Tema 13

Java amb bases de dades

1 Introducció

En este tema vorem com podem accedir des d'una aplicació Java a una base de dades. Concretament a una en MySQL però podria ser amb qualsevol altra.

Podrem crear bases de dades, taules, modificar-les, etc. I també podrem introduir dades, modificar-les i esborrar-les.

I per últim vorem com tractar les consultes a una base de dades.

2 Instal·lació

2.1 Instal·lar el gestor de BD MySQL

Per a poder treballar amb BD, és necessari instal·lar un gestor de BD, com pot ser MySQL, Postgres, Informix, Oracle, etc. Nosaltres instal·larem el MySQL. També instal·larem el MySQL Workbench per a treballar amb la BD en mode gràfic. Ací està resumit (per a més informació, consulteu l'annex vist en l'assignatura de BD).

a) Instal·lar MySQL

En Windows cal executar el paquet que baixarem de:

www.mysql.com/downloads/mysql

Exemple d'ús:

```
show databases;
create database proves;
use proves;
create table ... ;
select ... ;
show tables;
quit;
```

Les bd es guarden en:

Windows → c:\mysql\data

Linux → /var/lib/mysql

Exportar BD: mysqldump –u (usuari) –p (password) (nombd) > fitxer.sql

b) Instal·lar MySQL Workbench

b.1) Abaixar l'aplicació i instal·lar: http://dev.mysql.com/downloads/workbench

b.2) Crear una connexió a una BD:

Connection: Accés a la BD proves (un nom arreu)

Hostname: localhost (si la BD està al nostre ordinador)

Schema: proves (si no posem res, accedirem a totes les BD del host)

b.3) Entrar al MySQL Workbench:

Triar una connexió creada

Posar usuari (root o res) i password (root o res).

2.2 Instal·lar el driver de MySQL per a Java

Per a poder connectar Java amb una BD (MySQL en este exemple), necessitem el driver o connector de la BD. El driver és una classe per a fer d'enllaç entre la nostra aplicació i la base de dades.

Java (JDK) no inclou tots els drivers per a connectar-se amb els distints motors de BD del mercat. Per això caldrà abaixar-los d'Internet. Netbeans sí que l'inclou, però Eclispse no.

a) Abaixar el driver

Este driver se sol abaixar de la pàgina web del gestor de la BD. Per exemple, el de MySQL es pot abaixar de:

www.mysql.com/downloads/connector/j/

Haurem abaixat fitxer:

mysql-connector-java-5.1.X.tar.gz (últim disponible)

b) Extraure el fitxer corresponent

Del fitxer abaixat hem d'extraure el següent fitxer, que és el driver:

mysql-connector-java-5.1.X-bin.jar

c) Copiar el fitxer al diretori adequat

Per a que eixa classe estiga accessible per als nostres projectes Java, hem de copiar eixe jar en la carpeta <DIRECTORI_DE_JAVA>/jre/lib/ext.

Windows: C:\Archivos de programa\Java\jdk1.6.0_06\jre\lib\ext

Linux: /usr/lib/jvm/java-6-sun-1.6.0.16/jre/lib/ext

OS/X: /Library/Java/JavaVirtualMachines/jdk1.8.0_31.jdk /Contents/Home/jre/lib/ext

En eixa carpeta és on s'afegeixen els jar addicionals. Així ja els tindrem disponibles per a quan necessitem utilitzar-los altra vegada. No obstant això, si no funcionara, seguim els passos d'ac:

http://www.codigofantasma.com/blog/conectar-java-con-mysql-usando-jdbc/

3 Utilització

Una volta feta tota la instal·lació necessària, anem a utilitzar les BD des d'una aplicació Java en Netbeans.

3.1 Importar el paquet necessari

Al principi de la nostra aplicació importarem el paquet java.sql:

```
import java.sql.*;
```

Però només caldria importar 3 classes d'eixe paquet:

Connection	Per a fer la connexió a la bd
PreparedStatement	Per a preparar la sentència SQL
ResultSet	Per a guardar el resultat de la consulta SQL

3.2 Carregar el driver del gestor de BD

```
Class.forName("com.mysql.jdbc.Driver");          // Per a MySql
Class.forName("oracle.jdbc.Driver.OracleDriver"); // Per a Oracle
```

3.3 Establir la connexió amb la BD

```
String login = "";
String password = "";
String bd = "proves";
String url = "jdbc:mysql://localhost/" + bd;
Connection conn = null;     // la posarem global per a ser usada a les funcions
conn = DriverManager.getConnection(url, login, password);
```

La sintaxi de la url és:

- ✓ Per a MySQL: jdbc:mysql://servidor:port/nomBaseDades
- ✓ Per a Oracle: jdbc:oracle:thin:servidor:port:nomBaseDades

Si no indiquem el servidor, agafa el de per defecte: localhost. I si no indiquem el port, agafa el de per defecte (El de MySQL és 3306, el d'Oracle: 1521).

És a dir, per a accedir a la BD "proves" podríem fer-ho així:

```
conn = DriverManager.getConnection("jdbc:mysql://localhost/proves", "", "");
```

3.4 Fer les consultes a la BD

Es pot accedir a la BD de 2 formes: accés directe i amb preparció de la sentència.

<table>
<tr><th rowspan="2">TIPUS DE SENTÈNCIA</th><th colspan="2">TIPUS D'ACCÉS</th></tr>
<tr><th>DIRECTE</th><th>AMB PREPARACIÓ DE LA SENTÈNCIA</th></tr>
<tr><td>CREATE
INSERT
UPDATE
DELETE</td><td>Statement s = conn.createStatement();

int i = s.executeUpdate(sentència);</td><td>PreparedStatement ps =
conn.prepareStatement(sentència);
int i = ps.executeUpdate();</td></tr>
<tr><td rowspan="2">SELECT</td><td>Statement s = conn.createStatement();

ResultSet rs = s.executeQuery(sentència);</td><td>PreparedStatement ps =
conn.prepareStatement(sentència);
ResultSet rs = ps.executeQuery();</td></tr>
<tr><td colspan="2">while (rs.next()) {
System.out.println(rs.getFloat(1)); // 1a columna (la 0 no existeix)
System.out.println(rs.getInt("edat")); // columna "edat"
}</td></tr>
</table>

La forma de fer-ho amb l'accés directe és més ràpid si anem a executar la sentència només 1 vegada, però si està dins d'un bucle, és mes ràpid fer-ho amb preparació de la sentència: abans del bucle es fa el *prepare*; dins, l'*execute*. Ho vorem millor en el punt de sentències SQL parametritzades.

En el quadre anterior, ***sentència*** és un String on estarà la instrucció insert o delete o select, etc.

El mètode ***executeUpdate()*** retorna la quantitat de files afectades per la *where*.

La classe ***ResultSet*** és com una matriu (files x columnes) que haurem de recórrer amb el moviment d'un cursor.

El mètode ***rs.next()*** fixa el cursor a la primera fila de resultats. I cada volta que fem altre *rs.next()* avança el cursor una fila més. Retorna un booleà indicant si queden més files. Els mètodes ***get__()*** retornen el valor de les columnes de la select: *getString()* per a les cadenes de text, *getDouble()* per als decimals, *getDate()* per a les dates, etc.

3.5 Tancar la connexió

```
rs.close();
s.close();
conn.close();
```

Quan ja no anem a gastar la BD, hem de tancar la connexió. També és convenient tancar el ResultSet i el Statement, i en l'ordre invers a la creació.

3.6 Tractament d'excepcions de bases de dades

Les instruccions anteriors poden donar error si no existeix la BD, si la sentència no està ben construïda, si intentem inserim provocant duplicitat de clau primària, etc. Per tant, cal tractar eixes excepcions amb blocs try-catch:

INSTRUCCIÓ DE BD	EXCEPCIÓ GENERADA
Class.forName(...)	ClassNotFoundException
DriverManager.getConnection(..., "", "")	SQLException
createStatement() prepareStatement(...)	
executeUpdate() executeQuery()	
next() get__()	
close()	

Per a assegurar-nos que, passe el que passe, es tanca la connexió, convé fer-ho al bloc finally:

```
try {

        ...

}
catch (SQLExceptcion e) {
        System.out.println("Error d'accés a la BD, etc")
}
finally {
        try {
            if (rs != null) rs.close();
            if (s != null) s.close();
            if (conn != null) conn.close();
        }
        catch (SQLException e) {
            System.out.println("Error tancant la connexió: " + e);
        }
}
```

3.7 Exemple

```
import java.io.*;
import java.sql.*;
public static void main (String[] args) {
        Connection conn;
        PreparedStatement ps;
        ResultSet rs;

        // ----- CARREGUEM EL DRIVER
        try {
                Class.forName("com.mysql.jdbc.Driver");
        }
        catch ( ClassNotFoundException e) {System.out.println("No s'ha trobat el driver");}

        // ----- ESTABLIM LA CONNEXIÓ
        try {
                conn = DriverManager.getConnection("jdbc:mysql://localhost/proves", "", "");
        }
        catch ( SQLException e) {System.out.println("No s'ha fet la connexió");}

        try {
                // ----- PREPAREM LA CONSULTA
                 ps = conn.prepareStatement("SELECT * FROM amics");

                // ----- EXECUTEM LA CONSULTA
                rs = ps.executeQuery();

                // ----- ARREPLEGUEM ELS VALORS DE LA CONSULTA
                while (rs.next()) {
                        // Podem accedir a la columna amb el seu nom (edat, nom...)
                        int i = rs.getInt("edat");
                        System.out.print("\t" + i);
                        // O Podem fer-ho amb l'índex de la columna (1, 2...)
                        String str = rs.getString(1);
                        System.out.println("\t" + str);
                }
        }
        catch (SQLException e) {System.out.println("Error en la sentència SQL");}
        finally { conn.close(); }
}
```

Nota: el close() no es faria si alguna excepció d'un try anterior no ha pogut ser capturat. Per tant, seria millor posar els 3 try dins d'un altre i, en eixe altre, posar el finally amb el close();

Exercicis

1. Amb MySQL crea la base de dades "proves" amb les següents taules:

   CLIENTS = codi + nom + edat + deute

   EMPLEATS = codi + nom + sou + cap
   C. Ali: cap → EMPLEATS(codi)

2. Fes en Java el programa provesBD que faça el següent:

   - Funció quantsClients que retorne quants clients hi ha.
   - Funció deuteTotal que retorne el deute total de tots els clients.
   - Funció souMig que retorne la mitja dels sous de tots els empleats.
   - Funció mostrarClients, que se li passa com a paràmetre el codi del client i mostrarà, en fila, totes les dades del client. Si el codi del client és -1 es mostraran les dades de tots els clients, ordenats pel codi. Posa també una capçalera a la llista. Retornarà quants clients s'han mostrat.
   - Funció mostrarClients, que se li passa com a paràmetre el nom del client i mostrarà, en fila, totes les dades de tots els clients que es diguen així. Posa també una capçalera a la llista. Retornarà quants clients s'han mostrat.
   - Funcions mostrarEmpleats, anàlogues a les de mostrarClients.
   - Funció insClient que demane nom, edat i deute i inserisca eixe client. Tin en compte que el camp codi és clau. Hauràs de vore quin és el major dels codis guardats i inserir el nou client amb el número següent. Retornarà un booleà indicant si ha anat bé l'operació o no.
   - Funció insEmpleat que demane nom i sou. Després mostrarà els empleats (amb la funció corresponent) i demanarà el codi del cap del nou empleat. Si el codi no és correcte, avisarà. Si no, mostrarà les dades d'eixe cap i preguntarà si està segur. En cas afirmatiu, inserirà el nou empleat amb el codi del cap corresponent. Retornarà un booleà indicant si ha anat bé l'operació o no.

- Funció incrPercSou que li passes un codi d'empleat i un percentatge i incrementarà en eixe percentatge el sou de l'empleat. Si el codi de l'empleat és -1, ho farà amb tots els empleats. Retornarà la quantitat d'empleats afectats.

- Funció esborrarClient que li passes com a paràmetre un codi de client. Si és -1 caldrà demanar el codi del client a esborrar (mostrant prèviament els clients). Si no existeix el codi mostrarà error. Si no, mostrarà les dades del client a esborrar. Si està segur d'esborrar, ho esborrarà. La funció retornarà un booleà indicant si s'ha esborrat o no.

- Programa principal:

    - Pregunta per teclat el nom de la base de dades. Si no existeix mostrarà l'avís corresponent. Si sí, l'obrirà i mostrarà "Connexió a la bd tal".

    - Fes un bucle amb un menú amb les opcions anteriors, més l'opció d'eixir, que haurà de tancar la connexió amb la base de dades i mostarà "Desconnexió de la bd tal".

3. Modifica el joc de rol per a guardar les dades en una base de dades i poder seguir la partida en un altre moment.

3.8 Recorregut pel ResultSet

Mètodes de la classe ResultSet per a moure'ns pel conjunt de resultats:

Mètode	Posa el cursor...
next()	En la posició següent.
previous()	en la posició anterior.
first()	en la primera posició
last()	en l'última posició
relative(n)	n posicions més que l'actual (o menys si n és negatiu)

Altres mètodes de la classe ResultSet

Mètode	Descripció
getRow()	Retorna el número de fila del ResultSet. La primera és la 1. Si no apunta a cap fila, retorna el 0.
getInt(col) getFloat(col) get...	Retornen el valor del ResultSet de la columna col en la fila actual. La columna es pot indicar per un número o pel nom de la columna.

Per a obtindre la quantitat de files d'un resultSet:

```
rs.last();        // Posiciona el cursor a l'última fila del rs
System.out.println( rs.getRow() );  // Retorna el número de fila
```

Ara bé, si després volem recórrer el ResultSet, haurem de fer:

```
rs.first();       // Posiciona el cursor a la primera fila del rs
```

... i no posar el next com a primera instrucció dins del bucle. És a dir:

```
...
rs.last();
System.out.println( "Hi ha " + rs.getRow() + " files:");
rs.first();
do {
        ...  // Recorrem el resultSet
} while (rs.next());
```

4 Sentències SQL parametritzades

Podem tindre una sentència preparada amb la possibilitat de passar-li paràmetres en el moment de l'execució de la sentència.

S'utilitza en aquells casos que anem a executar diverses vegades una mateixa sentència però en diferents paràmetres. Així només la prepararem una vegada i, per tant, podrem guanyar en velocitat de procés.

Exemple:

```
// PREPAREM LA SENTÈNCIA PARAMETRITZADA

PreparedStatement ps;
ps = conn.prepareStatement("insert into amics values (?, ?)");
do {
        imprimir("Nom: "); String nom = llegirCadena();
        imprimir("Edat: "); int edat = llegirEnter();
        // PASSEM ELS PARÀMETRES
        ps.setString(1, nom);        // Li passem el 1r paràmetre
        ps.setInt(2, edat);          // Li passem el 2n paràmetre
        // EXECUTEM LA SENTÈNCIA (COM SEMPRE)
        ps.execeUpdate();   // O bé ps.executeQuery() si fóra un SELECT.
} while (...);
```

Altres llocs parametritzables:

```
UPDATE amics
        SET edat = edat + ?
        WHERE edat > ?

DELETE FROM amics
        WHERE edat = ?

SELECT *
        FROM amics
        WHERE edat > ?
```

5 DDL des de Java

Les sentències SQL es poden dividir en:

DML: Llenguatge de Manipulació de Dades: SELECT, INSERT, UPDATE, DELETE

DDL: Llenguatge de Definició de dades: CREATE, ALTER, DROP

Als apartats anteriors hem vist com executar sentències DML des de Java. Ara vorem com executar les de DDL:

<table>
<tr><th colspan="3">CREACIÓ D'UNA BASE DE DADES</th></tr>
<tr><th>PASSOS</th><th colspan="2">INSTRUCCIONS</th></tr>
<tr><td>Obrir connexió (a cap BD)</td><td colspan="2">conn = DriverManager.getConnection(... cap BD ...);</td></tr>
<tr><td>Crear BD</td><td colspan="2">Statement s = conn.createStatement();
s.executeUpdate("CREATE DATABASE nova");</td></tr>
<tr><td>Seleccionar BD</td><td>//Amb la mateixa connexió

s.executeUpdate("USE nova");</td><td>//Amb una altra connexió a la nova BD
conn = Driv...getConnection(... BD nova ...);
s = conn.createStatement();</td></tr>
<tr><td>Crear taules, esborrar-les...</td><td colspan="2">s.executeUpdate("CREATE TABLE..."); (*)</td></tr>
<tr><td>Tancar connexió</td><td colspan="2">conn.close();</td></tr>
</table>

És a dir, en Java, per a crear una base de dades (o esborrar-la), seleccionar-la i crear taules es fa amb el mètode executeUpdate.

(*) És necessari que la sentència que volem executar (s) ha d'estar creada a partir d'una BD. Per això cal tornar a fer el createStatement (per a la nova conn).

Exercicis

4. Modifica les funcions mostrarClient, insClient, esborrarClient, incrPercSou que has fet anteriorment per a que treballen amb sentències SQL parametritzades

5. Fes funcions per a posar en pràctica el DDL des de Java.

6 Metadades

6.1 Metadades d'una connexió

Treballarem amb la classe **DatabaseMetaData**.

Per a obtindre les BD:

```
DatabaseMetaData dbmd = conn.getMetaData();
ResultSet rs = dbmd.getCatalogs();

while ( rs.next() )
{
   System.out.println(rs.getObject(1));     // Nom del catàleg (BD)
}
```

> Altra forma d'obtindre les BD:
>
> ```
> Statement s = conn.createStatement();
> ResultSet rs = s.executeQuery("show databases");
> ```

Per a obtindre les taules:

```
DatabaseMetaData dbmd = conn.getMetaData();
ResultSet rs = dbmd.getTables("proves", null, null, null);
while ( rs.next() ) {
        String nomCataleg  = rs.getObject(1) ;
        String nomEsquema = rs.getObject(2) ;
        String nomTaula    = rs.getObject(3) ;
        ...
}
```

Els paràmetres de getTables() són:

Catàleg	(Si no s'indica, mostra les taules de la BD actual)
Esquema/es	(Admet el comodí %)
Taula/es	(Admet el comodí %)
Tipus	(Vector de tipus. Veieu l'API)

El resultat serà un ResultSet amb una fila per cada taula i fins a 10 columnes. La tercera té el nom de la taula. Les altres s'indiquen a l'API de Java:

http://download.oracle.com/javase/6/docs/api/java/sql/DatabaseMetaData.html

> Altra forma d'obtindre les taules:
>
> ```
> Statement s = conn.createStatement();
> ResultSet rs = s.executeQuery("show tables");
> ```

Per a obtindre les columnes:

```
DatabaseMetaData dbmd = conn.getMetaData();
ResultSet rs = dbmd.getColumns("proves", null, "amics", null);
while ( rs.next() )
{
        String nomCataleg   = rs.getObject(1) ;
        String nomEsquema = rs.getObject(2) ;
        String nomTaula     = rs.getObject(3) ;
        String nomColumna = rs.getObject(4) ;
        String tipusColumna = rs.getObject(6) ;
        ...
}
```

Els paràmetres de *getColumns()* són els mateixos que en *getTables()*;

El resultat serà un ResultSet amb una fila per cada columna i fins a 23 columnes. La 4a té el nom de la columna i la 6a el tipus. Les altres s'indiquen a l'API de Java.

6.2 Metadades d'una consulta

Treballarem amb la classe **ResultSetMetaData**.

Per a obtindre la quantitat de columnes i els noms i tipus:

```
ResultSet rs = st.executeQuery("select * from amics");
ResultSetMetaData rsmd = rs.getMetaData();
int QColumnes = rsmd.getColumnCount();
for(int i=1; i<=Qcolumnes; i++){
      String nomColumna = rsmd.getColumnName(i) ;
      String tipusColumna= rsmd.getColumnTypeName(i) ;
       ...
}
```

Exercicis

6. Crea la classe Connexio:

Atributs:

```
Connection conn;
String motor;        // "MySQL", "Oracle"
String bd;
```

Mètodes (pensa si algun és estàtic):

driver() : dóna a triar entre MySQL i Oracle i carregarà el driver.
driver(String motor) : espera "MySQL" o "Oracle i el carrega.
obri(String host, String bd, String usuari, String password)
obri() : pregunta per teclat les 4 coses per a obrir la connexió.
Els constructors (amb i sense paràmetres) han d'obrir la connexió.
tanca() : tanca la connexió.
mostraBasesDeDades(): mostra les BD existents.
mostraTaules(): mostra les taules de la BD actual.
mostraTaules(String bd): mostra les taules de la BD del paràmetre.
mostraColumnes(String taula): mostra les columnes de la taula.
mostraSelect(String taula): mostra el select *
getSelect(String taula): retorna el ResultSet del select *
getCount(String taula): retorna el count(*)
Fes altres funcions mostraSelect, getSelect i getCount passant també el paràmetre where, de tipus String.
Exemple de crida:

```
c.mostraSelect("amics", "edat>=18");
```

getBD(): retorna el nom de la BD actual
existeix(String bd, String taula, String col): retorna un booleà dient si existeix la columna de la taula de la bd parametritzada.
Possibles crides:

```
c.existeix("proves", "", "");
c.existeix("proves", "amics", "");
c.existeix("proves, "amics", "edat");
```

7 Transaccions

7.1 Definició i ús

Una transacció és un conjunt de sentències que constitueixen una operació única. És a dir: s'executen totes les sentències de la transacció o cap.

Imagina't que estàs fent un programa on t'hages d'assegurar que es fan 3 inserts o, si ha hagut algun error, que no se'n faça cap. JDBC permet agrupar instruccions SQL en una sola transacció.

El control de la transacció és realitzat per l'objecte Connection. Quan una connexió es crea, per defecte és en mode auto-commit. Això significa que cada instrucció individual SQL es tracta com una transacció en ella mateixa, i es comprometrà (es guardarà en la BD) en el mateix moment. Per això, per a iniciar una transacció de diverses sentències SQL, haurem de posar el mode auto-commit a false. Quan volem acabar la transacció, per a que es guarden els canvis en la BD haurem de cridar al mètode commit(). Per contra, si volem cancelar els canvis, cridarem al mètode rollback():

```
try {
        conn.setAutoCommit(false);          <-- Inici de la transacció
        Statement  s = conn.createStatement();
        s.executeUpdate(...);
        s.execcuteUpdate(...)
        conn.commit();                      <-- Fi de la transacció
} catch (SQLException e) {
        conn.rollback();                    <-- Avortament de la transacció
        System.out.println("La transacció ha fallat");
} finally{
        conn.setAutoCommit(true);
}
```

Nota: també podem crear *savepoints* i fer *rollbacks* a ells. Mireu els mètodes *setSavepoint* i *rollback* de la classe Connection.

7.2 Nivells d'aïllament

Què passa si mentre estem en una transacció llegim valors canviats a la base de dades per altra transacció? Java permet establir diferents efectes de lectura. Veiem primer quins són eixos efectes i després vorem com establir-los (modes d'aïllament).

7.2.1 Efectes de lectura

Hi ha 3 efectes de lectura diferents quan una transacció llig dades que podria haver canviat una altra transacció.

Lectura bruta: es dóna quan en una transacció llegim el valor modificat per una altra transacció que encara no està guardat.

Transacció A	Transacció B
`SELECT edad FROM usuaris WHERE id = 1;` `// llegirà 20`	
	`UPDATE usuaris SET edat = 21 WHERE id = 1;` `// No es fa commit`
`SELECT edad FROM usuaris WHERE id = 1;` `// llegirà 21`	
	`ROLLBACK;`

Lectura no repetible: es dóna quan en una transacció es llig una fila 2 voltes però els valors no coincideixen.

Transacció A	Transacció B
`SELECT * FROM usuaris WHERE id = 1;` `// llegirà 20`	
	`UPDATE usuaris SET edat = 21 WHERE id = 1;` `COMMIT;`
`SELECT * FROM usuaris WHERE id = 1;` `// llegirà 21`	

Lectura fantasma: es dóna quan en una transacció es fan 2 consultes idèntiques i s'obtenen resultats diferents.

Transacció A	Transacció B

`SELECT * FROM usuaris`
`WHERE edat BETWEEN 10 AND 30;`
`// llegirà n usuaris`	

	`INSERT INTO usuaris VALUES ( 3, 'Pepet', 27 );` `COMMIT;`

`SELECT * FROM usuaris`
`WHERE edat BETWEEN 10 AND 30;`
`// llegirà n+1 usuaris`	

7.2.2 Nivells d'aïllament

Dins d'una transacció hi ha diferents nivells d'aïllament respecte a altres transaccions per a permetre un efecte de lectura o altre. Podem seleccionar el nivell amb el mètode *setTransactionIsolation* de la interfície *Connection*.

```
connexio.setTransactionIsolation(nivell);
```

Com a paràmetre li passarem el nivell que volem mitjançant les constants predefinides en eixa interfície. Estes són les constants que indiquen el nivell d'aïllament, de menys restrictiu a més restrictiu:

		EFECTES DE LECTURA		
		LECTURA BRUTA	LECTURA NO REPETIBLE	LECTURA FANTASMA
CONSTANTS DELS NIVELLS D'AÏLLAMENT	TRANSACTION_READ_UNCOMMITED	pot ocórrer	pot ocórrer	pot ocórrer
	TRANSACTION_READ_COMMITED	-	pot ocórrer	pot ocórrer
	TRANSACTION_REPEATABLE_READ	-	-	pot ocórrer
	TRANSACTION_SERIALIZABLE	-	-	-

Hi ha una altra constant que podem passar-li a eixe mètode, que és TRANSACTION_NONE, per a indicar que no es podran usar transaccions.

Exemple:

```
connexio.setAutoCommit(false);

connexio.setTransactionIsolation(Connection.TRANSACTION_SERIALIZABLE);
```