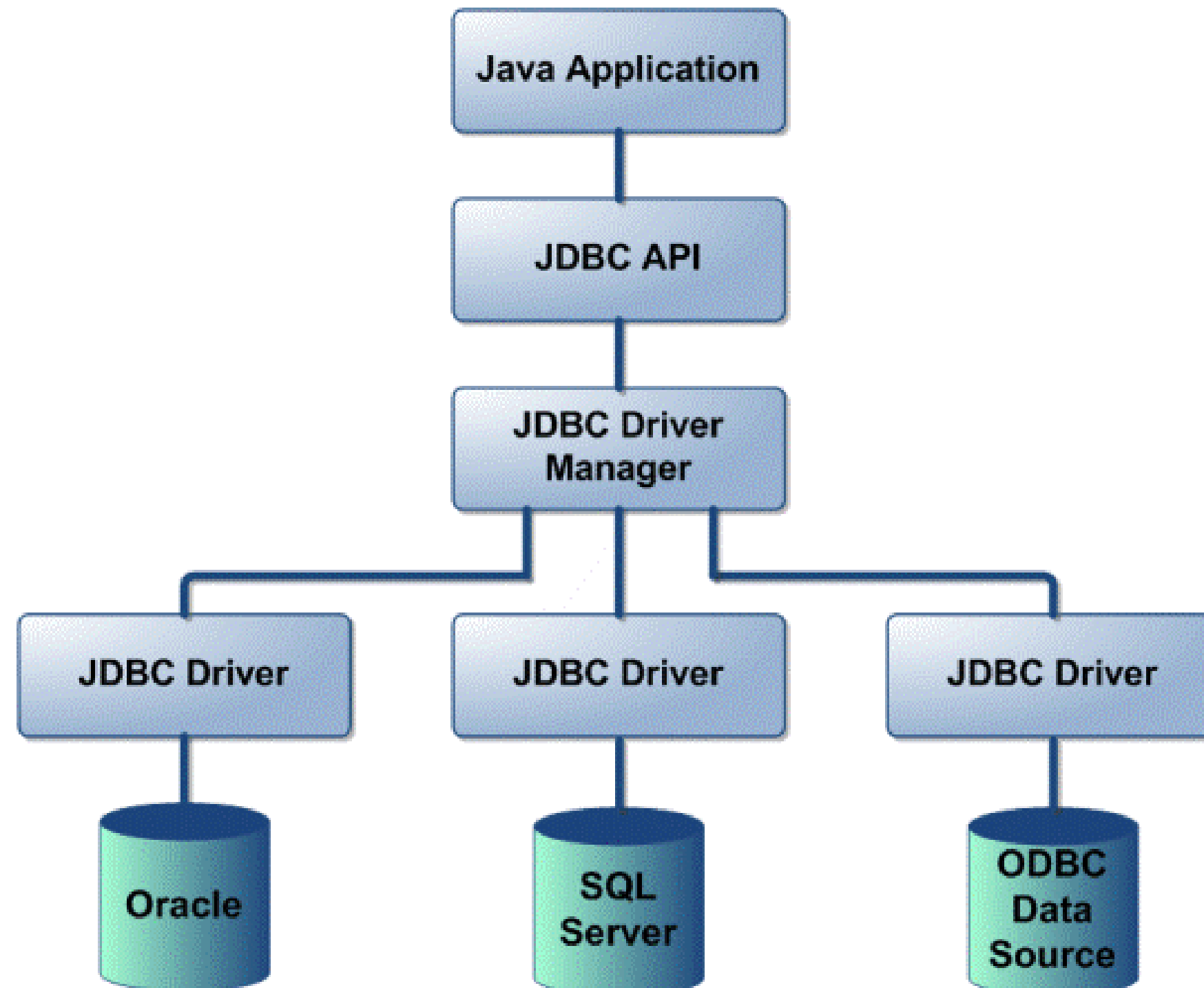


# JDBC



**NOTA: ODBC è un driver attraverso un'API standard per la connessione dal client al DBMS**

# JDBC: INTRODUZIONE

---

**In informatica JDBC (Java DataBase Connectivity), è un connettore e un driver per database che consente l'accesso e la gestione della persistenza dei dati sulle basi di dati da qualsiasi programma scritto con il linguaggio di programmazione Java, indipendentemente dal tipo di DBMS utilizzato.**

**È costituito da un'API object oriented orientata ai database relazionali, raggruppata nel package java.sql, che serve ai client per connettersi a un database fornendo i metodi per interrogare e modificare i dati.**

**L'architettura di JDBC, prevede l'utilizzo di un “driver manager”, che espone alle applicazioni un insieme di interfacce standard e si occupa di caricare a “run-time” i driver opportuni per “pilotare” gli specifici DBMS.**

**Le applicazioni Java utilizzano le "JDBC API" per parlare con il JDBC driver manager, mentre il driver manager usa le JDBC driver API per parlare con i singoli driver che pilotano i DBMS specifici.**

---

**L'API JDBC è costituita dai seguenti componenti principali:**

- **JDBC Drivers:** una raccolta di classi e interfacce per comunicare con il database;
- **Connections:** stabilisce una connessione con il database;
- **Statements:** dichiarazioni che servono per avviare le QUERY;
- **ResultSets:** gli elementi restituiti da una QUERY, La classe ResultSet fornisce un cursore che punta alla riga corrente nel result-set fornito dalla query;

**I passaggi per connettere il database utilizzando JDBC sono:**

- 1. Caricare il driver JDBC;**
  - 2. Connessione;**
  - 3. Statements;**
  - 4. Esecuzione statements;**
  - 5. Chiudere la connessione al database;**
-

**I driver da caricare cambiano in base al DBMS che utilizziamo. Vengono caricati a runTime grazie alla classe DriverManager con l'url di connessione al DB formattato secondo lo standard jdbc e varia in base al DBMS.**

**In MySQL l'url è: jdbc:mysql://localhost:3306/nomeSchema**

**Possiamo scomporlo in 3 punti:**

- 1. jdbc:mysql:// >> prefisso standard per le connessioni MySQL;**
- 2. localhost:3306 >> nome del server e porta che ospita il DB;**
- 3. nomeSchema >> nome dello schema a cui fare riferimento;**

**Una volta recuperato l'url di connessione è possibile effettuare la connessione al DB con le credenziali di accesso al DB. Questo ci permetterà di ricavare l'oggetto Connection grazie al DriverManager:**

```
Connection conn = DriverManager.getConnection(url,username,password);
```

**Per far funzionare la connessione abbiamo bisogno di importare il Driver "mysql-connector-j" per MySql nel progetto, questo permetterà al DriverManager di recuperare il driver necessario per la comunicazione con il DB.**

**Potrebbe essere necessario chiamare manualmente la registrazione del driver:**

- **String DB\_DRIVER = "com.mysql.jdbc.Driver";**
- **Class.forName(DB\_DRIVER);**

**Per utilizzare invece il DriverManager e Connection abbiamo bisogno di importare due librerie:**

- **import java.sql.Connection;**
  - **import java.sql.DriverManager;**
-

**Un esempio di sintassi con anche chiamata manuale del Driver:**

```
1.String DB_DRIVER = "com.mysql.jdbc.Driver";
2.String DB_URL = "jdbc:mysql://localhost:3306/world";
3.String DB_USERNAME = "root";
4.String DB_PASSWORD = "root";
5.Connection conn = null;
6.try {
7.    // Register the JDBC driver
8.    Class.forName(DB_DRIVER);
9.    // Open the connection
10.   conn = DriverManager.getConnection(DB_URL, DB_USERNAME, DB_PASSWORD);
11.   if (conn != null)
12.       System.out.println("Successfully connected.");
13.   else System.out.println("Failed to connect.");
14. } catch (Exception e) {
15.     e.printStackTrace(); }
```

**Le interfacce Statement definiscono i metodi e le proprietà che consentono di inviare comandi SQL e ricevere dati dal database. Ci sono tre tipi di statement:**

- **Statement** - non è in grado di accettare parametri;
- **PreparedStatement** - accetta parametri di input;
- **CallableStatement** - accetta parametri di input a runtime;

**La differenza principale tra Prepared e Callable è che PreparedStatement viene gestito in Java mentre CallableStatement viene recuperato dal database.**

**Quindi Statement viene usato per query fisse, statiche. Invece PreparedStatement ci permette di settare dei valori prima di eseguire la query.**

```
1.Statement stmt = conn.Statement ();  
2.ResultSet rs = stmt .executeQuery(QUERY);
```



**Una volta creato lo statement bisogna eseguirlo per recuperare un oggetto di ResultSet:**

```
1.PreparedStatement stmt = conn.prepareStatement(QUERY);  
2.ResultSet rs = stmt.executeQuery();
```

**Una volta recuperato il result-set è possibile accedervi come se fosse una matrice sfruttando un ciclo while per far avanzare il cursore del result-set su tutte le righe:**

```
1.while(rs.next())  
2.    {  
3.        System.out.println(rs.getString(1)); //First Column  
4.        System.out.println(rs.getString(2)); //Second Column  
5.        System.out.println(rs.getString(3)); //Third Column  
6.        System.out.println(rs.getString(4)); //Fourth Column  
7.    }
```



**È possibile anche indicare il nome della colonna nei getter dei valori per le colonne:**

```
1. System.out.println(rs.getString("column_name"));
```

**È possibile recuperare anche i METADATI dal result-set, ovvero tutte le informazioni riguardanti la struttura della tabella come ad esempio il tipo di una colonna o il numero di colonne o il nome della tabella ecc ecc...**

```
1. ResultSetMetaData metaData = rs.getMetaData();  
2. Integer columnCount = metaData.getColumnCount();
```

**Nell'esempio andiamo a recuperare il numero di colonne del result-set.**

**NOTA: è necessaria la libreria `java.sql.ResultSetMetaData`.**

**Le principali funzioni della classe ResultSetMetaData sono:**

- **getColumnCount()** - restituisce il numero di colonne;
  - **getColumnName(int columnNumber)** - restituisce il nome della colonna;
  - **getColumnLabel(int columnNumber)** - restituisce l'alias assegnato nella QUERY alla colonna;
  - **getTableName(int columnNumber)** - restituisce il nome della tabella;
  - **getColumnType(int columnNumber)** - restituisce il tipo di dato della colonna;
  - **isAutoIncrement(int columnNumber)** - indica se la colonna è auto-incrementata;
  - **isCaseSensitive(int columnNumber)** - indica se la colonna è case sensitive;
  - **isSearchable(int columnNumber)** - indica se possiamo usare la colonna nel WHERE;
  - **isNullable(int columnNumber):**
    - **restituisce:**
      - **0** se può avere NULL;
      - **1** se non può avere NULL;
      - **2** se è sconosciuto;
-

**Il result-set ottenuto è navigabile grazie ad un cursore, che di default avanza alle righe successive. La navigazione del cursore è modificabile tramite dei valori da passare al momento della creazione dello statement (se è supportato dal DBMS).**

**Possiamo usare diverse funzioni per far scorrere il cursore:**

- **next()** – passa alla riga successiva;
  - **previous()** – passa alla riga precedente;
  - **first()** – passa alla prima riga del ResultSet;
  - **last()** – passa all'ultima riga;
  - **beforeFirst()** – si sposta all'inizio, prima della prima riga;
  - **afterLast()** – si sposta alla fine, dopo l'ultima riga;
  - **relative(int numOfRows)** – si sposta del numero di righe indicate (anche in negativo);
  - **absolute(int rowNumber)** – passa alla riga specificata;
  - **getRow()** – recupera il numero della riga;
-

**Di default il result-set è di sola lettura, per renderlo modificabile è necessario indicare alla creazione dello statement che può essere modificato:**

```
1.PreparedStatement pstmt = dbConnection.prepareStatement(  
2. QUERY, // la query da eseguire  
3. ResultSet.TYPE_SCROLL_SENSITIVE, // indichiamo il tipo di scorrimento del cursore  
4. ResultSet.CONCUR_UPDATABLE); // indichiamo che il result-set è aggiornabile  
5.ResultSet rs = pstmt.executeQuery();
```

**Si possono inserire e modificare i record del result-set ottenuto:**

```
1.rs.updateDouble("salary", 1100.0); // possiamo passare anche il numero della colonna  
2.rs.updateRow(); // aggiorniamo il database
```

**Per aggiungere una o più righe dobbiamo prima spostarci con il cursore:**

```
1.rs.moveToInsertRow(); // sposta il cursore per inserire una riga
```

**Ora possiamo aggiornare i valori della nuova riga:**

```
1.rs.updateString("name", "Venkat"); //aggiorna il tipo string  
2.rs.updateString("position", "DBA"); //aggiorna il tipo string  
3.rs.updateDouble("salary", 925.0); // aggiorna il tipo double
```

**Infine possiamo inserire la nuova riga al database:**

```
1.rs.insertRow(); // aggiorna il database con la nuova riga
```

**Adesso possiamo spostarci alla riga a cui ci trovavamo prima del metodo moveToInsertRow():**

```
1.rs.moveToCurrentRow(); // sposta il cursore a prima di moveToInsertRow()
```

**Per rimuovere una riga invece bisogna posizionarsi con il cursore alla riga interessata e poi eliminarla:**

```
1.rs.absolute(2); // sposta il cursore alla riga indicata  
2.rs.deleteRow();
```

**Il recupero dei record di un result-set da una QUERY avviene tutto contemporaneamente in memoria. Possiamo indicare il numero di righe massimo da caricare in memoria contemporaneamente:**

```
1.PreparedStatement pstmt = dbConnection.prepareStatement(QUERY);  
2.pstmt.setFetchSize(10);  
3.ResultSet rs = pstmt.executeQuery();
```

**NOTA: ad esempio un risultato di 100 righe verrà visualizzato con 10 cicli da 10 righe di lettura al database. È possibile usare la stessa istruzione per indicare un limite di memoria al result-set.**

**Esercizio:**

**Usando jdbc recuperare e stampare una view che mostri le città italiane presenti nel database world, stampando la view sfruttando le informazioni ricavate dai metadati (quindi con un metodo di stampa generalizzato).**

**Esercizio:**

**Aggiungere tramite JDBC 10 città italiane non presenti nella tabella city di world con anche inserendo i corrispettivi dati. Da svolgere senza le query ma usando i metodi del ResultSet (JDBC).**

---