U2
JDBC

Accés a dades DAM

Continguts

1. Java DataBase Connectivity
2. Esquema d'utilització
3. Objectes
4. Transaccions
5. Excepcions
6. Transferir dades entre classes

1. Java DataBase Connectivity

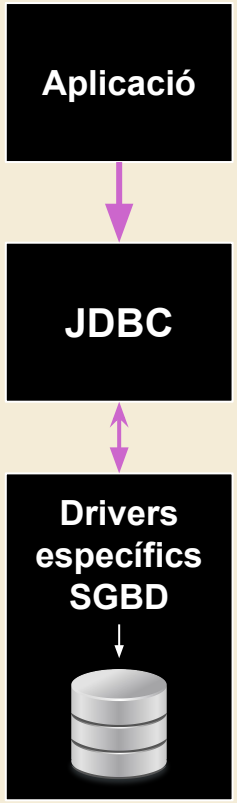
JDBC és una API de Java que permet connectar la nostra aplicació amb una base de dades.

Per a connectar-se a una base de dades JDBC necessita utilitzar un driver, un controlador, que normalment proporciona el mateix desenvolupador de la base de dades, a través del qual ens permetrà connectar-nos, executar consultes, instruccions DML i DDL, procediments emmagatzemats, ...

Per a poder connectar-no a una base de dades en concret haurem d'incloure el driver al classpath de l'aplicació. En NetBeans és suficient amb incloure el jar o la carpeta que conté el driver a les llibreries del projecte.

La nostra aplicació no haurà d'interactuar en cap moment amb el driver de la base de dades, només utilitzarà l'API JDBC.

Si canviam de base de dades, com a molt haurem de modificar alguna sentència SQL, però en general l'aplicació no s'haurà de modificar.



2. Esquema d'utilització

Per utilitzar aquesta api sempre seguirem la mateixa seqüència:

1. Instal·lar el controlador.
2. Obtenir una connexió amb la base de dades.
3. Crear una sentència amb l'SQL que volem executar i executar-la.
4. Si es tracta d'una consulta:
   - Tractar les distintes files que ens torna la consulta una a una.
5. Alliberar tots els recursos.

Totes les classe que utilitzarem seran dels paquets java.sql i javax.sql.

1. Instal·lar el controlador.

a) Encara que pugui semblar obvi, el primer que hem de fer és aconseguir el controlador jdbc per a la base de dades amb la que volem treballar. Normalment els podrem trobar a la web del desenvolupador de la base de dades.

b) Desam el controlador a un directori del nostre ordinador, si està comprimit el descomprimim. Normalment el controlador jdbc estarà dins un arxiu .jar.

c) Afegim el jar o el directori al *classpath* de l'aplicació. En Netbeans o eclipse serà suficient amb afegir el jar o la carpeta a les llibreries del projecte

2. Obtenir una connexió amb la base de dades.

Per obtenir una connexió hem d'utilitzar el mètode getConnection de la classe DriverManager. Ens tornarà un objecte de la classe Connection a partir del qual podrem treballar amb la base de dades.

```
Connection con = DriverManager.getConnection(url);
```

La url dependrà de cada DBMS, heu de consultar la documentació. Per exemple:

- MySQL: jdbc:mysql://<servidor>[:<port<]/<base de dades>?paràmetres

  ```
  jdbc:mysql://localhost:3306/aspirants?user=aidwc&password=contrasenya
  ```

- Oracle: jdbc:oracle:thin:@//[HOST][:PORT]/SERVICE

  ```
  jdbc:oracle:this:@//localhost:1521/Oracle?user=aidwc&password=contrasenya
  ```

Altres versions d'aquest mètode són:

```
Connection con = DriverManager.getConnection(url, user, password);
Connection con = DriverManager.getConnection(url, propietats);
```

on propietats és un objecte de la classe Properties, semblant a un HashMap on clau i valor són String.

3. Crear una sentència amb l'SQL que volem executar i executar-la.

Per executar una sentència SQL el primer que hem de fer és crear un objecte Statement a partir de la connexió establerta al punt anterior.

```
Statement st = con.createStatement();
```

El següent pas depèn del tipus de sentència SQL que volem executar. Si es tracta d'una consulta:

```
ResultSet rs = st.executeQuery("Select * from empleats");
```

Aquesta mètode executa el Select a la base de dades i ens torna un objecte ResultSet, un cursor. Si el que volem és executar una instrucció Insert, Delete, Update o una DDL:

```
int filesAfectades=st.executeUpdate("DELETE FROM empleats");
```

Ens torna el nombre de files afectades o zero si es tracta d'instruccions DDL com Create Table.

Les sentències SQL han de ser vàlides per el DBMS on l'executam, de forma que si canviam de DBMS segurament haurem de retocar algunes d'aquestes sentències.

4. Tractar les files tornades per la consulta.

Si hem executat una consulta SELECT haurem recollit el resultat dins un objecte ResultSet. Podem veure aquest objecte com un cursor de PL o un Iterator de Java. L'executeQuery torna el resultSet situat abans de la primera fila, exactament igual que un cursor. Per accedir a la següent fila hem d'executar el mètode next, que torna false quan ha arribat al final de la consulta.

```
while (rs.next()){ //executarà el while mentre quedin files.
```

Una vegada situat el resultSet a la fila, hem de recuperar les columnes una a una. Per fer-ho hem de saber el tipus de cada columna i el seu nom o la posició que ocupa dins el SELECT:

```
int codi = rs.getInt(1);
int codi = rs.getInt("codi");
String nom = rs.getString("nom");
```

5. Tractar les files tornades per la consulta.

En acabar d'utilitzar un objecte **ResultSet**, **Statement** o **Connection** l'hauríem de tancar per així lliberar els recursos que utilitzen.

Tots aquests objectes tenen un mètode close() que allibera els recursos,

```
if(rs!=null) rs.close();
if(st!=null) st.close();
if(con!=null) con.close();
```

En tancar un Statement es tancaran tots els ResultSet que en depenen.
En tancar una Connection es tancaran tots els Statement que en depenen.
Així i tot és millor tancar-los a ma un a un quan acabem d'utilitzar-los.

Les classes Connection, Statement i Resultset implementen la **interfície Autocloseable**. Això vol dir que es poden utilitzar amb el try-with-resources.

3. Objectes - Connection

Aquest objecte representa la connexió amb la base de dades. Ens permet entre d'altres:

Crear objectes com

- Statement,
- PreparedStatement
- CallableStatement, ...

Controlar les transaccions:

- setAutocommit: estableix si es fa un commit després de cada instrucció o no. Per defecte està posat a true.
- commit: realitza un commit
- rollback: realitza un rollback, complet o fins un savePoint.
- savePoint: estableix un savePoint amb nom o sense, setSavePoint, releaseSavePoint.

3. Objectes - Statement

Aquest objecte representa una sentència SQL. Es crea a partir de Connection:

- Statement st=con.createStatement()
- Statement st=con.createStatement(resultSetType, resultSetConcurrency)

- **resultSetType**:
  - TYPE_FORWARD_ONLY: El ResultSet només es podrà moure per endavant. Per defecte.
  - TYPE_SCROLL_INSENSITIVE: EL ResultSet es podrà moure endavant i endarrere, però no reflecteix els canvis que hi pugui haver a la base de dades.
  - TYPE_SCROLL_SENSITIVE: Aquest si que reflectirà els canvis.
- **resultSetConcurrency**: determina si es podran modificar les dades a partir del resultSet.
  - CONCUR_READ_ONLY. Per defecte.
  - CONCUR_UPDATABLE

Per a executar sentències SQL a la base de dades, Statement ens proporciona els següents mètodes:

- **executeQuery(“select”):** Executa consultes Select. Torna un objecte ResultSet amb el qual podem recorrer els reusltats.
- **executeUpdate(“insert”**): Permet executar qualsevol tipus de sentència SQL que no sigui select. Torna un sencer que indica les files afectades si es tracta d'un insert, update o delete, o zero si és una altra sentència.
- **addBatch(“sentència”)**: afegeix la sentència a un buffer per a ser executada més tard. Normalment s'utilitza amb inserts o updates, quan hem de fer moltes instruccions seguides, per carregar dades, per exemple. Dona millor rendiment que fer els executeUpdate un a un.
- **executeBatch()**: executa totes les instruccions del buffer. Torna un array de sencers amb les files afectades per a cada sentència.

3. Objectes - ResultSet

Representa el cursor tornat per un mètode Statement.executeQuery. Les seves característiques dependran en bona mesura dels paràmetres que s'hagin utilitzat en crear l'Statement.

Inicialment el cursor està posicionat abans de la primera fila. Una bona manera de recórrer el cursor és:

```
while(rs.next()){
      int codi=rs.getInt("identificador");
      ...
}
```

rs.next() torna fals en arribar al final del cursor. Per a cada fila hem de recuperar els distints camps amb un mètode getXxx() on Xxx és el tipus de la columna. Podem recuperar-lo posant el nom de la columna o la posició que ocupa dins la llista de columnes tornades per la consulta(perillós si es modifica aquesta llista).

Per cursors que es poden recórrer en els dos sentits tenim el mètode *previous*

Altres mètodes són:

- Posició del cursor:
  - isBeforeFirst, isFirst, isAfterLast, isLast indiquen si el cursor es troba respectivament abans de la primera fila, a la primera, després de la darrera, a la darrera.
  - first, last, absolute(index), relative(x positiu o negatiu): moven el cursor a la primera, darrera, fila index o x files envant o enrere si x és negatiu.
- Modificació del cursor
  - **updateXxx(index o nom, valor)** modifica la columna indicada amb el valor.
  - **updateRow()**: actualitza la fila a la base de dades amb els valors que hem canviat amb updateXxx.
- Inserció de files al cursor:
  - **moveToInsertRow():** Col·loca el cursor a la fila d'inserció. Els seus camps es modifiquen amb updateXxx
  - **insertRow()**: Inserta la fila a la base de dades.
  - **moveToCurrentRow()**: torna a la posició anterior del cursor.

3. Objectes - PreparedStatement

Si hem de repetir la mateixa sentència SQL diverses vegades amb valors diferents podem utilitzar aquest objecte. La base de dades pre-compilarà la sentència i per cada execució només haurà de canviar els paràmetres, amb la qual cosa el rendiment serà millor.

En definir la cadena que conté la sentència SQL hem de posar un ? on anirà cada un dels paràmetres:

```
String actualitzaSalaris = "update empleats set sou= ? where codiEmpleat= ?";
```

Llavors cream l'objecte a partir de Connection:

```
PreparedStatement sous = con.prepareStatement (actualitzaSalaris);
```

En aquest moment ja tenim la sentència pre-compilada al DMBS. Per a utilitzar-la hem d'assignar valors als paràmetres:

```
sous.setDouble(1,956.32); //Substituirà el primer ? de la cadena per 956.32
sous.setInt(2, 2345); //Substituirà el segon ? De la cadena per 2345
int n=sous.executeUpdate(); //Executa la sentència amb els valors subministrats.
```

4. Transaccions

D'una manera informal podríem definir una transacció com un conjunt de sentències SQL que s'han d'executar o totes o cap. Per exemple, per fer una transferència bancaria hem de restar doblers al que paga i sumar-ne al que cobra. No pot ser que només en facem una de les dues.

Per defecte JDBC fa un commit després de cada sentència. Per evitar-ho hem de canviar:

```
con.setAutoCommit(false);
```

Llavors podrem utilitzar:

```
con.commit(); //guardar la transacció
con.rollback(); //tornar enrere la transacció
```

També podem utilitzar savepoints:

```
SavePoint punt1=con.setSavePoint();
con.rollback(punt1);
con.releaseSavePoint(punt1); //anul·la el punt
```

5. Excepctions. SQLExceptions

Quan JDBC troba una errada durant la seva interacció amb el DBMS llança una excepció del tipus SQLException que conté:

- La descripció de l'error. Es pot recuperar amb `SQLException.getMessage();`
- SQLStateCode. Una cadena de cinc caràcters alfanumèrics que han estat estanditzats. Es recupera amb `SQLException.getSQLState();`
- El codi d'error. Depèn de la implementació del driver i hauria de ser el mateix que mostra el DBMS, el típic ORA-8234. Es recupera amb `SQLExceptioin.getErrorCode();`

Warnings

Un warning és una errada no lo suficientment greu per aturar el programa, alerten l'usuari que alguna cosa no ha anat tot lo bé que hauria d'haver anat. Els objectes Connection, Statement, ResultSet, ... poden tornar warnings. Els recuperam

```
SQLWarning warning=stmt.getWarning(); // recupera el primer
warning=warning.getNextWarning(); //Recupera el següent o null
```

6. Transferir dades entre classes

Seguint les característiques d'encapsulació dels llenguatges orientats a objectes, quan una aplicació Java ha d'accedir a una base de dades sol delegar tot el relacionat amb ella a una o una sèrie de classes, que es dediquen exclusivament a interactuar amb la base de dades, mentre que el tractament de dades, la interacció amb l'usuari, ... es deixa a altres classes.

Per a transferir la informació entre les classes de bases de dades i les altres es solen crear objectes que només tenen els atributs necessaris, els getters i setters (no sempre) d'aquests atributs i un constructor complet. Els mètodes que interactuen amb la base de dades solen tornar, o rebre com a paràmetres, objectes o col·leccions d'objectes d'aquest tipus.

Encara que no siguin exactament el mateix, podem dir que els següents noms fan referència al mateix concepte:

- **VO** (Value Object), **DO** (DataObject): en principi són immutables, no canvien.
- **DTO** (Data Transfer Object): pensat per transferir dades entre aplicacions, són serialitzables.
- **POJO** (Plain Old Java Object): qualsevol objecte Java sense cap requeriment especial: implementació d'interfícies, descendència d'una superclasse concreta, ...

Un exemple de classe d'aquest tipus podria ser la següent:

La classe Localitat té tres atributs, un constructor que rep com a paràmetres els valors d'aquests atributs, i els getters necessaris per recuperar els seus valors.

Com que es suposa que les dades no es modificaran, no s'han programat els setters.

La classe tampoc disposa de cap mètode addicional, el seu únic objectiu és mantenir la informació de l'objecte.

```
public class Localitat {

    private String codiLocalitat;
    private String nomLocalitat;
    private String codiIlla;

    public Localitat(String codiLocalitat,
           String nomLocalitat, String codiIlla) {
        this.codiLocalitat = codiLocalitat;
        this.nomLocalitat = nomLocalitat.replace("\'",
"''");
        this.codiIlla=codiIlla;
    }

    public String getNomLocalitat() {
        return nomLocalitat;
    }

    public String getCodiLocalitat() {
        return codiLocalitat;
    }

    public String getCodiIlla() {
        return codiIlla;
    }
}
```