkValores,
     String[] campos, String[] valores)
     throws SQLCommandException;

private String getEsquema(String esquema);
```

```
  public int executeUpdate(String esquema, String tabla,
       Map<String, Integer[]> mapa, int operacion,
       String[] pkArgumentos, String[] pkValores,
       String[] campos, String[] valores)
       throws SQLCommandException;
}
```

Paquete es.ubu.alu.mydatabasejc.actions

Este paquete engloba las clases con la lógica de negocio de la aplicación y gestiona el flujo de ejecución del programa apoyadas en el fichero de configuración struts.xml.

LoginAction

Clase de la que heredan las demás, al menos aquellas que deban validar una conexión a la base de datos (en nuestro trabajo, todas).

```
public class LoginAction extends ActionSupport {
  protected String CONEXION = "conexion";
  private String url;
  private String usuario;
  private String password;

  public String ayuda(String propiedad);

  /**
   * Realiza la validación de los campos de formulario
   */
  @Override
  public void validate();

  /**
   * Presenta siempre la pantalla para logarse
   * @return "success"
   */
  public String logear();

  /**
   * Se valida el intento de login. Si se produce un error en una
   * conexión
   * previamente establecida, se intentará realizar de nuevo la
   * conexión
   * con los datos previos mediante llamada al método login()
   * @return "error" si no se puede validar el login
   * "success" en caso de que el login se haya validado
   * correctametne
   */
  public String validarLogin();

  /**
   * Se valida el intento de login. Si se produce un error en una
   * conexión
   * previamente establecida, se intentará realizar de nuevo la
```

```
   * conexión
   * con los datos previos mediante llamada al método login()
   * @return "error" si no se puede validar el login
   * "success" en caso de que el login se haya validado
   * correctametne
   * @param connectionImpl Objeto que se debe validar
   */
  protected String validarLogin(ConnectionImpl connectionImpl);

  /**
   * Establece una conexión física con la base de datos a partir de
   * los
   * datos recibidos desde el formulario HTML en las propiedades
   * pertinentes
   * y añade el objeto a la sesión web del usuario
   * @return "error" si no se puede hacer login
   * "success" en caso de que se haya realizado el login
   * correctamente
   */
  public String login();

  /**
   * Intenta conseguir la conexión física con la base de datos. Si
   * la consigue
   * la guarda en la sesión del usuario
   * @param url URL a la base de datos
   * @param usuario Usuario que se quiere conectar a la base de
   * datos
   * @param password Clave del usuario
   * @return "error" si no se consigue la conexión, "success" en
   * caso contrario
   */
  private String login(String url, String usuario, String password);

    ...
}
```

MenuAction

Hereda de LoginAction y de ella heredan todas las acciones que finalmente muestren el menú de usuario.

```
public class MenuAction extends LoginAction implements SessionAware {
  protected String ACTION_ERROR = "ACTION_ERROR";
  protected String ACTION_MESSAGE = "ACTION_MESSAGE";
  private List<Menu> menus;
  protected String[] filtroArgumentos;
  protected String[] filtroValores;
  protected Map<String, Object> sesion;

  protected void cierraRS(ResultSet rs, PreparedStatement ps);

   /**
   * Obtiene en una List<List> el resultset recibido. El primer
```

```
     * elemento de
     * la lista representa los nombres de los campos del resultset
     * subyacente
     * @param resultSet Resultset a convertir en List<List>
     * @param metodoLink Indica si el método que proporciona el
     * resultset debe
     * tener un link cuando sea presentado en la jsp (true) o no
     * (false)
     * @linkParametros Cuando se requiera incluir una columna de link
     * (metodoLink = true)
     * se deberán indicar en este argumento la lista de parámetros que
     * se deben
     * incluir en la primera columna de la lista con la información
     * requerida
     * para realizar el link correctamente.
     * @return
     * @throws SQLException
     */
    protected List<List> getListInfo(ResultSet resultSet, boolean
           metodoLink, String[] linkParametros) throws SQLException,
           ResultSetException;

    /**
     * Obtiene en una List<List> el resultset recibido. Los elementos
     * de la lista
     * están organizados en forma de formulario, el primer elemento
     * contiene el
     * nombre del campo, el segundo contiene el valor. En principio
     * pensado para
     * resultset con un solo registro, aunque si tuviera más no
     * cambiaría la
     * organización de los datos.
     * @param resultSet Resultset a convertir en List<List>
     * @param metodoLink Indica si el método que proporciona el
     * resultset debe
     * tener un link cuando sea presentado en la jsp (true) o no
     * (false)
     * @param linkParametros Cuando se requiera incluir una columna de
     * link (metodoLink = true)
     * se deberán indicar en este argumento la lista de parámetros que
     * se deben
     * incluir en la primera columna de la lista con la información
     * requerida
     * para realizar el link correctamente.
     * @param argumentos Si este parametro es null, la primera columna
     * contendrá
     * la cedena de la forma
     * linkParmetro1=valor1&linkParametro2=valor2. Pero si
     * este parametro no es nulo la cadena será de la forma
     * [argumentos]=linkParametro1&
     * [valores]=valor1&[argumentos]=linkParametro2&[valores]=valor2
     * @param valores Contiene el nombre del parámetro bajo el que se
     * incluyen los
     * valores
     * @return
     * @throws SQLException
     */
    protected List<List> getListInfoReverse(ResultSet resultSet);

    /**
```

```
   * Obtiene en una List<List> el resultset recibido. El primer
   * elemento de
   * la lista representa los nombres de los campos del resultset
   * subyacente
   * @param resultSet Resultset a convertir en List<List>
   * @param metodoLink Indica si el método que proporciona el
   * resultset debe
   * tener un link cuando sea presentado en la jsp (true) o no
   * (false)
   * @param linkParametros Cuando se requiera incluir una columna de
   * link (metodoLink = true)
   * se deberán indicar en este argumento la lista de parámetros que
   * se deben
   * incluir en la primera columna de la lista con la información
   * requerida
   * para realizar el link correctamente.
   * @param argumentos Si este parametro es null, la primera columna
   * contendrá
   * la cedena de la forma
   * linkParmetro1=valor1&linkParametro2=valor2. Pero si
   * este parametro no es nulo la cadena será de la forma
   * [argumentos]=linkParametro1&
   * [valores]=valor1&[argumentos]=linkParametro2&[valores]=valor2
   * @param valores Contiene el nombre del parámetro bajo el que se
   * incluyen los
   * valores
   * @return
   * @throws SQLException
   */
  protected List<List> getListInfo(ResultSet resultSet, boolean
         metodoLink, String[] linkParametros, String argumentos,
         String valores) throws SQLException, ResultSetException;

  public List<Menu> getMenus()

    ...
}
```

DatabaseMetaDataAction

```
public class DatabaseMetaDataAction extends MenuAction implements
Preparable, SessionAware {

  private ConnectionImpl connectionImpl;
  private List<PropiedadValor> listInicial;
  private String metodo;
  private List<List> listInfo;
  private String parametros;
  Parameter[] arrayParametros; // parametros del método invocado
  private String linkAction;
  private String linkNamespace;
  private int linkColumnNumber;
  private boolean metodoLink;
  private String[] linkParametros;

  public Object getParameter(int i);
```

```
public List getArrayParametros();

/**
 * Invoca el método de DatabaseMetaData con los parámetros
 * adecuados y forma la
 * lista con la información a presentar
 * @return "success" si no se produce error, "error" en caso
 * contrario
 */
public String info();

/**
 * Invoca el método de DatabaseMetaData con los parámetros
 * adecuados y forma la
 * lista con la información a presentar
 * @return "success" si no se produce error, "error" en caso
 * contrario
 */
public String resultset();

/** Obtenidos los tipos de los parámetros, se convierten los
 * valores
 * recibidos a los tipos correspondientes y se ponen en la sesión
 * para
 * sucesivas llamadas a este o a otros métodos que usen el mismo
 * parámetro (con el mismo nombre)
 * Convierte cada valor de filtroValores al tipo necesario según
 * se indica
 * en parameterTypes y lo añade a la sesión del usuario con el
 * nombre
 * correspondiente en filtroArgumentos
 *
 * @param parameterTypes Array de clases de los argumentos del
 * método que
 * se va a invocar en un momento posterior. Este array marca el
 * orden en el
 * que se procesan los elementos de filtroArgumentos y de
 * filtroValores
 */
private void setParametrosSesion(Class[] parameterTypes);

private Class[] getParameterTypes(String parametros) throws
       IOException, ClassNotFoundException ;

/**
 * Ejecuta el método del objeto databaseMetaData indicado.
 * Devuelve el resultset
 * resultado de la ejecución.
 * @param databaseMetaData
 * @param metodo Nombre del método a invocar
 * @param parametros Array de clases que conforman los parámetros
 * del método
 * codificado en Base64
 * @return
 * @throws IllegalAccessException
 * @throws IllegalArgumentException
 * @throws NoSuchMethodException
 * @throws InvocationTargetException
 * @throws DatabaseMetaDataException Si el resultado de la
```

```
   * ejecución del método
   * no produce un objeto de tipo ResultSet se genera una excepción
   * de este tipo
   */
  private Object getInvoke(DatabaseMetaData databaseMetaData,
         String metodo, Class[] parameterTypes)
         throws IllegalAccessException, IllegalArgumentException,
            NoSuchMethodException, InvocationTargetException,
            DatabaseMetaDataException, IOException,
            ClassNotFoundException;

   /**
   * Se valida la conexión obtenida. Si error, se pasará a login.jsp
   */
  @Override
  public void validate();

  public String inicial();

  /**
   * Se obtiene la conexión de la sesión del usuario
   *
   * @throws Exception
   */
  @Override
  public void prepare() throws Exception;

    ...
}
```

TablasAction

```
public class TablasAction extends MenuAction implements Preparable,
SessionAware {
  private ConnectionImpl connectionImpl;
  private Map<String, Object> sesion;
  private String TABLE_CAT;
  private String TABLE_SCHEM;
  private String TABLE_NAME;
  private String TABLE_TYPE;
  private List<List> listInfo;
  private Set<String> arrayParametros;
  private String[] pkArgumentos;
  private String[] pkValores;
  private boolean metodoLink;
  private String[] formCampos;
  private String[] formValores;
  private Map<String, Integer> mapaEditables;
  private Map<String, Integer[]> mapaCompleto;

  public boolean isEditDisabled(String campo);

  public String insertarGuardar();

  public String insertar();
```

```
  public Object getPkValores(int i);

  public String borrar();

  public Object getParameter(int i);

  /** Obtenidos los tipos de los parámetros, se convierten los
   * valores
   * recibidos a los tipos correspondientes y se ponen en la sesión
   * para
   * sucesivas llamadas a este método
   * Convierte cada valor de filtroValores al tipo necesario según
   * se indica
   * en el mapa obtenido de la sesion y grabado previamente, en la
   * primera
   * ejecución y lo añade a la sesión del usuario con el nombre
   * correspondiente en filtroArgumentos
   *
   * @param tabla nombre del esquema y la tabla. Se buscará en la
   * sesión un objeto map
   * con este nombre. Contendrá un mapa de los campos del resultset
   * y su tipo
   * @return mapa del resultset de la tabla con
   * @throws SQLException
   */
  private Map<String, Integer> setParametrosSesion(String tabla);

  /**
   * Consulta la tabla recibida en TABLE_NAME y en TABLE_SCHEM y la
   * presenta
   * en jsp en forma de lista
   * @return
   */
  public String consulta();

  /**
   * Se valida la conexión obtenida. Si error, se pasará a login.jsp
   */
  @Override
  public void validate();

  @Override
  public void setSession(Map<String, Object> map);

  /**
   * Se obtiene la conexión de la sesión del usuario
   *
   * @throws Exception
   */
  @Override
  public void prepare() throws Exception;
}
```

3.3. Diseño procedimental

A continuación describiremos cada uno de los subsistemas que se han presentado. Realizaremos una descripción de la funcionalidad del sistema mediante la vista de casos de usos, una descripción del modelo de datos que soporta.

A continuación se muestra el diagrama inicial de casos de uso con indicación de los agentes que intervienen en las operaciones:

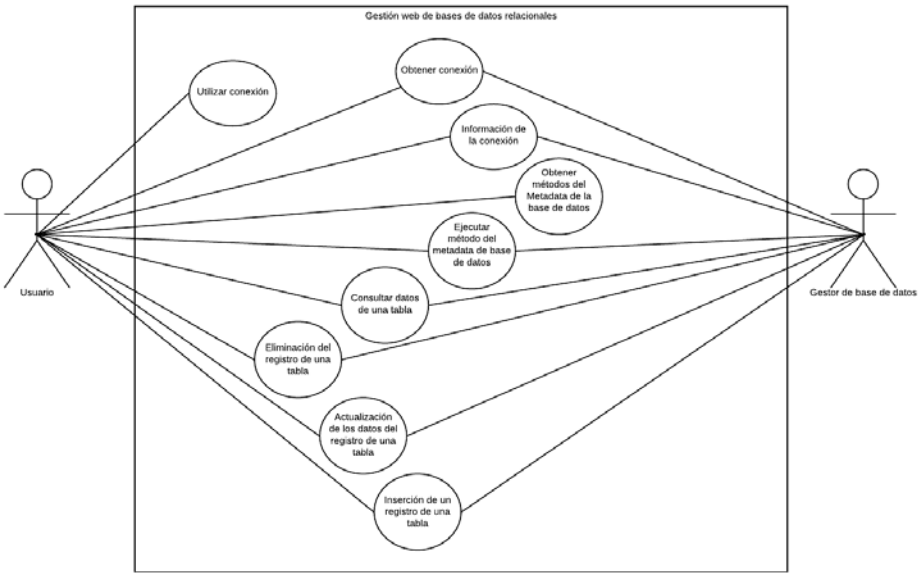

Figura 3: Diagrama inicial de casos de uso

Conexión a la base de datos

Para un usuario nuevo, una solicitud a la página principal de la aplicación web (index.html) redirige a la acción pertinente que presentará al usuario la página donde deberá introducir los datos necesarios para establecer la conexión:

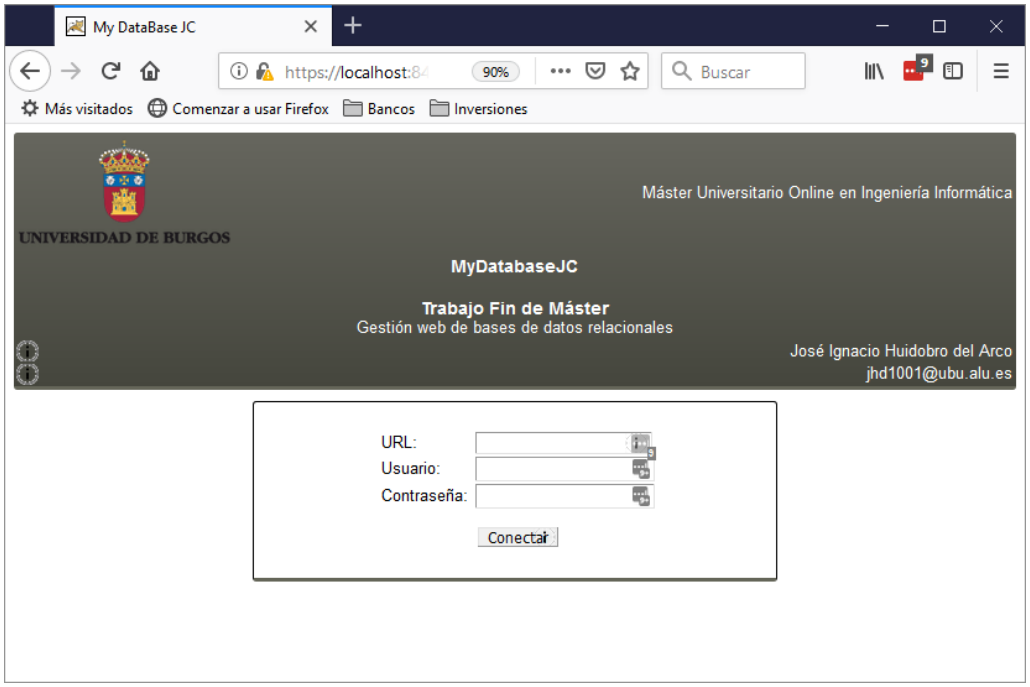

Figura 4: Pantalla de conexión con la base de datos

El proceso que recibe la request con los datos solicitados debe analizar la URL y en función de ella realizar la carga del driver JDBC correspondiente a la base de datos a la que se quiere conectar. Una vez producida la carga del driver, el proceso deberá intentar realizar una conexión física con la base de datos según los datos recibidos.

Si el intento de conexión es válido, el proceso instanciará un objeto de la clase ConnectionImpl que almacenará en la sesión del usuario mantenida por el servidor de aplicaciones web.

Escenario de casos de uso Obtener conexión	
Caso de uso	Obtener conexión
Actores	Usuario, Base de datos
Descripción	Para que el usuario realice una conexión debe enviar los datos de conexión. La base de datos debe validar los datos de conexión
Precondición	1. El usuario debe conocer los datos de conexión. 2. La aplicación debe disponer del driver JDBC apropiado para la base de datos a la que se quiere conectar
Flujo normal	1. El usuario rellena el formulario de conexión.

Escenario de casos de uso Obtener conexión	
	2. El sistema decide que driv.er debe cargar en función de la URL. 3. El sistema carga el driver JDBC. 4. El sistema solicita una conexión a la base de datos con los datos recibidos. 5. La base de datos acepta la solicitud y retorna una conexión válida a la aplicación. 6. La aplicación almacena en la sesión del usuario la conexión y los datos de conexión para su uso entre llamadas por parte del mismo usuario.
Flujo alterno	Flujo alterno A La URL no informa de un driver posible. 1. El sistema analiza la URL proporcionada y en función de su contenido determina qué driver JDBC debe ser cargado. 2. El sistema informa al usuario de que la URL no proporciona un driver disponible. 3. El usuario se mantiene en la pantalla de login. Flujo alterno B Error al cargar el driver. 1. El sistema intenta cargar el driver determinado por la URL. 2. El sistema no puede cargar el driver. 3. Se informa al usuario del error al cargar el driver. 4. El usuario se mantiene en la pantalla de login. Flujo alterno C Error estableciendo conexión. 1. El sistema intenta establecer una conexión. 2. La base de datos informa de un error al sistema en el intento de conexión. 3. El sistema informa al usuario del error en el intento de conexión. 4. El usuario se mantiene en la pantalla de login.
Postcondición	El usuario ha realizado una conexión a la base de datos.

Tabla 37: Caso de uso Obtener conexión

Cada vez que el usuario realice una solicitud a la aplicación que requiera la utilización de la conexión a la base de datos, el sistema debe realizar una comprobación de varias situaciones necesarias para que la conexión pueda utilizarse.

Escenario de casos de uso Utilizar conexión	
Caso de uso	Utilizar conexión.
Actores	Usuario, Base de datos.
Descripción	Para que el usuario utilice una conexión previamente establecida, se deben realizar una serie de pasos que se indican a continuación.
Precondición	1. El usuario debe haber establecido previamente una conexión. 2. La conexión debe haber sido almacenada en la sesión del usuario en el servidor web.
Flujo normal	1. El usuario solicita una función que requiere la utilización de una conexión. 2. La aplicación confirma que el usuario tiene en la sesión un objeto con una conexión. 3. La aplicación solicita a la base de datos que la conexión sea válida y que esté abierta. 4. La base de datos valida e informa de que la conexión es correcta y se encuentra abierta. 5. La aplicación proporciona al proceso la conexión solicitada.
Flujo alterno	Flujo alterno A Si el sistema detecta que el usuario no dispone de una sesión válida, se retorna al usuario a la pantalla de login y se cancela el proceso original. Flujo alterno B Si el sistema detecta que el usuario no dispone en la sesión de un objeto de la clase ConnectionImpl, se retorna al usuario a la pantalla de login y se cancela el proceso original. Flujo alterno C Si el sistema detecta que la conexión no es válida o no está abierta, se retorna al usuario a la pantalla de login y se cancela el proceso original. En todos los flujos alternos, el usuario recibe el correspondiente error en la pantalla de login a la que se ve desplazado.
Postcondición	El sistema proporciona la conexión con la base de datos.

Tabla 38: Caso de uso Utilizar conexión

Consulta de información de conexión

El usuario, una vez obtenida la conexión, recibirá la información de la conexión mostrando las propiedades más importantes de la misma.

Para obtener dicha información, se ejecutarán todos los métodos de la clase ConnectionInfo y se mostrará la información obtenida en forma de tabla html.

Escenario de casos de uso Información de la conexión	
Caso de uso	Información de la conexión.
Actores	Usuario, Base de datos.
Descripción	El usuario solicita la información de la conexión y se le muestra en pantalla.
Precondición	1. El usuario obtiene una conexión válida y activa.
Flujo normal	1. El usuario solicita la información de la conexión. 2. El sistema instancia un objeto de la clase ConnectionInfo. 3. El sistema obtiene una lista de los métodos de la clase. 4. Para cada método, el sistema ejecuta el método y almacena el resultado. 5. El sistema proporciona al usuario la lista de métodos y valores obtenidos en su ejecución.
Flujo alterno	
Postcondición	

Tabla 39: Caso de uso Información de la conexión

Métodos de los metadata de la base de datos

Para que el usuario pueda realizar todas las operaciones requeridas con la base de datos, se le proporcionará una lista con los métodos disponibles relacionados con la base de datos. Existen una gran cantidad de métodos para obtener información de la base de datos, aunque solo nos centraremos en dos tipos:

- Métodos que, con o sin parámetros, devuelven información en forma de resultset.
- Métodos que, sin parámetros, devuelven información simple, un solo dato.

Para realizar esta funcionalidad, se crea una clase DatabaseMetaDataImpl que implementa todos los métodos del interface DatabaseMetaData y además implementa una serie de métodos que engloban en categorías

aquellos métodos que, sin parámetros, devuelven datos simples. Estos métodos implementados retornarán la información en forma de lista de objetos PropiedadValor. Por lo tanto, finalmente, el sistema solo deberá contemplar de la clase DatabaseMetaDataImpl:

- Métodos que, con o sin parámetros, devuelven información en forma de resultset
- Métodos que, sin parámetros, devuelven información en forma de lista de PropiedadValor

Escenario de casos de uso Obtener métodos del Metadata de la base de datos	
Caso de uso	Obtener métodos del metadata de la base de datos.
Actores	Usuario, Base de datos
Descripción	El usuario solicita la lista de métodos. El sistema proporciona la lista de métodos junto con los tipos de parámetros necesarios para su ejecución.
Precondición	1. El usuario obtiene una conexión válida y activa.
Flujo normal	1. El usuario solicta la información de la base de datos. 2. El sistema instancia un objeto de la clase DatabaseMetaDataImpl. 3. El sistema recopila la lista de métodos de la clase. 4. Para cada método, almacena nombre y clases de los argumentos si el tipo devuelto es resultset o list. 5. El sistema proporciona al usuario la lista de métodos con sus nombres y la lista de tipos de sus argumentos.
Flujo alterno	
Postcondición	Para conseguir enviar al usuario la lista de los tipos de argumentos, esta debe ser convertida a un outputstream y posteriormente codificada en base64 puesto que la información va a viajar en la response y posteriormente volverá a la aplicación en una request.

Tabla 40: Caso de uso Obtener métodos del Metadata de la base de datos

Java Reflection es un módulo integrado en el J2EE que permite, entre otras cosas, la ejecución de un método de una clase y la recepción del resultado en el objeto adecuado. Una vez que el usuario dispone de los métodos proporcionados por JDBC para consulta de bases de datos, el sistema le debe proporcionar la posibilidad de ejecutarlos adecuadamente y recoger y devolver los resultados al usuario.

Escenario de casos de uso Ejecutar un método del metadata de la base de datos.	
Caso de uso	Ejecutar un método del metadata de la base de datos.
Actores	Usuario, Base de datos.
Descripción	El usuario solicita la ejecución de un determinado método de los metadatos de la base de datos proporcionados por la clase DatabaseMetaDataImpl. El sistema le presenta los resultados obtenidos. Opcionalmente, el usuario puede añadir a la request uno o más argumentos para la ejecución del método. El nombre del parámetro debe coincidir con el nombre del argumento para que su asociación tenga validez.
Precondición	1. El usuario obtiene una conexión válida y activa. 2. El usuario dispone del nombre del método y de los tipos de sus argumentos así como de los valores que debe asignar a cada uno. 3. La lista de tipos de los argumentos debe estar codificada en base64 para que le lleguen correctamente al sistema y pueda, a partir de ahí, obtener los tipos de los argumentos en el formato adecuado.
Flujo normal	1. El usuario envía al sistema una request en la que se indica el método y los tipos de los argumentos que debe ejecutarse. 2. El sistema averigua los nombres de los argumentos del método que el usuario desea ejecutar. 3. Si el usuario incluye en la request algún parámetro que corresponda con un argumento de la ejecución del método, el sistema lo convierte al tipo adecuado y lo almacena en la sesión del usuario. 4. El sistema busca en la sesión del usuario los datos de los argumentos por su nombre. 5. El sistema invoca al método pasándole los valores de los argumentos necesarios. 6. El sistema obtiene el resultado de la invocación (resultset o List) y lo presenta al usuario.
Flujo alterno	Flujo alterno A Los tipos de argumentos pasados al sistema no se corresponden con los que el método requiere. Se presentará al usuario el error correspondiente.

Escenario de casos de uso Ejecutar un método del metadata de la base de datos.	
	Flujo alterno B Los valores de los argumentos pasados al sistema en la request no pueden ser convertidos al tipo adecuado. Se presentará al usuario el error correspondiente. Flujo alterno C El número de datos obtenidos excede el límite establecido. Se presenta el mensaje al usuario.
Postcondición	Los datos son presentados al usuario en forma de List<List> que permite recorrerlos mediante un doble bucle y presentarlos en forma de tabla HTML

Tabla 41: Caso de uso Ejecutar un método del metadata de la base de datos

Trabajo con tablas

Una de las operaciones más importantes de la aplicación debe ser, como no podía ser de otra manera, la posibilidad de consultar, modificar, borrar o insertar registros en las tablas de la base de datos, puesto que, al fin y al cabo, esta es la utilidad última de una base de datos.

Para realizar estas operaciones, los sistemas de bases de datos relacionales disponen de un lenguaje de consulta y manipulación de datos en tablas denominado SQL.

La aplicación debe proporcionar los mecanismos necesarios para que el usuario pueda lanzar los comandos SQL más importantes:

- SELECT
- DELETE
- UPDATE
- INSERT

En todos estos comandos debe indicarse el nombre de la tabla sobre la que se debe realizar la operación. Puede ir o no acompañada del nombre del esquema propietario de la tabla.

En algunos casos se debe incluir la colección de campos/valor que permiten filtrar o acotar el ámbito de datos de ejecución del comando, y en otros se debe incluir además, los datos nuevos que deben ser almacenados en la base de datos,

bien en una operación de actualización (UPDATE) bien en una operación de inserción (INSERT).

Para componer una correcta instrucción SQL se crea la clase SQLCommand que se explica en el diccionario de datos.

A continuación se indican los distintos casos de uso dentro de esta funcionalidad que corresponden con las operaciones básicas sobre datos solicitadas en los requerimientos.

Escenario de casos de uso Consultar datos de una tabla.	
Caso de uso	Consultar datos de una tabla.
Actores	Usuario, Base de datos.
Descripción	El usuario solicita la consulta de los datos de una tabla. El sistema solicita a la base de datos la consulta de los datos de la tabla y devuelve al usuario los datos obtenidos.
Precondición	1. El usuario obtiene una conexión válida y activa. 2. El usuario dispone del nombre de la tabla y, opcionalmente, la colección de campos y valores que representan el filtro a aplicar en la colección de datos.
Flujo normal	1. El usuario envía al sistema una request en la que se indica la tabla que se desea consultar. 2. Opcionalmente, el usuario puede indicar el esquema al que pertenece la tabla. 3. Opcionalmente, el usuario puede indicar uno o varios campos y sus correspondientes valores para conformar la matriz de filtrado de datos. 4. El sistema convierte los datos recibidos como parte del filtro en el tipo adecuado. Para ello debe conocer previamente los tipos de cada uno de los campos de la tabla. 5. El sistema guarda en la sesión del usuario los valores recibidos como parámetros de filtrado para que el usuario pueda reutilizarlos en consultas posteriores. 6. El sistema compone la consulta en lenguaje SQL estándar. 7. El sistema traslada la consulta SQL a la base de datos. 8. La base de datos comienza la transferencia de datos en forma de ResultSet. 9. El sistema convierte los datos a formato List<List> y los entrega al usuario.

Escenario de casos de uso Consultar datos de una tabla.	
Flujo alterno	Flujo alterno A Alguno de los datos enviados por el usuario no forma parte de los posibles campos de consulta de la tabla. Se presenta el correspondiente error al usuario. Flujo alterno B Los valores de los campos pasados al sistema como parámetros del filtro en la request no pueden ser convertidos al tipo adecuado. Se presentará al usuario el error correspondiente. Flujo alterno C El número de datos obtenidos excede el límite establecido. Se presentan solo el número de registros establecido en el límite.
Postcondición	Los datos son presentados al usuario en forma de List<List> que permite recorrerlos mediante un doble bucle y presentarlos en forma de tabla HTML.

Tabla 42: Caso de uso Consultar datos de una tabla

En los casos de uso que vienen a continuación, los comandos SQL que se utilizan para realizar la operación de modificación de datos no precisan la obtención previa de un resultset. Cuando pretendemos borrar un registro con una conexión JDBC tenemos dos opciones:

a. Obtenemos en un resultset el registro o registros que deseamos eliminar (o actualizar si este fuera el caso). A continuación, y antes de cerrar el resultset, utilizamos el método deleteRow() (o el método updateRow()) para trasladar al registro subyacente de la base de datos los cambios realizados a nivel del resultset.
b. Utilizamos el método executeUpdate de la clase PreparedStatement para enviar al gestor de base de datos el correspondiente comando SQL y dar por finalizada la operación.

En el proyecto se ha utilizado la opción b. por lo que no es necesario establecer un resultset previo que seleccione el registro a eliminar o actualizar.

Escenario de casos de uso Eliminación del registro de una tabla.	
Caso de uso	Eliminar un registro de una tabla.
Actores	Usuario, Base de datos.
Descripción	El usuario solicita la eliminación del registro de una tabla. El sistema solicita a la base de datos la eliminación del registro de la tabla y devuelve al usuario el resultado de la operación.
Precondición	1. El usuario obtiene una conexión válida y activa.

Escenario de casos de uso Eliminación del registro de una tabla.	
	2. El usuario dispone del nombre de la tabla. 3. El usuario dispone del nombre y valor de los campos que forman la primary key del registro que se quiere eliminar.
Flujo normal	1. El usuario envía al sistema una request en la que se indica la tabla que se desea consultar. 2. Opcionalmente, el usuario puede indicar el esquema al que pertenece la tabla. 3. El usuario debe indicar, obligatoriamente, los campos que conforman la primary key de la tabla y los valores para ellos que localizan unívocamente al registro a eliminar. 4. El sistema convierte los datos recibidos como parte del filtro (la primary key) en el tipo adecuado. Para ello debe conocer previamente los tipos de cada uno de los campos de la tabla. 5. El sistema compone la consulta en lenguaje SQL estándar. 6. El sistema traslada la consulta SQL a la base de datos. 7. La base de datos ejecuta la sentencia SQL retornando el número de registros afectados por la operación. 8. El sistema presenta al usuario el resultado de la operación mediante el mensaje oportuno.
Flujo alterno	Flujo alterno A Alguno de los datos enviados por el usuario no forma parte de los posibles campos de la primary key de la tabla. Se presenta el correspondiente error al usuario. Flujo alterno B Los valores de los campos pasados al sistema como parámetros de la primary key en la request no pueden ser convertidos al tipo adecuado. Se presentará al usuario el error correspondiente.
Postcondición	

Tabla 43: Caso de uso Eliminación del registro de una tabla

Escenario de casos de uso Actualización de los datos del registro de una tabla.	
Caso de uso	Actualización de los datos del registro de una tabla.
Actores	Usuario, Base de datos.
Descripción	El usuario solicita la modificación de un registro de una tabla. El sistema solicita a la base de datos la

Escenario de casos de uso Actualización de los datos del registro de una tabla.	
	modificación del registro de la tabla y devuelve al usuario el resultado de la operación.
Precondición	1. El usuario obtiene una conexión válida y activa. 2. El usuario dispone del nombre de la tabla. 3. El usuario dispone del nombre y valor de los campos que forman la primary key del registro que se quiere actualizar. 4. El usuario dispone de los nombres de los campos y los valores que desea modificar.
Flujo normal	1. El usuario envía al sistema una request en la que se indica la tabla que se desea consultar. 2. Opcionalmente, el usuario puede indicar el esquema al que pertenece la tabla. 3. El usuario debe indicar, obligatoriamente, los campos que conforman la primary key de la tabla y los valores para ellos que localizan unívocamente al registro a actualizar. 4. El usuario debe indicar, obligatoriamente, los campos que se desean modificar junto a los nuevos valores para dichos campos. 5. El sistema convierte los datos recibidos como parte del filtro (la primary key) y del conjunto de campos que se desean modificar, en el tipo adecuado. Para ello debe conocer previamente los tipos de cada uno de los campos de la tabla. 6. El sistema compone la consulta en lenguaje SQL estándar. 7. El sistema traslada la consulta SQL a la base de datos. 8. La base de datos ejecuta la sentencia SQL retornando el número de registros afectados por la operación. 9. El sistema presenta al usuario el resultado de la operación mediante el mensaje oportuno.
Flujo alterno	Flujo alterno A Alguno de los datos enviados por el usuario no forma parte de los posibles campos de la primary key o del conjunto de campos modificables de la tabla. Se presenta el correspondiente error al usuario. Flujo alterno B Los valores de los campos pasados al sistema como parámetros de la primary key o del conjunto de campos a modificar en la request no pueden ser convertidos al tipo adecuado. Se presentará al usuario el error correspondiente.

Escenario de casos de uso Actualización de los datos del registro de una tabla.
Postcondición

Tabla 44: Caso de uso Actualización de los datos del registro de una tabla

Escenario de casos de uso Inserción de un registro de una tabla.	
Caso de uso	Inserción del registro de una tabla.
Actores	Usuario, Base de datos.
Descripción	El usuario solicita la inserción de un registro de una tabla. El sistema solicita a la base de datos la inserción del registro de la tabla y devuelve al usuario el resultado de la operación.
Precondición	1. El usuario obtiene una conexión válida y activa. 2. El usuario dispone del nombre de la tabla. 3. El usuario dispone de los nombres de los campos y los valores que desea insertar.
Flujo normal	1. El usuario envía al sistema una request en la que se indica la tabla que se desea consultar. 2. Opcionalmente, el usuario puede indicar el esquema al que pertenece la tabla. 3. El usuario debe indicar, obligatoriamente, los campos que se desean insertar junto a los nuevos valores para dichos campos en un nuevo registro. 4. El sistema convierte los datos recibidos como parte del conjunto de campos que se desean insertar, en el tipo adecuado. Para ello debe conocer previamente los tipos de cada uno de los campos de la tabla. 5. El sistema compone la consulta en lenguaje SQL estándar. 6. El sistema traslada la consulta SQL a la base de datos. 7. La base de datos ejecuta la sentencia SQL retornando el número de registros afectados por la operación. 8. El sistema presenta al usuario el resultado de la operación mediante el mensaje oportuno.
Flujo alterno	Flujo alterno A Alguno de los datos enviados por el usuario no forma parte de los posibles campos de la primary key o del conjunto de campos modificables de la tabla. Se presenta el correspondiente error al usuario. Flujo alterno B Los valores de los campos pasados al sistema del conjunto de campos a insertar en la

Escenario de casos de uso Inserción de un registro de una tabla.	
	request no pueden ser convertidos al tipo adecuado. Se presentará al usuario el error correspondiente.
Postcondición	

Tabla 45: Caso de uso Inserción de un registro de una tabla

Diagrama estructurado o modelo de casos de uso

A continuación se muestran en un diagrama estructurado las interrelaciones entre los casos de uso descritos. Se han incorporado dos especializaciones del caso de uso general “Ejecutar método del metadata de la base de datos”:

- Ejecutar método getTables: Es el caso de uso especialidado, utilizado para conocer los nombres de las tablas disponibles en el sistema.
- Ejecutar método getPrimaryKey: Es el caso de uso especializado que es necesario ejecutar para obtener la información de la primary key de una tabla determinada. Esta información es necesaria para saber si, sobre dicha tabla, es posible realizar o no, operaciones de modificación e inserción de datos.

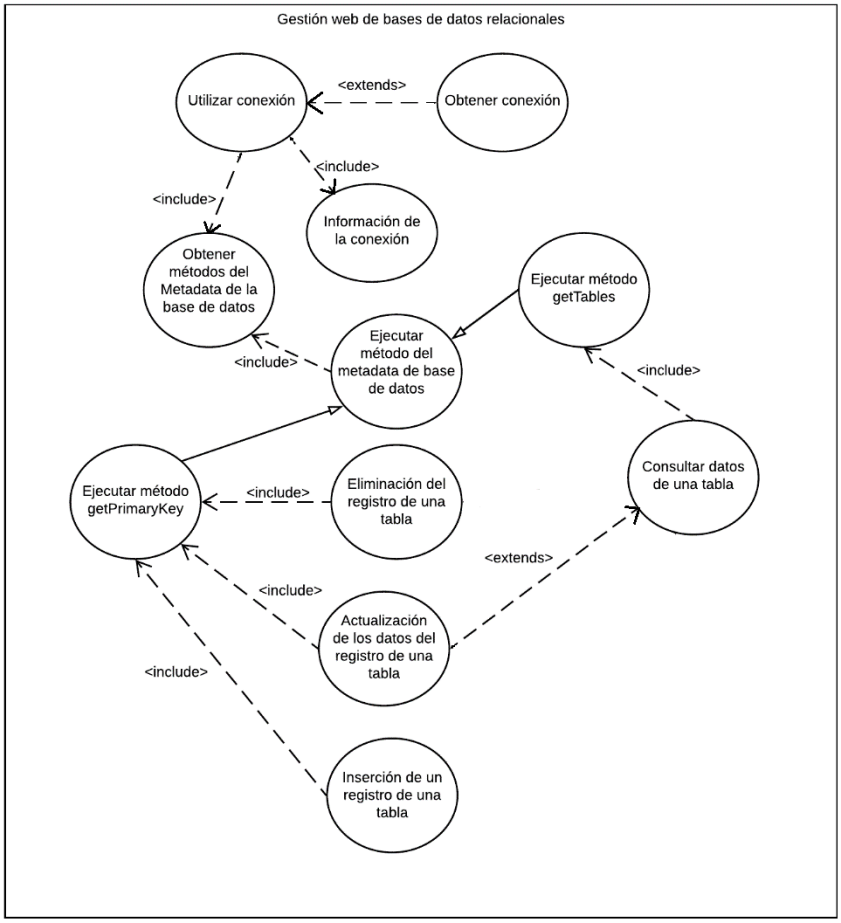

Figura 5: Diagrama estructurado de casos de uso

Interesante comentar que los casos “Eliminación…”, “Actualización...” e “Inserción…” constituyen una extensión del caso de uso “Consultar…” puesto que la vía inicial para llegar a su ejecución está en este último caso de uso. Así mismo, los 3 incluyen el subcaso de uso “Ejecutar método getPrimaryKey” para dar a entender que sólo si la tabla dispone de una clave primaria podrá, sobre ella, realizarse una operación de modificación de datos.

3.4. Diseño arquitectónico

A continuación, se describe el diseño de la aplicación en cuanto al flujo del programa atendiendo a cada uno de los casos de uso explicados. Para ello nos ayudaremos de un conjunto de gráficos y explicaciones junto con las correspondientes llamadas e instanciaciones de clases necesarias para realizar las tareas encomendadas.

Conexión a la base de datos

Obtener conexión

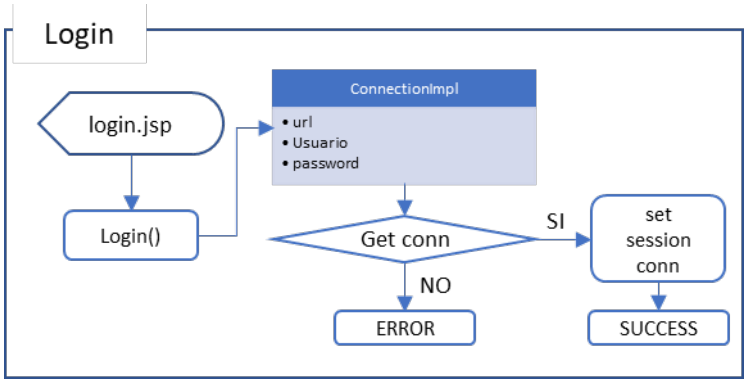

Figura 6 : Obtener conexión

En login.jsp, el formulario html recoje la información del usuario para establecer la conexión.

Mapeado por struts.xml, la ejecución continúa en el método login() de la clase LoginAction que instancia un objeto de ConnectionImpl y establece la conexión.

Una vez establecida la conexión se almacena el objeto ConnectionImpl en la sesión del usuario para poderse recuperar entre request distintas.

Utilizar conexión

Siempre que un usuario necesita una conexión con la base de datos se debe seguir el siguiente método para que el sistema se la proporcione:

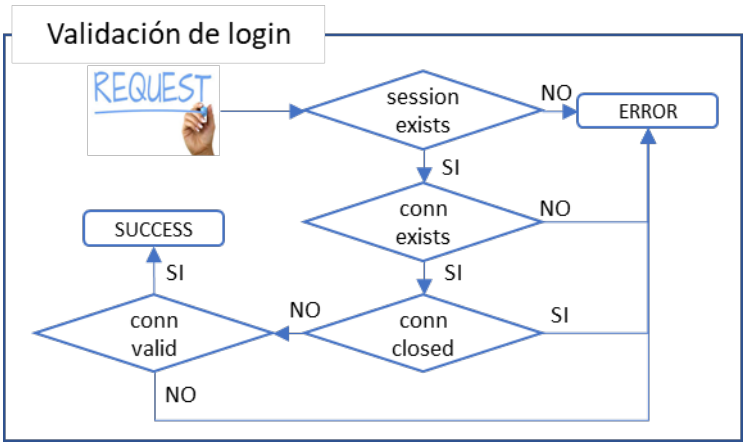

Figura 7: Utilización de login

En primer lugar, el sistema comprueba que la sesión del usuario exista. Para ello se llama al método validarLogin() de la clase LoginAction. Esta llamada se debe realizar en cada petición de conexión, por lo que la clase LoginAction es la clase padre de todas las clases actions de la aplicación y en todas ellas existe una llamada a dicho método a través de el método validate() que debe estar en cada clase action y en LoginAction.

Consulta de información de conexión

Información de la conexión

De forma automática, una vez que se inicia la aplicación y después de obtenida la conexión, se presenta al usuario la información general de la conexión mediante la llamada a una serie de métodos del objeto instanciado de la clase ConnectionInfo.

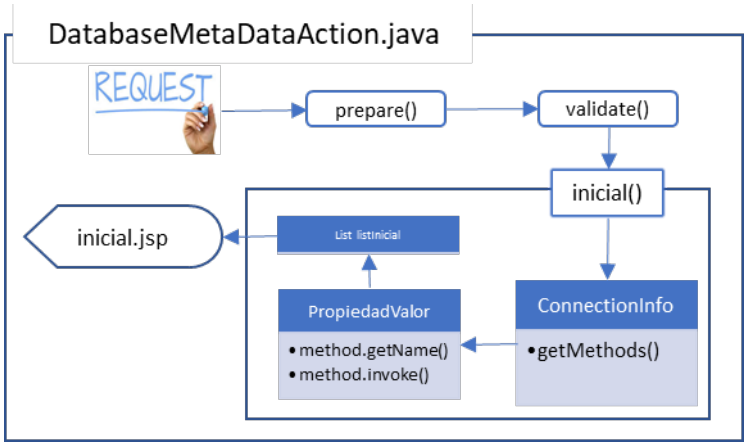

Figura 8: Presentación inicial de pantalla

El foco de ejecución del programa pasa al método inicial() de la clase DatabaseMetaDataAction que instancia un objeto de la clase ConnectionInfo y mediante java Reflection ejecuta los métodos en ella definidos y compone, junto a los nombres de esos métodos, con los resultados obtenidos de su ejecución una lista de objetos de la clase PropiedadValor que posteriormente son presentados al usuario en la página inicial.jsp.

En inicial.jsp, se recorre la lista de información presentándola al usuario.

Métodos de los metadata de la base de datos

Obtener métodos del Metadata de la base de datos

Constituyen lo que viene a ser el menú de trabajo de la aplicación. Al usuario se le presentan los distintos métodos que nos interesan del objeto DatabaseMetaData de la conexión para lo que implementamos nuestra propia clase DatabaseMetaDataImpl que utilizaremos para obtener estos métodos. El sistema obtiene la lista de métodos con la información necesaria de ellos para, en una fase posterior, poder realizar una invocación de estos métodos y obtener su resultado y presentarlo al usuario.

La lista de menús se obtiene del método getMenus() de la clase MenuAction, que hereda directamente de LoginAction y de la que heredan todas las acciones en las que sea necesario manejar el menú, como por ejemplo DatabaseMetaDataAction.

En el fichero menu.jsp se procede a la representación de las opciones del menú definiendo los enlaces para visualizar los resultados de seguimiento cada una de las opciones del menú:

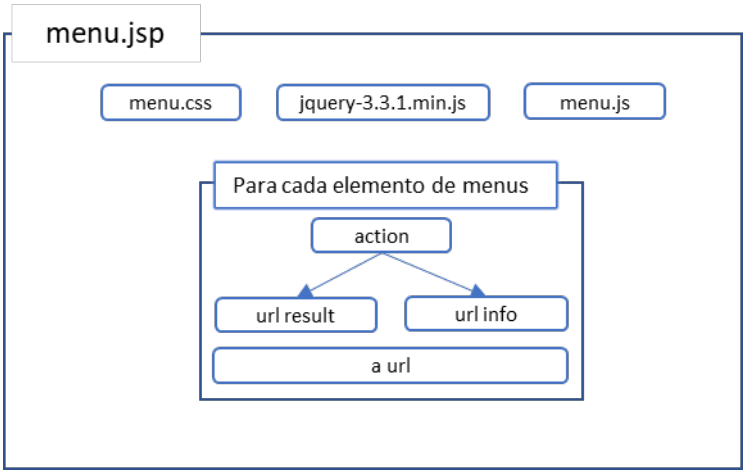

Figura 9: menu.jsp

Ejecutar un método del metadata de la base de datos

Existen dos métodos que permiten ejecutar cualquier método del metadata de la base de datos desde el menú del usuario. Su comportamiento es básicamente el mismo y difieren en la fase final, en la presentación de la información al usuario. El método resultset() de DatabaseMetaDataAction, al final, debe realizar una conversión de los datos desde un objeto de clase ResultSet a un objeto de clase List<List>. Además, el método debe tener en cuenta que, para invocar los métodos de este tipo, deben proporcionarse los valores de los argumentos del tipo adecuado. El método info() de la misma clase difiere en que el resultado ya está en forma List<List> y no es necesario proporcionar valores en la invocación de los métodos.

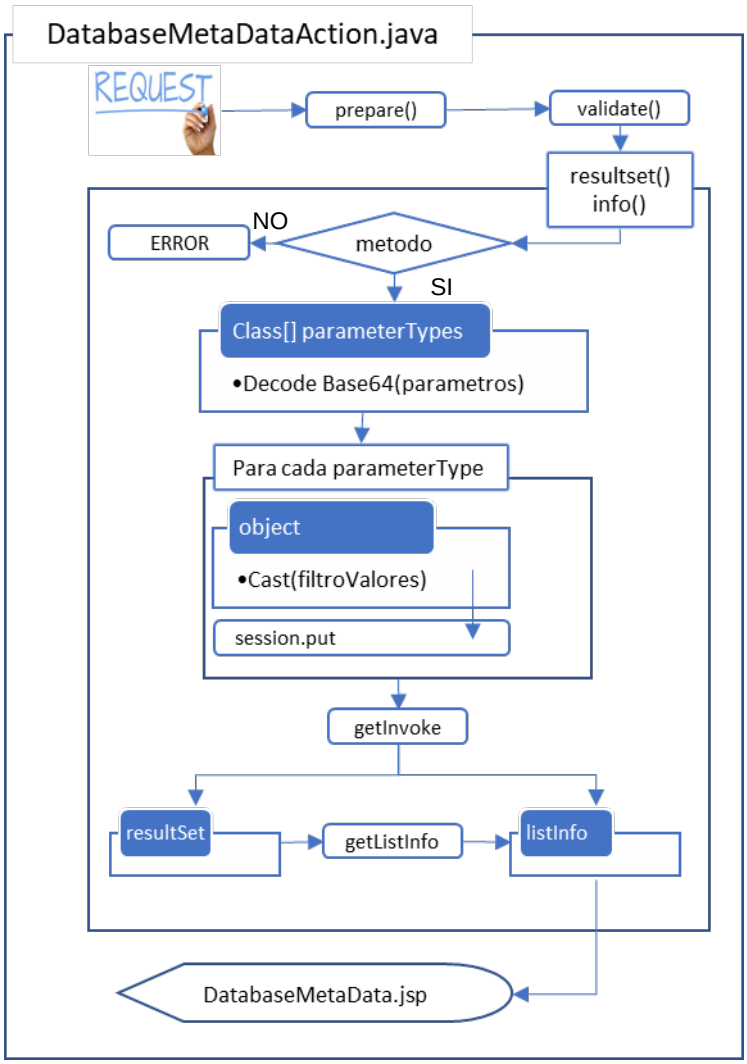

Figura 10: Ejecutar un método del metadata de una base de datos

Es interesante seguirle la pista al método getParameterTypes que recibe los parámetros de la request y los convierte en un array de clases.

También es interesante el método setParametrosSesion que convierte los parámetros recibidos de la request, los convierte al tipo adecuado y los guarda en la sesión del usuario para su utilización posterior.

A continuación se ejecuta el método getInvoke() que obtiene, bien un resultset, bien un List<List> en función del método requerido.

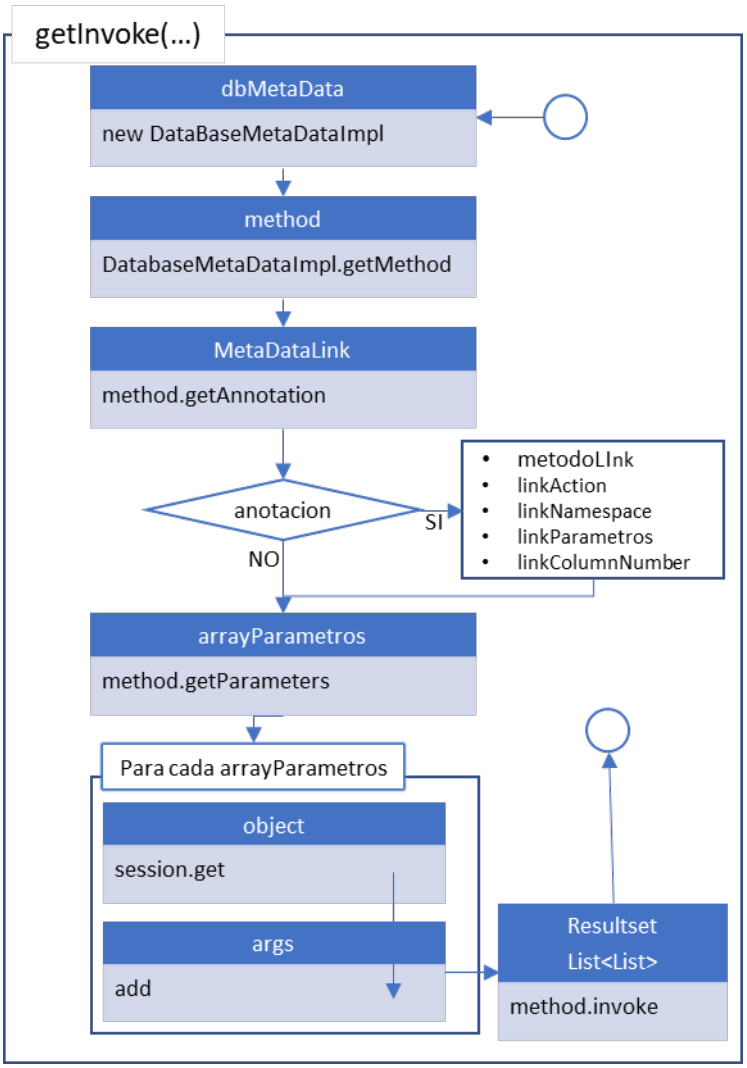

Figura 11: getInoke()

Finalmente, considerar el método getListInfo() de la clase MenuAction que, a partir de un ResultSet, devuelve una lista de listas (List<List>) para su posterior presentación al usuario.

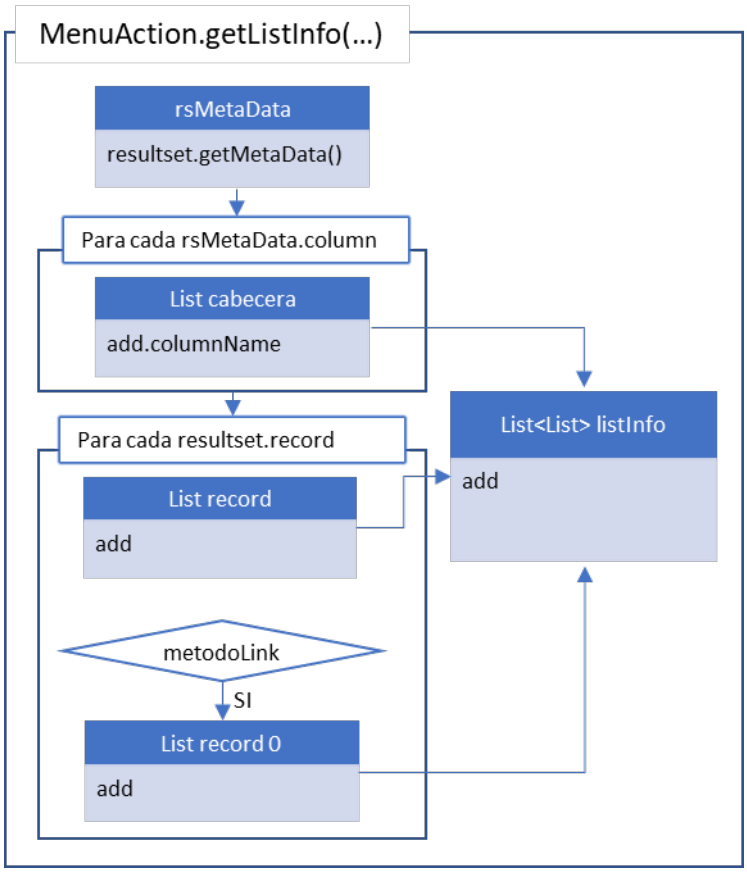

Figura 12: getListInfo()

Como paso final, solo queda presentar la información al usuario mediante un doble bucle que recorre la estructura List<List> incluyendo, si fuera necesario, elementos de link obtenidos a partir de la utilización de la anotación MetaDataLink en los métodos así requeridos de los metadata de la base de datos (en el trabajo solo se ha utilizado en el método getTables()). Tener en cuenta el tratamiento diferenciado de la primera fila como fila de títulos de la tabla html.

También es interesante indicar como se construye el formulario html que permitirá al usuario enviar al sistema los valores de los argumentos de invocación de los métodos. El código se encuentra en filtro.jsp y, utilizando un iterador, permite adaptarse a cualquier número de argumentos, variable en función del método a invocar. Los campos se organizan en dos columnas.

El objeto arrayParametros referenciado es establecido en el método getInvoke.

Trabajo con tablas

Consultar datos de una tabla

Esta sería la primera funcionalidad que se debería implementar, para presentar al usuario los registros filtrados de la tabla deseada. Los comandos de consulta en bases de datos relacionales utilizan el lenguaje SQL, y el comando particular de consulta es la sentencia SELECT. Esta, y todas las demás desarrolladas en este trabajo, están agrupadas en una clase diseñada a tal efecto denominada SQLCommand. Cualquier operación comenzará con la instanciación de un objeto de esta clase para lo cual es necesario pasar una conexión JDBC válida.

Después, solo deberemos invocar al método adecuado con los datos pertinentes para realizar la operación contra la base de datos oportuna.

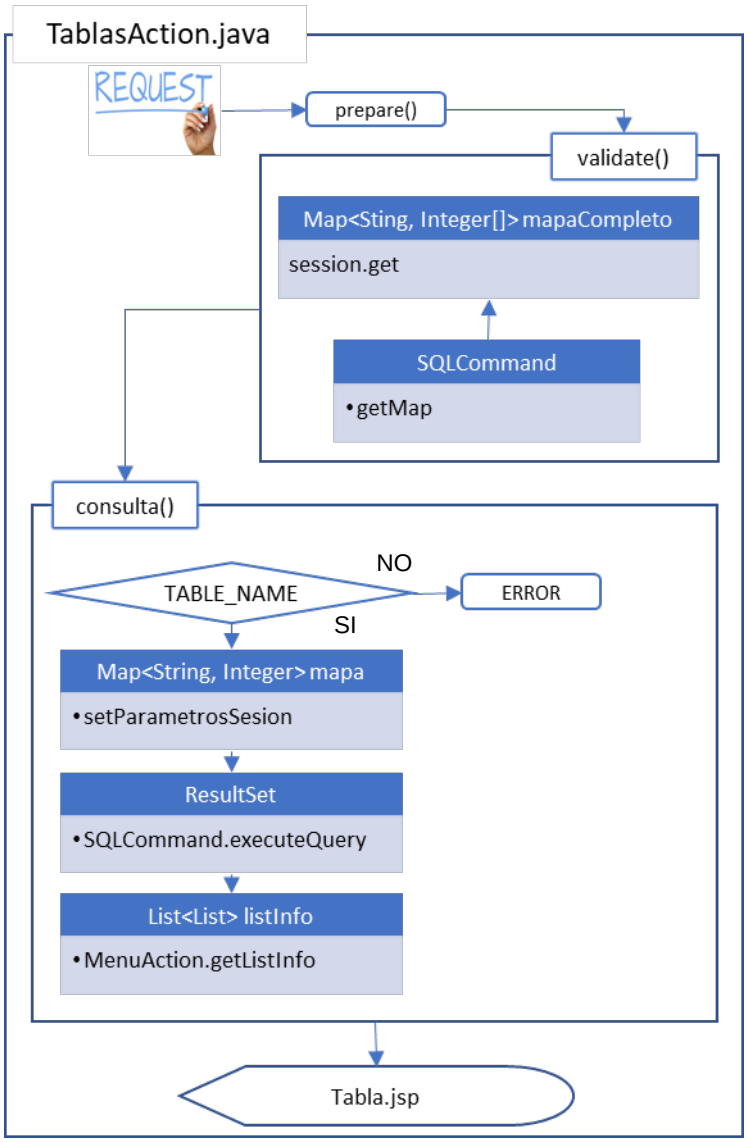

Figura 13: TablasAction. consulta()

Como se puede ver, el mapeado de la request lleva a la aplicación a ejecutar el método consulta() de la clase TablasAction, pero antes, se incluye en la validación el control de los datos necesarios para una correcta llamada posterior. En el método validate() se instancia un mapa con todos los campos de la tabla a consultar y se guarda en la sesión para no tener que realizarlo más veces para esa tabla.

A continuación, el método consulta() insta a la ejecución del comando SQL y al final carga una List<List> utilizando el método ya explicado getListInfo.

En la pantalla tabla.jsp se presenta al usuario el conjunto de datos de una forma similar a la explicada en DatabaseMetaData.jsp. Solo he de comentar que, con el fin de enlazar la consulta con las operaciones de eliminación, edición o inserción, se incluyen a modo de ejemplo.

En la clase SQLCommand, el método executeQuery nos permite obtener el resultset de la tabla.

Que nos lleva al método setSQLPartList para informar las variables oportunas.

Y al método executeQuery con los parámetros adecuados.

Finalmente, mostrar el formulario html que le permite al usuario aportar valores para realizar los filtros adecuados mediante el mismo código ya utilizado para la invocación de métodos del metadata de la base de datos.

Eliminar un registro de una tabla

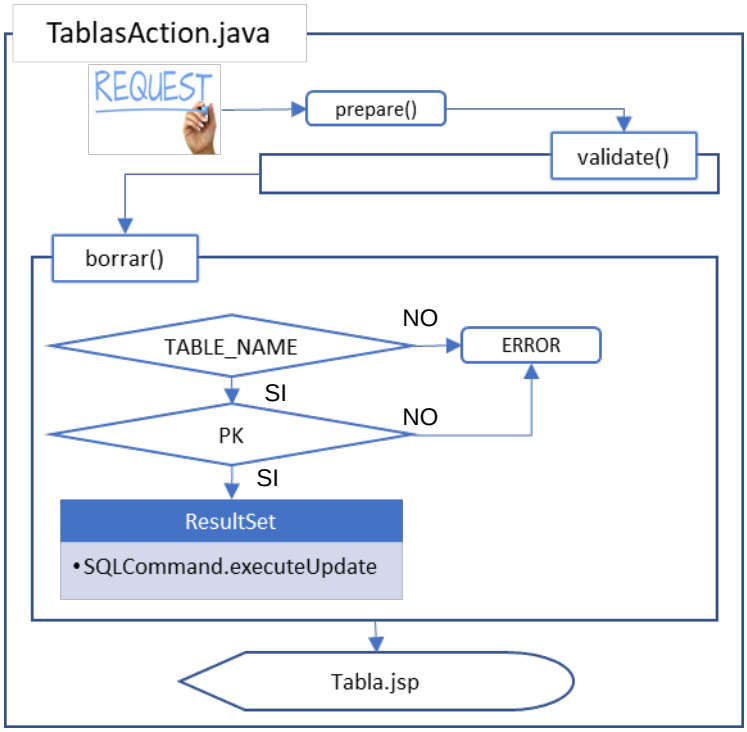

Figura 14: borrar()

Con una llamada a SQLCommand.executeUpdate que se comentará en el punto siguiente.

Editar un registro de una tabla

Con un paso a través de un formulario en el que se presentan los campos del registro, el usuario podrá en dicho formulario modificarlos y retornarlos al sistema para que se lance el correspondiente comando a la base de datos, teniendo en cuanta que necesitamos no solo los datos que han de ser modificados sino también los campos que conforman la primary key de la tabla.

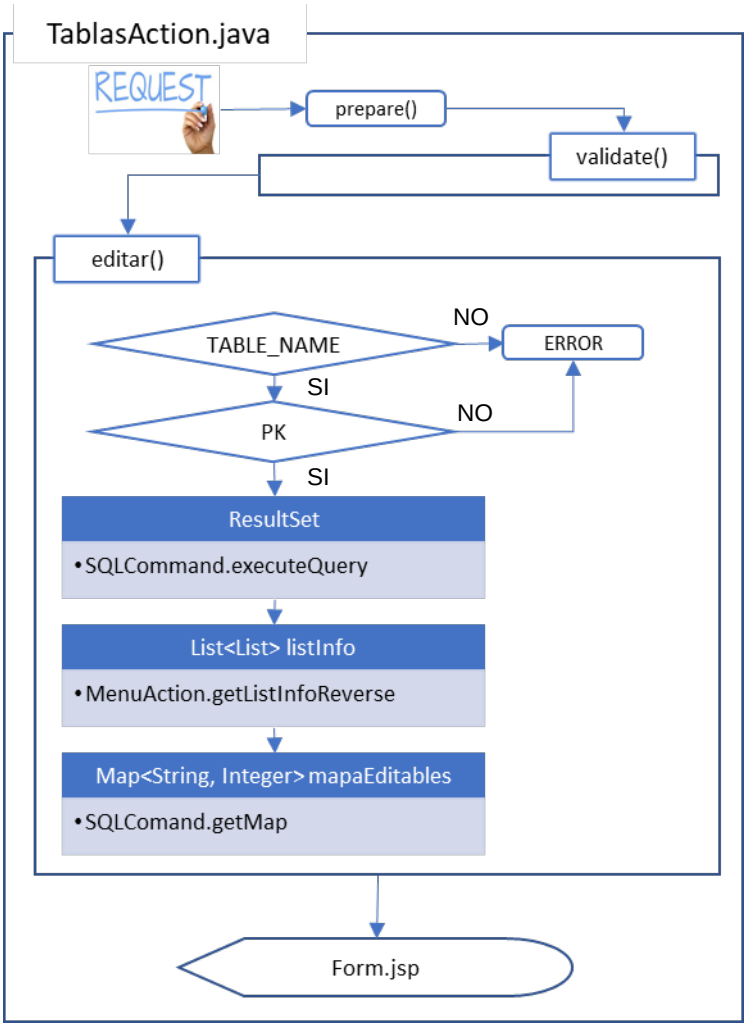

Figura 15: editar()

El proceso siguiente es el encargado de obtener los datos del formulario y guardarlos en la base de datos.

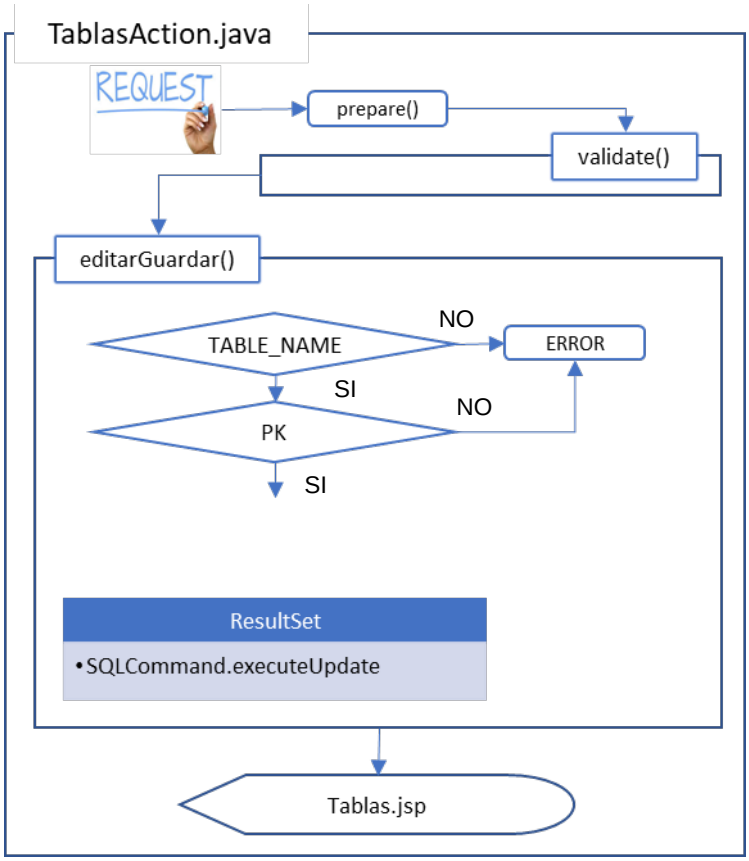

Figura 16: editarGuardar()

Que llama a la clase executeUpdate de SQLCommand.

Y este a su vez hace dos llamadas al setSQLPartList.

Inserción de un registro en una tabla

De forma análoga se realizaría la funcionalidad de inserción de datos en una tabla de la base de datos.

Apéndice D

Documentación técnica de programación

4.1. Introducción

Como ampliación de la sección anterior, desarrollamos el siguiente capítulo indicando la estructura de directorios, los detalles de programación más relevantes atendiendo a los casos de uso explicados, las instrucciones de compilación, instalación y ejecución del proyecto y detallando las pruebas realizadas para comprobar el correcto funcionamiento de la aplicación.

4.2. Estructura de directorios

MyDataBaseJC/src/main

Carpeta raiz de códigos fuente.

java

Carpeta con código fuente de clases java.

resources

Carpeta con properties y ficheros de configuración xml.

webapp

Estructura de carpetas para aplicación web. En ella incluimos los siguientes directorios.

Carpeta	Descripción
.	Se ubican aquí los ficheros jsp y html.
META-INF	Ubicación de context.xml.
WEB-INF	Ubicación de carpeta jspf con páginas jsp que se reutilizan en varias páginas jsp diferentes y web.xml.
resources	Ubicación de ficheros de recursos: imágenes, hojas de estilo y javascript.

Tabla 46: Estructura de carpetas

MyDataBaseJC/src/test

Carpeta raiz de códigos fuente para test.

java

Carpeta con código fuente de clases java JUnit.

MyDataBaseJC/target

Carpeta raiz de soluciones compiladas. En ella nos encontramos una versión comprimida en formato war (myDatabaseJC-1.0-SNAPSHOT.war) junto con la aplicación en formato desplegado que puede ser automáticamente trasladado a un servidor de aplicaciones web bajo la carpeta myDataBaseJC-1.0-SNAPSHOT.

4.3. Manual del programador

A continuación, se describe el diseño de la aplicación en cuanto al flujo del programa atendiendo a cada uno de los casos de uso explicados. En los procesos ya explicados en el apartado 3.4 Diseño arquitectónico, detallamos ahora las cuestiones técnicas relativas a la programación. Este capítulo es, por tanto, una ampliación del indicado anteriormente.

Conexión a la base de datos

Obtener conexión

En login.jsp, el formulario html recoje la información del usuario para establecer la conexión:

```
<s:form action="login" method="POST" namespace="/"
    id="filtro">
    <s:textfield name="url" key="URL"
                 title='%{ayuda("URL")}' class="ayuda"/>
    <s:textfield name="usuario" key="Usuario"/>
    <s:password name="password" key="Contrasena"/>
    <s:submit key="Conectar" name="conectar"
              title='%{ayuda("Conectar")}' class="ayuda"/>
</s:form>
```

Mapeado por struts.xml, la ejecución continúa en el método login() de la clase LoginAction que instancia un objeto de ConnectionImpl y establece la conexión:

```
private String login(String url, String usuario,
                     String password) {
    ConnectionImpl connectionImpl =
       new ConnectionImpl(url, usuario, password);
    HttpServletRequest request = ServletActionContext
       .getRequest();
    request.getSession()
       .setAttribute(CONEXION, connectionImpl);
    return SUCCESS;
}
```

Una vez establecida la conexión se almacena el objeto ConnectionImpl en la sesión del usuario para poderse recuperar entre request distintas.

Utilizar conexión

Siempre que un usuario necesita una conexión con la base de datos se debe seguir el siguiente método para que el sistema se la proporcione.

En primer lugar, el sistema comprueba que la sesión del usuario exista. Para ello se llama al método validarLogin() de la clase LoginAction. Esta llamada se debe realizar en cada petición de conexión, por lo que la clase LoginAction es la clase padre de todas las clases actions de la aplicación y en todas ellas existe una llamada a dicho método a través de el método validate() que debe estar en cada clase action:

```
@Override
public void validate() {
```

```
    super.validarLogin(connectionImpl);
}
```

Y en LoginAction:

```
public String validarLogin() {
    // control de validez de sesión iniciada
    HttpServletRequest request = ServletActionContext
       .getRequest();
    if (request.getSession().isNew()) return ERROR;
    ConnectionImpl connectionImpl =
       (ConnectionImpl)request.getSession()
       .getAttribute(CONEXION);
    return validarLogin(connectionImpl);
}
```

También en la misma clase:

```
protected String validarLogin(
    ConnectionImpl connectionImpl) {
    if (connectionImpl==null) {
         addActionError(getText("Conexión no realizada"));
         return ERROR;
    }

    // si la conexión se ha cerrado, se intenta establecer nuevamente
    if (connectionImpl.isClosed())
         return login(
            connectionImpl.getUrl(),
            connectionImpl.getUsuario(),
            connectionImpl.getPassword());

    // si la conexión no es válida, se intenta establecer nuevamente
    if (!connectionImpl.isValid(0))
         return login(
            connectionImpl.getUrl(),
            connectionImpl.getUsuario(), connectionImpl.getPassword());

         // validación de login correcta
         return SUCCESS;
}
```

Consulta de información de conexión

Información de la conexión

De forma automática, una vez que se inicia la aplicación y después de obtenida la conexión, se presenta al usuario la información general de la

conexión mediante la llamada a una serie de métodos del objeto instanciado de la clase ConnectionInfo.

El foco de ejecución del programa pasa al método inicial() de la clase DatabaseMetaDataAction que instancia un objeto de la clase ConnectionInfo y mediante java Reflection ejecuta los métodos en ella definidos y compone, junto a los nombres de esos métodos, con los resultados obtenidos de su ejecución una lista de objetos de la clase PropiedadValor que posteriormente son presentados al usuario en la página inicial.jsp:

```
  public String inicial() {
    // Modela la clase de información de la conexión
             para su presentación
    ConnectionInfo connectionInfo =
            new ConnectionInfo(connectionImpl
                .getConnection());

    // inicia la lista de valores
    listInicial = new ArrayList<>();

    // obtiene los métodos de la clase
    Method[] methods = ConnectionInfo.class
            .getMethods();

    // para cada método
    for (Method method : methods) {
      // solo de la clase base 8excluye los de clase padre
      if (method.getDeclaringClass().equals(
                          connectionInfo.getClass())) {
        Object o = null;
        try {
          o = method.invoke(
                                connectionInfo,
                                new Object[]{});
        } catch (Exception e) {
          o = e.getLocalizedMessage();
        }

        // crea el metodo/valor. La propiedad es el nombre del método
        // y el valor es el resultado de la ejecución del método
        PropiedadValor p = new PropiedadValor(
                          method.getName(), o);

        // y lo añade a la lista
        listInicial.add(p);
      }
    }
    return SUCCESS;
  }
```

En inicial.jsp, se recorre la lista de información presentándola al usuario:

```
<s:iterator value="listInicial">
    <tr>
```

```
        <td title=
           '<s:property value="%{ayuda(propiedad)}"/>'>
           <s:property value="%{getText(propiedad)}"/>
        </td>
        <td><s:property value="valor"/></td>
    </tr>
</s:iterator>
```

Métodos de los metadata de la base de datos

Obtener métodos del Metadata de la base de datos

Constituyen lo que viene a ser el menú de trabajo de la aplicación. Al usuario se le presentan los distintos métodos que nos interesan del objeto DatabaseMetaData de la conexión para lo que implementamos nuestra propia clase DatabaseMetaDataImpl que utilizaremos para obtener estos métodos. El sistema obtiene la lista de métodos con la información necesaria de ellos para, en una fase posterior, poder realizar una invocación de estos métodos y obtener su resultado y presentarlo al usuario.

La lista de menús se obtiene del método getMenus() de la clase MenuAction, que hereda directamente de LoginAction y de la que heredan todas las acciones en las que sea necesario manejar el menú, como por ejemplo DatabaseMetaDataAction:

```
  public List<Menu> getMenus() { //throws IOException {
    // inicia la lista de valores del menu
    menus = new ArrayList<>();

    // A continuación obtiene los métodos de la
    MetaData que devuelven
    // objetos de tipo ResultSet que serán añadidos
           como opciones de menú
    Method[] methods = DatabaseMetaDataImpl.class
           .getMethods();

    int index;
    // para cada método
    for (Method method : methods) {
      // solo si el método retorna un tipo ResultSet
      if (method.getReturnType()==ResultSet.class
                  || method.getReturnType()==List.class) {
          // crea el menu.
          Menu menu = new Menu(
                          method.getReturnType()==
                                 ResultSet.class ? "resultset" :
                                 "info",
           method.getName(),
```

```
           method.getParameterTypes());
          // lo busca en la lista
          if ((index =
                                menus.indexOf(menu))==-1) {
            // y lo añade el nombre del
                                método a la lista si no existe aún
            menus.add(menu);
          }
      }
    }
    Collections.sort(menus);
    return menus;
  }
```

En el fichero menu.jsp se procede a la representación de las opciones del menú definiendo los links para visualizar los resultados de seguir cada una de las opciones del menú:

```
<s:iterator value="menus">
    <li class="ayuda">
          <s:url action="%{action}" var="urlTag"
                namespace="/DatabaseMetaData">
            <s:param name="metodo">
                <s:property value="metodo"/>
            </s:param>
            <s:param name="parametros">
                <s:property value="parametros"/>
            </s:param>
          </s:url>
          <s:a href="%{urlTag}" title='%{ayuda(metodo)}'>
            <s:property value="%{getText(metodo)}"/>
          </s:a>
    </li>
</s:iterator>
```

Ejecutar un método del metadata de la base de datos

Existen dos métodos que permiten ejecutar cualquier método del metadata de la base de datos desde el menú del usuario. Su comportamiento es básicamente el mismo y difieren en la fase final, en la presentación de la información al usuario. El método resultset() de DatabaseMetaDataAction, al final, debe realizar una conversión de los datos desde un objeto de clase ResultSet a un objeto de clase List<List>. Además, el método debe tener en cuenta que, para invocar los métodos de este tipo, deben proporcionarse los valores de los argumentos del tipo adecuado. El método info() de la misma clase difiere en que el resultado ya está en forma List<List> y no es necesario proporcionar valores en la invocación de los métodos.

```
  public String resultset() {
    // Recoge los mensajes de error previos y los
                establece en esta acción
```

```
    if (sesion.get(ACTION_ERROR)!=null) {
      addActionError(getText((String)sesion
                .get(ACTION_ERROR)));
      sesion.remove(ACTION_ERROR);
    }

    // Si metodo no contiene ningún valor, error
    if (metodo==null || metodo.length()==0) {
      addActionError(
              getText("Llamada.a.la.accion.resultset.incorrecta"));
      return ERROR;
    }
    ResultSet rs = null;
    try {
      Class[] parameterTypes =
                    getParameterTypes(parametros);
      setParametrosSesion(parameterTypes);
      rs = (ResultSet)getInvoke(connectionImpl
                    .getConnection().getMetaData(),
                    metodo, parameterTypes);
      listInfo = getListInfo(rs, metodoLink,
                    linkParametros);
    } catch (ResultSetException | SQLException |
                NoSuchMethodException | SecurityException |
                IllegalAccessException |
                IllegalArgumentException |
                InvocationTargetException | IOException |
                ClassNotFoundException |
                DatabaseMetaDataException ex) {
      try {if (rs!=null) rs.close();}
      catch (SQLException ex1) {;}
      addActionError(ex.getMessage());
      return ERROR;
    }
    return SUCCESS;
  }
```

Es interesante seguirle la pista al método getParameterTypes que recibe los parámetros de la request y los convierte en un array de clases:

```
  private Class[] getParameterTypes(String parametros)
          throws IOException, ClassNotFoundException {
    // se decodifica y obtiene el array de bytes que
                forman los parámetros del métod
    ByteArrayInputStream bais =
            new ByteArrayInputStream(Base64
                .decodeBase64(parametros));
    ObjectInputStream ois =
            new ObjectInputStream(bais);
    // se obtiene el array de parámetros a partir de
                los datos obtenidos
    return (Class[])ois.readObject();
  }
```

También es interesante el método setParametrosSesion que convierte los parámetros recibidos de la request, los convierte al tipo adecuado y los guarda en la sesión del usuario para su utilización posterior:

```
private void setParametrosSesion(
       Class[] parameterTypes) {
  if (filtroArgumentos == null ||
             filtroValores == null) return;
  for (int i = 0; i < parameterTypes.length; i++) {
    String atributo = filtroArgumentos[i];
    Object valor = null;
    if (!"".equals(filtroValores[i]))
      try {
        switch (parameterTypes[i].getName()) {
          case "[Ljava.lang.String;":
            String[] valores = filtroValores[i]
                        .split(",",0);
            List<String> lvalores = new ArrayList();
            for (int j = 0; j< valores.length; j++)
              if (!"".equals(valores[j]))
                              lvalores.add(valores[j].trim());
            valor = lvalores.toArray(valores);
            break;
          case "boolean": valor = Boolean
                        .valueOf(filtroValores[i]); break;
          case "int": valor = Integer
                        .valueOf(filtroValores[i]); break;
          case "short": valor = Short
                        .valueOf(filtroValores[i]); break;
          default: valor = filtroValores[i];
        }
      } catch (Exception e) {
        addActionError(
         getText(
           "Error conviertiendo valor {0} en campo {1}",
           new String[]{filtroValores[i],
           filtroArgumentos[i]}));
      }
    sesion.put(atributo, valor);
  }
}
```

A continuación, se ejecuta el método getInvoke() que obtiene, bien un resultset, bien un List<List> en función del método requerido.

```
private Object getInvoke(DatabaseMetaData databaseMetaData,
       String metodo, Class[] parameterTypes)
       throws IllegalAccessException, IllegalArgumentException,
          NoSuchMethodException, InvocationTargetException,
          DatabaseMetaDataException, IOException,
          ClassNotFoundException {
  // Obtiene el objeto DatabaseMetaDataImpl
  DatabaseMetaDataImpl dbMetadata =
       new DatabaseMetaDataImpl(databaseMetaData);
  // Se invoca el método solicitado, con los parámetros calculados
  Method method = DatabaseMetaDataImpl.class.getMethod(
          metodo, parameterTypes);
```

```
    // obtiene la anotación del tipo MetaDataLink para determinar
            si el método
    // debe conllevar un link asociado a cada registro del resultset
    MetaDataLink anotacion = method
            .getAnnotation(MetaDataLink.class);
    metodoLink = false;
    // si existe la anotación, se definen la acción, el namespace
            y los parámetros
    if (anotacion != null) {
      metodoLink = true;
      linkAction = anotacion.action();
      linkNamespace = anotacion.namespace();
      linkParametros = anotacion.parametros();
      linkColumnNumber = anotacion.columnNumber();
    }

    // Obtiene los nombres de los parametros
    arrayParametros = method.getParameters();

    // crea una lista de objetos que serán finalmente los
            parametros a pasar al metodo invocado
    List<Object> listaParametros = new ArrayList<>();

    // busca cada parametro en la sesión del usuario
    for (Parameter parametro : arrayParametros) {
      // y lo asigna a la lista de objetos a pasar
      Object o = sesion.get(parametro.getName());
      if (o!=null) if ("".equals(o.toString())) o = null;
      listaParametros.add(o);
    }
    // Obtiene la lista de objetos en forma de array. Serán los
            argumentos de la función invocada
    Object[] args = listaParametros.toArray();

    // e invoca al método pasandole los argumentos necesarios
    Object o = method.invoke(dbMetadata, args);
    if (o instanceof ResultSet) return (ResultSet)o;
    if (o instanceof List) return (List<List>)o;
    throw new DatabaseMetaDataException(
            getText(
                "El.objeto.obtenido.no.es.del.tipo.adecuado"), metodo);
  }
```

Finalmente, considerar el método getListInfo() de la clase MenuAction que, a partir de un ResultSet, devuelve una lista de listas (List<List>) para su posterior presentación al usuario.

```
  protected List<List> getListInfo(ResultSet resultSet,
         boolean metodoLink, String[] linkParametros,
         String argumentos, String valores)
         throws SQLException, ResultSetException {
    List<List> lista = new ArrayList();
```

```
    // se obtiene el metadata del result set
    ResultSetMetaData rsMetadata = resultSet.getMetaData();

    // y se almacenan los nombres de las columnas como primer
           Elemento de la lista
    int i = 0;
    List cabecera = new ArrayList();

    // si es un método que debe llevar link, se añade un objeto
           antes de los datos
    // en cada registro. La primera columna tendrá los parámetros
           del link
    if (metodoLink) cabecera.add(null);
    for (i = 1; i <= rsMetadata.getColumnCount(); i++) {
      cabecera.add(rsMetadata.getColumnName(i));
    }
    lista.add(cabecera);

    // para cada registro del resultset se añade una lista con sus
           Valores a la lista del resultset
    int n = 1;
    while (resultSet.next()) {

      // si se supera el máximo de registros, se manda finaliza
      if (n++>getNumMaxRecords())
        break;
      List record = new ArrayList();
      String urlParametro = "";
      if (metodoLink) record.add(urlParametro);
      for (int j = 1; j < i; j++) {
        if (metodoLink) {
          for (String linkParametro : linkParametros)
            if (linkParametro.equals(
                        resultSet.getMetaData().getColumnName(j))) {
              if (argumentos==null) {
                if (resultSet.getObject(j)!=null)
                  urlParametro = urlParametro + "&"
                                    + linkParametro + "=" +
                                    resultSet.getObject(j);
              } else {
                if (resultSet.getObject(j)!=null) {
                  urlParametro = urlParametro + "&"
                                    + argumentos + "=" +
                                    linkParametro;
                  urlParametro = urlParametro + "&"
                                    + valores + "=" +
                                    resultSet.getObject(j);
                }
              }
            }
        }
        record.add(resultSet.getObject(j));
      }
      if (metodoLink) record.set(0,urlParametro.substring(1));
      lista.add(record);
    }
    return lista;
  }
```

Como paso final, solo queda presentar la información al usuario mediante un doble bucle que recorre la estructura List<List> incluyendo, si fuera necesario, elementos de link obtenidos a partir de la utilización de la anotación MetaDataLink en los métodos así requeridos de los metadata de la base de datos (en el trabajo solo se ha utilizado en el método getTables()). Tener en cuenta el tratamiento diferenciado de la primera fila como fila de títulos de la tabla html.

```
<s:iterator value="listInfo" var="record" status="status">
    <s:if test="#status.first == true"><thead></s:if>
         <tr>
           <s:iterator value="#record" status="stat">
               <s:set name="propiedad"><s:property /></s:set>
               <s:if test="#status.first == true">
                   <s:if test="!metodoLink && #stat.count != 1">
                         <th><s:property /></th>
                   </s:if>
                   <s:elseif test="#stat.count != 1">
                         <th><s:property /></th>
                   </s:elseif>
                   <s:elseif test='#propiedad != null'>
                         <th class="ayuda"><s:property/></th>
                   </s:elseif>
               </s:if>
               <s:else>
                   <s:if test="metodoLink">
                         <s:if test="#stat.count == 1">
                               <s:url action="%{linkAction}"
                                     namespace="%{linkNamespace}"
                                     var="urlTag"/>
                               <s:set var="url">
                                     <s:property value="urlTag"/>
                                           ?
                                     <s:property/>
                               </s:set>
                         </s:if>
                         <s:elseif test="#stat.count == 2">
                               <td title='
                                     <s:property
                                     value="%{ayuda(propiedad)}"/>'>
                                     <s:if test="#stat.count-1 ==
                                           linkColumnNumber">
                                           <s:a href="%{url}"
                                                 id="ir" >
                                           <s:text
                                           name="%{propiedad}"/>
                                           </s:a>
                                     </s:if>
                                     <s:else>
                                           <s:text
                                           name="%{propiedad}"/>
                                     </s:else>
                               </td>
                         </s:elseif>
```

```
                        <s:else>
                              <td>
                                    <s:if test="#stat.count-1 ==
                                          linkColumnNumber">
                                          <s:a href="%{url}"
                                          id="ir"><s:property/>
                                          </s:a>
                                    </s:if>
                                    <s:else>
                                          <s:property/>
                                    </s:else>
                              </td>
                        </s:else>
                  </s:if>
                  <s:elseif test="#stat.count == 1">
                        <td title='<s:property
                              value="%{ayuda(#propiedad)}"/>'>
                              <s:text name="%{propiedad}"/>
                        </td>
                  </s:elseif>
                  <s:else>
                        <td>
                              <s:property/>
                        </td>
                  </s:else>
               </s:else>
            </s:iterator>
         </tr>
    <s:if test="#status.first == true"></thead></s:if>
</s:iterator>
```

También es interesante indicar como se construye el formulario html que permitirá al usuario enviar al sistema los valores de los argumentos de invocación de los métodos. El código se encuentra en filtro.jsp y, utilizando un iterador, permite adaptarse a cualquier número de argumentos, variable en función del método a invocar. Los campos se organizan en dos columnas:

```
<s:form name="filtro" method="POST" theme="simple" id="filtro">
    <table class="filtro">
         <s:iterator value="arrayParametros" status="stat">
            <s:if test="#stat.odd == true">
                <tr id="odd">
            </s:if>
            <s:set var="filtro">
                <s:property escapeHtml="false"/>
            </s:set>
            <td>
                <s:text name="%{filtro}" />:
                <s:hidden name="filtroArgumentos" value="%{filtro}"/>
            </td>
            <td>
                <s:textfield name="filtroValores"
                    value="%{getParameter(#stat.count-1)}"/>
            </td>
            <s:if test="#stat.even == true">
                </tr>
            </s:if>
```

```
        </s:iterator>
    </table>
    <s:hidden name="metodo"/>
    <s:hidden name="parametros"/>
    <s:hidden name="TABLE_CAT"/>
    <s:hidden name="TABLE_SCHEM"/>
    <s:hidden name="TABLE_NAME"/>
    <s:submit name="filtrar" value="filtrar"/>
</s:form>
```

El objeto arrayParametros referenciado es establecido en el método getInvoke.

Trabajo con tablas

Consultar datos de una tabla

Esta sería la primera funcionalidad que se debería implementar, para presentar al usuario los registros filtrados de la tabla deseada. Los comandos de consulta en bases de datos relacionales utilizan el lenguaje SQL, y el comando particular de consulta es la sentencia SELECT. Esta, y todas las demás desarrolladas en este trabajo, están agrupadas en una clase diseñada a tal efecto denominada SQLCommand. Cualquier operación comenzará con la instanciación de un objeto de esta clase para lo cual es necesario pasar una conexión JDBC válida.

Después, solo deberemos invocar al método adecuado con los datos pertinentes para realizar la operación contra la base de datos oportuna.

Como se puede ver, el mapeado de la request lleva a la aplicación a ejecutar el método consulta() de la clase TablasAction, pero antes, se incluye en la validación el control de los datos necesarios para una correcta llamada posterior. En el método validate() se instancia un mapa con todos los campos de la tabla a consultar y se guarda en la sesión para no tener que realizarlo más veces para esa tabla:

```
  public void validate() {
    super.validarLogin(connectionImpl);
    // si no hay errores, se carga el mapa de la tabla consultada
    if (!hasActionErrors()) {
      // busca el mapa del resultset en la sesión del usuario
      mapaCompleto = (Map<String, Integer[]>)sesion.get(
          Funciones.getCatalog(TABLE_CAT,TABLE_SCHEM).equals("") ?
            TABLE_NAME :
```

```
                Funciones.getCatalog(TABLE_CAT,TABLE_SCHEM) + "." +
TABLE_NAME));
      // si el mapa aún no se ha creado
      if (mapaCompleto == null) {
        // se calcula y se guarda en la sesión
        SQLCommand sqlCommand =
                    new SQLCommand(connectionImpl);
        mapaCompleto = sqlCommand

    .getMap(Funciones.getCatalog(TABLE_CAT,TABLE_SCHEM), TABLE_NAME);
        sesion.put(
          Funciones.getCatalog(TABLE_CAT,TABLE_SCHEM).equals("") ?
TABLE_NAME :
                Funciones.getCatalog(TABLE_CAT,TABLE_SCHEM) + "." +
TABLE_NAME),
          mapaCompleto);
      }
    }

  }
```

A continuación, el método consulta() insta a la ejecución del comando SQL y al final carga una List<List> utilizando el método ya explicado getListInfo:

```
  public String consulta() {
    // Recoge los mensajes de error previos y los establece en
           esta acción
    if (sesion.get(ACTION_ERROR)!=null) {
      addActionError(getText((String)sesion.get(ACTION_ERROR)));
      sesion.remove(ACTION_ERROR);
    }

    // Recoge los mensajes informativos previos y los establece en
           esta acción
    if (sesion.get(ACTION_MESSAGE)!=null) {
      addActionMessage(
                getText((String)sesion.get(ACTION_MESSAGE)));
      sesion.remove(ACTION_MESSAGE);
    }

    // comprueba que los parámetros se reciban correctamente
    if (TABLE_NAME == null || "".equals(TABLE_NAME)) {
      sesion.put(ACTION_ERROR,
                "Faltan.esquema.o.nombre.de.tabla");
      return "back";
    }
    metodoLink = false; // Será true cuando exista una primary key
           en la tabla consultada

    // establece y recoge los parámetros de búsqueda en sesión del
           usuario
    Map<String, Integer> mapa = setParametrosSesion(
        Funciones.getCatalog(TABLE_CAT,TABLE_SCHEM).equals("") ?
TABLE_NAME :

(Funciones.getCatalog(TABLE_CAT,TABLE_SCHEM).equals("null") ?
TABLE_NAME :
                Funciones.getCatalog(TABLE_CAT,TABLE_SCHEM) + "." +
TABLE_NAME)));
```

```
    // define el array de posibles parámetros para presentarlos en
            la jsp
    arrayParametros = mapa.keySet();

    //PreparedStatement ps = null;
    ResultSet rs = null;
    ResultSet rs2 = null;
    try {
      SQLCommand sqlCommand = new SQLCommand(connectionImpl);
      rs =
sqlCommand.executeQuery(Funciones.getCatalog(TABLE_CAT,TABLE_SCHEM),
                TABLE_NAME,
                arrayParametros, sesion);
      // se obtiene una lista con los campos primary key
      List<String> pkList = new ArrayList<>();
      String[] linkParametros = {};
      rs2 = connectionImpl.getConnection().getMetaData()
                .getPrimaryKeys(null,
                Funciones.getCatalog(TABLE_CAT,TABLE_SCHEM),
                TABLE_NAME);
      while (rs2.next())
        pkList.add(rs2.getString("COLUMN_NAME"));
      if (pkList.size()!=0) metodoLink = true;
      // se transforma el resultset en una lista para su visualización
      listInfo = getListInfo(rs, metodoLink,
                pkList.toArray(linkParametros), "pkArgumentos",
                "pkValores");
    } catch (ResultSetException | SQLException |
                SQLCommandException ex) {
      addActionError(ex.getMessage());
      return ERROR;
    } finally {
      cierraRS(rs, null);
      cierraRS(rs2, null);
    }
    return SUCCESS;
  }
```

En la pantalla tabla.jsp se presenta al usuario el conjunto de datos de una forma similar a la explicada en DatabaseMetaData.jsp. Solo hay que comentar que, con el fin de enlazar la consulta con las operaciones de eliminación, edición o inserción, se incluyen a modo de ejemplo:

```
<ul class="bqY ocultar">
    <s:url action="borrar" var="urlBru" includeParams="get"
      escapeAmp="false"/>
    <s:set var="url">
         <s:property value="urlBru"/>&<s:property/>
    </s:set>
    <li class="bqX" title="<s:property
              value='%{ayuda("borrar.registro")}'/>">
         <s:a href="%{url}" class="acciones bru">  </s:a>
  </li>
```

En la clase SQLCommand, el método executeQuery nos permite obtener el resultset de la tabla:

```
public ResultSet executeQuery(String TABLE_SCHEM,
       String TABLE_NAME, Set<String> arrayParametros,
       Map<String, Object> sesion) throws SQLCommandException {
  setSQLPartList(TABLE_SCHEM, TABLE_NAME, arrayParametros,
          sesion);
  return executeQuery(TABLE_SCHEM, TABLE_NAME,
          ResultSet.TYPE_FORWARD_ONLY, ResultSet.CONCUR_READ_ONLY);
}
```

Que nos lleva al método setSQLPartList para informar las variables oportunas:

```
public void setSQLPartList(String TABLE_SCHEM, String TABLE_NAME,
       Set<String> arrayParametros, Map<String, Object> sesion) {
  inicializa();
  // para cada columna del resultset
  for (String columna : arrayParametros) {
    // se busca en la sesión del usuario
    Object o = sesion.get(getEsquema(TABLE_SCHEM) + TABLE_NAME
             + "." + columna);
    // si el parámetro existe y no es "", se añade la
             condición a la cadena where
    // y el valor a la lista de parametros
    if (o != null && !"".equals(o.toString())) {
      // si no es el primer elemento se antepone AND
      if (listaParametrosWhere.size() != 0) {
        cadenaWhere = cadenaWhere + "AND ";
      } // si es el primer filtro, se antepone where
      else {
        cadenaWhere = "where ";
      }
      // y a continuación se añade el nombre del campo y la
                posición del parámetro.
      // esto evitará ataques por SQL inyection
      cadenaWhere = cadenaWhere + columna + " = ? ";
      listaParametrosWhere.add(o);
    }
  }
}
```

Y al método executeQuery con los parámetros adecuados:

```
private ResultSet executeQuery(String TABLE_SCHEM, String
         TABLE_NAME, int TYPE_SCROLL, int CONCUR) throws
         SQLCommandException {
  // Se obtiene la cadena de consulta definitiva
  String sql = String.format("SELECT * FROM %s%s %s",
         getEsquema(TABLE_SCHEM), TABLE_NAME, cadenaWhere);
  PreparedStatement ps = null;
  try {
    // se obtiene el PreparedStatemtne
    ps = connectionImpl.getConnection().prepareStatement(sql,
             TYPE_SCROLL, CONCUR);
    // se asignan los parámetros
```

```
      int parameterIndex = 1;
      for (Object o : listaParametrosWhere) {
        ps.setObject(parameterIndex++, o);
      }
      ResultSet rs = null;
      // y se resuelve el resultset
      return ps.executeQuery();
    } catch (SQLException ex) {
      throw new SQLCommandException(this, ex);
    }
  }
```

Finalmente, mostrar el formulario html que le permite al usuario aportar valores para realizar los filtros adecuados mediante el mismo código ya utilizado para la invocación de métodos del metadata de la base de datos.

Eliminar un registro de una tabla

```
  public String borrar(){
    if (TABLE_NAME == null || "".equals(TABLE_NAME)) {
      sesion.put(ACTION_ERROR,
                 "Faltan.esquema.o.nombre.de.tabla");
      return "tablas";
    }
    if (pkArgumentos == null || pkArgumentos.length == 0) {
      sesion.put(ACTION_ERROR, "Falta.primary.key");
      return "consulta";
    }

    try {
      SQLCommand sqlComand = new SQLCommand(connectionImpl);
      if
(sqlComand.executeUpdate(Funciones.getCatalog(TABLE_CAT,TABLE_SCHEM),
                 TABLE_NAME,
                 mapaCompleto, SQLCommand.OPERACION_DELETE,
                 pkArgumentos, pkValores, null, null)==0) {
        sesion.put(ACTION_ERROR,
                   "No.se.ha.eliminado.ningún.registro");
      } else {
        sesion.put(ACTION_MESSAGE,
                   "El.registro.ha.sido.eliminado");
      }
    } catch (SQLCommandException ex) {
      sesion.put(ACTION_ERROR, ex.getLocalizedMessage());
    } finally {
      return "consulta";
    }
  }
```

Con una llamada a SQLCommand.executeUpdate que se comentará en el punto siguiente.

Editar un registro de una tabla

Con un paso de valores a través de un formulario en el que se presentan los campos del registro, el usuario podrá en dicho formulario modificarlos y retornarlos al sistema para que se lance el correspondiente comando a la base de datos, teniendo en cuanta que necesitamos no solo los datos que han de ser modificados sino también los campos que conforman la primary key de la tabla.

```
public String editar(){
  if (TABLE_NAME == null || "".equals(TABLE_NAME)) {
    sesion.put(ACTION_ERROR,
              "Faltan.esquema.o.nombre.de.tabla");
    return "tablas";
  }
  if (pkArgumentos == null || pkArgumentos.length == 0) {
    sesion.put(ACTION_ERROR, "Falta.primary.key");
    return "consulta";
  }

  // obtiene el resultset del registro a editar
  ResultSet rs = null;
  try {
    SQLCommand sqlComand = new SQLCommand(connectionImpl);
    // y se resuelve el resultset
    rs = sqlComand.executeQuery(
              Funciones.getCatalog(TABLE_CAT,TABLE_SCHEM),
              TABLE_NAME,
              mapaCompleto, pkArgumentos, pkValores,
              ResultSet.TYPE_SCROLL_INSENSITIVE,
              ResultSet.CONCUR_READ_ONLY);
    // se transforma el resultset en una lista para su visualización
    listInfo = getListInfoReverse(rs);
    // se consigue el mapa de campos editables.
    mapaEditables = sqlComand.getMap(mapaCompleto,
              SQLCommand.ISDEFINITELYWRITABLE +
              SQLCommand.ISWRITABLE);

  } catch (SQLCommandException ex) {
    addActionError(ex.getMessage());
    return ERROR;
  } finally {
    cierraRS(rs,null);
  }
  return SUCCESS;
}
```

El proceso siguiente es el encargado de obtener los datos del formulario y guardarlos en la base de datos.

```
public String editarGuardar() {
  if (TABLE_NAME == null || "".equals(TABLE_NAME)) {
    sesion.put(ACTION_ERROR,
              "Faltan.esquema.o.nombre.de.tabla");
    return "tablas";
  }
```

```
  if (pkArgumentos == null || pkArgumentos.length == 0) {
    sesion.put(ACTION_ERROR, "Falta.primary.key");
    return "consulta";
  }
  if (formCampos == null || formCampos.length == 0 ||
         formValores == null || formValores.length == 0) {
    sesion.put(ACTION_ERROR,
              "Faltan.datos.para.hacer.la.operacion");
    return "consulta";
  }

  try {
    SQLCommand sqlComand = new SQLCommand(connectionImpl);
    if (sqlComand.executeUpdate(TABLE_SCHEM
              Funciones.getCatalog(TABLE_CAT,TABLE_SCHEM),
              TABLE_NAME,
              mapaCompleto, SQLCommand.OPERACION_UPDATE,
              pkArgumentos, pkValores, formCampos, formValores)==0)
              {
      sesion.put(ACTION_MESSAGE,
                   "No.se.ha.actualizado.ningún.registro");
    } else {
      sesion.put(ACTION_MESSAGE,
                   "El.registro.ha.sido.actualizado");
    }
  } catch (SQLCommandException ex) {
    sesion.put("ACTION_ERROR", ex.getLocalizedMessage());
  } finally {
    return "consulta";
  }
}
```

Que llama a la clase executeUpdate de SQLCommand:

```
public int executeUpdate(String esquema, String tabla,
       Map<String, Integer[]> mapa, int operacion,
       String[] pkArgumentos, String[] pkValores, String[] campos,
       String[] valores) throws SQLCommandException {
  int retorno = 0;
  if (operacion == OPERACION_UPDATE) {
    setSQLPartList(mapa, pkArgumentos, pkValores, campos,
              valores);
    sql = String.format("UPDATE %s%s SET %s %s",
        getEsquema(esquema),
        tabla,
        cadenaSet,
        cadenaWhere);
  } else if (operacion == OPERACION_DELETE) {
    setSQLPartList(mapa, pkArgumentos, pkValores,
              OPERACION_WHERE);
    sql = String.format("DELETE FROM %s%s %s",
        getEsquema(esquema),
        tabla,
        cadenaWhere);
  }
```

```
    if ("".equals(sql)) {
      throw new SQLCommandException(this, operacion);
    }
    PreparedStatement ps = null;
    try {
      ps = connectionImpl.getConnection().prepareStatement(sql);
      // se asignan los parámetros
      int parameterIndex = 1;
      // en primer lugar la parte update, no habrá parámetros si
                no hay parte update
      for (Object o : listaParametrosSet) {
        ps.setObject(parameterIndex++, o);
      }
      // y luego la parte where
      for (Object o : listaParametrosWhere) {
        ps.setObject(parameterIndex++, o);
      }
      retorno = ps.executeUpdate();
    } catch (SQLException ex) {
      throw new SQLCommandException(this, ex);
    } finally {
      try {
        if (ps != null) {
          ps.close();
        }
      } catch (SQLException ex) {;
      }
    }
    return retorno;
  }
}
```

Y este a su vez hace dos llamadas al setSQLPartList:

```
  private void setSQLPartList(Map<String, Integer[]> mapa,
         String[] pkArgumentos, String[] pkValores, String[] campos,
         String[] valores) throws SQLCommandException {
    inicializa();
    int n = 0;
    for (String columna : campos) {
      try {
       Object o = getValor(mapa.get(columna)[0], valores[n++]);
       if (listaParametrosSet.size() != 0) {
         cadenaSet = cadenaSet + ", ";
       }
       cadenaSet = cadenaSet + columna + " = ? ";
       listaParametrosSet.add(o);
      } catch (Exception e) {
       ConversionException ce =
         new ConversionException(pkArgumentos[n], e);
       throw new SQLCommandException(ce);
      }
    }
    n = 0;
    for (String columna : pkArgumentos) {
      try {
       Object o = getValor(mapa.get(columna)[0],
         pkValores[n++]);
```

```
       if (listaParametrosWhere.size() != 0) {
         cadenaWhere = cadenaWhere + "AND ";
       } else {
         cadenaWhere = "where ";
       }
       cadenaWhere = cadenaWhere + columna + " = ? ";
       listaParametrosWhere.add(o);
      } catch (Exception e) {
       ConversionException ce =
         new ConversionException(pkArgumentos[n], e);
       throw new SQLCommandException(ce);
      }
    }
  }
```

Inserción de un registro en una tabla

De forma análoga se realizaría la funcionalidad de inserción de datos en una tabla de la base de datos.

Tratamiento de errores de datos

Antes de ejecutar un comando SQL en el que se incluyen datos asociados a campos de una tabla, el sistema realiza una conversión de los datos que, normalmente, son recibidos desde el usuario mediante la utilización de formularios web y, por tanto, son recibidos en formato texto, realiza una conversión al tipo adecuado en función del tipo de campo al que debe ir asociado, ya sea en un apartado where, en un apartado set o en un apartado insert.

Es posible que el usuario, de forma errónea, introduzca un dato que no pueda ser convertido al tipo de dato requerido en cuyo caso se debería informar al usuario de tal situación.

La función de conversión está centralizada en el método estático SQLCommand.getValor que recibe como parámetros el tipo de dato en forma de int y el valor a convertir en forma de String. El método devuelve el valor correspondiente en forma de Object que es la clase padre que puede englobar a todos los tipos que pueden ser devueltos.

A continuación vemos un estracto del método:

```
public static Object getValor(int tipo, String valor) {
  valor = valor==null
   ? null
```

```
 : ("".equals(valor) ? null : valor);
switch (tipo) {
  case Types.ARRAY:
    String[] valores = valor.split(",", 0);
    List<String> lvalores = new ArrayList();
    for (int k = 0; k < valores.length; k++) {
      if (!"".equals(valores[k])) {
        lvalores.add(valores[k].trim());
      }
    }
    return lvalores.toArray(valores);
  // para tipos Integer
  case Types.BIGINT:
  case Types.INTEGER:
  case Types.ROWID:
    return valor == null
      ? (Integer) null
      : Integer.valueOf(valor);
```

En primer lugar asigna el valor null si la cadena es una cadena vacía. A continuación, en función del tipo, se realiza la conversión al tipo java adecuado mediante el uso de la función correspondiente. Por ejemplo, en el caso de tipo Integer, la función es valueOf(valor).

Finalmente devuelve el valor obtenido en la conversión.

Si la función de conversión produce una excepción, dicha excepción será retomada por el método que llama a este método. Existen dos métodos, fundamentalmente, desde los que se llaman al método getValor:

- TablasAction.setParametrosSesion: Este método es el encargado de concretar el filtro en una consulta a una base de datos. En un bucle for se recorren todos los argumentos recibidos y se convierten al tipo adecuado mediante llamadas al método getValor. El bloque de código correspondiente se encuentra en un bloque try … catch que maneja la excepción de la siguiente manera:

  ```
  addActionError(filtroArgumentos[i] + ": " + ex.toString());
  ```

  Con lo que se le informa al usuario de que en el campo indicado se ha producido el correspondiente error.
- SQLComand.setSQLPartList: Este método es el encargado de formar los distintos bloques de un comando SQL, por lo tanto, es en él en el que se produce la conversión de los datos de tipo String en datos del tipo java adecuado. Igual que en el método anterior, se engloba la llamada al método getValor en un bloque try … catch como el siguiente:

```
} catch (Exception e) {
  ConversionException ce =
   new ConversionException(pkArgumentos[n], e);
  throw new SQLCommandException(ce);
}
```

La diferencia con el caso anterior es que, ahora, se instancia un objeto de la clase ConversionException que nos permitirá realizar un seguimiento en el log de la aplicación y este objeto es el que se transmite al método llamante mediante un throw sobre un nueva SQLCommandException. Al final, la excepción llega al útlimo método, por ejemplo, el método que intenta guardar los cambios en un registro, que se encargará de mostrar el mensaje al usuario:

```
try {
  SQLCommand sqlComand = new SQLCommand(connectionImpl);
  if (sqlComand.executeUpdate(
     Funciones.getCatalog(TABLE_CAT,TABLE_SCHEM),
     TABLE_NAME,
    mapaCompleto, SQLCommand.OPERACION_UPDATE,
    pkArgumentos, pkValores, formCampos, formValores)
    == 0) {
   ...
  }
} catch (SQLCommandException ex) {
  addActionError(ex.getLocalizedMessage());
  return editar();
}
```

En el bloque catch se recoge y muestra la excepción y se vuelve a editar el registro que mostrará de nuevo los datos del mismo registro.

4.4. Compilación, instalación y ejecución del proyecto

[2]

Conceptos básicos de ciclo de vida de construcción

Maven está basado en el concepto central del ciclo de vida de construcción de software lo que indica que el proceso para constuir y distribuir un proyecto está predefinido, es decir, que solo es necesario aprender un pequeño conjunto

de comandos para construir cualquier proyecto de Maven y el fichero POM se asegurará de que se obtengan los resultados deseados.

Existen tres ciclos de vida de compilación incorporados: default, clean y site. El ciclo default maneja la implementación del proyecto, clean maneja la limpieza y site se encarga de la creación de la documentación del sitio del proyecto.

Un ciclo de vida de construcción se compone de fases.

Cada uno de estos ciclos está definido por una lista diferente de fases de compilación donde una fase representa una etapa en el ciclo de vida. Por ejemplo, el ciclo **default** se compone de las fases:

- **validate**: valida que el proyecto es correcto.
- **compile**: Compila el código fuente del proyecto.
- **test**: prueba el código fuente compilado en un marco de prueba adecuado.
- **package**: toma el código compilado y lo guarda en su formato distribuible.
- **verify**: Verifica los resultados de las pruebas de integración.
- **install**: instala el paquete en el repositorio local para usarlo como una dependencia en otros proyectos a nivel local.
- **deploy**: copia el paquete final al repositorio remoto para compartirlo con otros desarrolladores.

  Estas fases se ejecutan de forma secuencia para completar el ciclo **default**.

Llamadas de línea de comandos habituales

En el entorno de desarrollo el siguiente comando crea e instala proyectos en el repositorio local:

```
mvn install
```

Este comando ejecutará cada fase predeterminada del ciclo de vida en orden (validate, compile, package, etc.) antes de ejecutar install.

El siguiente comando crea e implementa proyectos de forma limpia en el repositorio compartido:

```
mvn clean deploy
```

Una fase se compone de objetivos de complemento (plugin goals)

La manera en que se lleva a cabo las responsabilidades de un paso específico en el ciclo de vida de la compilación puede variar. Esto se hace declarando los objetivos del complemento vinculados a esas fases.

Un objetivo de complemento representa una tarea específica, más fina que una fase, que contribuye a la construcción y gestión de un proyecto. Puede estar vinculado a cero o más fases de construcción. El orden de ejecución depende del orden en que se invocan los objetivos y las fases de creación. Por ejemplo, en el siguiente comando:

```
mvn clean dependency:copy-dependencies package
```

clean y **package** son fases, mientras **dependency:copy-dependencies** es un objetivo de complemento.

Si se ejecuta esto, la fase **clean** se ejecutará primero y luego el objetivo **dependency:copy-dependencies**, antes de finalmente ejecutar la fase **package**.

Algunas fases no se suelen llamar desde la línea de comandos

Las fases nombradas con palabras con guión (pre-*, post-* o process-*) generalmente no se llaman directamente desde la línea de comando.

Configuración del proyecto

El ciclo de vida de la compilación es bastante sencillo de usar pero ¿cómo asignamos tareas a cada una de las fases de compilación?.

Packaging

Lo primero y más habitual es fijar el empaquetado del proyecto a través del elemento POM <**packaging**>. Algunos valores válidos para este término son **jar, war, ear** y **pom**. En nuestro caso, utilizaremos el valor **war**.

Cada paquete contiene una lista de objetivos para vincular a una fase particular.

Plugins

La segunda forma de agregar objetivos a las fases es configurar los complementos en su proyecto. Los plugins son proyectos que proporcionan objetivos a Maven. Además, un plugin puede tener uno o más objetivos en los que cada objetivo representa una capacidad de ese plugin. Por ejemplo, el plugin de compilar contiene dos objetivos: compile y testCompile. El primero compila el código fuente y el segundo compila el código fuente de su código de prueba.

En nuestro proyecto es necesario indicar un plugin de compilación de maven que fuerce a realizar una compilación en versión java 1.8 y con un argumento de compilación específico: -parameters. Este argumento específico hace que en la fase de compilación, los argumentos de los métodos mantengan su nombre para que java Reflections pueda utilizarlos. Si no se hiciera, los argumentos tendrían los nombres genéricos arg0, arg1, etc.

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-compiler-plugin</artifactId>
  <version>3.8.0</version>
  <configuration>
    <source>1.8</source>
    <target>1.8</target>
    <compilerArguments>
      <endorseddirs>${endorsed.dir}</endorseddirs>
    </compilerArguments>
    <compilerArgs>
      <arg>-parameters</arg>
    </compilerArgs>
  </configuration>
</plugin>
```

Referencia del ciclo de vida

A continuación se enumeran todas las fases de compilación de los ciclos de vida default, clean y site:

Ciclo de vida clean

Fase	Descripción
pre-clean	Ejecutar los procesos necesarios antes de la limpieza real del proyecto.
clean	Eliminar todos los archivos generados por la compilación anterior.
post-clean	Ejecutar los procesos necesarios para finalizar el proyecto de limpieza.

Tabla 47: ciclo de vida clean

Ciclo de vida default

Fase	Descripción
validate	Validar el proyecto si es correcto y toda la información necesaria está disponible.
initialize	Inicializar el estado de compilación, por ejemplo, establecer propiedades o crear directorios.
generate-sources	Genera cualquier código fuente para incluirlo en la compilación.
process-sources	Procesa el código fuente, por ejemplo para filtrar cualquier valor.
generate-resources	Genera recursos para su inclusión en el paquete.
process-resources	Copia y procesa los recursos en el directorio de destino, listo para empaquetar.
compile	Compila el código fuente del proyecto.
process-classes	Postprocesa los archivos generados a partir de la compilación, por ejemplo, para hacer una mejora del código de bytes en las clases de Java.
generate-test-sources	Genera cualquier código fuente de prueba para su inclusión en la compilación.
process-test-sources	Procesa el código fuente de prueba, por ejemplo, para filtrar cualquier valor.
generate-test-resources	Crea recursos para la prueba.
process-test-resources	Copia y procesa los recursos en el directorio de destino de prueba.
test-compile	Compila el código fuente de prueba en el directorio de destino de prueba.
process-test-classes	Postprocesa los archivos generados a partir de la compilación de prueba, por ejemplo, para hacer una mejora del código de bytes en las clases de Java.

Fase	Descripción
test	Ejecuta pruebas utilizando un marco de prueba de unidad adecuada. Estas pruebas no deben requerir que el código esté empaquetado o implementado.
prepare-package	Realiza las operaciones necesarias para preparar un paquete antes del embalaje real. Esto a menudo resulta en una versión desempaquetada y procesada del paquete.
package	Toma el código compilado y lo empaqueta en su formato distribuible, como un JAR.
pre-integration-test	Realiza las acciones necesarias antes de que se ejecuten las pruebas de integración. Esto puede implicar cosas como configurar el entorno requerido.
integration-test	Procesa y despliega el paquete, si es necesario, en un entorno donde se puedan ejecutar pruebas de integración.
post-integration-test	Realiza las acciones requeridas después de que las pruebas de integración hayan sido ejecutadas. Esto puede incluir la limpieza del medio ambiente.
verify	Ejecuta cualquier verificación para verificar que el paquete es válido y cumple con los criterios de calidad.
install	Instala el paquete en el repositorio local, para usarlo como una dependencia en otros proyectos a nivel local.
deploy	Hecho en un entorno de integración o lanzamiento, copia el paquete final al repositorio remoto para compartirlo con otros desarrolladores y proyectos.

Tabla 48: ciclo de vida default

Ciclo de vida site

Fase	Descripción
pre-site	Ejecuta los procesos necesarios antes de la generación del sitio del proyecto real.
site	Genera la documentación del sitio del proyecto.
post-site	Ejecuta los procesos necesarios para finalizar la generación del sitio y preparar la implementación del sitio.
site-deploy	Implementa la documentación del sitio generado en el servidor web especificado.

Tabla 49: ciclo de vida site

Encuadernaciones de ciclo de vida incorporadas

Algunas fases tienen objetivos vinculados a ellos por defecto. Y para el ciclo de vida default estos enlaces dependen del valor de empaquetado. Estos son algunos de los enlaces de objetivo a la fase de built-in.

Enlaces de ciclo de vida clean

Fase	plugin:objetivo
clean	clean:clean

Tabla 50: Enlaces de ciclo de vida limpio

Enlaces de ciclo de vida default – packaging war

Fase	plugin:objetivo
process-resources	resources:resources
compile	Compiler:compile
Process-test-resources	Resources:testResources
Test-compile	Compiler:testCompile
Test	Surefire:test
Package	War:war
Install	Install:install
Deploy	Deploy:deploy

Tabla 51: Enlaces de ciclo de vida default - packaging war

Enlaces de ciclo de vida site

Phase	plugin:goal
site	site:site
site-deploy	site:deploy

Tabla 52: Enlaces de ciclo de vida site

Fijaciones de complementos para el packaging war

[3]

```
<phases>
 <process-resources>
  org.apache.maven.plugins:maven-resources-plugin:2.6:resources
 </process-resources>
 <compile>
  org.apache.maven.plugins:maven-compiler-plugin:3.1:compile
 </compile>
 <process-test-resources>
  org.apache.maven.plugins:maven-resources-plugin:2.6:testResources
 </process-test-resources>
 <test-compile>
  org.apache.maven.plugins:maven-compiler-plugin:3.1:testCompile
 </test-compile>
 <test>
  org.apache.maven.plugins:maven-surefire-plugin:2.12.4:test
 </test>
 <package>
  org.apache.maven.plugins:maven-war-plugin:2.2:war
 </package>
 <install>
  org.apache.maven.plugins:maven-install-plugin:2.4:install
 </install>
 <deploy>
  org.apache.maven.plugins:maven-deploy-plugin:2.7:deploy
 </deploy>
</phases>
```

4.5. Pruebas del sistema

JUnit

[6]

JUnit es un conjunto de clases o framework que permite realizar la ejecución de clases Java de manera controlada para poder evaluar si el funcionamiento de cada uno de los métodos de la clase se comporta como se espera. Si la clase

cumple la especificación, JUnit devolverá que el método pasó exitosamente la prueba.

Las pruebas se han centrado en los métodos de las clases Action de struts, que conforman la base del Controlador en el MVC.

A modo de ejemplo, y sin pretender describir todas las posibles pruebas que se pueden realizar, se acompañan 3 clases que muestran como se prueban y testean dos de los métodos iniciales: el login y la obtención de la información inicial de la conexión con la base de datos.

DatosConexion.java

Con el objeto de indicar los datos para establecer una conexión se define esta clase con 3 propiedades de la que heredan todas las clases de test. Las propiedades contienen los datos necesarios para establecer la conexión con una base de datos:

```
package es.ubu.alu.mydatabasejc.jdbc;

/**
 *
 * @author jhuidobro
 */
public class DatosConexion {
  protected String url = "";
  protected String usuario = "";
  protected String password = "";
}
```

LoginActionTest.java

Esta clase probará los métodos definidos en la clase LoginAction, utilizada principalmente para obtener una conexión válida o validarla si ya existe, con la base de datos. Inicialmente hereda de DatosConexion.java y declara un objeto de la clase LoginAction:

```
public class LoginActionTest extends DatosConexion {
  LoginAction action = null;
```

En el método de inicio del test, se instancia el objeto action y se pasan los parámetros para realizar la conexión:

```
@Before
public void setUp() {
  action = new LoginAction();
  action.setUrl(url);
  action.setUsuario(usuario);
  action.setPassword(password);
}
```

El método que testea el login, hace una llamada a action.login() y controla la respuesta:

```
@org.junit.Test
public void testLogin() {
  System.out.println("testLogin");
  if (!action.login().equals("success"))
    fail("El login falló");
}
```

DatabaseMetaDataActionTest.java

En esta clase se prueban los métodos más relevantes de la clase DatabaseMetaDataAction, en concreto probamos el método inicial() que completa un List con las propiedades y valores más importantes de la conexión previamente establecida. Comenzamos con la declaración heredada de DatosConexion y la definición de las propiedades action y List:

```
DatabaseMetaDataAction action;
private List<PropiedadValor> listInicial;
private ConnectionImpl conexion;
```

En la inicialización de la prueba, se instancia action, se establece la conexión con la base de datos y se inicializa la lista:

```
@Before
public void setUp() throws ConnectionException {
  conexion = new ConnectionImpl(
          url, usuario, password);
  action = new DatabaseMetaDataAction();
  action.setConnectionImpl(conexion);
  listInicial = new ArrayList<PropiedadValor>();
}
```

Se crea el test que compueba la ejecución del método inicial() y se presentan en pantalla los datos de la lista obtenida:

```
@org.junit.Test
public void testListInicial() {
  System.out.println("testListInicial");
  if (!action.inicial().equals("success"))
    fail("El testInfo falló");
  listInicial = action.getListInicial();
  System.out.println("listInicial: " +
```

```
            listInicial);
  }
```

Finalmente, se cierra la conexión una vez finalizado el test:

```
@After
public void tearDown() {
  conexion.close();
}
```

Apache JMeter

[4]

La aplicación Apache JMeter es un software de código abierto, 100% java, diseñada para cargar el comportamiento funcional de la prueba y medir el rendimiento.

Se puede utilizar para probar el rendimiento tanto en recursos estáticos como dinámicos de aplicaciones web.

Características:

- La capacidad de carga y el rendimiento prueban muchas aplicaciones, servidores y tipos de protocolo diferentes.
- Permite la grabación rápida del plan de prueba.
- Dispone de un modo de línea de comandos para cargar la prueba desde cualquier sistema operativo compatible Java.
- Informe HTML completo.
- Marco completo de subprocesos múltiples que permite el muestreo simultáneo por muchos subprocesos y el muestreo simultáneo de diferentes funciones por grupo de subprocesos separados.

No es un navegador, funciona a nivel de protocolo. Se parece a un navegador, sin embargo, JMeter no realiza todas las accciones admitidas por los navegadores, no ejecuta el javascript que se encuentra en las páginas HTML. Tampoco representa las páginas HTML como lo hace un navegador.

Se utilizó esta herramienta para realizar un paquete de pruebas y de carga de la aplicación suficiente, siempre teniendo en cuenta que el objetivo del proyecto es puramente educativo.

Se plantean cuatro request a la aplicación, una para establecer la conexión, otra ejecutando getTables, otra ejecutando getTables del esquema JHA y otra para hacer una select de una tabla. El resultado obtenido es:

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
HTTP Request conexion	600	133	93	1936	84.85	0.00%	2.69647	11.27	0.71	4278
HTTP Request getTables	600	90	59	800	48.34	0.00%	2.71935	11.41	0.95	4297
HTTP Request gettables jha	600	48	34	296	22.79	100.00%	2.72124	3.38	1.44	1273
HTTP Request select contrato	600	86	58	406	37.32	0.00%	2.72168	11.42	0.55	4297
TOTAL	2400	89	34	1936	61.37	25.00%	10.77722	37.22	3.61	3536.2

Tabla 53: Resultados pruebas JMeter

Apéndice E

Documentación de usuario

5.1. Introducción

Para que el usuario pueda utilizar la aplicación y comprenda el alcance y las posibilidades de esta, se indica a continuación, en primer lugar, los requisitos de usuario que cumple, el sistema o procedimiento de instalación y un pequeño manual de usuario que le ayudará a utilizar la aplicación.

5.2. Requisitos de usuario

Los requisitos de usuario vienen indicados en los objetivos con los que la aplicación se desarrolló. Brevemente los indicamos nuevamente:

- Permitir el mantenimiento de una base de datos remota vía internet.
- Ofrecer un entorno amigable e intuitivo.
- Permitir efectuar cualquier operación básica de mantenimiento sobre una base de datos relacional.
- Se dispondrá de una ayuda interactiva en línea.

5.3. Instalación

En el paquete de distribución se proporcionan dos posibilidades de instalación:

- Paquete war: Se proporciona un archivo empaquetado en formato war que se puede desplegar en un servidor web que cumpla los requisitos establecidos a tal efecto.

- Carpeta: Además se proporciona una carpeta con la aplicación compilada y la estructura de ficheros ya definida. Si se desea utilizar esta carpeta directamente, es necesario cargar un fichero XML que indique al servidor la ruta y el nombre que se le dará a la aplicación. A modo de ejemplo, podría ser:

```
<?xml version="1.0" encoding="UTF-8"?>
<Context docBase="ruta\target\myDataBaseJC-1.0-SNAPSHOT"
     path="/myDataBaseJC"/>
```

5.4. Manual de usuario

A continuación detallamos a modo de manual de usuario las distintas pantallas, normas, instrucciones y pasos que debe seguir un usuario para la correcta utilización de la aplicación.

Conexión a la base de datos

Al invocar por primera vez a la aplicación, o siempre que no se disponga de una conexión válida a la base de datos, el usuario visualizará la pantalla para introducir los datos de login.

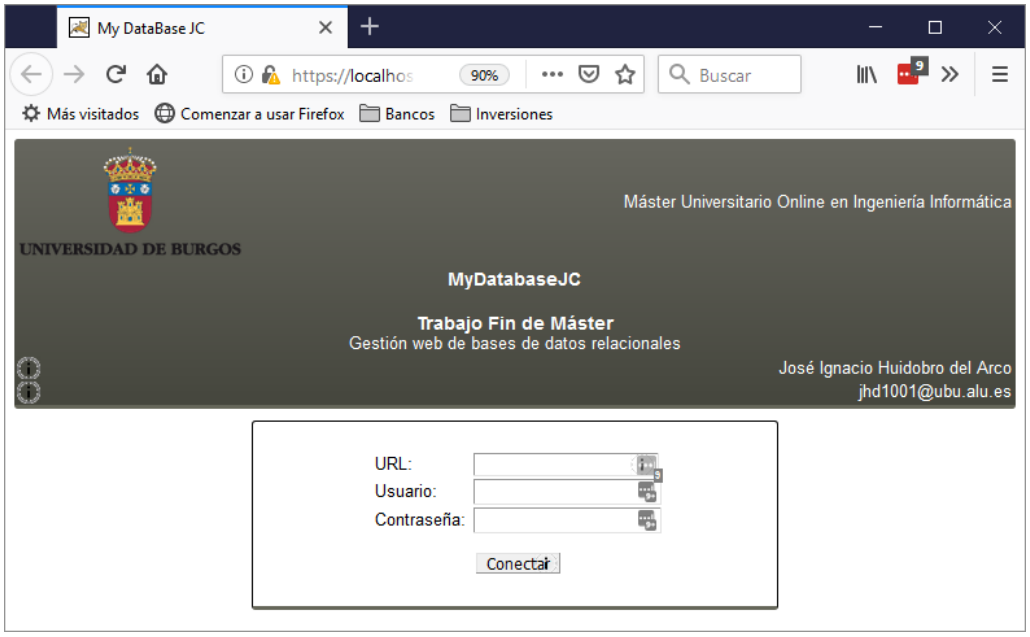

Figura 17: Pantalla de login

El usuario deberá introducir los datos solicitados tras lo que la aplicación intentará establecer la conexión. Si no pudiera establecerse dicha conexión se informará al usuario de la causa en la misma pantalla.

Si el usuario obtiene la conexión a la base de datos se pasa al siguiente apartado.

Información inicial de la base de datos

Obtenida la conexión se le presenta al usuario la siguiente pantalla.

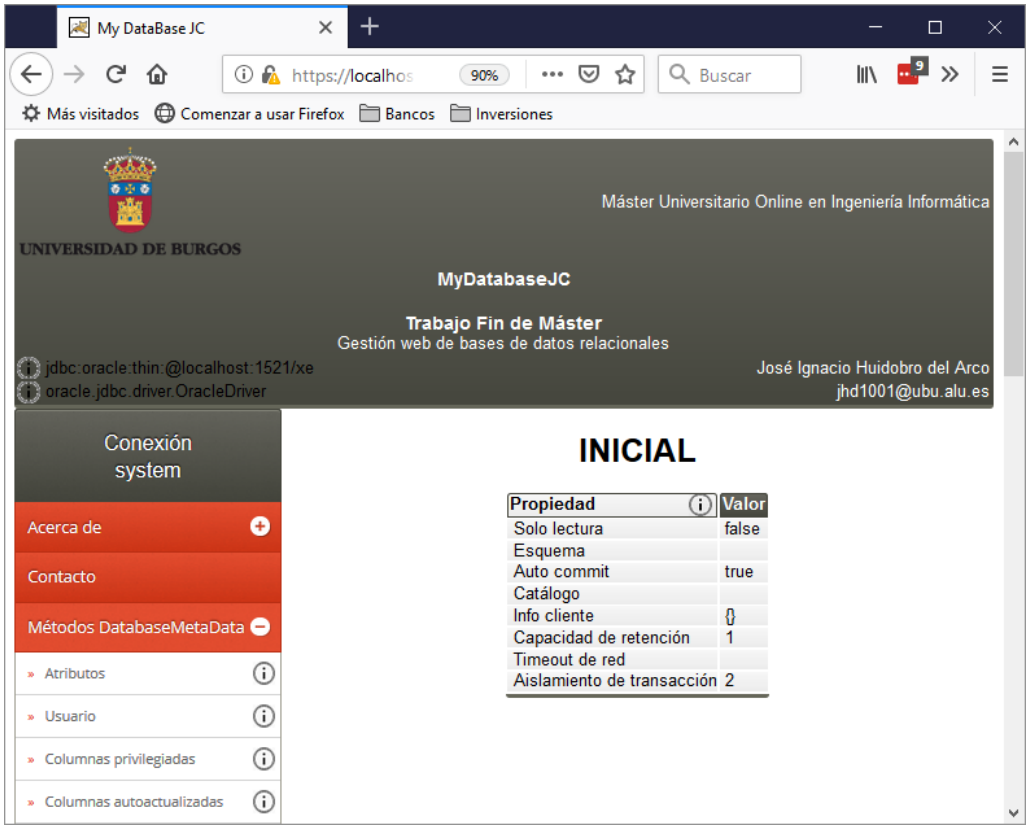

Figura 18: Pantalla inicial

En esta pantalla podemos distinguir tres zonas informativas:

- La banda superior muestra información general como universidad, nombre, aplicación, así como la url de conexión y el driver utilizado.
- La banda lateral es un menú para que el usuario elija la operación a realizar. Además muestra en la parte superior el usuario de conexión.
- La zona central muestra una colección de propiedades de la conexión establecida.

Cuando el usuario elije una determinada opción del menú lateral, en la zona central se presenta la información correspondiente.

El usuario puede colocar el ratón sobre aquellos elementos distinguidos con un símbolo de información (una i latina rodeada por un círculo) tras lo que

aparecerá junto al cursor una explicación de lo referenciado a modo de ayuda interactiva o contextual.

Como caso especial, la opción de menú **Tablas**, muestra al usuario las tablas de la base de datos con unos determinados atributos:

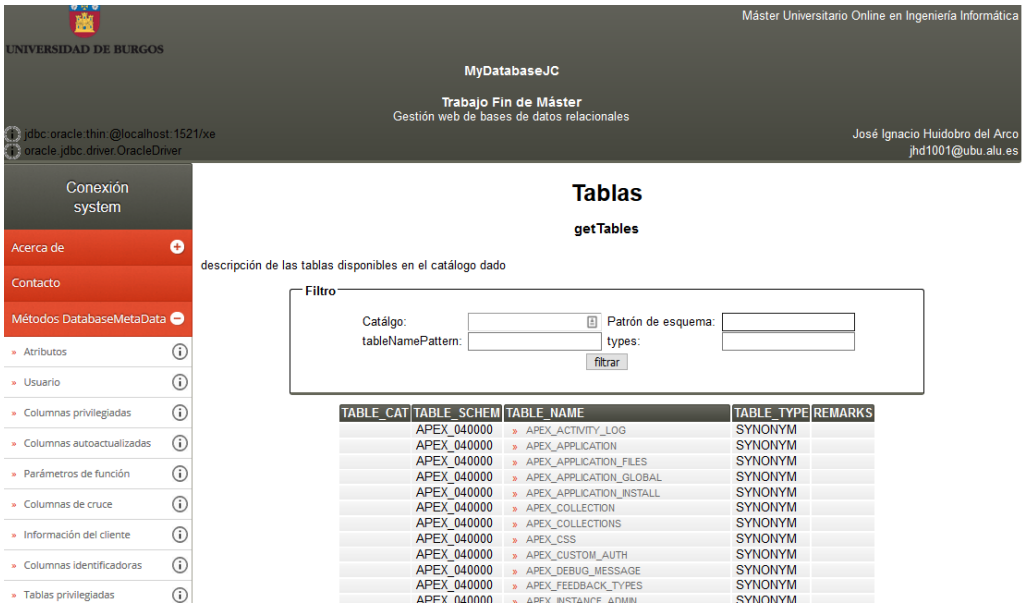

Figura 19: pantalla getTables sin argumentos

La primera vez que se utiliza esta opción, es posible que aparezcan una gran cantidad de registros puesto que el método, sin argumentos, devuelve todas las tablas de la base de datos. Si vamos al final de la tabla veremos que, inicialmente, se nos habrán visualizado 1000 registros.

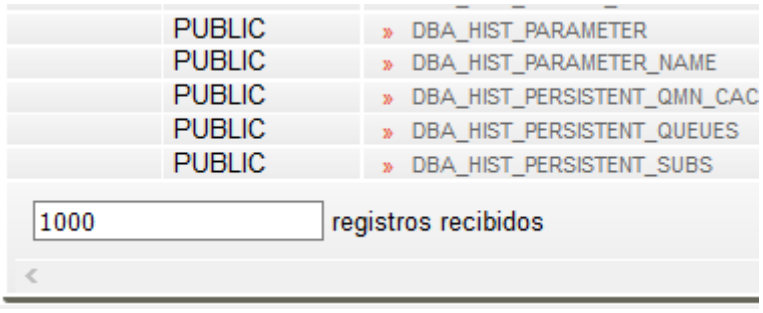

Figura 20: Máximo número de registros recibidos

Cuando se reciben el máximo número de registros posible, de acuerdo con la limitación establecida, que el usuario pueda indicar un número diferente y

descargar, de este modo, más registros (o menos, según se indique). Si el número de registros es inferior al máximo establecido, el usuario no podrá cambiar este número aunque seguirá viendo el número de registros recibidos.

Se puede filtrar el número de tablas, en el caso que nos ocupa, o el número de registros en general de una opción de menú, ingresando los valores deseados en el formulario filtro que aparece. En nuestro ejemplo, si filtramos mediante el patrón de esquema JHA obtenemos la siguiente pantalla:

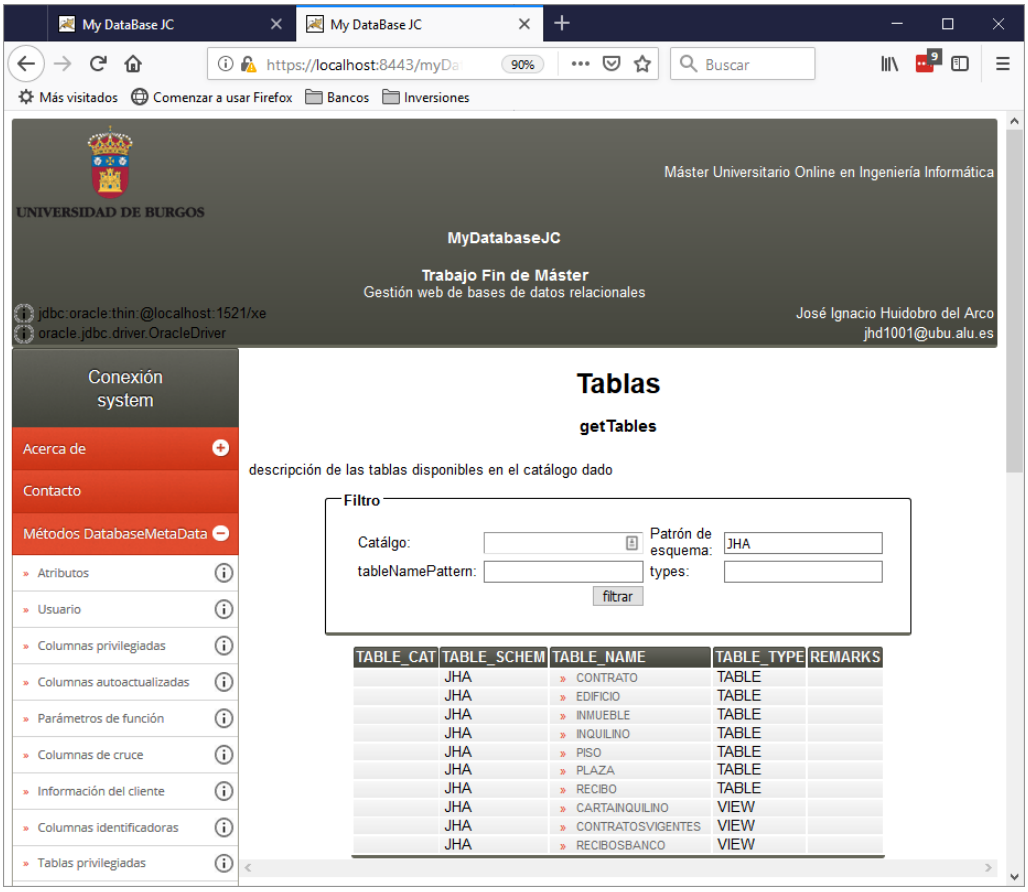

Figura 21: Pantalla getTables de un patrón de esquema concreto

Como se puede observar, aparecen las tablas que cumplen con los criterios establecidos en el filtro.

Este sistema de filtrado, como ya se ha comentado, funciona con cualquiera de las funciones invocadas en el menú lateral.

Solo en el caso de Tablas, junto al TABLE_NAME aparece un link que nos permitirá mostrar el contenido de una tabla concreta.

Mantenimiento de una tabla

De manera general no se hacen comprobaciones previas antes de realizar la operación requerida. Existe una transformación o adaptación de los datos enviados al servidor al tipo adecuado lo cual podrá provocar algún tipo de excepción que será tratada y devuelta al usuario indicándole cuál es el motivo del error.

Así mismo, una vez enviado el comando SQL al gestor de base de datos, este puede producir un error que será tratado de forma similar al anterior por el sistema, o puede ser ejecutado sin errores lo que provocará el retorno del programa a la pantalla de consulta de la tabla indicando que la operación se realizó correctamente.

Si seguimos el link de alguna de las tablas visualizadas aparecerá la siguiente pantalla:

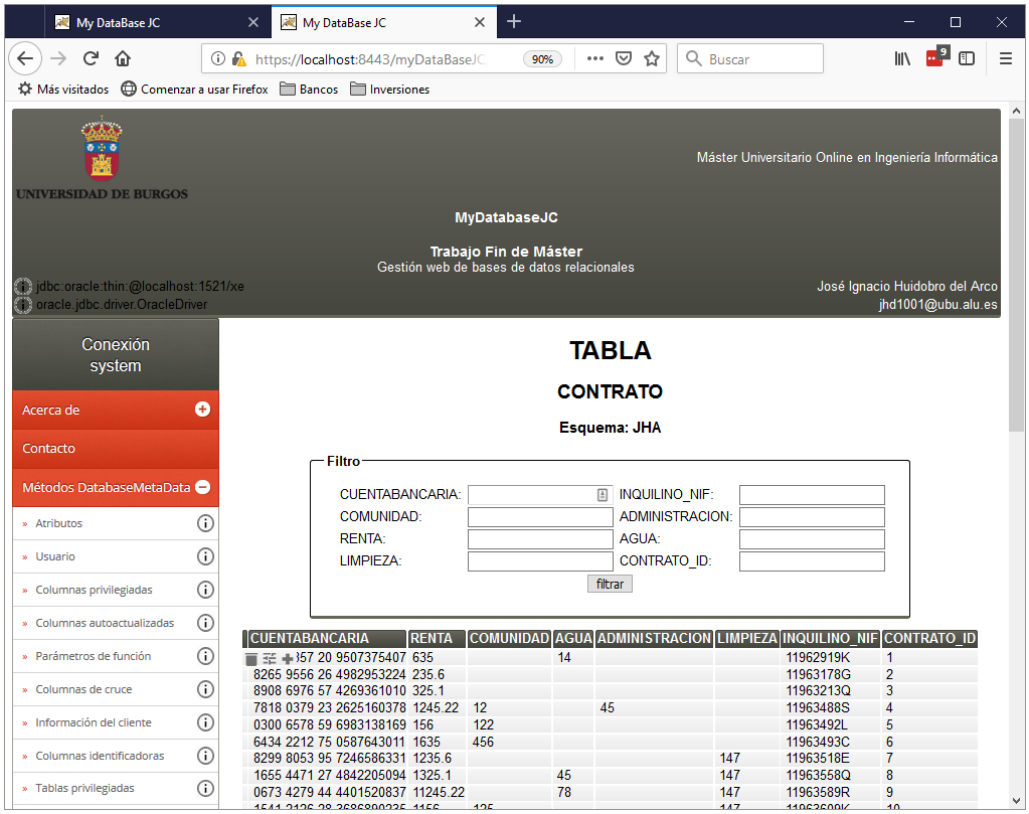

Figura 22: Pantalla de una consulta de tabla

De manera similar a lo indicado anteriormente, la zona central consta de un formulario para realizar filtrado de datos y una zona inferior en la que aparecen los datos de la tabla seleccionada.

En cada registro de la tabla, cuando recibe el foco del ratón, se activan 3 iconos[1] que le permiten al usuario eliminar, editar o insertar un nuevo registro.

Modificar un registro

Al hacer clic en el icono de edición, aparece la siguiente pantalla:

[1] Sólo si la tabla dispone de una primary key

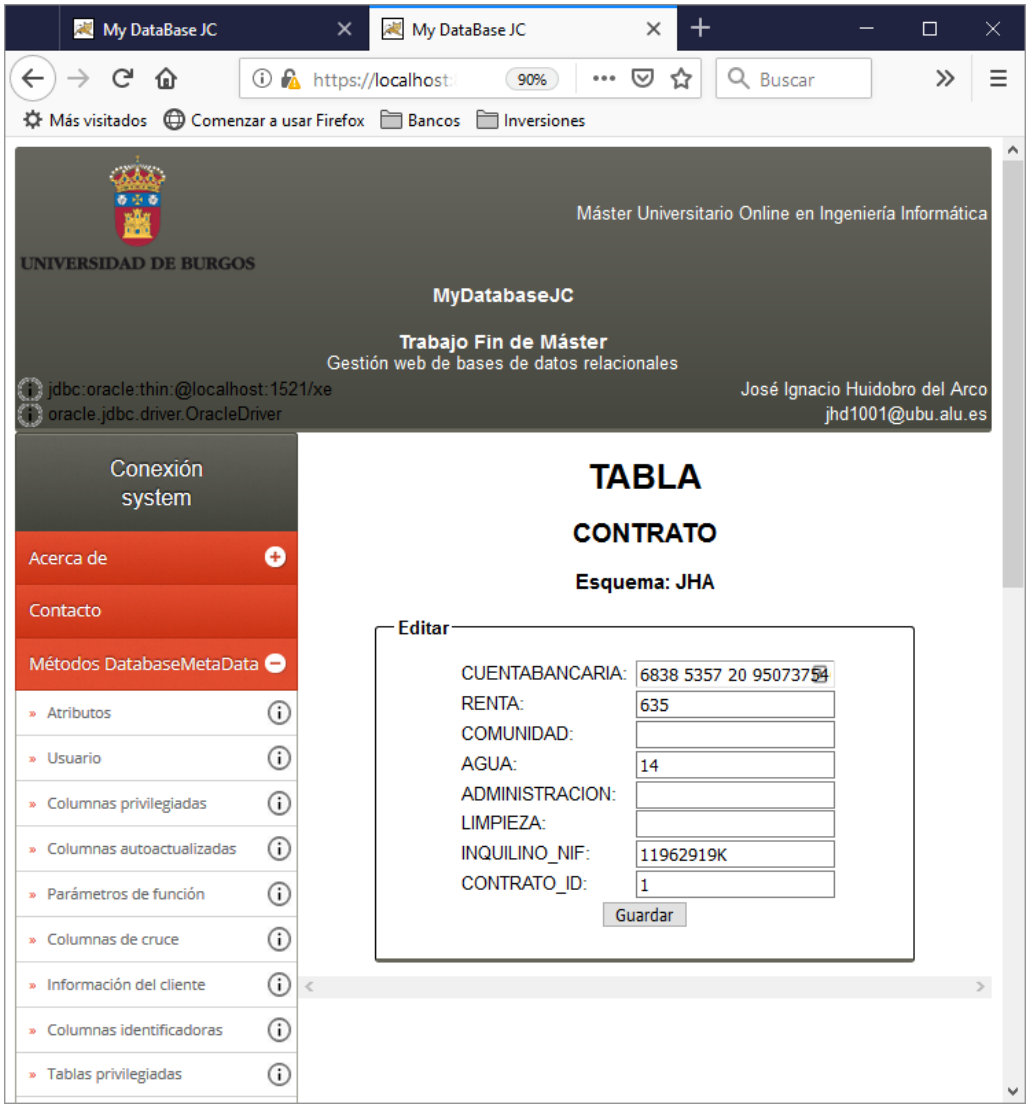

Figura 23: Editar registro de una tabla

En la pantalla se ve un formulario con todos los campos editables de la tabla, con los datos que actualmente están en el registro y la posibilidad de cambiarlos y finalmente, mediante el botón **Guardar**, grabar los cambios en la tabla correspondiente.

Insertar registro

Independientemente del registro en que se realice, un clic en el icono de insertar registro aparecerá la siguiente pantalla:

Figura 24: Insertar registro en una tabla

De manera similar a la pantalla de edición, el usuario completará los datos del registro y al hacer clic en **Guardar**, se almacenarán los datos en un nuevo registro en la tabla seleccionada.

Bibliografía

[1] D. d. V. Roque, «gestiopolis,» 25 11 2014. [En línea]. Available: https://www.gestiopolis.com/estimacion-de-costos-de-desarrollo-de-software/.

[2] Administración electrónica, Gobierno de España, «Estudio de viabilidad del sistema,» [En línea]. Available: https://administracionelectronica.gob.es/pae_Home/dam/jcr:75728be8-964e-45e8-865e-e3582c232cc8/METRICA_V3_Estudio_de_Viabilidad_del_Sistema.pdf.

[3] M. y. Barrado, «Especificación de Requisitos Software según el estándar IEEE 830,» 22 10 2008. [En línea]. Available: https://wikis.fdi.ucm.es/ELP/Especificaci%C3%B3n_de_Requisitos_Software_seg%C3%BAn_el_est%C3%A1ndar_IEEE_830.

[4] The Apache Software Foundation, «Introduction to the Build Lifecycle,» 20 01 2019. [En línea]. Available: http://maven.apache.org/guides/introduction/introduction-to-the-lifecycle.html.

[5] The Apache Software Foundation, «Plugin Bindings for default Lifecycle Reference,» 24 10 2018. [En línea]. Available: http://maven.apache.org/ref/3.6.0/maven-core/default-bindings.html.

[6] JUnit, «JUnit,» 2019. [En línea]. Available: https://junit.org/junit5/.

[7] The Apache Software Foundation, «Apache JMeter,» 2018. [En línea]. Available: https://jmeter.apache.org.

[9] chuidiang, «DataBaseMetaData y ResultSetMetaData con java y MySQL,» 4 2 2007. [En línea]. Available: http://www.chuidiang.org/java/mysql/ResultSet-DataBase-MetaData.php.

[10] Apache Software Fundatio, «Tomcat Web Application Deployment,» 9 11 2018. [En línea]. Available: https://tomcat.apache.org/tomcat-7.0-doc/deployer-howto.html.

[11] Apache Software Foundation, «Security Considerations,» 9 11 2018. [En línea]. Available: https://tomcat.apache.org/tomcat-7.0-doc/security-howto.html.

[12] Apache Software Foundation, «Apache Maven Compiler Plugin,» 26 7 2018. [En línea]. Available: https://maven.apache.org/plugins/maven-compiler-plugin/.

[13] Apache Software Foundation, «Apache Maven WAR Plugin,» 3 6 2018. [En línea]. Available: https://maven.apache.org/plugins/maven-war-plugin/.

[14] Apache Software Foundation, «Apache Maven Dependency Plugin,» 19 5 2018. [En línea]. Available: https://maven.apache.org/plugins/maven-dependency-plugin/.

[15] Apache Software Foundation, «Tomcat Setup,» 9 11 2018. [En línea]. Available: https://tomcat.apache.org/tomcat-7.0-doc/setup.html.

[16] Apache Software Foundation, «Manager App HOW-TO,» 9 11 2018. [En línea]. Available: https://tomcat.apache.org/tomcat-7.0-doc/manager-howto.html.

[17] Apache Software Foundation, «JNDI Datasource HOW-TO,» 9 11 2018. [En línea]. Available: https://tomcat.apache.org/tomcat-7.0-doc/jndi-datasource-examples-howto.html.

[18] Apache Software Foundation, «Class Loader HOW-TO,» 9 11 2018. [En línea]. Available: https://tomcat.apache.org/tomcat-7.0-doc/class-loader-howto.html.

[19] Apache Software Foundation, «Connectors How To,» 9 11 2018. [En línea]. Available: https://tomcat.apache.org/tomcat-7.0-doc/connectors.html.

[20] Programación en castellano, S.L., «Servlets básico,» [En línea]. Available: https://programacion.net/articulo/servlets_basico_108/2.

[21] Stack Exchange, Inc., «What is the difference between javaee-api and javaee-web-api?,» 2013. [En línea]. Available: https://stackoverflow.com/questions/16789020/what-is-the-difference-between-javaee-api-and-javaee-web-api.

[22] Fundación Wikimedia, Inc., «Java EE,» 2018. [En línea]. Available: https://es.wikipedia.org/wiki/Java_EE.

[23] Fundación Wikimedia, Inc., «Java Servlet,» 2018. [En línea]. Available: https://es.wikipedia.org/wiki/Java_Servlet.

[24] Fundación Wikimedia, Inc., «Java Server Page,» 12 06 2018. [En línea]. Available: https://es.wikipedia.org/wiki/JavaServer_Pages.

[25] Oracle Corporation, «Java JDBC API,» 2018. [En línea]. Available: https://docs.oracle.com/javase/8/docs/technotes/guides/jdbc/.

[26] Oracle Corporation, «Industry Support,» [En línea]. Available: https://www.oracle.com/technetwork/java/index-136695.html.

[27] Oracle Corporation, «JDBC Overview,» [En línea]. Available: https://www.oracle.com/technetwork/java/overview-141217.html.

[28] Oracle Corporation, «Features,» [En línea]. Available: https://www.oracle.com/technetwork/java/features-135464.html.

[29] L. Hernández, «API JDBC como interfaz de acceso a bases de datos SQL,» 28 9 2004. [En línea]. Available: http://www.iuma.ulpgc.es/users/lhdez/inves/pfcs/memoria-ivan/node8.html.

[30] Fundación Wikipedia, inc., «Tomcat,» 15 12 2018. [En línea]. Available: https://es.wikipedia.org/wiki/Tomcat.

[31] Wikimedia Foundation, Inc., «Apache Struts 2,» 18 11 2018. [En línea]. Available: https://en.wikipedia.org/wiki/Apache_Struts_2.

[32] Apache Software Foundation, «Key Technologies Primer,» 2018. [En línea]. Available: https://struts.apache.org/primer.html.

[33] Apache Software Foundation, «How To Create A Struts 2 Web Application,» 2018. [En línea]. Available: https://struts.apache.org/getting-started/how-to-create-a-struts2-web-application.html.

[34] Wkimedia Foundation, Inc., «Maven,» 8 1 2019. [En línea]. Available: https://es.wikipedia.org/wiki/Maven.

[35] Wikimedia Foundation, Inc., «Netbeans,» 28 12 2018. [En línea]. Available: https://es.wikipedia.org/wiki/NetBeans.

[36] Apache Software Foundation, «Creating an Enterprise Application Using Maven,» 2018. [En línea]. Available: https://netbeans.org/kb/docs/javaee/maven-entapp.html.