JDBC

Leonardo De Boni

December 1, 2022

CONTENTS

1 OTTENERE UNA CONNESSIONE

1.1 Import

La classe da importare è:
    java.io.Connection

1.2 Aprire una connessione

Esistono vari metodi che restituiscono una connessione:

- DriverManager.getConnection(String url)
- DriverManager.getConnection(String url, String user,String pass)
- DriverManager.getConnection(String url, Properties properties)

Formato dell'url:
    jdbc:subprotocol://host:porta/database
Eventualmente l'url può contenere anche username e password e nel nostro caso l'url per MariaDb diventa:
    jdbc:mariadb://localhost:3306/database?user=root&password=myPassword
Connessione:
    Connection cn = DriverManager.getConnection(String url);

1.3 Chiudere una connessione

Dopo avere eseguito le operazioni che mi interessano sulla connessione devo **sempre** chiudere la connessione:
    cn.close();

2 ECCEZIONI

Le connessioni sono delicate e possono generare errori e possono essere gestite tramite il costrutto try with resources:

```
try (...dichiarazione oggetti autoclosable...) {
...
}
```

dove all'interno delle parentesi tonde posso instanziare un oggetto che implementa **AutoClosable** come ad esempio nel nostro caso è **Connection**.

3 STATEMENT

3.1 Creare uno statement

Dopo aver creato una connessione come visto prima possono creare uno Statement a partire da essa:

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

3.2 Eseguire codice sql

Per eseguire codice sql posso usare:

```
st.executeUpdate(String sql);
```

che posso usare per operazioni DML + DDL (insert, update, delete, drop, create, ...) e restituisce un **int** corrispondente al numero di righe modificate

```
st.executeQuery(Stringsql);
```

che posso usare per operazioni DQL (select) e retituisce un oggetto di tipo **ResultSet**.

3.3 Eseguire codice sql generico

Quando non so che tipo di query devo eseguire (es: di update o ricerca) posso usare il metodo:

```
st.execute();
```

che ritorna un **boolean** equivalente a:

- true se il primo risultato è un **ResultSet** che posso ottenere con st.getResultSet()
- false se il primo risultato è il numero delle righe modificate che posso ottenere con st.getUpdateCount()

Per controllare se ci sono altri risultati uso st.getMoreResults(). So di avere finito i risultati perchè l'ultimo result sarà un count dal valore di -1. Codice di esempio.

3.4 Eseguire query multiple

//fare

4 RESULTSET

Quando ottengo un **ResultSet** posso muovermi su esso con un cursore che partirà dalla riga o (inesistente). Quindi ho vari modi per muovere il cursore ed ottenere i dati relativi alla sua posizione.

4.1 Estrarre dati dal ResultSet

Per iterare sul **ResultSet** ho a disposizione il seguente metodo:

```
boolean next();
```

Esempio:

```
while ( rs.next() ) {
...
}
```

Esistono getters per ottenere i campi. Per esempio per ottenere un campo String ho varie opzioni:

- rs.getString("nomeCampo")
- rs.getString(1)

Il primo funziona in base al nome del campo che voglio ottenere il secondo in base alla posizione del campo(sconsigliato).

4.2 Movimenti avanzati sul ResultSet

Oltre a next() ho a disposizione altri metodi per muovermi nel **ResultSet**:

- previous()
- first()
- last()
- absolute(numero)
- relative(numero)

4.3 Concorrenza

//fare

5 RESULTSETMETADATA

//fare

6 PREPAREDSTATEMENT

6.1 Creare un PreparedStatement

Il **PreparedStatement** serve per scrivere una query parametrica, ovvero una query che al posto dei dati che devo personalizzare ogni volta metto dei placeholder. Posso creare un **PreparedStatement** in vari modi:

- cn.prepareStatement(sql)
- cn.prepareStatement(sql,int): INSERT
- cn.prepareStatement(sql,int[]): INSERT
- cn.prepareStatement(sql,String[]): INSERT

- cn.prepareStatement(sql,int,int)

Esempio di query sql parametrica:
INSERT INTO autori (id, cognome, nome) VALUES (?,?,?)
Da notare l'utilizzo del carattere ? come placeholder. Quindi il codice completo per creare un **PreparedStatement** può essere:

```
String sql = "INSERT INTO autori (id, cognome, nome) VALUES (?,?,?)";
PreparedStatement ps = cn.prepareStatement(sql);
```

Ora come faccio a impostare un valore arbitrario al posto dei placeholder?

6.2 Operare su PreparedStatement

Per sostituire i placeholder con valori arbitrari ho a disposizione vari setters che accettano 2 argomenti di cui il primo contiene la posizione del placeholder che voglio sostituire e il secondo il valore che voglio inserire. Esempio:

- ps.setInt(posizione,valore);
- ps.setString(posizione,valore);

6.3 Eseguire un PreparedStatement

Una volta inseriti i dati al posto dei placeholder non ci resta che eseguire il **PreparedStatement** in modo tale da operare sul database:

```
ps.executeUpdate();
```

7 TRANSAZIONI

La transazione è un elenco di operazioni che vengono eseguite o **tutte o nessuna**. Di default la connessione è in modalità auto commit e in automatico dopo ogni operazione viene applicata al database in automatico. Posso disabilitare questa modalità e decidere io quando applicare le modifiche con:

```
cn.setAutoCommit(false);
```

Per applicare le modifiche al database devo quindi usare:

```
cn.commit();
```

Invece se voglio annullare le modifiche che ancora non ho applicato pur mantenendo la connessione uso:

```
cn.rollback();
```

8 ORM (MAPPING)