Programació

Java DataBase Connectivity

JAVA DATA BASE CONNECTIVITY

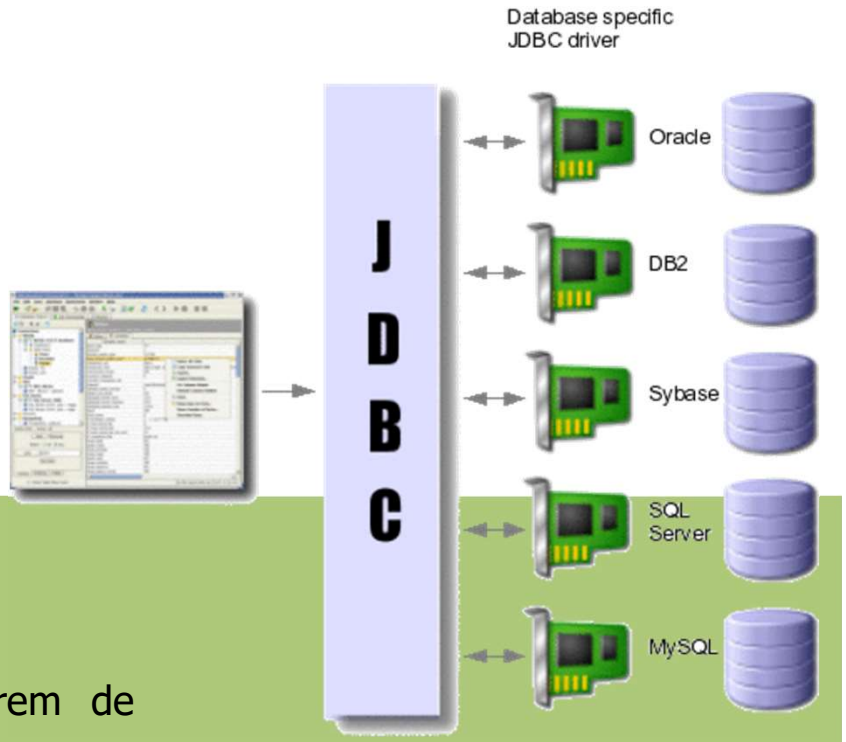
JDBC és una API de Java que permet connectar la nostra aplicació amb una base de dades.

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

ESQUEMA D'UTILITZACIÓ

Per utilitzar aquesta api sempre seguirem la mateixa seqüència:

1. Instal·lar el controlador.
2. Carregar el controlador. A partir de la versió 4.0 de JDBC aquest pas ja no és necessari.
3. Obtenir una connexió amb la base de dades.
4. Crear una sentència amb l'SQL que volem executar i executar-la.
5. Si es tracta d'una consulta:
    1. Tractar les distintes files que ens torna la consulta una a una.
6. Alliberar tots els recursos.

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

1. Instal·lar el controlador

1 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.

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

3 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. Carregar el controlador.

En les darreres versions de jdbc aquest pas ja no és necessari, però per si de cas vos toca barallar-vos amb aplicacions que utilitzin les versions més antigues ho explicarem.

El primer que heu de fer és consultar la documentació del controlador per a saber quin és el nom de la classe del driver, el nom complet incloent el paquet.

Llavors executarem la següent instrucció:

```
Class.forName("nom_driver_jdbc");
```

Per exemple, per utilitzar una base de dades MySQL:

```
Class.forName("com.mysql.jdbc.Driver");
```

I per utilitzar una base de dades Oracle:

```
Class.forName("oracle.jdbc.driver.OracleDriver");
```

El major inconvenient que té aquest mètode és que si el desenvolupador del controlador decideix canviar el nom de la classe haurem de modificar, recompilar i redistribuir la nostra aplicació. Per això actualment aquest pas no és necessari i després d'instal·lar el controlador podem passar directament al següent.

3. 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.

4. 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'executem, de forma que si canviem de DBMS segurament haurem de retocar algunes d'aquestes sentències.

5. 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");
```

6. Tancar els recursos oberts.

En acabar d'utilitzar un objecte ResultSet, Statement o Connection l'hauríem de tancar per així alliberar 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.

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.

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.
```

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 fem 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
```

SQLEXCEPTION

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 estandarditzats. 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 recuperem

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