JDBC

JDBC (Java Database Connectivity) è un'interfaccia completamente Java utilizzata per eseguire istruzioni SQL sui database.

L'**API JDBC** si trova nel pacchetto java.sql; contiene poche classi concrete, è composta principalmente da interfacce indipendenti dal database.

Le **API JDBC** consentono di accedere a qualsiasi tipo di dati tabulari, in particolare ai dati memorizzati in database relazionali

- JDBC consente di scrivere applicazioni Java che gestiscono queste tre attività di programmazione:
  - Connettere un'origine dati (e.g., database)
  - Inviare query e istruzioni di aggiornamento per il database
  - Recuperare ed elaborare i risultati ricevuti

JDBC include:

- **API JDBC** - Un insieme di interfacce che fanno parte della piattaforma Java e costituiscono le API per il programmatore
- **JDBC Driver Manager** - Gestore di driver che permette a driver di terze parti di connettersi ad un DB specifico

Un driver JDBC permette di

- Connettersi ad un DB
- Inviare un comando SQL
- Processare il risultato

L'impiego di JDBC solitamente si articola attraverso quattro passi:

1. Per prima cosa, è necessario caricare il **driver** idoneo per l'utilizzo del particolare database che si intende sfruttare. Può essere caricato un apposito driver JDBC installato in precedenza nel sistema, oppure può essere sfruttato il ponte JDBC-ODBC. Non è importante il nome o il funzionamento interno del particolare driver selezionato: l'interfaccia di programmazione sarà sempre la medesima.

2. Si **apre una connessione** verso il particolare database necessario all'applicazione, sfruttando il driver caricato al passo precedente.

3. Si impiegano l'**interfaccia di JDBC** ed il linguaggio **SQL** per interagire con la base di dati. Generalmente, viene sottoposta al DBMS una query volta all'ottenimento di alcuni risultati.

4. I risultati ottenuti possono essere manipolati sfruttando le classi JDBC e del codice Java studiato per il compito.

```
private Connection con =null ;

private final String URL = "jdbc:mysql://localhost:3306/nomeDB";
private final String USER = "username";
private final String PASS = "password";

public Connection connetti() {
    try {
        if(this.con==null) {
        this.con= DriverManager.getConnection(URL, USER, PASS);
        System.out.println("Siamo connessi!");
        }

    } catch (SQLException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();

    }
    return this.con;
};
```

```
    private Connection con = null;
    private Statement statement = null;
    private PreparedStatement ps=null;
    private ResultSet rs = null;

    private Connessione c = new Connessione();

// Scorro il ResultSet usando il metodo next() e mostro i risultati.
//in questo caso riempio un ArrayList di libri e lo ritorno

        List<Libro> libri = new ArrayList<>();
        this.con = c.connetti();
        try {
            this.statement = this.con.createStatement();
        } catch (SQLException e1) {
            // TODO Auto-generated catch block
            e1.printStackTrace();
        }
        try {
            this.rs = this.statement.executeQuery(FIND_ALL);
            while(this.rs.next()) {
                Libro l = new Libro();
                l.setId(rs.getInt("id"));
                l.setPagine(rs.getInt("pagine"));
                l.setEditore_id(rs.getInt("editore_id"));
                l.setPrezzo(rs.getDouble("prezzo"));
//              l.setP_iva(rs.getDouble("p_iva"));
```

```
                l.setTitolo(rs.getString("titolo"));
                libri.add(l);


            }
        } catch (SQLException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }

        return libri;
```

---

connettersi ad un database

Da JDBC 4 non è più necessario caricare in memoria il driver corrispondente, questo è già disponibile.

Se necessario, la sintassi per effettuare l'operazione è la seguente:

```
Class.forName(stringa_driver);
```

- source

---

java.sql.Connection

A questo punto entrano in gioco

- l'interfaccia `java.sql.Connection`
- e la classe `java.sql.DriverManager`.

La prima descrive le funzionalità necessarie per entrare in comunicazione con uno specifico database, mentre `DriverManager` offre una serie di metodi statici, utili per stabilire qualsiasi tipo di connessione consentita dai driver `JDBC` già caricati in memoria.

Il modello generalmente osservato è il seguente:

```
    Connection conn = DriverManager.getConnection(url_database);
```

Una volta ottenuta una connessione attiva, diventa possibile sfruttare i metodi descritti da `Connection`. I più frequentemente utilizzati sono:

- `createStatement()` Crea e restituisce un oggetto java.sql.Statement, utile per interagire con il database mediante dei comandi SQL.
- `close()` Chiude la connessione.

L'interfaccia Statement

Il linguaggio SQL comprende istruzioni utili per interagire con una base di dati.

In particolare, mediante SQL è possibile compiere tre principali operazioni:

1. Eseguire selezioni e ricerche all'interno di una o più tabelle, con l'istruzione SELECT.
2. Modificare il contenuto di una tabella, con istruzioni come DELETE, INSERT e UPDATE.
3. Modificare la struttura del database, ad esempio con CREATE TABLE.

L'interfaccia `java.sql.Statement` comprende i metodi necessari per fornire al DBMS le istruzioni SQL appena descritte:

- `executeQuery()` commissiona le istruzioni di tipo SELECT.
- `executeUpdate()` commissiona le istruzioni di aggiornamento delle tabelle (DELETE, INSERT e UPDATE) e della base di dati (CREATE TABLE, INDEX e così via).

---

metodo executeQuery()

- `executeQuery()` restituisce sempre un oggetto che implementa l'interfaccia **java.sql.ResultSet**. Grazie ad essa è possibile prendere in esame i risultati restituiti dall'istruzione SQL di ricerca commissionata ad DBMS.

metodo executeUpdate()

- `executeUpdate()`, non ha risultati da restituire.
- ritorna un **intero** che riporta il numero delle righe coinvolte dall'esecuzione di istruzioni di tipo DELETE, INSERT e UPDATE.
- dove non c'è nulla da restituire, il valore di ritorno sarà 0 (zero).

L'interfaccia ResultSet

- L'interfaccia **java.sql.ResultSet** comprende i metodi indispensabili per scorrere l'insieme dei risultati restituiti da una query SQL.
- Il metodo **next()** scorre in avanti tale insieme.

in pratica

- Inizialmente, il cursore del corrente oggetto ResultSet sarà posizionato prima del primo dei record restituiti.

- In questa condizione, non è possibile svolgere operazioni di analisi dei risultati: nessun record è puntato dal cursore corrente.
- Una prima chiamata a next() farà in modo che il cursore venga spostato sul primo record restituito dalla query.
- Ogni volta che un record è puntato dal cursore, diventa possibile estrarne i contenuti.
- Quando non ci sono più record nel ResultSet, il metodo next() ritorna **false**
- Non conoscendo il numero di righe restituite dal db, un ResultSet solitamente viene passato in rassegna con un codice del tipo:

```
while (resultSet.next()) {
    // Esamina il record corrente.
}
```

Un ciclo di questo tipo termina non appena tutti i record restituiti dalla query eseguita sono stati passati in rassegna.

Leggere i valori del record corrente

Quando un record è correttamente puntato dal cursore, è possibile esaminare i suoi campi attraverso dei metodi che hanno tutti la forma: `getTipo(int indiceColonna)`

Ad esempio, si supponga di voler ottenere il contenuto del primo campo del record corrente, sotto forma di **stringa**:

```
String stringa = resultSet.getString(1);
```

Se si conoscono i **nomi** associati ai singoli campi del record, è possibile usare la variante: `getTipo(String nomeColonna)`

Ad esempio: `String nome = resultSet.getString("Nome");`

I metodi per leggere

Il seguente elenco riporta i metodi di questa famiglia più frequentemente utilizzati:

- `getBoolean()` Restituisce il campo specificato sotto forma di **boolean**.
- `getByte()` Restituisce il campo specificato sotto forma di **byte**.
- `getDate()` Restituisce il campo specificato sotto forma di **oggetto java.util.Date**.
- `getDouble()` Restituisce il campo specificato sotto forma di **double**.
- `getFloat()` Restituisce il campo specificato sotto forma di **float**.
- `getInt()` Restituisce il campo specificato sotto forma di **int**.
- `getLong()` Restituisce il campo specificato sotto forma di **long**.
- `getShort()` Restituisce il campo specificato sotto forma di **short**.
- `getString()` Restituisce il campo specificato sotto forma di **oggetto java.lang.String**.
