

Unidad 12

Gestión de bases de datos relacionales

Programación
1º D.A.M.

1

Contenido

1. Bases de datos relacionales

2. JDBC

3. Operaciones con bases de datos

4. Clases interesantes (JDBC)

5. Singleton Pattern

6. Transacciones

2

1. Bases de datos relacionales

3

1. Bases de datos relacionales

- Sistema gestor de bases de datos
- Base de datos relacional
- Diagrama Entidad – Relación (DER)
- Lenguaje estructurado de consultas (SQL)
 - Lenguaje de definición de datos (DDL)
 - Lenguaje de manipulación de datos (DML)
 - Consultas (*queries*)
 - SELECT
 - Modificaciones (*updates*)
 - INSERT INTO
 - UPDATE
 - DELETE FROM

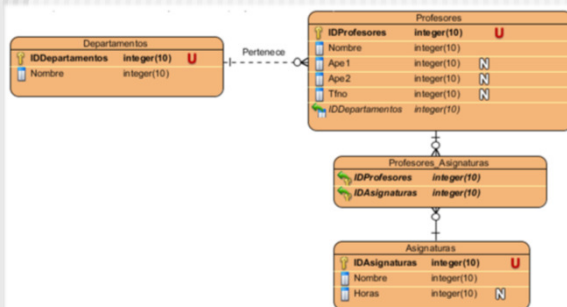
4

1. Bases de datos relacionales

- Base de datos relacional
 - Tabla
 - Campo
 - Clave primaria (*primary key*)
 - Clave foránea (*foreign key*)
 - Registro
 - Relaciones
 - 1 – N : Migración de la clave primaria (de 1 a N)
 - N – N : Nueva tabla para la relación y migración de ambas claves
 - 1 – 1 : Migración de una clave primaria
 - Restricciones de integridad (*constraints*)
 - Clave única
 - Integridad referencial

5

1. Bases de datos relacionales



6

1. Bases de datos relacionales

IDDEPARTAMENTOS	NOMBRE
1	Informática
2	Administración

IDSIGNATURAS	NOMBRE	HORAS
1	Programación	8
2	Formación laboral	3
3	Sistemas operativos	6

IDPROFESORES	NOMBRE	APE1	APE2	TFNO	IDDEPARTAMENTOS
1	Sergio	San Victoriano	Huertas	666666666	1
2	Sandra	López	Jiménez	655555555	1
3	Luis	Gómez	Lares	699999999	2

IDPROFESORES	IDSIGNATURAS
1	1
2	3
3	2

7

1. Bases de datos relacionales

- `select * from departamentos`
- `select nombre, apel, tfno from profesores`
- `select nombre, apel from profesores
where iddepartamentos=1 and nombre like 'Se%'`
- `select p.nombre, d.nombre
from profesores p, departamentos d
where p.iddepartamentos=d.iddepartamentos`

IDDEPARTAMENTOS	NOMBRE
1	Informática
2	Administración

NOMBRE	APE1	TFNO
Sergio	San Victoriano	666666666
Sandra	López	655555555
Luis	Gómez	699999999

NOMBRE	APE1
Sergio	San Victoriano

NOMBRE	NOMBRE
Sergio	Informática
Sandra	Informática
Luis	Administración

8

1. Bases de datos relacionales

- `insert into departamentos (nombre)
values ('Geografía')`
- `update departamentos
set nombre='Geografía e Historia'
where iddepartamentos=22`
- `delete from departamentos
where iddepartamentos=22`

IDDEPARTAMENTOS	NOMBRE
1	Informática
2	Administración
22	Geografía

IDDEPARTAMENTOS	NOMBRE
1	Informática
2	Administración
22	Geografía e Historia

IDDEPARTAMENTOS	NOMBRE
1	Informática
2	Administración

9

2. JDBC

10

2. JDBC

- *Java DataBase Connectivity*
- API para conectar programas Java con BDs
- Ubicada en el paquete java.sql
 - Contiene clases e interfaces para
 - Ejecutar consultas y actualizaciones
 - Invocar procedimientos y funciones
 - Acceder a los recursos de la base de datos
- Ofrece interfaces que definen CÓMO
 - Establecer la conexión con la base de datos
 - Ejecutar sentencias (consultas y actualizaciones)
 - Liberar la conexión
 - ...

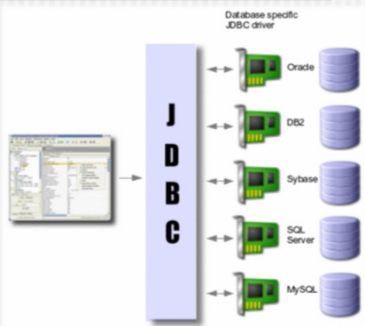
11

2. JDBC

- Driver JDBC
 - Clases que implementan las interfaces JDBC
 - Paquetes ofrecidos por los fabricantes de SGBD
 - Normalmente como parte del propio producto
 - Oracle, MySQL, PostgreSQL, DB2, HSQL, ...
 - Consiguen que JDBC funcione para cualquier SGBD
 - Mismo programa funciona para distintos SGBDs
 - Sólo hay que usar y cargar el driver correspondiente
- Programa Java con acceso a BD por JDBC
 - Cargar el driver
 - Establecer la conexión
 - Ejecutar la consulta e iterar y mostrar resultados
 - Cerrar la conexión

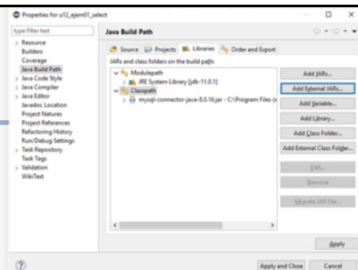
12

2. JDBC



13

2. JDBC



- Necesario añadir el jar del driver
 - En el momento de la ejecución
 - Añadirlo manualmente
 - Añadirlo con el IDE usado (Eclipse, ...)
 - Botón derecho sobre el proyecto
 - Build Path | Configure Build Path | Classpath | Add External JARs
 - Seleccionar el jar del driver a usar (proporcionado por el SGBD)
 - C:\Program Files (x86)\MySQL\Connector J 8.0\mysql-connector-java-8.0.16.jar

14

2. JDBC

- Pasos genéricos de manejo
 1. Establecimiento de parámetros de la conexión
 2. Carga (levantamiento) del driver a memoria
 3. Establecimiento de la conexión con la BD
 4. Preparación y ejecución de la consulta SQL
 5. Recorrido del conjunto de filas obtenido
 6. Cierre de los recursos empleados

15

2. JDBC

1. Establecimiento de parámetros de la conexión

- Único código dependiente del SGBD y de la BD
- Se indica
 - Nombre del driver JDBC que se empleará
 - URL de acceso a la base de datos
 - Localizador: SGBD, servidor, puerto
 - Usuario con el que se accederá a la base de datos
 - Contraseña del usuario con que se accederá a la base de datos

```
// Parámetros de la conexión
String driver = "com.mysql.cj.jdbc.Driver";
String database = "Instituto";
String hostname = "localhost";
String port = "3306";
String url = "jdbc:mysql://" + hostname + ":" + port + "/" + database + "?useSSL=false&serverTimezone=UTC";
String usr = "root";
String pwd = "1234";
```

16

2. JDBC

2. Carga (levantamiento) del driver a memoria

```
try{
    // Se levanta el driver
    Class.forName(driver);
}catch(ClassNotFoundException e){
    e.printStackTrace();
}
```

3. Establecimiento de la conexión con la BD

- Se proporciona la URL, usuario y contraseña
- Se requiere captura de SQLException
- Obtención de objeto Connection

```
// Se establece la conexión con la base de datos
con = DriverManager.getConnection(url, usr, pwd);
```

17

2. JDBC

4. Preparación y ejecución de la consulta SQL

- Definición de la cadena de consulta (String)

```
// Se define la consulta
String sql = "SELECT IDDepartamentos, nombre FROM Departamentos";
```

- Generación de sentencia SQL
 - A partir de la conexión con la BD
 - Obtención de objeto PreparedStatement

```
// Se prepara la sentencia a ejecutar
pstm = con.prepareStatement(sql);
```

- Ejecución de la consulta SQL
 - A partir de la sentencia SQL
 - Obtención de objeto ResultSet

```
// Se ejecuta la sentencia
rs = pstm.executeQuery();
```

18

2. JDBC

5. Recorrido del conjunto de filas obtenido

- Uso de métodos de ResultSet
 - next
 - Obtención del siguiente registro (fila)
 - Devuelve true si existía un siguiente registro
- getInt, getFloat, getString, getDate, ...
 - Obtención de un campo a partir de su nombre o posición
 - Uso de un método u otro atendiendo a su tipo de datos

```
// Se itera por los resultados
while(rs.next()){
    // Se muestran los campos del registro actual
    System.out.println("Departamento de ID " + rs.getInt("IDDepartamentos")
    }
    + " y nombre " + rs.getString("nombre"));
```

19

2. JDBC

6. Cierre de los recursos empleados

- Cierre del resultado, la sentencia SQL y la conexión
- Cierre en orden inverso al orden de apertura

```
try{
    // Se cierran los recursos
    if(rs != null)    rs.close();
    if(pstm != null) pstm.close();
    if(con != null)  con.close();
}catch(Exception e){
    e.printStackTrace();
    throw new RuntimeException(e);
}
```

20

2. JDBC

```
try{
    // Se levanta el driver
    Class.forName(driver);
}catch(ClassNotFoundException e){
    e.printStackTrace();
}
try{
    // Se establece la conexión con la base de datos
    con = DriverManager.getConnection(url, usr, pwd);
    // Se define la consulta
    String sql = "SELECT IDDepartamentos, nombre FROM Departamentos";
    // Se prepara la sentencia a ejecutar
    pstm = con.prepareStatement(sql);
    // Se ejecuta la sentencia y se obtienen los resultados
    rs = pstm.executeQuery();
    // Se itera por los resultados
    while(rs.next()){
        // Se muestran los campos del registro actual
        System.out.println("Departamento de ID " + rs.getInt("IDDepartamentos")
    }
}catch(SQLException e){
    e.printStackTrace();
    throw new RuntimeException(e);
}finally{
    try{
        // Se cierran los recursos en orden inverso a su creación
        if(rs != null)    rs.close();
        if(pstm != null) pstm.close();
        if(con != null)  con.close();
    }catch(Exception e){
        e.printStackTrace();
        throw new RuntimeException(e);
    }
}
```

21

3. Operaciones con BDs

1. Select
2. Select con Join
3. Insert
4. Update
5. Delete
6. Procedimiento almacenado (update)
7. Procedimiento almacenado (select)

22

3. Operaciones con BDs

■ SELECT

- Levantar driver JDBC
 - `Class.forName("com.mysql.cj.jdbc.Driver");`
- Establecer conexión con la BD
 - `Connection con=DriverManager.getConnection(url,usr,pwd);`
- Preparar sentencia SQL a ejecutar
 - `PreparedStatement pstmt = con.prepareStatement(sql);`
- Ejecutar la consulta y recoger resultados
 - `ResultSet rs = pstmt.executeQuery();`
- Recorrer registros de los resultados
 - `while(rs.next()) { rs.getXXX(); }`
- Cerrar resultado, sentencia y conexión
 - `rs.close(); pstmt.close(); con.close();`

23

select

```
// Parámetros de la conexión
String driver = "com.mysql.cj.jdbc.Driver";
String database = "Instituto";
String hostname = "localhost";
String port = "3306";
String url = "jdbc:mysql://" + hostname + ":" + port + "/" + database + "?useSSL=false&serverTimezone=UTC";
String usr = "root";
String pwd = "1234";

Connection con = null;
PreparedStatement pstmt = null;
ResultSet rs = null;

try {
    Class.forName(driver);
} catch (ClassNotFoundException e) {
    e.printStackTrace();
}

try {
    con = DriverManager.getConnection(url, usr, pwd);
    String sql = "SELECT IDDepartamento, nombre FROM Departamentos";
    pstmt = con.prepareStatement(sql);
    rs = pstmt.executeQuery();
    while(rs.next()) {
        System.out.println("Departamento : " + rs.getString("nombre"));
    }
} catch (SQLException e) {
    e.printStackTrace();
    throw new RuntimeException(e);
} finally {
    try {
        if(rs != null) rs.close();
        if(pstmt != null) pstmt.close();
        if(con != null) con.close();
    } catch (Exception e) {
        e.printStackTrace();
        throw new RuntimeException(e);
    }
}
```

24

3. Operaciones con BDs

- **SELECT** (con JOIN de tablas)
 - Análogo al SELECT normal
 - Precaución con nombres de campo repetidos
 - Establecer alias para evitar conflictos

```
SELECT p.nombre AS nomProf, p.apel, d.nombre AS nomDepto
FROM Profesores p, Departamentos d
WHERE p.IDDepartamentos=d.IDDepartamentos
```

25

select (con Join)

```
// Parámetros de la conexión
String driver = "com.mysql.cj.jdbc.Driver";
String database = "Instituto";
String hostname = "localhost";
String port = "3306";
String url = "jdbc:mysql://" + hostname + ":" + port + "/" + database + "?useSSL=false&serverTimezone=UTC";
String usr = "root";
String pwd = "1234";

Connection con = null;
PreparedStatement pstmt = null;
ResultSet rs = null;

try {
    Class.forName(driver);
} catch (ClassNotFoundException e) {
    e.printStackTrace();
}

try {
    con = DriverManager.getConnection(url, usr, pwd);
    String sql = "SELECT p.nombre AS nomProf, p.apel, d.nombre AS nomDepto ";
    sql += "FROM Profesores p, Departamentos d ";
    sql += "WHERE p.IDDepartamentos=d.IDDepartamentos";
    pstmt = con.prepareStatement(sql);
    rs = pstmt.executeQuery();
    while (rs.next()) {
        System.out.println(rs.getString("nomProf") + " de " + rs.getString("nomDepto"));
    }
} catch (SQLException e) {
    e.printStackTrace();
} finally {
    try {
        if (rs != null) rs.close();
        if (pstmt != null) pstmt.close();
        if (con != null) con.close();
    } catch (Exception e) {
        e.printStackTrace();
        throw new RuntimeException(e);
    }
}
```

26

3. Operaciones con BDs

- **INSERT**
 - Levantar driver JDBC
 - `Class.forName("com.mysql.cj.jdbc.Driver");`
 - Establecer conexión con la BD
 - `Connection con=DriverManager.getConnection(url,usr,pwd);`
 - Preparar sentencia SQL a ejecutar (parametrizar)
 - `PreparedStatement pstmt = con.prepareStatement(sql);`
 - Dar valor a los parámetros de la sentencia SQL
 - `pstmt.setXXX(param, valorParam);`
 - Ejecutar la consulta y comprobar resultado
 - `int resultado = pstmt.executeUpdate();`
 - Cerrar sentencia y conexión
 - `pstmt.close(); con.close();`

27

insert

```
// Parámetros de la conexión
String driver = "com.mysql.cj.jdbc.Driver";
String database = "instituto";
String hostname = "localhost";
String port = "3306";
String url = "jdbc:mysql://" + hostname + ":" + port + "/" + database + "?useSSL=false&serverTimezone=UTC";
String usr = "root";
String pwd = "1234";

Connection con = null;
PreparedStatement pstmt = null;

try{
    Class.forName(driver);
}catch(ClassNotFoundException e){
    e.printStackTrace();
}

try{
    con = DriverManager.getConnection(url, usr, pwd);
    String sql = "INSERT INTO Asignaturas (Nombre, Horas) ";
    sql += "VALUES (?, ?)";
    pstmt = con.prepareStatement(sql);
    pstmt.setString(1, "Entornos de Desarrollo");
    pstmt.setInt(2, 3);
    int resultado = pstmt.executeUpdate();
    if(resultado == 1)
        System.out.println("1 fila insertada correctamente");
    else
        throw new RuntimeException("No se pudo insertar la fila");
}catch(SQLException e){
    e.printStackTrace();
    throw new RuntimeException(e);
}finally{
    try{
        if(pstmt != null) pstmt.close();
        if(con != null) con.close();
    }catch(Exception e){
        e.printStackTrace();
        throw new RuntimeException(e);
    }
}
```

28

3. Operaciones con BDs

■ UPDATE

- Levantar driver JDBC
 - `Class.forName("com.mysql.cj.jdbc.Driver");`
- Establecer conexión con la BD
 - `Connection con=DriverManager.getConnection(url,usr,pwd);`
- Preparar sentencia SQL a ejecutar (parametrizar)
 - `PreparedStatement pstmt = con.prepareStatement(sql);`
- Dar valor a los parámetros de la sentencia SQL
 - `pstmt.setXXX(param, valorParam);`
- Ejecutar la consulta y comprobar resultado
 - `int resultado = pstmt.executeUpdate();`
- Cerrar sentencia y conexión
 - `pstmt.close();` `con.close();`

29

update

```
// Parámetros de la conexión
String driver = "com.mysql.cj.jdbc.Driver";
String database = "instituto";
String hostname = "localhost";
String port = "3306";
String url = "jdbc:mysql://" + hostname + ":" + port + "/" + database + "?useSSL=false&serverTimezone=UTC";
String usr = "root";
String pwd = "1234";

Connection con = null;
PreparedStatement pstmt = null;

try{
    Class.forName(driver);
}catch(ClassNotFoundException e){
    e.printStackTrace();
}

try{
    con = DriverManager.getConnection(url, usr, pwd);
    String sql = "UPDATE Asignaturas SET Horas=? ";
    sql += "WHERE Nombre=?";
    pstmt = con.prepareStatement(sql);
    pstmt.setInt(1, 5);
    pstmt.setString(2, "Entornos de Desarrollo");
    int resultado = pstmt.executeUpdate();
    System.out.println(resultado + " filas actualizadas correctamente");
}catch(SQLException e){
    e.printStackTrace();
    throw new RuntimeException(e);
}finally{
    try{
        if(pstmt != null) pstmt.close();
        if(con != null) con.close();
    }catch(Exception e){
        e.printStackTrace();
        throw new RuntimeException(e);
    }
}
```

30

3. Operaciones con BDs

■ DELETE

- Levantar driver JDBC
 - `Class.forName("com.mysql.cj.jdbc.Driver");`
- Establecer conexión con la BD
 - `Connection con=DriverManager.getConnection(url,usr,pwd);`
- Preparar sentencia SQL a ejecutar (parametrizar)
 - `PreparedStatement pstmt = con.prepareStatement(sql);`
- Dar valor a los parámetros de la sentencia SQL
 - `pstmt.setXXX(param, valorParam);`
- Ejecutar la consulta y comprobar resultado
 - `int resultado = pstmt.executeUpdate();`
- Cerrar sentencia y conexión
 - `pstmt.close();` `con.close();`

31

delete

```
// Parámetros de la conexión
String driver = "com.mysql.cj.jdbc.Driver";
String database = "Instituto";
String hostname = "localhost";
String port = "3306";
String url = "jdbc:mysql://" + hostname + ":" + port + "/" + database + "?useSSL=false&serverTimezone=UTC";
String usr = "root";
String pwd = "1234";

Connection con = null;
PreparedStatement pstmt = null;
try{
    Class.forName(driver);
}catch(ClassNotFoundException e){
    e.printStackTrace();
}
try{
    con = DriverManager.getConnection(url, usr, pwd);
    String sql = "DELETE FROM Asignaturas ";
    sql += "WHERE Nombre=?";
    pstmt = con.prepareStatement(sql);
    pstmt.setString(1, "Entornos de Desarrollo");
    int resultado = pstmt.executeUpdate();
    System.out.println(resultado + " filas eliminadas correctamente");
}catch(SQLException e){
    e.printStackTrace();
    throw new RuntimeException(e);
}finally{
    try{
        if(pstmt != null) pstmt.close();
        if(con != null) con.close();
    }catch(Exception e){
        e.printStackTrace();
        throw new RuntimeException(e);
    }
}
```

32

3. Operaciones con BDs

■ Procedimiento almacenado (Actualización)

- Levantar driver JDBC
 - `Class.forName("com.mysql.cj.jdbc.Driver");`
- Establecer conexión con la BD
 - `Connection con=DriverManager.getConnection(url,usr,pwd);`
- Preparar sentencia de invocación (parametrizar)
 - `CallableStatement pstmt = con.prepareCall(sql);`
- Dar valor a los parámetros de la invocación
 - `pstmt.setXXX(param, valorParam);`
- Ejecutar la consulta
 - `pstmt.execute();`
- Cerrar sentencia y conexión
 - `pstmt.close();` `con.close();`

33

```
// Parámetros de la conexión
String driver = "com.mysql.cj.jdbc.Driver";
String database = "instituto";
String hostname = "localhost";
String port = "3306";
String url = "jdbc:mysql://" + hostname + ":" + port + "/" + database + "?useSSL=false&serverTimezone=UTC";
String usr = "root";
String pwd = "1234";
Connection con = null;
CallableStatement pstmt = null;
try{
    Class.forName(driver);
}catch(ClassNotFoundException e){
    e.printStackTrace();
}
try{
    con = DriverManager.getConnection(url, usr, pwd);
    pstmt = con.prepareCall("{ call PA_insertaAsignatura(?, ?) }");
    pstmt.setString("nomAsig", "Bases de datos");
    pstmt.setInt("horasAsig", 6);
    pstmt.executeUpdate();
    System.out.println("Fin del procedimiento.");
}catch(SQLException e){
    e.printStackTrace();
    throw new RuntimeException(e);
}finally{
    try{
        if(pstmt != null) pstmt.close();
        if(con != null) con.close();
    }catch(Exception e){
        e.printStackTrace();
        throw new RuntimeException(e);
    }
}
```

34

3. Operaciones con BDs

■ Procedimiento almacenado (Selección)

- Levantar driver JDBC
 - `Class.forName("com.mysql.cj.jdbc.Driver");`
- Establecer conexión con la BD
 - `Connection con=DriverManager.getConnection(url,usr,pwd);`
- Preparar sentencia de invocación (parametrizar)
 - `CallableStatement pstmt = con.prepareCall(sql);`
- Dar valor a los parámetros de la invocación
 - `pstmt.setXXX(param, valorParam);`
- Ejecutar la consulta
 - `pstmt.executeUpdate();`
- Recoger y recorrer resultados resultados
 - `ResultSet rs = pstmt.getResultSet();`
 - `while(rs.next()){ rs.getXXX(campo); }`
- Cerrar sentencia y conexión
 - `pstmt.close(); con.close();`

35

```
// Parámetros de la conexión
String driver = "com.mysql.cj.jdbc.Driver";
String database = "instituto";
String hostname = "localhost";
String port = "3306";
String url = "jdbc:mysql://" + hostname + ":" + port + "/" + database + "?useSSL=false&serverTimezone=UTC";
String usr = "root";
String pwd = "1234";
Connection con = null;
CallableStatement pstmt = null;
try{
    // Se levanta el driver
    Class.forName(driver);
}catch(ClassNotFoundException e){
    e.printStackTrace();
}
try{
    // Se establece la conexión con la base de datos
    con = DriverManager.getConnection(url, usr, pwd);

    // Se prepara la sentencia a ejecutar
    pstmt = con.prepareCall("{ call PA_consultaAsignatura(?) }");

    // Se ejecuta el procedimiento y recoge el resultado
    pstmt.executeUpdate();

    ResultSet rs = pstmt.getResultSet();

    // Se itera por los resultados
    while(rs.next()){
        // Se muestran los campos del registro actual
        System.out.println("Asignatura de nombre " + rs.getString(1) + " y " + rs.getString(2) + " horas.");
    }
}catch(SQLException e){
    e.printStackTrace();
    throw new RuntimeException(e);
}finally{
    // Se cierran los recursos en orden inverso a su creación
    try{
        if(pstmt != null) pstmt.close();
        if(con != null) con.close();
    }catch(Exception e){
        e.printStackTrace();
        throw new RuntimeException(e);
    }
}
```

36

4. Clases interesantes (JDBC)

1. DriverManager
2. Connection
3. PreparedStatement
4. CallableStatement
5. ResultSet
6. SQLException

37

4. Clases interesantes (JDBC)

■ DriverManager

- Clase de gestión del controlador de la BD
 - Cargado con `Class.forName(driver);`
- Métodos
 - `getConnection(url, user, password)`
 - Estático
 - Obtención de la conexión con una base de datos
 - Requiere url (para identificar la base de datos)
 - Requiere credenciales (usuario y contraseña)
 - Requiere captura de `SQLException`

38

4. Clases interesantes (JDBC)

■ Connection

- Clase de gestión de la conexión a la BD
 - Objeto obtenido con
 - `DriverManager.getConnection(url, usr, passwd)`
- Métodos
 - `prepareStatement(sql)`
 - Recibe cadena SQL de consulta o actualización
 - Devuelve un `PreparedStatement`
 - Para ejecución de sentencias SQL
 - `prepareCall(sql)`
 - Recibe cadena SQL de invocación a procedimiento
 - Devuelve un `CallableStatement`
 - Para ejecución de procedimientos
 - `close()`
 - Cierre de la conexión

39

4. Clases interesantes (JDBC)

■ PreparedStatement

- Gestión de sentencias de consulta o actualización
 - Obtenido con `prepareStatement` de `Connection`
- Métodos
 - `executeQuery()`
 - Ejecuta la consulta y devuelve los resultados (`ResultSet`)
 - `executeUpdate()`
 - Ejecuta la actualización e indica nº de registros afectados
 - `setXXX(param, valorParam)`
 - Coloca valores de consulta parametrizada
 - `setInt, setString, ...`
 - A partir del nombre de un parámetro
 - A partir del índice de un parámetro
 - `close()`
 - Cierre de la sentencia

40

4. Clases interesantes (JDBC)

■ CallableStatement

- Gestión de invocaciones a procedimientos
 - Obtenido con `prepareCall` de `Connection`
- Métodos
 - `setXXX(param, valorParam)`
 - Coloca valores de invocación parametrizada del procedimiento
 - `setInt, setString, ...`
 - A partir del nombre de un parámetro o de su índice
 - `execute()`
 - Invocación del procedimiento asociado a la sentencia
 - `getResultSet()`
 - Obtiene el conjunto de registros de la consulta (`ResultSet`)
 - `close()`
 - Cierre de la sentencia

41

4. Clases interesantes (JDBC)

■ ResultSet

- Representa la "tabla" de resultados de la consulta
 - Obtenido a partir de `executeQuery` de `PreparedStatement`
 - Obtenido a partir de `getObject` de `CallableStatement`
- Métodos
 - `next()`
 - Determina si hay un siguiente registro (true) o no (false)
 - `getXXX(campo)`
 - `getInt, getString, ...`
 - Obtiene el valor de un campo del registro actual, identificado por
 - Nombre ó posición que ocupa
 - `wasNull()`
 - Determina si se leyó un valor nulo (true) o no (false)
 - `close()`
 - Cierre del conjunto actual de resultados

42

4. Clases interesantes (JDBC)

■ SQLException

- Excepciones ocurridas al manejar BDs
- Ubicada en java.sql
- Métodos interesantes
 - `getSQLState`
 - Describe el error según las convenciones XOPEN
 - `getMessage`
 - Recoge la información del error proporcionada por el driver
 - `getErrorCode`
 - Devuelve el código del error ocurrido
 - `getNextException`
 - Permite ver la siguiente excepción ocurrida
 - Útil en transacciones y operaciones complejas

43

5. Singleton pattern

44

5. Singleton pattern

■ "Patrón simple"

- Patrón de diseño
 - Solución eficiente para resolver un tipo de problema
 - Generalmente aceptada por todos (estandarizada)
- Problema resuelto por el *patrón simple*
 - Obtención de una única instancia de una clase
 - Evitar varias instancias de la misma
 - Si aún no hay una instancia, instanciar
 - Si hay una instancia, tomar esa

45

5. Singleton pattern

- Aplicación al acceso a BBDDs
 - Una única conexión con la BBDD
 - Problema a resolver
 - Establecer cada conexión tiene un alto coste
 - Procesamiento, tráfico de red y tiempo
 - Muchos accesos a la BBDD por aplicación
 - Solución
 - Al requerir la conexión
 - Si no está instanciada, hacerlo (abrir conexión)
 - Si lo está, tomar esa instancia (no abrir otra conexión)
 - Al cerrar el programa
 - Cerrar la conexión, si está abierta

46

5. Singleton pattern

- Solución "*singleton pattern*"
 - Definir objeto que se desea único
 - Estático
 - Privado
 - Definir método de acceso al objeto
 - Si el objeto es null, instanciar y devolverlo
 - Si el objeto no es null, devolverlo

47

5. Singleton pattern

- Solución "*singleton pattern*"
 - En el caso de las conexiones a BBDD: cerrar
 - Justo antes de terminar el programa
 - Usar gancho (*hook*) similar a escuchador (*listener*)
 - Pendiente de la ocurrencia de eventos
 - Se indica el objeto que manejará el evento
 - Dicho objeto hará las acciones pertinentes ante el evento

```
Runtime.getRuntime().addShutdownHook(  
    new ObjetoManejador());
```

48

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.util.ResourceBundle;

public class UConnection {

    private static Connection con = null;

    public static Connection getConnection(){
        try{
            if(con == null){
                Runtime.getRuntime().addShutdownHook(new MISHutdownHook());

                ResourceBundle rb = ResourceBundle.getBundle("jdbc");
                String driver = rb.getString("driver");
                String url = rb.getString("url");
                String pwd = rb.getString("pwd");
                String usr = rb.getString("usr");

                Class.forName(driver);
                con = DriverManager.getConnection(url, usr, pwd);
            }
            return con;
        }catch(Exception e){
            e.printStackTrace();
            throw new RuntimeException("Error al crear la conexión", e);
        }
    }

    static class MISHutdownHook extends Thread{
        public void run(){
            try{
                con = UConnection.getConnection();
                con.close();
            }catch(Exception e){
                e.printStackTrace();
                throw new RuntimeException(e);
            }
        }
    }
}
```

49

6. Transacciones

50

6. Transacciones

- Transacción
 - Conjunto atómico (indivisible) de acciones
 - O se ejecutan todas correctamente
 - O no se ejecuta ninguna
 - Resultados posibles
 - Estado de "no haber ejecutado ninguna acción"
 - Todas han finalizado sin errores
 - Estado de "haber ejecutado todas las acciones"
 - Alguna/s ha/n finalizado con errores

51

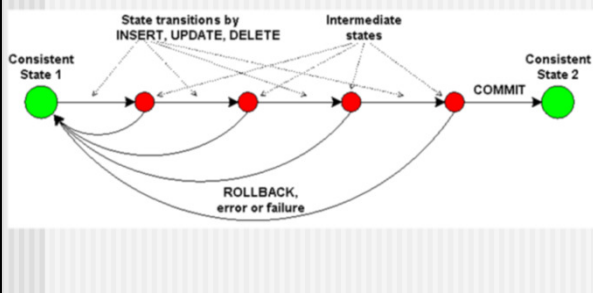
6. Transacciones

■ Manejo de transacciones

- Establecer operaciones que la forman
- Llevar a cabo las acciones
- Tras ejecutarlas
 - Confirmarlas si procede
 - *Commit*
 - Deshacerlas si procede
 - *Rollback*

52

6. Transacciones



53

6. Transacciones

■ Métodos de la interfaz `connection`

- `commit`
 - Confirma el éxito de las operaciones
 - Compromete su resultado
- `rollback`
 - Deshace las operaciones
 - Retorno al punto anterior a la primera
- `setAutoCommit`
 - Establece a `true` o `false` el `auto commit`
 - Para comprometer automáticamente cada operación
 - Por defecto vale `true`
 - Necesario a `false` para manejar transacciones
 - Comprometer manualmente con `commit`

54

Unidad 12

Gestión de bases de datos relacionales

Programación
1º D.A.M.
