Que

Modifica el programa anterior para que, utlitzant varis connectors, puga almacenar l'informació a diverses Base de Dades

Para qué

Este ejercicio ha resultado útil para aprender y practicar el funcionamiento de los conectores. También ha sido beneficioso para adquirir conocimientos sobre bases de datos, ya que fue necesario ejecutar varias consultas. En mi experiencia, la dificultad de este programa reside en las consultas y la gestión de poderse conectar a otras bases de datos, además de la variabilidad en la sintaxis SQL entre distintas bases de datos. En resumen, la actividad fue útil para comprender mejor el funcionamiento de conectores, así como para aprender a realizar consultas, entre otros aspectos.

Pseudocodigo

Conexion

```
Enumeración DatabaseType:
    MYSQL
    POSTGRESQL

Clase Conexion:

    Constante URL: "jdbc:mysql://localhost/"
    Constante DB: "10813358"
    Constante USER: "10813358"
    Constante PASSWORD: "10813358"
    Variable `con` inicializada a nulo

    Función getConnection:
        Intentar:
              con -> Llamar a DriverManager.getConnection concatenando URL,
DB, USER y PASSWORD
        Capturar SQLException:
            Mensaje de error
        Devolver con

    Función getDataBaseType:
        Variable databasetype inicializada a nulo
        Si URL contiene "postgresql":
            databasetype -> POSTGRESQL
        Sino, si URL contiene "mysql":
            databasetype -> MYSQL
        Devolver databasetype
```

gestor

```
Clase gestor:
    Crear una instancia de ReadClient llamada rc
    Crear una instancia de Conexion llamada conexion

    Función testConexion:
        Intentar:
                connection = Llamar a getConnection desde la instancia de
conexion
            Si connection no es nulo y no está cerrado:
                Llamar a Colors.okMsg con el mensaje "¡Conexión exitosa!"
                Devolver verdadero
            Sino:
                   Llamar a Colors.errMsg con el mensaje "La conexión está
cerrada o es nula."
        Capturar SQLException:
            Mensaje de error
        Devolver falso

    Función executeUpdate con parámetro query:
        Intentar:
                connection = Llamar a getConnection desde la instancia de
conexion
            Si connection no es nulo:
                Intentar:
                      statement = Crear un PreparedStatement con la query y
la conexión
                    Llamar a statement.executeUpdate
                Capturar SQLException:
                    Mensaje de error
            Sino:
                Mensaje de error
        Capturar SQLException:
            Mensaje de error

    Función select con parámetros select, from, where, returnType y params:
          Crear la query concatenando "SELECT ", select, " FROM ", from, "
WHERE ", where
        Inicializar result a nulo

        Intentar:
            rs -> Llamar a executeSelect con la query y los parámetros
            Si rs.next():
                Si returnType es Integer:
                    Asignar a result el valor casteado de rs.getInt(1)
                Sino, si returnType es String:
                    Asignar a result el valor casteado de rs.getString(1)
        Capturar SQLException:
            Mensaje de error
        Devolver result

    Función executeSelect con parámetros query y params:
        Intentar:
```

```
        Intentar:
               connection -> Llamar a getConnection desde la instancia de
conexion
            Si connection no es nulo:
                Intentar:
                     statement -> Crear un PreparedStatement con la query y
la conexión
                    Por cada parámetro en params:
                           Llamar a statement.setObject con el índice y el
parámetro
                    Devolver statement.executeQuery
                Capturar SQLException:
                    Mensaje de error
        Capturar SQLException:
            Mensaje de error
        Devolver nulo

          Función createTable con parámetros tableName, queryMYSQL y
queryPOSTGRESQL:
        Obtener el tipo de base de datos desde la instancia de Conexion
        Si la tabla no existe:
             Llamar a Colors.debMsg con el mensaje "Tabla " + tableName + "
creada."
            Inicializar createTableQuery a cadena vacía

            Si el tipo de base de datos es MYSQL:
                        Asignar a createTableQuery el valor de queryMYSQL
formateado con tableName
            Sino, si el tipo de base de datos es POSTGRESQL:
                Asignar a createTableQuery el valor de queryPOSTGRESQL
            Llamar a executeUpdate con createTableQuery

    Función getTitulo con parámetro tableName:
        Crea un mensaje de titulo para las listas

    Función write con parámetros path y datos:
        Intentar:
            Crear un objeto File con path
            Si el archivo existe, borrarlo
            Sino, crear un nuevo archivo
            Intentar:
                     Crear un BufferedWriter con un FileWriter asociado al
archivo
                Escribir los datos en el archivo
                Mensaje de OK
            Capturar IOException:
                 Mensaje de error
        Capturar IOException:
             Mensaje de error

    Función read con parámetro path:
        Crear una lista de cadenas llamada lines

        Intentar:
            Crear un objeto File con path
```

```
            Crear un objeto File con path
            Si el archivo no existe:
                Mensaje de error
                Devolver la lista de líneas
            Intentar:
                    Crear un BufferedReader con un FileReader asociado al
archivo
                Leer cada línea del archivo y añadirla a la lista de líneas
        Capturar IOException:
            Mensaje de error
        Capturar IOException:
            Mensaje de error
        Devolver la lista de líneas

    Función tableExists con parámetro tableName:
        Obtener el tipo de base de datos desde la instancia de Conexion
        Si el tipo de base de datos es nulo:
            Mensaje de error
            Devolver falso
        Sino:
            Crear query según el tipo de base de datos

            Intentar:
                Ejecutar la query con executeSelect
                Obtener el resultado del conteo de tablas
                Devolver verdadero si el resultado es mayor a 0, sino falso
            Capturar SQLException:
                Mensaje de error
                Devolver falso
```

gsAlumnos

```
Clase gsAlumnos extiende gestor:

    Crear una instancia de ReadClient llamada rc
    Constante ALUMNOSPATH con valor "./res/alumnos.txt"

    Función alta:
        Pedir nombre, apellidos y nia al usuario
        Pedir día, mes y año de nacimiento al usuario
        Dar formato a la fecha
        Llamar a insertAlumno con los datos proporcionados
        Mensaje de OK

    Función baja:
        Obtener el ID del alumno con NIA
        Si el ID es 0, imprimir "Se ha cancelado la eliminación."
          Sino, llamar a dropAlumno con el ID y luego imprimir que se a
eliminado correctamente

    Función mostrarAlumnos:
        Crear la query selectQueryAlumnos
        Imprimir el título obtenido con la función getTitulo
```

```
        Intentar:
            Ejecutar la query y obtener el ResultSet
            Iterar sobre el ResultSet e imprimir los datos de los alumnos
        Capturar SQLException:
            Mensaje de Error

    Función getIDConNIA:
         Pedir NIA al usuario y retornar el resultado de encontrarID con el
NIA

    Función encontrarID con parámetro nia:
             Llamar a select para obtener el ID del alumno con el NIA
proporcionado
          Si el resultado es diferente de nulo, retornar el resultado, sino
retornar -1

    Función createTable:
          Crear la tabla de alumnos con la función createTable de la clase
padre

    Función exportTable:
        Crear la query para obtener datos de alumnos
        Obtener el ResultSet ejecutando la query
        Inicializar una cadena datos
        Iterar sobre el ResultSet y construir la cadena de datos
        Llamar a write con ALUMNOSPATH y datos

    Función importTable:
        Leer las filas del archivo ALUMNOSPATH
        Iterar sobre las filas y dividirlas en datos
        Para cada fila:
            Obtener el ID del alumno con el NIA proporcionado
                   Si el ID es diferente de -1, construir una query de
actualización, sino una query de inserción
            Ejecutar la query con executeUpdate

    Función comprobarNia con parámetros nia y exist:
        Obtener el ID del alumno con el NIA proporcionado
         Si exist es true, retornar si el ID es diferente de -1, sino si es
-1

    Función pedirNia con parámetro exist:
        Inicializar cancel y errMsg
        Si exist es true, asignar valores específicos
        Pedir NIA al usuario hasta que sea válido según exist
        Retornar el NIA

    Función insertAlumno con parámetros name, surname, fecha y nia:
        Construir la query de inserción con los datos proporcionados
        Llamar a executeUpdate con la query

    Función dropAlumno con parámetro id:
         Construir las queries para eliminar el alumno de las matrículas y
de la tabla de alumnos
```

```
        Llamar a executeUpdate con ambas querys
```

gsModulos

```
Clase gsModulo extiende gestor:

    Crear una instancia de ReadClient llamada rc
    Constante MODULOSPATH con valor "./res/modulos.txt"

    Función baja:
        Obtener el ID del módulo con nombre
        Si el ID es diferente de -1, llamar a dropModulo con el ID

    Función alta:
        Pedir nombre de nuevo módulo al usuario hasta que sea único
        Llamar a insertModulo con el nombre proporcionado
        Imprimir "Módulo registrado correctamente."

    Función createTable:
          Crear la tabla de módulos con la función createTable de la clase
padre

    Función exportTable:
        Crear la query para obtener datos de módulos
        Obtener el ResultSet ejecutando la query
        Inicializar una cadena datos
        Iterar sobre el ResultSet y construir la cadena de datos
        Llamar a write con MODULOSPATH y datos

    Función importTable:
        Leer las filas del archivo MODULOSPATH
        Iterar sobre las filas y dividirlas en datos
        Para cada fila:
            Obtener el ID del módulo con el nombre proporcionado
                   Si el ID es diferente de -1, construir una query de
actualización, sino una query de inserción
            Ejecutar la query con executeUpdate

    Función mostrarModulos:
        Crear la query selectQueryAlumnos
        Imprimir el título obtenido con la función getTitulo
        Intentar:
            Ejecutar la query y obtener el ResultSet
            Iterar sobre el ResultSet e imprimir los nombres de los módulos
        Capturar SQLException:
              Llamar a Colors.errMsg con el mensaje "Imposible mostrar los
módulos"

    Función pedirIDconNombre:
        Pedir nombre del módulo al usuario
            Si el nombre es "/c", imprimir "Operación cancelada por el
usuario." y retornar -1
```

```
        Obtener el ID del módulo con el nombre proporcionado
        Si el ID es -1, imprimir "No existe ningún módulo con ese nombre."

    Función encontrarIDconNombre con parámetro name:
            Llamar a select para obtener el ID del módulo con el nombre
proporcionado
         Si el resultado es diferente de nulo, retornar el resultado, sino
retornar -1

    Función dropModulo con parámetro id:
          Construir las queries para eliminar el módulo de las matrículas y
de la tabla de módulos
        Llamar a executeUpdate con ambas queries
        Imprimir "Modulo eliminado exitosamente"

    Función insertModulo con parámetro modulo:
        Construir la query de inserción con el nombre proporcionado
        Llamar a executeUpdate con la query
```

gsMatriculas

```
Clase gsMatriculas extiende gestor:

    Crear una instancia de ReadClient llamada rc
    Crear una instancia de gsAlumnos llamada gsa
    Crear una instancia de gsModulo llamada gsm
         Crear una constante String llamada MATRICULASPATH con valor
"./res/matriculas.txt"

    Función crearMatricula:
        Obtener el ID del alumno con NIA desde gsa
        Obtener el ID del módulo con nombre desde gsm
        Si el ID de la matrícula con IDs obtenidos es -1:
                Llamar a insertMatricula con el ID del alumno, el ID del
módulo, y notas nulas
        Sino:
            Imprimir "La matricula ya existe"

    Función eliminarMatricula:
        Obtener el ID de la matrícula mediante pedirID
        Si el ID es diferente de -1:
            Llamar a dropMatricula con el ID
            Imprimir "Matricula eliminada correctamente"

    Función modificarNotas:
        Obtener el ID de la matrícula mediante pedirID
        Si el ID es diferente de -1:
            Obtener el ID del módulo asociado a la matrícula
            Obtener el nombre del módulo con el ID obtenido
             Llamar a mostrarNotas con el ID de la matrícula y el nombre del
módulo
            Obtener la longitud de las notas
```

```
            Si la longitud de las notas es mayor a 0:
                Pedir al usuario el número de notas a modificar
                Crear un HashMap llamado notasUpdate
                Para cada iteración hasta el número de notas a modificar:
                    Pedir al usuario la posición de la nota a modificar
                    Pedir al usuario la nueva nota
                    Agregar la posición y la nueva nota al HashMap
                   Llamar a modNotas con el ID de la matrícula y las notas
actualizadas
                Imprimir "Notas actualizadas correctamente"

    Función mostrarModuloAlumno:
        Obtener el ID del alumno con NIA desde gsa
        Si el ID del alumno es diferente de 0:
            Obtener el ID del módulo con nombre desde gsm
            Si el ID del módulo es diferente de -1:
                Obtener el nombre del módulo con el ID obtenido
                 Llamar a mostrarNotas con el ID del módulo y el nombre del
módulo
        Sino:
            Imprimir "Operación cancelada por el usuario."

    Función mostrarModulosAlumno con parámetro almID:
          Obtener el ResultSet de la consulta SELECT a la tabla matriculas
con el ID del alumno
        Crear un HashSet llamado matAlmIdProcesados
        Crear variables almName, almSurnames y almNia
        Mientras haya filas en el ResultSet:
            Obtener el ID del alumno y el ID del módulo
            Obtener el nombre del módulo con el ID obtenido
             Llamar a mostrarNotas con el ID de la matrícula y el nombre del
módulo

    Función mostrarCentro:
        Obtener el ResultSet de la consulta SELECT a la tabla matriculas
        Crear un HashSet llamado matAlmIdProcesados
        Crear variables almName, almSurnames y almNia
        Mientras haya filas en el ResultSet:
                        Obtener el ID del alumno y añadirlo al HashSet
matAlmIdProcesados
        Imprimir título de matriculas
        Para cada matAlmIdProcesado:
              Obtener el ResultSet de la consulta SELECT a la tabla alumnos
con el ID del alumno
            Mientras haya filas en el ResultSet de alumnos:
                Obtener el nombre, apellidos y NIA del alumno
            Imprimir el nombre, apellidos y NIA del alumno
            Llamar a mostrarModulosAlumno con el ID del alumno

    Función createTable:
        Definir el nombre de la tabla como "matriculas"
        Definir la consulta SQL para MySQL y PostgreSQL
             Llamar a super.createTable con el nombre de la tabla y las
consultas SQL
```

```
    Función exportTable:
          Definir la consulta SQL para seleccionar todas las filas de la
tabla matriculas
        Obtener el ResultSet de la consulta
        Inicializar una cadena llamada datos
        Mientras haya filas en el ResultSet:
            Obtener los valores de ID, ID del alumno, ID del módulo y notas
            Concatenar los valores a la cadena datos
        Llamar a super.write con la ruta y los datos
        Imprimir mensaje de éxito o error

    Función importTable:
        Leer las filas del archivo especificado en MATRICULASPATH
        Para cada fila en las filas:
            Separar los datos utilizando el delimitador ";"
            Obtener el ID de la matrícula con IDs obtenidos
            Si el ID es diferente de -1:
                Definir la consulta SQL para actualizar la matrícula
            Sino:
                Definir la consulta SQL para insertar la matrícula
            Llamar a super.executeUpdate con la consulta SQL
    Función qualificar:
        Obtener el ID del alumno con NIA desde gsa
        Obtener el ID del módulo con nombre desde gsm
        Obtener el ID de la matrícula con IDs obtenidos
        Si el ID de la matrícula es -1:
              Pedir al usuario si desea matricular al alumno directamente
(y/n)
            Si la opción es "n":
                Imprimir "No se ha podido qualificar al alumno"
                Retornar
            Sino:
                  Llamar a insertMatricula con el ID del alumno, el ID del
módulo, y notas nulas
                Obtener el ID de la matrícula con IDs obtenidos

        Pedir al usuario la cantidad de notas a añadir
        Crear un array de double llamado notas
        Para cada iteración hasta la cantidad de notas:
            Pedir al usuario la nota a añadir
            Agregar la nota al array de notas
        Llamar a addNotas con el ID de la matrícula y el array de notas
        Imprimir "Notas actualizadas correctamente"

    Función mostrarNotas con parámetros matrID y nombreModulo:
        Obtener el array de notas con getNotas usando matrID
        Imprimir el nombre del módulo
        Si la longitud de notas es igual a 0:
            Imprimir "El alumno aún no tiene notas en este módulo."
        Para cada nota en el array de notas:
            Imprimir la posición y la nota

    Función pedirID:
        Inicializar las variables nia, modulo e id en 0, repetir en true
```

```
        Mientras repetir sea true:
            Si nia es igual a 0:
                Obtener el ID del alumno con NIA desde gsa
            Si nia es igual a 0:
                Asignar false a repetir
            Sino:
                Obtener el ID del módulo con nombre desde gsm
                Si el ID del módulo es igual a -1:
                    Asignar false a repetir
                Sino:
                    Obtener el ID de la matrícula con IDs obtenidos
                    Si el ID es igual a -1:
                            Imprimir "El alumno no está matriculado en ese
módulo"
                    Sino:
                        Asignar false a repetir
        Retornar el ID

    Función modNotas con parámetros matrID y notasUpdate:
        Obtener el array de notas actuales con getNotasString usando matrID
        Para cada entrada en notasUpdate:
            Obtener la posición y la nueva nota
            Si la posición está dentro del rango de las notas actuales:
                Actualizar la posición con la nueva nota
            Sino:
                Imprimir mensaje de error
        Crear un StringBuilder llamado result
        Para cada nota en las notas actuales:
            Agregar la nota al StringBuilder
        Eliminar el último carácter del StringBuilder
        Definir la consulta SQL para actualizar las notas
        Llamar a super.executeUpdate con la consulta SQL

    Función getNotas con parámetro matrID:
          Obtener la cadena de notas en formato String con getNotasString
usando matrID
        Si la cadena no es null ni "null" ni vacía:
            Separar las notas utilizando el delimitador "#"
             Crear un array de double llamado notas con la longitud de las
notas separadas
            Para cada índice y nota en las notas separadas:
                Convertir la nota a double y asignarla al array de notas
            Retornar el array de notas
        Sino:
            Retornar un array de double vacío

    Función getNotasString con parámetro matrID:
          Obtener la cadena de notas en formato String usando select con
matrID
        Si la cadena es "null" o null:
            Asignar una cadena vacía a la variable notas
        Retornar la cadena de notas

    Función addNotas con parámetros matrID y notas:
```

```
             Obtener la cadena de notas actuales en formato String con
getNotasString usando matrID
             Obtener la cadena de notas a añadir en formato String con
createNotasString usando notas
        Si la cadena de notas actuales no es null:
            Si no está vacía:
                Concatenar la cadena de notas a añadir a la cadena de notas
actuales con el delimitador "#"
        Definir la consulta SQL para actualizar las notas
        Llamar a super.executeUpdate con la consulta SQL

    Función createNotasString con parámetro notas:
        Si la longitud de notas es 0:
            Imprimir "No se proporcionaron notas para agregar."
            Retornar una cadena vacía
        Crear un StringBuilder llamado result
        Para cada nota en las notas:
            Agregar la nota al StringBuilder con el delimitador "#"
        Eliminar el último carácter del StringBuilder
        Retornar el resultado

    Función encontrarIDconIDs con parámetros alumnoID y moduloID:
         Obtener el ID de la matrícula con IDs obtenidos usando select con
alumnoID y moduloID
        Si el resultado es diferente de null:
            Retornar el resultado
        Sino:
            Retornar -1
    Función insertMatricula con parámetros alumnoID, moduloID y notas:
             Definir la consulta SQL para insertar la matrícula con los
parámetros proporcionados
        Llamar a super.executeUpdate con la consulta SQL
        Imprimir "Matrícula creada correctamente"

    Función dropMatricula con parámetro matrID:
            Definir la consulta SQL para eliminar la matrícula con el ID
proporcionado
        Llamar a super.executeUpdate con la consulta SQL
```

menus

```
Clase menus:

    Crear una instancia de ReadClient llamada rc
    Crear una instancia de gsAlumnos llamada gsAl
    Crear una instancia de gsModulo llamada gsMod
    Crear una instancia de gsMatriculas llamada gsMat

    Función mainMenu:
        Llamar a crearTablas
        Declarar y asignar true a la variable repit
        Mientras repit sea true:
```

```
            Imprimir menú principal
            Pedir al usuario la opción del menú
            Según la opción seleccionada:
                Caso 0:
                    Imprimir "Programa cerrado"
                    Asignar false a repit
                    Romper el bucle
                Caso 1:
                    Llamar a menuAlumnos
                Caso 2:
                    Llamar a menuModulos
                Caso 3:
                    Llamar a menuMatriculas
                Caso 4:
                    Llamar a importar
                Caso 5:
                    Llamar a exportar
                Defecto:
                    Imprimir "Debes introducir un valor valido"

    Función menuAlumnos:
        Declarar y asignar 0 a la variable menu
        Declarar y asignar true a la variable repetir
        Mientras repetir sea true:
            Imprimir menú de alumnos
            Pedir al usuario la opción del menú
            Según la opción seleccionada:
                Caso 0:
                    Imprimir "Has salido del menu de Alumnos"
                    Asignar false a repetir
                    Romper el bucle
                Caso 1:
                    Llamar a gsAl.alta
                Caso 2:
                    Llamar a gsAl.baja
                Caso 3:
                    Llamar a gsAl.mostrarAlumnos
                Defecto:
                    Imprimir "Debes introducir un valor valido"

    Función menuModulos:
        Declarar y asignar 0 a la variable menu
        Declarar y asignar true a la variable repetir
        Mientras repetir sea true:
            Imprimir menú de módulos
            Pedir al usuario la opción del menú
            Según la opción seleccionada:
                Caso 0:
                    Imprimir "Has salido del menu de Modulo"
                    Asignar false a repetir
                    Romper el bucle
                Caso 1:
                    Llamar a gsMod.alta
                Caso 2:
                    Llamar a gsMod.baja
```

```
                Llamar a gsMod.baja
            Caso 3:
                Llamar a gsMod.mostrarModulos
            Defecto:
                Imprimir "Debes introducir un valor valido"

Función menuMatriculas:
    Declarar la variable menu
    Declarar y asignar true a la variable repetir
    Mientras repetir sea true:
        Imprimir menú de matrículas
        Pedir al usuario la opción del menú
        Según la opción seleccionada:
            Caso 0:
                Imprimir "Has salido del menu de Evaluar"
                Asignar false a repetir
                Romper el bucle
            Caso 1:
                Llamar a gsMat.crearMatricula
            Caso 2:
                Llamar a gsMat.eliminarMatricula
            Caso 3:
                Llamar a gsMat.qualificar
            Caso 4:
                Llamar a gsMat.modificarNotas
            Caso 5:
                Llamar a gsMat.mostrarModuloAlumno
            Caso 6:
                Obtener el ID del alumno con NIA desde gsAl
                Llamar a gsMat.mostrarModulosAlumno con el ID obtenido
            Caso 7:
                Llamar a gsMat.mostrarCentro
            Defecto:
                Imprimir "Debes introducir un valor valido"

Función crearTablas:
    Llamar a gsAl.createTable
    Llamar a gsMod.createTable
    Llamar a gsMat.createTable

Función importar:
    Llamar a gsAl.importTable
    Llamar a gsMod.importTable
    Llamar a gsMat.importTable

Función exportar:
    Llamar a gsAl.exportTable
    Llamar a gsMod.exportTable
    Llamar a gsMat.exportTable
```

Practica_06

```
Clase Practica_06:
    Función main:
        Crear una instancia de gestor llamada gs
        Si gs.testConexion es verdadero:
            Crear una instancia de menus llamada menu
            Llamar a menu.mainMenu
```

Como

Conexión

Esta clase está diseñada para gestionar la conexión a una base de datos. Con las variables constantes, podemos obtener la conexión a la base de datos. Para lograrlo, debemos inicializar un objeto de tipo Connection. La interfaz Connection en Java es parte de la API JDBC (Java Database Connectivity) y se encuentra en el paquete java.sql. Esta interfaz proporciona métodos para establecer una conexión con una base de datos y para gestionar transacciones entre la aplicación Java y la base de datos.

Para establecer una conexión, se utiliza el método DriverManager.getConnection(). Este método toma como parámetros la URL de conexión a la base de datos, el nombre de usuario y la contraseña asociados a esa conexión. Para eso, tengo un método `getConnection()` que devuelve una conexión establecida con la base de datos. Además, tengo una enumeración anidada, `DatabaseType`, con dos tipos de bases de datos y una función `getDatabaseType()` que devuelve según la URL el tipo de base de datos que estás usando.

Gestor

La clase Gestor está diseñada para facilitar la gestión de operaciones en una base de datos, integrando funcionalidades para realizar conexiones, ejecutar consultas SQL, crear y verificar tablas, así como leer y escribir datos en archivos. Utiliza la clase Conexion para establecer conexiones JDBC con la base de datos.

El método `testConexion()` verifica la conexión a la base de datos, mostrando mensajes de éxito o error. Para ello, creamos un objeto Connection y llamamos a la función `conexion.getConnection()`.

El método `executeUpdate()` crea una conexión y otro objeto del tipo `PreparedStatement` para preparar la sentencia SQL especificada en la cadena `query` (que es el String de la consulta SQL que tiene la función como parámetro) para su ejecución. Este objeto proporciona métodos para establecer valores en los parámetros de la sentencia y ejecutarla de manera eficiente. El uso de `PreparedStatement` mejora la compatibilidad con diferentes bases de datos. La mayoría de los controladores JDBC están diseñados para trabajar con sentencias SQL preparadas, lo que facilita la portabilidad de la aplicación entre diferentes sistemas de gestión de bases de datos (Database Management System, DBMS). Finalmente, se ejecuta la consulta con la función `executeUpdate()` del objeto `PreparedStatement`.

El método `executeSelect()` es similar al anterior, pero como parámetros tiene el String de la consulta y una lista de Objetos. Se crea una conexión y un `PreparedStatement`, pero además de eso se añaden al `PreparedStatement` los objetos de la lista, con un bucle que va añadiendo todos los parámetros al `PreparedStatement` mediante la función `setObject(int, Object)`. Esta función requiere la posición del parámetro que se debe modificar y el objeto que se debe colocar en dicho parámetro. Es común utilizar el carácter '?' en la consulta SQL para indicar los lugares donde se insertarán los parámetros, y luego sustituir esos marcadores de posición por los objetos deseados, asignándoles valores secuenciales comenzando desde 1, siendo el primer parámetro en la posición 1 al realizar una consulta de inserción. Por último, después de tener todos los parámetros sustituidos, ejecutamos `executeQuery()`, otro método de el `PreparedStatement`. Este método nos devuelve un `ResultSet`, que es una tabla de datos que representa un conjunto de resultados de la base de datos. Este `ResultSet` también lo devuelve la función `executeSelect`.

El método `select()` se usa para simplificar la función `executeSelect()`. Esta función toma como referencia un select y pide el contenido de ella por separado, es decir, si la sintaxis de un select es la siguiente `SELECT ... FROM ... WHERE ...`, esta función pide como parámetros el contenido del Select, el contenido del From y el contenido del Where. También tiene como parámetro una clase, para saber qué tipo de parámetro debe devolver y una lista de objetos, para pasársela a la función `executeSelect()`. La función `select()`, crea una query con los tres parámetros principales e inicializa una variable resultado a null, llama a `executeSelect()` pasándole como parámetros la query y la lista de objetos. El resultado de esa función se guarda en un `ResultSet`. Luego se comprueba si ha devuelto algo la función con un `if` que tiene como condición la función `next()` del `ResultSet`, este método mueve el cursor hacia adelante una fila desde su posición actual. Inicialmente se coloca un cursor antes de la primera fila, por eso debemos llamar a la función y esta función devuelve true si la nueva fila actual es válida. Si no hay más filas o no hay, devuelve false. Por tanto, si no hubiera ningún resultado de la consulta, devolveríamos null. Por otra parte, si existiera un resultado, comprobaríamos qué tipo de objeto se espera que se devuelva e llamamos a la respectiva función para obtener ese objeto de la clase `ResultSet`.

El método `tableExists()` tiene como parámetros la tabla que se va a comprobar si existe y devuelve un booleano según si existe o no. Según el tipo de base de datos al que te quieras conectar, ejecuta una consulta con `executeSelect()` para saber si existe o no y devuelve el resultado del `ResultSet`.

El método `createTable()` tiene como parámetros el nombre de la tabla, la consulta en MySQL y la consulta en POSTGRESQL. Primero, comprueba si la tabla no existe; si es así, comprueba qué tipo de base de datos estás utilizando e llama a `executeUpdate()` con la consulta del tipo de base de datos.

El método `write()` tiene como parámetros la ruta del fichero y los datos. Esta función sobrescribe los datos de un fichero, además de gestionar si existe o no.

El método `read()` tiene como parámetros la ruta del fichero y añade cada fila del fichero a un `ArrayList` de `string` para devolverlo.

Por último, el método `getTitulo` convierte el texto introducido como parámetro a un texto con un poco más de formato.

gsAlumnos

La clase `gsAlumnos` extiende la funcionalidad de la clase `Gestor` y se centra en la gestión de operaciones relacionadas con alumnos en una base de datos.

El método `createTable()` define dos consultas para crear la tabla de alumnos, establece el nombre de la tabla e invoca la función `createTable()` de `Gestor`.

El método `encontrarID()` tiene como parámetro el NIA e invoca la función `select()` de la clase superior. Si el resultado de la selección es `null`, devuelve `-1`.

El método `comprobarNia()` ya está explicado en la práctica anterior. El método `pedirNia()` ya está explicado en la práctica anterior.

El método `getIDConNIA()` llama a la función `pedirNia()` y con el NIA invoca la función `encontrarID()` para devolver el ID.

El método `insertAlumno()` tiene como parámetros los datos del alumno (nombre, apellidos, fecha de nacimiento y NIA). Con esos datos, crea una consulta para insertar el alumno y llama a la función `executeUpdate()` del padre.

El método `dropAlumno()` tiene como parámetro el ID del alumno y crea dos consultas para eliminar el alumno con ese ID y la matrícula asociada a ese mismo ID de alumno.

El método `alta()` pide los datos del alumno al usuario y llama a la función `insertAlumno()`.

El método `baja()` llama a la función `getIDConNIA()` y luego elimina ese ID invocando la función `dropAlumno()`.

El método `mostrarAlumnos()` imprime los alumnos del centro creando una consulta para seleccionar todos los alumnos. Llama a la función `super.getTitulo()` e imprime el resultado, luego invoca la función `executeSelect()` para obtener un `ResultSet` y mostrar toda la información de cada alumno por pantalla.

El método `importTable()` guarda en un ArrayList de Strings la llamada a la función `super.read()` pasándole como parámetro la variable constante (`ALUMNOSPATH`) de la ruta del archivo donde se encuentran los datos de los alumnos, definida al principio de la clase. Luego, separa los campos y comprueba si existe el alumno con la función `encontrarID()`. Si el alumno existe, se actualizan los datos; si no, se añaden.

Por último, el método `exportTable()` invoca la función `executeSelect()` para obtener los datos de toda la tabla. Luego, con ese `ResultSet`, se obtienen los valores de cada fila y se añaden con un formato separado por ";" a un String para después llamar a la función `super.write()` pasándole la `ALUMNOSPATH` y el string.

gsModulos

La clase `gsModulos` extiende la funcionalidad de la clase `Gestor` y se centra en la gestión de operaciones relacionadas con módulos en una base de datos. Sus métodos son similares a los de la

clase alumnos, por lo tanto, solo se describirá brevemente lo que hace cada método:

- `createTable()`: crea la tabla módulos si no existe.
- `pedirIDconNombre()`: pide el nombre y devuelve su ID con `encontrarIDconNombre()`.
- `encontrarIDconNombre()`: busca en la DB el módulo y devuelve su ID.
- `insertModulo()`: inserta un módulo en la DB.
- `dropModulo()`: elimina un módulo de la DB.
- `alta()`: pide los datos del módulo y llama a `insertModulo()`.
- `baja()`: pide el nombre del módulo y llama a `dropModulo()`.
- `mostrarModulos()`: muestra los módulos.
- `exportTable()`: exporta a un fichero los módulos.
- `importTable()`: importa de un fichero los módulos.

gsMatriculas

Esta clase `gsMatriculas` es un gestor que proporciona métodos para gestionar las matrículas de alumnos en módulos. Esta clase tiene algunos métodos similares a las dos clases anteriores, por lo que me centraré solo en los métodos diferentes:

- `insertMatricula()`: inserta una matrícula en la DB.
- `dropMatricula()`: elimina una matrícula de la DB.
- `crearMatricula()`: pide los datos de la matrícula y llama a `insertMatricula()`.
- `eliminarMatricula()`: llama a `pedirID()` y elimina la matrícula.
- `modificarNotas()`: pide al usuario los datos de las notas a modificar y llama a `modNotas()`.
- `mostrarModuloAlumno()`: muestra las notas de un módulo de un alumno llamando a la función `mostrarNotas()`.
- `mostrarModulosAlumno()`: muestra las notas de los módulos de un alumno.
- `mostrarCentro()`: muestra las notas del centro.
- `qualificar()`: pide los datos y las notas para añadir notas a una matrícula y llama a `addNotas()`. Si el alumno no está matriculado, puede, según la opción del usuario, matricularlo.
- `pedirID()`: pide el ID del alumno y el de la matrícula y devuelve el ID del módulo llamando a `encontrarIDconIDs()`.
- `createTable()`: crea la tabla matrículas si no existe.
- `exportTable()`: exporta a un fichero las matrículas.
- `importTable()`: importa de un fichero las matrículas.
- `encontrarIDconIDs()`: devuelve el ID de la matrícula con el ID del alumno y del módulo.
- `getNotas()`: obtiene las notas de una matrícula en forma de cadena llamando a `getNotasString()` y las pasa a una lista.
- `getNotasString()`: obtiene las notas de una matrícula llamando a `select()` de la clase padre.
- `addNotas()`.
- `createNotasString()`: convierte una lista de notas a una cadena separada por "#".
- `modNotas()`: añade notas a una matrícula.
- `mostrarNotas()`: muestra las notas de un alumno y de un módulo.

Menus

Estas funciones organizan las opciones disponibles para el usuario en distintos menús y gestionan la ejecución de acciones relacionadas con alumnos, módulos y evaluaciones, conectándose con las clases `gsAlumnos`, `gsModulos` y `gsMatriculas` para realizar las operaciones correspondientes en la base de datos. Además de las funciones ya explicadas, tiene tres métodos adicionales:

- `crearTablas()`: llama a la función `createTable()` de los alumnos, módulos y matrículas.
- `importar()`: llama a la función `exportTable()` de los alumnos, módulos y matrículas.
- `exportar()`: llama a la función `exportTable()` de los alumnos, módulos y matrículas.

Practica_06

Esta clase comprueba que la conexión sea correcta llamando a la función `testConexion()` de `Gestor`. Si es correcta, llama a la función `mainMenu()` de 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. 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, descubrí algunas funciones que no conocía, como la `enumeración anidada` en Java.

La práctica de diferentes consultas 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 extensión, la posibilidad de errores y la gestión de diferentes bases de datos. Sin embargo, estoy satisfecho con los resultados obtenidos y considero que la dificultad fue proporcional al aprendizaje adquirido.