Memoria Practica 8, ExistDB

Que

Modifica el programa anterior para que puga almacenar l'informació en una base de datos EXISTdb

Para que

Este ejercicio ha resultado útil para aprender y practicar el funcionamiento de los conectores con la base de datos noSQL ExistDB. También ha sido beneficioso para adquirir conocimientos sobre ExistDB, ya que fue necesario informarme sobre como es su funcionamiento. En mi experiencia, la dificultad de este programa reside la gestión por software, además de nuevas implementaciones en el programa. En resumen, la actividad fue útil para comprender mejor el funcionamiento de ExistDB, así como para aprender y saber como se programan este tipo de bases de datos, entre otros aspectos.

Como

Paquete `com.existdb.utilidades`

Este paquete ya ha sido explicado en las anteriores prácticas. Se encuentran dos clases: `ReadClient` sirve para preguntar datos por consola y `Colors` para sacar mensajes por consola con colores.

Paquete `com.existdb.modelos`

Dentro de este paquete se encuentran tres clases para gestionar los objetos de tipo alumno, módulo y matrícula. Estas simplemente tienen los getters y setters, el toString y una función para pasar los datos a XML.

Paquete `com.existdb.gestor`

En este paquete se encuentra las clases para gestionar la base de datos.

Clase `Conexion`

Esta clase gestiona la conexión a una base de datos existente en eXist-DB, un sistema de gestión de base de datos nativo XML. Utiliza la API XML:DB para Java (xmldb-api) para interactuar con la base de datos eXist. eXist-DB almacena datos en formato XML y proporciona un entorno de base de datos XML completo.

La conexión se realiza utilizando las variables constantes proporcionadas, que incluyen la URI del servidor de eXist-DB, el puerto y las credenciales de usuario (nombre de usuario y contraseña). La clase utiliza el controlador de eXist-DB y registra la base de datos mediante el método

`startConexion()`. El método `getCollection()` devuelve un objeto `Collection`, que representa la conexión establecida con la base de datos.

La clase también incluye un método `getDocument(nombreDocumento)`, que obtiene un recurso XML de la base de datos por su nombre, y un método `cerrarConexion()` para cerrar la conexión cuando sea necesario.

Además, se proporciona un método `isConexionEstablecida()` que permite verificar si la conexión con la base de datos está activa en un momento dado.

Clase `CrudManager`

La clase `CrudManager` está diseñada para gestionar operaciones CRUD (Crear, Leer, Actualizar, Eliminar) en una base de datos basada en documentos XML, utilizando eXist-DB y la API XML:DB para Java. La clase utiliza la conexión proporcionada por la clase `Conexion` para interactuar con la base de datos eXist-DB.

El método `createResource` simplifica la creación o actualización de documentos en la base de datos. Recibe el nombre del recurso, el contenido XML y el nombre del elemento raíz. Comienza obteniendo la conexión a la base de datos con `getDatabase` de `Conexion`. Intenta obtener el recurso existente; si no existe, crea un nuevo documento. Luego, parsea el contenido XML y crea un nuevo elemento. Este elemento se agrega al documento existente como hijo del elemento raíz. Se convierte el documento modificado a cadena XML y, si el recurso no existía, se crea y almacena en la base de datos. El método proporciona una interfaz intuitiva para gestionar documentos en la base de datos eXist-DB, manejando la creación y actualización con eficiencia.

El método `createNewDocument()` crea un nuevo documento XML utilizando la API DOM (Document Object Model). Recibe el nombre del elemento raíz como parámetro (`root`). Comienza creando un `DocumentBuilder` a través de `DocumentBuilderFactory` para construir documentos XML. Luego, crea un nuevo documento con `builder.newDocument()`. Añade un elemento raíz al documento utilizando el nombre proporcionado (`root`) y lo retorna.

El método `parseXmlContent()` recibe una cadena XML (`xmlContent`) y la convierte en un documento DOM utilizando la API DOM. Crea un `DocumentBuilder` mediante `DocumentBuilderFactory` y, a partir de la cadena XML proporcionada, crea un objeto `InputSource`. Luego, parsea esta entrada para obtener un documento DOM, el cual es devuelto por la función.

El método `createElementFromXml()` toma una cadena XML (`xmlContent`) y un documento DOM (`document`). Utiliza un `DocumentBuilder` para parsear la cadena XML y crear un nuevo documento (`newDocument`). Luego, importa el elemento raíz de este nuevo documento al documento original, devolviendo el elemento resultante. Este proceso permite agregar contenido XML a un documento existente.

El método `convertDocumentToString()` transforma un documento DOM en una cadena XML. Utiliza `TransformerFactory` para obtener un `Transformer` y `StringWriter` para almacenar la salida transformada. Posteriormente, realiza la transformación del documento a una cadena

mediante `transform(new DOMSource(document), new StreamResult(writer))` y devuelve la cadena resultante.

La función `readResource(String resourceName)` tiene como objetivo obtener un recurso XML de la base de datos con un nombre específico (resourceName). Utiliza la clase `Conexion` para obtener la colección actual de la base de datos a través del método `getDatabase`. Luego, intenta obtener el recurso XML de la colección utilizando el nombre proporcionado y lo retorna.

La función `updateFieldById` comienza obteniendo la conexión a la base de datos a través de `Conexion.getDatabase()`. Luego, intenta obtener el recurso XML deseado de la colección. Posteriormente, extrae el contenido actual del documento XML y lo parsea a un objeto Document. El proceso continúa con la obtención de una lista de nodos que coinciden con el nombre especificado. Se itera sobre esta lista para localizar el nodo con un atributo "id" igual al valor objetivo. Los nodos representan unidades individuales dentro de un documento. Una vez encontrado, se actualiza el valor del campo designado con el nuevo valor proporcionado. La cadena XML modificada se genera convirtiendo el Document actualizado a una cadena mediante el uso de `convertDocumentToString`. Esta cadena actualizada se establece como el nuevo contenido del recurso XML. Luego, el recurso actualizado se almacena nuevamente en la base de datos mediante `collection.storeResource(resource)`.

La función `deleteResource` elimina un elemento específico de un documento XML almacenado en una base de datos. Su funcionamiento es similar al de la función anterior.

La función `getElementsByTagName` busca y devuelve una lista de nodos en un documento XML representado por la cadena `xmlContent`, que tienen el nombre especificado por `elementName`. Primero, se crea un objeto `DocumentBuilderFactory` y un `DocumentBuilder` para procesar el contenido XML. Luego, se parsea la cadena XML a un objeto `Document` utilizando un `InputSource`. Posteriormente, se invoca el método `getElementsByTagName(elementName)` en el documento, que devuelve una lista de nodos coincidentes con el nombre del elemento especificado.

La función `exportToXml` exporta el contenido de un recurso XML de la base de datos a un archivo especificado. Primero, se obtiene el recurso con el nombre dado utilizando la clase `Conexion` y la conexión actual. Si el recurso existe, se obtiene su contenido como una cadena XML. Luego, se crea un objeto `File` representando el archivo de destino especificado por `filePath`. Si el archivo ya existe, su contenido se borra para sobrescribirlo. A continuación, se utiliza un bloque `try-with-resources` para manejar la escritura del contenido XML en el archivo.

La función `importarDesdeXml` implementa la lógica para gestionar que no haya alumnos repetidos y llama a `addAlumno`.

La función `eliminarAlumno` implementa la lógica para gestionar que no exista el alumno y llama a `deleteAlumnoById`.

La función `exportar` llama a la función del padre para exportar pasándole los parámetros del documento y el path del archivo.

La función `importar` llama a la función del padre para importar pasándole los parámetros del documento y el path del archivo.

Clase `AlumnosDB`

La función `getAllAlumnos()` obtiene todos los registros de alumnos almacenados en la base de datos eXist-DB. Primero, utiliza la función `readResource(documento)` para obtener el recurso XML correspondiente al documento especificado. Luego, se extrae el contenido XML del recurso y se obtiene una lista de nodos "alumno" mediante `getElementsByTagName(xmlContent, "alumno")`. A continuación, se itera sobre estos nodos para crear objetos `Alumno` a partir de los elementos XML de cada nodo, utilizando la función `parseAlumnoFromElement(alumnoElement)`. Estos objetos `Alumno` se agregan a una lista (`alumnosList`). Finalmente, la función retorna la lista de alumnos.

La función `getAlumnoByNia` busca y retorna un objeto `Alumno` específico según el número de identificación del alumno (NIA). Primero, obtiene la lista completa de todos los alumnos. Luego, utiliza un flujo (`Stream`) para filtrar y encuentra el alumno con el NIA proporcionado. El resultado se maneja con un `Optional`, y se retorna el objeto `Alumno` encontrado o `null` si no hay coincidencias, proporcionando una gestión segura de la búsqueda.

La función `obtenerMaximoId` busca y devuelve el valor máximo del atributo "id" de los elementos "alumno" en un documento XML. Primero, obtiene el contenido del documento mediante la clase `Conexion`. Luego, parsea el contenido a un objeto `Document` para realizar operaciones XML. Posteriormente, utiliza un bucle para iterar sobre los elementos "alumno" y extraer sus atributos "id", buscando el máximo valor.

La función `parseAlumnoFromElement` convierte un elemento XML representando un alumno en un objeto de la clase `Alumno`. Recupera los valores de los atributos y elementos del elemento XML utilizando métodos auxiliares. La función `getChildValue` obtiene el valor de un elemento hijo del elemento padre.

La función `findAlumnoById` busca y recupera un objeto `Alumno` a partir de un contenido XML, utilizando un identificador específico. Se crea un documento XML a partir del contenido proporcionado y se recorren los elementos "alumno" comparando sus atributos "id". Si se encuentra un elemento con el identificador buscado, se utiliza la función `parseAlumnoFromElement` para convertirlo en un objeto `Alumno` y se devuelve.

La función `addAlumno` añade un alumno utilizando la función del padre.

La función `deleteAlumnoById` elimina un alumno utilizando la función del padre.

Clase `ModulosDB`

Esta clase es bastante similar a la de alumnos pero aplicada a los módulos, por ello considero que no es necesario explicarla, por no añadir que lo único que cambia es que tiene un método para listar los módulos.

Clase `MatriculasDB`

Esta clase también es bastante similar a las dos anteriores, por ello solo explicaré la función `updateMatricula` ya que es la única diferente.

La función `updateMatricula` actualiza las notas de una matrícula en la base de datos. Primero, busca la matrícula por su ID utilizando funciones auxiliares como `readResource` y `findMatriculaById`. Si la matrícula existe, llama a `updateFieldById` para modificar las notas de la matrícula en el recurso XML correspondiente. La actualización se realiza a través de la función `super.updateFieldById`, donde se especifica el recurso, el ID de la matrícula, el elemento XML a actualizar ("matricula"), el campo específico a modificar ("notas") y el nuevo valor. Finalmente, se imprime un mensaje indicando el éxito de la operación o informando si no se encontró la matrícula.

Clase `ClientCommands`

Esta clase se encarga de gestionar la lógica para pedir datos al cliente por consola y llamar a las funciones respectivas de las clases, por tanto simplemente las mencionaré y diré lo que hacen, ya que creo que ya han sido varias veces explicadas en las anteriores prácticas.

- `altaAlumno()`: Solicita datos de un nuevo alumno, muestra un resumen y permite agregarlo a la base de datos si el usuario confirma.
- `bajaAlumno()`: Pide el NIA de un alumno, confirma la eliminación y procede a dar de baja al alumno en la base de datos.
- `listarAlumnos()`: Imprime en consola la información de todos los alumnos almacenados en la base de datos.
- `altaModulo()`: Solicita el nombre de un nuevo módulo, muestra un resumen y lo agrega a la base de datos si el usuario lo confirma.
- `bajaModulo()`: Muestra los módulos existentes, permite seleccionar uno para dar de baja y confirma la eliminación en la base de datos.
- `listarModulos()`: Muestra en consola la información de todos los módulos almacenados en la base de datos.
- `altaMatricula()`: Matricula a un alumno en un módulo seleccionado, verificando la existencia del alumno y mostrando los módulos disponibles.
- `bajaMatricula()`: Permite dar de baja una matrícula seleccionada para un alumno específico, mostrando las matrículas disponibles.
- `anadirNota()`: Añade una nota a la matrícula seleccionada para un alumno, mostrando las matrículas y solicitando la nota.
- `modificarNota()`: Permite modificar una nota específica de la matrícula seleccionada para un alumno, mostrando las matrículas y solicitando la nueva nota.
- `eliminarNota()`: Elimina una nota específica de la matrícula seleccionada para un alumno, mostrando las matrículas y solicitando la nota a eliminar.
- `listarCentor()`: Muestra en consola la información detallada de todos los alumnos, sus matrículas y notas.
- `exportarAll()`: Exporta los datos de alumnos, módulos y matrículas a archivos XML en ubicaciones predefinidas.
- `importarAll()`: Importa los datos de alumnos, módulos y matrículas desde archivos XML en ubicaciones predefinidas.

Paquete `com.existdb`

En este paquete se encuentra la clase `Menu` ya explicada varias veces. Esta clase se encarga de mostrar un menú al cliente e llamar a las respectivas funciones de la clase `ClientCommands` con elecciones del usuario. Finalmente, se encuentra la clase `Main` que establece la conexión y llama a la clase `Menu`.

Conclusión

En conclusión, considero que esta actividad ha sido fascinante y ha aportado de manera significativa a mejorar mis habilidades de programación con conectores y bases de datos ExistDB. La utilización de nuevas funciones para acceder y modificar la base de datos desde mi programa ha ampliado mi comprensión y destrezas en este ámbito. A lo largo de la actividad.

La práctica ha sido beneficiosa y ha reforzado mi conocimiento en el manejo de bases de datos. Además, me intriga conocer las diferentes aproximaciones que mis compañeros han tomado para abordar esta actividad, ya que estoy consciente de que hay varias formas de implementarla. Consultar a mis amigos y obtener explicaciones sobre sus enfoques podría proporcionarme valiosas perspectivas y aprender nuevas técnicas.

En general, aunque la actividad fue fácil de entender conceptualmente, la implementación resultó ser desafiante debido a su implementacion de nuevas funciones, la posibilidad de errores y la gestión de un nuevo funcionamiento en la base de dato. Sin embargo, estoy satisfecho con los resultados obtenidos y considero que la dificultad fue proporcional al aprendizaje adquirido.