

Basi di Dati

Java e Database

JDBC



UNIVERSITÀ DEGLI STUDI DI SALERNO

.DIEM

SQL e Applicazioni

- In applicazioni complesse, l'utente non vuole eseguire comandi SQL, ma programmi
- SQL non basta, sono necessarie altre funzionalità, per gestire:
 - **input** (scelte dell'utente e parametri)
 - **output** (con dati che non sono relazioni o se si vuole una presentazione complessa)
 - per gestire il **controllo**

SQL e Applicazioni



Esistono diversi approcci che le applicazioni possono utilizzare per interrogare ed utilizzare il database



Call Level Interface

- Indica genericamente interfacce che permettono di inviare richieste a DBMS per mezzo di parametri trasmessi a funzioni
- Funzioni per la connessione al database, per l'esecuzione di interrogazione, per l'aggiornamento dei dati, e così via.
 - standard SQL/CLI ('95 e poi parte di SQL-3)
 - ODBC: implementazione proprietaria di SQL/CLI
 - JDBC: una CLI per il mondo Java

Call Level Interface

- Standard a "livello di chiamata"
- Approccio dinamico per la programmazione delle basi di dati:
 - Per l'accesso alla base di dati viene usata una libreria di funzioni che da un lato fornisce maggiore flessibilità e non richiede la presenza di alcun preprocessore
 - Comporta, però, che la verifica della sintassi e altri controlli sui comandi SQL avvenga solo al momento dell'esecuzione del programma.

Call Level Interface: ODBC

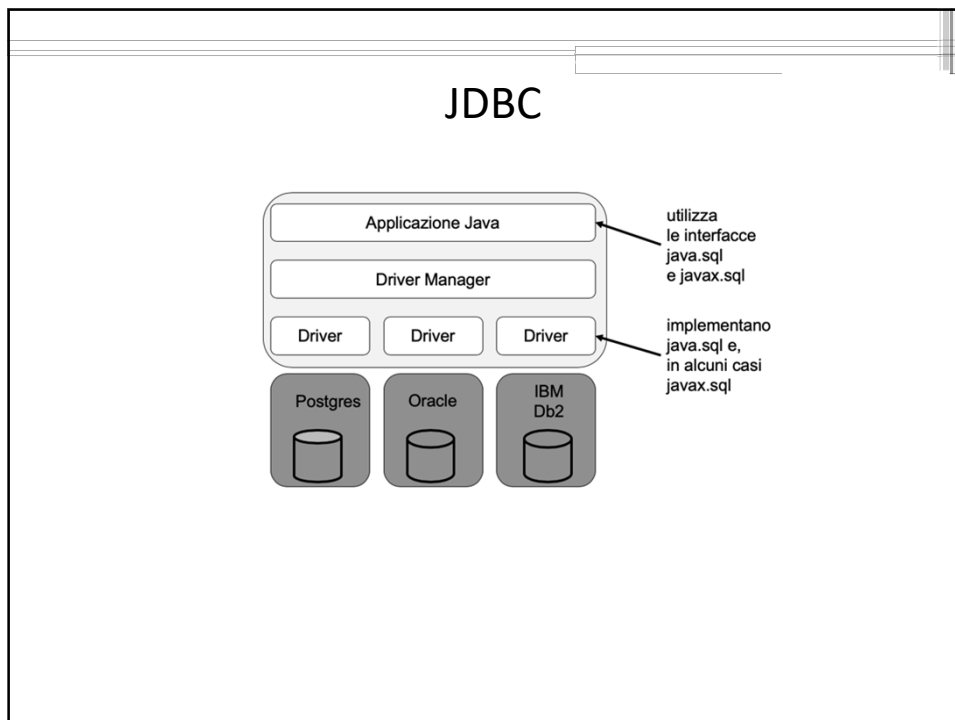
- ODBC Open DataBase Connectivity: è uno standard di fatto definito da Microsoft per l'accesso a basi di dati da applicazioni
- Tutti i principali DBMS possono essere acceduti tramite ODBC, il supporto ad ODBC è integrato nei sistemi operativi della famiglia Windows.
- ODBC Open DataBase Connectivity: è uno standard di fatto definito da Microsoft per l'accesso a basi di dati da applicazioni
- Tutti i principali DBMS possono essere acceduti tramite ODBC, il supporto ad ODBC è integrato nei sistemi operativi della famiglia Windows.
- ADO (ActiveX Data Object) e ADO.NET: sono le API Object Oriented di Microsoft per l'accesso ai dati da applicazioni COM+ e .NET
- OLE DB: è una API di basso livello usata dai tools.

Call Level Interface: ODBC

- ODBC Open DataBase Connectivity: è uno standard di fatto definito da Microsoft per l'accesso a basi di dati da applicazioni
- Tutti i principali DBMS possono essere acceduti tramite ODBC, il supporto ad ODBC è integrato nei sistemi operativi della famiglia Windows.
- ADO (ActiveX Data Object) e ADO.NET: sono le API Object Oriented di Microsoft per l'accesso ai dati da applicazioni COM+ e .NET
- OLE DB: è una API di basso livello usata dai tools.

JDBC

- JDBC é un interfaccia di programmazione (API) ad oggetti basata sullo standard CLI e definita dalla Sun Microsystems per l'accesso ad un database in maniera indipendente dalla piattaforma.
- JDBC è divenuto oramai uno "standard de facto" per lo sviluppo di applicazioni JAVA "DB-oriented" e fa parte del pacchetto software JDK dalla versione 1.1.
- Vantaggi:
 - Astrazione dal DBMS
 - Riuso
 - Sintassi semplice



- ## Tipi di Driver in JDBC
- **Tipo 1:**
 - Implementano un mapping verso un'altra API
 - Esempio: JDBC-ODBC
 - **Tipo 2:**
 - Scritti in parte in Java ed in parte in un altro linguaggio.
 - Usano librerie native del DBMS (scritte ad esempio in C)
 - Esempio: Oracle's OCI (Oracle Call Interface).
 - **Tipo 3:**
 - Usano un middleware server con un protocollo indipendente dal database.
 - **Tipo 4:**
 - Sono scritti completamente in JAVA
 - Esempio: il driver JDBC PostgreSQL

Confronto tra i diversi driver

- Type 1 (Ponte JDBC-ODBC)
 - prestazioni scadenti
 - non indipendente dalla piattaforma
 - fornito a corredo di SDK
- Type 2
 - migliori prestazioni
 - non indipendente dalla piattaforma
- Type 3
 - client indipendente dalla piattaforma
 - servizi avanzati (caching)
 - architettura complessa
- Type 4
 - indipendente dalla piattaforma
 - buone prestazioni
 - scaricabile dinamicamente
 - Per approfondimenti:

<http://java.sun.com/products/jdbc/driverdesc.html>

Documentazione ufficiale JDBC

<https://docs.oracle.com/javase/tutorial/jdbc/index.html>

ORACLE  Java® Documentation

The Java™ Tutorials

[Previous](#) • [Trail](#) • [Next](#) •

[Home Page](#)

The Java Tutorials have been written for JDK 8. Examples and practices described in this page don't take advantage of improvements introduced in later releases.

Trail: JDBC(TM) Database Access

The JDBC™ API was designed to keep simple things simple. This means that the JDBC makes everyday database tasks easy. This trail walks you through examples of using JDBC to execute common SQL statements, and perform other objectives common to database applications.

This trail is divided into these lessons:

[JDBC Introduction](#) Lists JDBC features, describes JDBC Architecture and reviews SQL commands and Relational Database concepts.

[JDBC Basics](#) Covers the JDBC API, which is included in the Java™ SE 8 release.

By the end of the first lesson, you will know how to use the basic JDBC API to create tables, insert values into them, query the tables, retrieve the results of the queries, and update the tables. In this process, you will learn how to use simple statements and prepared statements, and you will see an example of a stored procedure. You will also learn how to perform transactions and how to catch exceptions and warnings.

[Previous](#) • [TOC](#) • [Next](#) •

About Oracle | Contact Us | Legal Notices | Terms of Use | Your Privacy Rights
Copyright © 1996, 2017 Oracle and/or its affiliates. All rights reserved.

Documentazione ufficiale JDBC

<https://docs.oracle.com/javase/tutorial/jdbc/index.html>

ORACLE  Documentation

The Java™ Tutorials

Hide TOC

JDBC Basics

Getting Started
Processing SQL
Statements with JDBC
Establishing a Connection
Connecting with
DataSource Objects
Handling SQLExceptions
Setting Up Tables
Retrieving and Modifying
Values from Result Sets
Using Prepared
Statements
Using Transactions
Using RowSet Objects
Using JDBCRowSet
Objects
Using
CachedRowSetObjects
Using JoinRowSet Objects
Using FilteredRowSet
Objects
Using WebRowSet
Objects
Using Advanced Data
Types
Using Large Objects
Using XML Objects
Using Array Objects
Using DISTINCT Data
Type
Using Structured Objects
Using Customized Type
Mappings

« Previous • Trail • Next »

Home Page > JDBC(TM) Database Access

The Java Tutorials have been written for JDK 8. Examples and practices described in this page don't take advantage of improvements introduced in later releases.

Lesson: JDBC Basics

In this lesson you will learn the basics of the JDBC API.

- Getting Started sets up a basic database development environment and shows you how to compile and run the JDBC tutorial samples.
- Processing SQL Statements with JDBC outlines the steps required to process any SQL statement. The pages that follow describe these steps in more detail:
 - Establishing a Connection connects you to your database.
 - Connecting with DataSource Objects shows you how to connect to your database with DataSource objects, the preferred way of getting a connection to a data source.
 - Handling SQLExceptions shows you how to handle exceptions caused by database errors.
 - Setting Up Tables describes all the database tables used in the JDBC tutorial samples and how to create and populate tables with JDBC API and SQL scripts.
 - Retrieving and Modifying Values from Result Sets develop the process of configuring your database, sending queries, and retrieving data from your database.
 - Using Prepared Statements describes a more flexible way to create database queries.
 - Using Transactions shows you how to control when a database query is actually executed.
- Using RowSet Objects introduces you to RowSet objects; these are objects that hold tabular data in a way that make it more flexible and easier to use than result sets. The pages that follow describe the different kinds of RowSet objects available:
 - Using JDBCRowSet Objects

Preparare l'ambiente

■ Prima di realizzare l'applicazione, è necessario preparare l'ambiente.


- Scegliere il tipo di driver da usare
- Installarlo/configurarlo
- Preparare l'ambiente di sviluppo.

■ Per PostgreSQL, usiamo un driver di Tipo 4 (pure java)

- Posizionamento dei file su disco
- Settaggio della variabile CLASSPATH (i.e., aggiungere il .jar al progetto)
- Connessione da Java tramite il driver

JDBC PostgreSQL: scaricare il driver

<https://jdbc.postgresql.org/>

 PostgreSQL JDBC Driver

The world's most advanced open source database

[Home](#) [About](#) [Download](#) [Documentation](#) [Community](#) [Development](#)

31 March 2020

PostgreSQL JDBC Driver 42.2.12 Released

Notable changes

Changed

Added

Fixed

See full [changelog for 42.2.12](#)

09 March 2020

PostgreSQL JDBC Driver 42.2.11 Released

Notable changes

Changed

Added

LATEST RELEASES

42.2.12 - 31 Mar 2020 - [Notes](#)

42.2.11 - 09 Mar 2020 - [Notes](#)

42.2.10 - 30 Jan 2020 - [Notes](#)

42.2.9 - 06 Nov 2019 - [Notes](#)

Download | [Screenshots](#)

42.2.12

[GitHub project](#)


[Documentation](#)

[Getting started](#)


[FAQ](#)

SUPPORT US

PostgreSQL is free. Please support our work by making a [donation](#)



JDBC PostgreSQL: scaricare il driver

 PostgreSQL JDBC Driver

The world's most advanced open source database

[Home](#) [About](#) [Download](#) [Documentation](#) [Community](#) [Development](#)

Download

- About
- Current Version
- Supported Versions
- Other Versions
- Archived Versions

About

Binary JAR file downloads of the JDBC driver are available here and the current version with [Maven Repository](#). Because Java is platform neutral, it is a simple process of just downloading the appropriate JAR file and dropping it into your classpath. Source versions are also available here for recent driver versions.

This is the current version of the driver. Unless you have unusual requirements (running old applications or JVMs), this is the driver you should be using. It supports PostgreSQL 8.2 or newer and requires Java 6 or newer. It contains support for SSL and the `javax.sql` package.

- If you are using Java 8 or newer then you should use the JDBC 4.2 version.
- If you are using Java 7 then you should use the JDBC 4.1 version.
- If you are using Java 6 then you should use the JDBC 4.0 version.
- If you are using a Java version older than 6 then you will need to use a JDBC3 version of the driver, which will by necessity not be supported here.

Current Version 42.2.12

[PostgreSQL JDBC 4.2 Driver - 42.2.12](#)

[PostgreSQL JDBC 4.1 Driver - 42.2.12.jar](#)

[PostgreSQL JDBC 4.0 Driver - 42.2.12.jar](#)

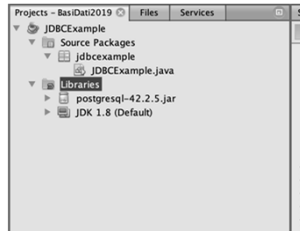
Other Versions

Many other versions of the JDBC driver are available. This includes development versions, compatibility with older JDKs, and previous versions of the driver.

JDBC PostgreSQL: importare il driver nel progetto

Nota bene: il **driver.jar** del driver deve essere nel **CLASSPATH**.
Per farlo è possibile importare il driver all'interno del progetto

- Creare un progetto e aggiungere il driver postgresql-xx.x.x.jar
- Il file .jar contiene il driver per PostgreSQL



Applicazione: passi fondamentali

- Una volta configurato l'ambiente, è possibile realizzare un applicazione in Java in grado di utilizzare il database.
- Una generica applicazione in Java che utilizza il database, deve eseguire i seguenti step:
 1. Caricamento del driver
 2. Apertura della connessione
 3. Esecuzione di comandi SQL
 4. Elaborazione dei risultati
 5. Chiusura della connessione

1. Caricare il driver

- La classe `java.sql.DriverManager` implementa i servizi di base per gestire i driver JDBC e per l'apertura delle connessioni.

- Prima di poter aprire una connessione, dobbiamo "caricare" il driver e indicare al **DriverManager** che vogliamo usare un particolare driver e renderlo disponibile

- Se il driver è disponibile nel **ClassPath**

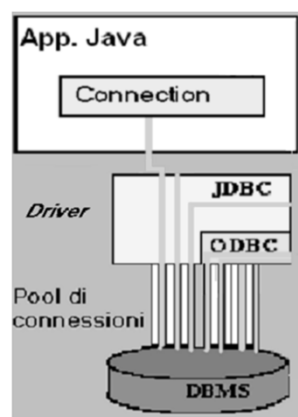
`Class.forName("org.postgresql.Driver");`

- Il nome della classe che implementa il driver è indicato con una stringa che può essere letta da un file di configurazione

- Disaccoppiamento dallo specifico DBMS

- Non è necessario ricompilare il codice se si cambia il DBMS, basta modificare la stringa nel file di configurazione

2. Apertura della connessione



2. Apertura della connessione

- Un oggetto di tipo **java.sql.Connection** rappresenta una connessione (sessione) con un database.

- I comandi al DBMS ed i risultati sono inviati e ricevuti tramite la connessione.

- Il metodo `DriverManager.getConnection()` restituisce un oggetto di tipo `Connection` per la gestione della connessione.

```
DriverManager.getConnection (String url, String user, String password)
```

2. Apertura della connessione

- Quali parametri passare a `getConnection()`?

- URL

- In JDBC ogni database viene univocamente identificato da una particolare stringa di connessione detta JDBC URL.

- Tale stringa adotta un formalismo simile alla definizione degli URL e serve a definire dove si trova il db e come accedere: specifica server, porta e database

- Sintassi della JDBC URL in PostgreSQL

```
jdbc:postgresql://<host>:<porta>/<baseDati>
```

2. Apertura della connessione

■ Quali parametri passare a getConnection()?

■ URL per PostgreSQL

```
jdbc:postgresql:<baseDati>  
jdbc:postgresql://<host>/<baseDati>  
jdbc:postgresql://<host>:<porta>/<baseDati>
```

Esempi

```
jdbc:postgresql:university  
jdbc:postgresql://127.0.0.1/university  
jdbc:postgresql://193.204.161.14:5432/university  
jdbc:odbc:university
```

2. Apertura della connessione

■ Quali parametri passare a getConnection()?

```
DriverManager.getConnection (String url, String  
user, String password)
```

■ Gli altri due parametri sono il nome dell'utente e la password per accedere al database

2. Apertura della connessione

```
import java.sql.*;
import java.util.logging.Level;
import java.util.logging.Logger;

public class JDBCConnection {

    public static void main(String[] args) {
        String url = "jdbc:postgresql://localhost/negozi";
        String user = "pippo";
        String pwd = "Pluto01";
        Connection conn = null;
        try {
            //Carichiamo la classe contenente il Driver
            Class.forName("org.postgresql.Driver");

            //Utilizziamo il DriverManager per aprire una connessione
            conn = DriverManager.getConnection(url, user, pwd);

            //Se non viene lanciata un'eccezione, allora la connessione
            //è stata creata correttamente
            System.out.println("Connessione aperta con il database");

            /* INSERIRE CODICE PER INTERAGIRE CON IL DATABASE */

            //Alla fine, chiudere la connessione
            conn.close();
        } catch (ClassNotFoundException ex) {
            Logger.getLogger(JDBCConnection.class.getName()).log(Level.SEVERE, null, ex);
        } catch (SQLException ex) {
            Logger.getLogger(JDBCConnection.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
}
```

2. Apertura della connessione

- L'operazione di connessione è costosa in termini di tempo
- Per questo solitamente si usa, all'inizio dell'applicazione, aprire tutte le connessioni necessarie, riutilizzando una stessa connessione per eseguire più operazioni, per poi chiuderle alla fine dell'applicazione
- La connessione viene chiusa con il metodo **close()**

Focus su Gestione degli utenti in PostgreSQL

SQL standard leaves the definition of users to the implementation

In PostgreSQL

CREATE USER *name* [[WITH] *option* [...]]

where *option* can be:

SUPERUSER | NOSUPERUSER
| CREATEDB | NOCREATEDB
| CREATEROLE | NOCREATEROLE
| INHERIT | NOINHERIT
| LOGIN | NOLOGIN
| REPLICATION | NOREPLICATION
| BYPASSRLS | NOBYPASSRLS
| CONNECTION LIMIT *connlimit*
| [ENCRYPTED] PASSWORD '*password*' | PASSWORD NULL | VALID UNTIL '*timestamp*'
| IN ROLE *role_name* [, ...]
| IN GROUP *role_name* [, ...]
| ROLE *role_name* [, ...]
| ADMIN *role_name* [, ...]
| USER *role_name* [, ...]
| SYSID *uid*

```
create user pippo password 'Pluto01'  
drop user pippo
```

E' definito a livello di cluster di database

Focus su Gestione degli utenti in PostgreSQL

Privilegi degli utenti

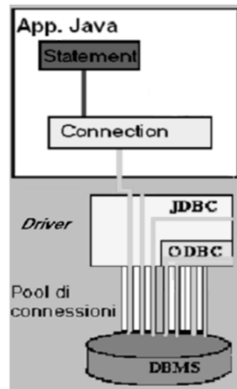
- insert (tabelle e viste)
- update (tabelle e viste)
- delete (tabelle e viste)
- select (tabelle e viste)
- references (tabelle ed attributi)
- usage (domini)
- all privileges

grant *privilegio* **on** *risorsa* **to** *utente* [with grant option]

revoke *privilegio* **on** *risorsa* **from** *utente* [restrict | cascade]

```
grant select on prodotti to pippo
```

3. Esecuzione di comandi SQL



3. Esecuzione di comandi SQL

Attraverso la connessione (oggetto Connection) è possibile inviare al database comandi utilizzando tre tipologie di oggetti diversi:

1. **Statement**: creati dal metodo **createStatement**, rappresentano comandi SQL e dovrebbero essere usati per comandi senza parametri inseriti dall'utente
2. **PreparedStatement**: creati dal metodo **prepareStatement**, rappresentano comandi SQL che possono essere precompilati nel database ed accettare parametri (di IN) al momento dell'esecuzione
3. **CallableStatement**: creati dal metodo **prepareCall**, che possono essere usati per eseguire stored procedure ed accettano anche parametri di OUT e INOUT.

Statement

- **Statement** è un'interfaccia i cui oggetti consentono di inviare, tramite una connessione, istruzioni SQL e di ricevere i risultati forniti

- La creazione di un oggetto di tipo **Statement** è effettuata da un oggetto di tipo **Connection**:

```
Connection con = DriverManager.getConnection(...);  
Statement stmt = con.createStatement();
```

- L'oggetto **Statement** permette di eseguire un comando SQL (ad esempio con il metodo `executeQuery`, ottenendo gli eventuali risultati (e.g., risultato di una query)

Statement

- L'oggetto **Statement** ha tre metodi per l'esecuzione di un comando

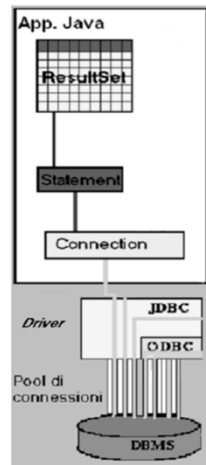
- `executeQuery`: Per statement SQL che producono un singolo set di risultati (es. `SELECT`).

```
Connection con = DriverManager.getConnection(...);  
Statement stmt = con.createStatement();  
stmt.executeQuery("Select * From Utente");
```

- `executeUpdate`: Per statement DML quali `INSERT`, `UPDATE`, o `DELETE` e DDL come `CREATE TABLE`, `DROP TABLE`, and `ALTER TABLE`. Il metodo restituisce un intero che indica il numero di righe modificate per le istruzioni DML ed è sempre 0 per le istruzioni DDL.

- `execute`: Per metodi che possono restituire più di un set di risultati.

4. Elaborazione dei risultati



ResultSet

- I risultati delle interrogazioni sono forniti in oggetti di tipo **ResultSet** (interfaccia definita in java.sql) che li contiene organizzati per riga e colonna
- Un result set è una **sequenza di ennuple** su cui si può "**navigare**" (in avanti, indietro e anche con accesso diretto)

1	2	3	4	5	6	7	8	9	10
NOME	COGNOME	UNAME	PIU	CF	LIVELLO	TEL	VIA	CITTA	CAP
					Dati				

- Si possono estrarre i valori degli attributi dalla ennupla *corrente*

Statement e ResultSet

- Un oggetto `ResultSet` è restituito dall'esecuzione di un comando effettuata con uno `statement` (e.g., `executeQuery()` restituisce un `ResultSet` contenente le righe risultato dell'esecuzione della query)

```
ResultSet rs = stmt.executeQuery("SELECT a,b FROM  
Table2;");
```

- Il risultato dell'esecuzione di un comando tramite `Statement` deve essere inserito in un oggetto di tipo `ResultSet`

Statement e ResultSet

```
public static void main(String[] args) {  
    String url = "jdbc:postgresql://localhost/megoz";  
    String user = "alippo";  
    String pwd = "Pluto88";  
    Connection conn = null;  
    Statement stmt = null;  
  
    try {  
        //Carichiamo la classe contenente il Driver  
        Class.forName("org.postgresql.Driver");  
  
        //Utilizziamo il DriverManager per aprire una connessione  
        conn = DriverManager.getConnection(url, user, pwd);  
  
        //Se non viene lanciata un'eccezione, allora la connessione  
        //è stata creata correttamente  
        System.out.println("Connessione aperta con il database");  
  
        stmt = conn.createStatement();  
        ResultSet rs = stmt.executeQuery("SELECT * FROM prodotti");  
  
        //Alla fine, chiudere le risorse  
        stmt.close();  
        conn.close();  
    } catch (ClassNotFoundException ex) {  
        Logger.getLogger(JDBCStatement.class.getName()).log(Level.SEVERE, null, ex);  
    } catch (SQLException ex) {  
        Logger.getLogger(JDBCStatement.class.getName()).log(Level.SEVERE, null, ex);  
    }  
}
```

ResultSet

Metodi principali di ResultSet:

- `next()`
- `getXXX(posizione)`
e.g.: `getString(3); getInt(2)`
- `getXXX(nomeAttributo)`
e.g.: `getString("Cognome"); getInt("Codice")`
- I metodi `getTipo(String nomeColonna)` o `getTipo(int indiceColonna)` permettono di prelevare i dati dalla riga corrente.
- Il `ResultSet` appena ottenuto è posizionato prima dell'inizio.
- Il metodo `ResultSet.next()` permette di avanzare riga per riga e restituisce false quando si giunge al termine.
- **NB:** Il **ResultSet** rimane collegato allo **Statement** da cui è stato generato. `executeQuery` ed `executeUpdate` chiudono il **ResultSet** precedentemente creato dallo **Statement** su cui sono invocati.

ResultSet

Metodi di get

X - indica che il metodo è il metodo d'elezione per quel tipo SQL.

x – indica che il metodo può essere usato per valori di quel tipo SQL.

[illegible]

ResultSet: Esempio

```
public static void main(String[] args) {
    String url = "jdbc:postgresql://localhost/negozi";
    String user = "pippo";
    String pwd = "123456";
    Connection conn = null;
    Statement stmt = null;

    try {
        //Carichiamo la classe contenente il Driver
        Class.forName("org.postgresql.Driver");
        //Utilizziamo il DriverManager per aprire una connessione
        conn = DriverManager.getConnection(url, user, pwd);
        //Se non viene lanciata un'eccezione, allora la connessione
        //è stata creata correttamente
        System.out.println("Connessione aperta con il database");
        //Creare uno statement utilizzando la connessione
        stmt = conn.createStatement();
        //Utilizzando lo statement è possibile eseguire query. I risultati
        //della query sono restituiti come oggetto ResultSet
        ResultSet rs = stmt.executeQuery("SELECT * FROM prodotti");

        //Utilizziamo il ResultSet per fare una stampa dei risultati della
        //query SELECT * FROM prodotti
        System.out.println("PID \t Nome \t\t\t\t Colore");
        while(rs.next()) {
            System.out.println(rs.getString(1) + "\t" + rs.getString("nome") +
                               "\t" + rs.getString("colore"));
        }
        //Alla fine, chiudere le risorse
        rs.close();
        stmt.close();
        conn.close();
    } catch (ClassNotFoundException ex) {
        Logger.getLogger(JDBCResultSet.class.getName()).log(Level.SEVERE, null, ex);
    } catch (SQLException ex) {
        //
    }
}
```

ResultSet: Esempio

PID	Nome	Colore
P1	Volante	Nero
P3	Carrozzeria	Nero
P4	Carrozzeria	Rosso
P5	Carrozzeria	Verde
P6	Cerchione	Nero
P7	Cerchione	Rosso
P8	Ruota	Nero
P9	Sedile	Nero
P10	Sedile	Rosso
P11	Sedile	Verde
P12	Tappetino	Nero
P13	Tappetino	Rosso
P14	Tappetino	Verde
P15	Casco	Rosso
P16	Casco	Verde
P2	Volante	Rosso

BUILD SUCCESSFUL (total time: 0 seconds)

ResultSet: valori null

Lettura di valori SQL NULL

- Un valore NULL può essere convertito da JDBC in **null**, **0** o **false**
- **null** si ottiene da quei metodi che restituiscono oggetti java (getString, getBigDecimal, getBytes, getDate, getTime, getTimeStamp, getAsciiStrea ...)
- **0** (zero) si ottiene da getByte, getShort, getInt, getLong, getFloat e getDouble
- **False** si ottiene da getBoolean

ResultSet: cursore

Il ResultSet è un cursore

Il ResultSet funziona come un cursore. Una volta spostato su un elemento, tale elemento viene "consumato", ciò significa che una volta letto un valore, se provo a rileggerlo posso ottenere un errore (dipende anche dall'implementazione del driver, ma JDBC non richiede che il valore sia ancora disponibile dopo un primo accesso ad esso).

```
while (rst.next()) {  
    System.out.print(rst.getString("PID")+" ");  
    System.out.print(rst.getString("PID")+" ");  
    System.out.println();  
}
```

Potrebbe non funzionare

ResultSet

Altri metodi per "muoversi" tra le righe del ResultSet

- next()
 - previous()
 - first()
 - last()
 - beforeFirst()
 - afterLast()
 - absolute(int pos)
 - relative (int pos)
-
- Non sempre è possibile spostarsi sulle righe in tutte le direzioni: dipende da come è stato definito il ResultSet.

ResultSet – spostarsi all'indietro

```
try {
    //Carichiamo la classe contenente il Driver
    Class.forName("org.postgresql.Driver");
    //Utilizziamo il DriverManager per aprire una connessione
    conn = DriverManager.getConnection(url, user, pwd);
    //Se non viene lanciata un'eccezione, allora la connessione
    //è stata creata correttamente
    System.out.println("Connessione aperta con il database");
    //Creare uno statement utilizzando la connessione
    stmt = conn.createStatement();
    //Utilizzando lo statement è possibile eseguire query. I risultati
    //della query sono restituiti come oggetto ResultSet
    ResultSet rs = stmt.executeQuery("SELECT * FROM prodotti");

    //Utilizziamo il ResultSet per fare una stampa dei risultati della
    //query SELECT * FROM prodotti
    System.out.println("PID \t Nome \t\t\t\t Colore");
    while(rs.next()){
        System.out.println(rs.getString(1) + "\t" + rs.getString("nome") +
            "\t\t" + rs.getString("colore"));
    }
}

rs.previous();
System.out.println("Sposta cursore indietro: \n" +
    rs.getString("PID") + "\t\t" + rs.getString("nome"));

//Alla fine, chiudere le risorse
rs.close();
stmt.close();
conn.close();
}
```

ResultSet – spostarsi all'indietro

```
rs      nuova      nero
P9      Sedile     Nero
P10     Sedile     Rosso
P11     Sedile     Verde
P12     Tappetino  Nero
P13     Tappetino  Rosso
P14     Tappetino  Verde
P15     Casco      Rosso
P16     Casco      Verde
mag 11, 2020 5:38:27 PM basidati.JDBCResultSetErrorePrevious main
GRAVE: null
org.postgresql.util.PSQLException: L'operazione richiede un «ResultSet» scorribile mentre questo è «FORWARD_ONLY».
    at org.postgresql.jdbc.PgResultSet.checkScrollable(PgResultSet.java:279)
    at org.postgresql.jdbc.PgResultSet.previous(PgResultSet.java:854)
    at basidati.JDBCResultSetErrorePrevious.main(JDBCResultSetErrorePrevious.java:50)

BUILD SUCCESSFUL (total time: 0 seconds)
```

ResultSet: Tipologie

Esistono infatti tre tipologie di ResultSet, ognuno supporta metodi diversi per lo spostamento tra le righe.

- **ResultSet.TYPE_FORWARD_ONLY (default):**

Permette solo la navigazione in un senso attraverso il metodo next(). Solo rieseguendo la query io possono ritornare all'inizio.

- **ResultSet.TYPE_SCROLL_INSENSITIVE**

Si possono navigare in qualunque modo anche con accesso diretto. Eventuali modifiche alla base dati fatte da altri utenti non saranno visibili

- **ResultSet.TYPE_SCROLL_SENSITIVE**

identico al caso precedente ma sensibile alle modifiche alla base dati

Il tipo di ResultSet desiderato deve essere specificato all'atto della creazione dello statement, come parametro di conn.createStatement()

ResultSet scrollable

```
try {
    //Carichiamo la classe contenente il Driver
    Class.forName("org.postgresql.Driver");
    //Utilizziamo il DriverManager per aprire una connessione
    conn = DriverManager.getConnection(url, user, pwd);
    //Se non viene lanciata un'eccezione, allora la connessione
    //è stata creata correttamente

    System.out.println("Connessione aperta con il database");

    //Se viene eseguito un statement utilizzando la connessione
    stmt = conn.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,ResultSet.CONCUR_READ_ONLY);
    //Utilizziamo lo statement a possibile eseguire query. I risultati
    //della query sono restituiti come oggetto ResultSet
    ResultSet rs = stmt.executeQuery("SELECT * FROM prodotti");

    //Utilizziamo il ResultSet per fare una stampa dei risultati della
    //query SELECT * FROM prodotti
    System.out.println("PID \t Nome \t\t\t\t Colore");
    while(rs.next()){
        System.out.println(rs.getString(1) + "\t" + rs.getString("nome") +
            "\t" + rs.getString("colore"));
    }

    rs.previous();
    System.out.println("Sposta cursore indietro: \n" +
        rs.getString("PID") + "\t" + rs.getString("nome"));

    //Alla fine, chiudere le risorse
    rs.close();
    stmt.close();
    conn.close();
}
```

ResultSet scrollable

```

P9      Sedile          Nero
P10     Sedile          Rosso
P11     Sedile          Verde
P12     Tappettino     Nero
P13     Tappettino     Rosso
P14     Tappettino     Verde
P15     Casco           Rosso
P16     Casco           Verde

Sposta cursore indietro:
P16     Casco

BUILD SUCCESSFUL (total time: 0 seconds)

```


ResultSet: Tipologie

Il **ResultSet** può essere utilizzato anche per modificare i dati del database, utilizzando i metodi `updateXXX()` (vedi slide successive). Per poter impostare il dataset come "sola lettura" o "modificabile", si utilizza un secondo parametro del metodo `createStatement`, che può avere i seguenti valori:

- **ResultSet.CONCUR_READ_ONLY**: Il contenuto non può essere modificato.
- **ResultSet.CONCUR_UPDATABLE**: Il contenuto può essere modificato

ResultSet: Tipologie

Per scegliere il particolare tipo di **ResultSet** che si intende ottenere, è necessario specificare un secondo parametro nell'esecuzione del comando tramite lo **Statement**

```
Connection con = DriverManager.getConnection(
    "jdbc:my_subprotocol:my_subname");
Statement stmt = con.createStatement(
    ResultSet.TYPE_SCROLL_SENSITIVE
    ResultSet.CONCUR_UPDATABLE)
```

In questo esempio, lo **Statement** **stmt** produrrà **ResultSet** Scrollable, sensibili ai cambiamenti del DB, modificabile.

ResultSet: Update

Per modificare i contenuti della riga corrente di un RecordSet sono disponibili opportuni metodi di update.

```
updateType(int col,Type value)
updateType(String colName,Type value)
```

Dove

Type è il tipo java corrispondente (il driver si incarica di trasformare i valori secondo necessità),
col e colname sono rispettivamente l'indice o il nome della colonna*

NB Le modifiche sono trasferite al DB alla chiamata del metodo `updateRow`. Dato che questo trasferisce solo la riga corrente è fondamentale invocarlo quando ancora si è posizionati su questa

* L'indice della colonna nel ResultSet può essere diverso dallo stesso nella tabella

ResultSet: Update

```
//Utilizziamo il DriverManager per aprire una connessione
conn = DriverManager.getConnection(url, user, pwd);
//Se non viene lanciata un'eccezione, allora la connessione
//è stata creata correttamente
System.out.println("Connessione aperta con il database");
//Creare uno statement utilizzando la connessione
stmt = conn.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE, ResultSet.CONCUR_UPDATABLE);
//Utilizzando lo statement è possibile eseguire query. I risultati
//della query sono restituiti come oggetto ResultSet
ResultSet rs = stmt.executeQuery("SELECT * FROM prodotti");
//Utilizziamo il ResultSet per fare una stampa dei risultati della
//query SELECT * FROM prodotti
System.out.println("PID \t Nome \t\t Colore");
while(rs.next()){
    System.out.println(rs.getString(1) + "\t" + rs.getString("nome") +
        "\t" + rs.getString("colore"));
}

//Modifichiamo il colore del primo prodotto in Giallo
rs.first();
rs.updateString("colore", "Giallo");
rs.updateRow();
rs.beforeFirst();
System.out.println("Prodotto modificato");

//Stampiamo di nuovo i prodotti per vedere la modifica effettuata
while(rs.next()){
    System.out.println(rs.getString(1) + "\t" + rs.getString("nome") +
        "\t" + rs.getString("colore"));
}
```

ResultSet: Delete

```
rs.deleteRow(); // Elimina la riga corrente

//Carichiamo la classe contenente il Driver
Class.forName("org.postgresql.Driver");
//Utilizziamo il DriverManager per aprire una connessione
conn = DriverManager.getConnection(url, user, pwd);
//Se non viene lanciata un'eccezione, allora la connessione
//è stata creata correttamente
System.out.println("Connessione aperta con il database");
//Creare uno statement utilizzando la connessione
stmt = conn.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE, ResultSet.CONCUR_UPDATABLE);
//Utilizzando lo statement è possibile eseguire query. I risultati
//della query sono restituiti come oggetto ResultSet
ResultSet rs = stmt.executeQuery("SELECT * FROM prodotti");
//Utilizziamo il ResultSet per fare una stampa dei risultati della
//query SELECT * FROM prodotti
System.out.println("PID \t Nome \t\t\t Colore");
while(rs.next()){
    System.out.println(rs.getString(1) + "\t" + rs.getString("nome") +
        "\t" + rs.getString("colore"));
}

//Eliminiamo il primo prodotto
rs.first();
rs.deleteRow();
rs.beforeFirst();
System.out.println("Prodotto eliminato");

//Stampiamo di nuovo i prodotti per vedere la modifica effettuata
while(rs.next()){
    System.out.println(rs.getString(1) + "\t" + rs.getString("nome") +
        "\t" + rs.getString("colore"));
}
```

```
//Carichiamo la classe contenente il Driver
Class.forName("org.postgresql.Driver");

//Creiamo il DriverManager per aprire una connessione
conn = DriverManager.getConnection(url, user, pwd);
//Se non viene lanciata un'eccezione, allora la connessione
//è stata creata correttamente

System.out.println("Connessione aperta con il database");
//Creare uno statement utilizzando la connessione
stmt = conn.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,ResultSet.CONCUR_UPDATABLE);
//Utilizzando lo statement è possibile eseguire query. I risultati
//della query sono restituiti come oggetto ResultSet
ResultSet rs = stmt.executeQuery("SELECT * FROM prodotti");
//Utilizzando il ResultSet per fare una stampa dei risultati della
//query SELECT * FROM prodotti
System.out.println("PID \t Nome \t\t\t Colore");
while(rs.next()){
    System.out.println(rs.getString(1) + "\t" + rs.getString("nome") +
        "\t" + rs.getString("colore"));
}

//Eliminiamo il primo prodotto
rs.first();
rs.deleteRow();
rs.beforeFirst();
System.out.println("Prodotto eliminato");

//Stampiamo di nuovo i prodotti per vedere la modifica effettuata
while(rs.next()){
    System.out.println(rs.getString(1) + "\t" + rs.getString("nome") +
        "\t" + rs.getString("colore"));
}
```

ResultSet: Insert

La **insert row** è una riga associata ad un ResultSet ma non ancora parte di questo:

rs.moveToInsertRow();	// sposta il cursore alla insert row
rs.updateObject(1, myArray);	// inserisce il primo elemento
rs.updateInt(2, 3857);	// inserisce il secondo elemento
rs.updateString(3, "Mysteries");	// inserisce il terzo elemento
rs.insertRow();	// inserisce la insert row nel ResultSet
rs.moveToCurrentRow();	// si posiziona all'ultima riga visitata

NB

- Se si legge un valore dalla **insert row** prima di averlo assegnato il valore è indefinito
- I valori vengono immessi nel DB alla chiamata del metodo insertRow, (ciò può generare una SQLException)
- moveToCurrentRow riporta il cursore alla posizione precedente la chiamata di moveToInsertRow

```
rs.moveToInsertRow();           // sposta il cursore alla insert row
rs.updateObject(1, myArray);    // inserisce il primo elemento
rs.updateInt(2, 3857);          // inserisce il secondo elemento
rs.updateString(3, "Mysteries"); // inserisce il terzo elemento
rs.insertRow();                 // inserisce la insert row nel ResultSet
rs.moveToCurrentRow();          // si posiziona all'ultima riga visitata
```

- Se si legge un valore dalla **insert row** prima di averlo assegnato il valore è indefinito
- I valori vengono immessi nel DB alla chiamata del metodo **insertRow**, (ciò può generare una **SQLException**)
- **moveToCurrentRow** riporta il cursore alla posizione precedente la chiamata di **moveToInsertRow**

ResultSet: Insert

```
try {
    //Carichiamo la classe contenente il Driver
    Class.forName("org.postgresql.Driver");
    //Utilizziamo il DriverManager per aprire una connessione
    conn = DriverManager.getConnection(url, user, pwd);
    //Se non viene lanciata un'eccezione, allora la connessione
    //è stata creata correttamente
    System.out.println("Connessione aperta con il database");
    //Creare una statement utilizzando la connessione
    stat = conn.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE, ResultSet.CONCUR_UPDATABLE);
    //Utilizzando lo statement è possibile eseguire query. I risultati
    //della query sono restituiti come oggetto ResultSet
    ResultSet rs = stat.executeQuery("SELECT * FROM prodotti");

    System.out.println("Inserimento nuovo prodotto con ResultSet");
    //Sposto il cursore sulla insertRow
    rs.moveToInsertRow();
    //Modifico i valori della insertRow
    rs.updateString("PID", "P20");
    rs.updateString("Nome", "Specchietto");
    rs.updateString("colore", "nero");
    //Inserisco la riga nel database
    rs.insertRow();
    rs.beforeFirst();
    System.out.println("Prodotto Inserito");

    //Stampiamo tutti i prodotti per vedere anche il nuovo prodotto inserito
    while(rs.next()){
        System.out.println(rs.getString(1) + "\t" + rs.getString("nome") +
            "\t" + rs.getString("colore"));
    }

    //Alla fine, chiudere le risorse
}
```

ResultSet: Insert

```
RUN:
Connessione aperta con il database
Inserimento nuovo prodotto con ResultSet
Prodotto Inserito
P3      Carrozzeria      Nero
P4      Carrozzeria      Rosso
P5      Carrozzeria      Verde
P6      Cerchione        Nero
P7      Cerchione        Rosso
P8      Ruota            Nero
P9      Sedile           Nero
P10     Sedile           Rosso
P11     Sedile           Verde
P12     Tappetino        Nero
P13     Tappetino        Rosso
P14     Tappetino        Verde
P15     Casco            Rosso
P16     Casco            Verde
P1      Volante          Giallo
P2      Volante          Nero
P20     Specchietto      nero
BUILD SUCCESSFUL (total time: 0 seconds)
```

Statement: executeUpdate

- Il metodo executeUpdate consente di effettuare query che modificano (insert, update, delete) i dati oppure un comando DDL
- In questo modo è possibile modificare, cancellare e inserire nuovi dati utilizzando direttamente query SQL (invece del resultSet)

```
int a = stm.executeUpdate(query);
```

- Viene restituito
 - un numero intero (contatore di aggiornamento) rappresentante il numero di righe che sono state inserite/aggiornate/cancellate
 - il valore 0 per statement di tipo DDL.

Statement: executeUpdate

```
try {
    Class.forName("org.postgresql.Driver");
    conn = DriverManager.getConnection(url,user,pwd);
    stat = conn.createStatement();
    Scanner scanner = new Scanner(System.in);
    System.out.print("Inserisci ID prodotto: ");
    String idProdotto = scanner.nextLine();
    System.out.print("Inserisci nome prodotto: ");
    String nomeProdotto = scanner.nextLine();

    //Inserimento prodotto con query SQL
    int a = stat.executeUpdate("INSERT INTO prodotti(PID,nome) values('"+
        idProdotto+"', '"+nomeProdotto+"')");

    System.out.println("Inserito " + a + " nuovo prodotto");

    ResultSet rs = stat.executeQuery("SELECT * FROM prodotti");

    //Utilizziamo il ResultSet per fare una stampa dei risultati della
    //query SELECT * FROM prodotti
    System.out.println("PID \t Nome \t\t\t Colore");
    while(rs.next()){
        System.out.println(rs.getString(1) + "\t" + rs.getString("nome") +
            "\t\t\t " + rs.getString("colore"));
    }
}
```

SQL Injection

Esempio di **SQL Injection**: avviene quando inseriamo come parametro della query un valore inserito dall'utente attraverso l'interfaccia della nostra applicazione senza fare nessun controllo. L'utente potrebbe inserire codice malevolo invece dei dati richiesti.

Inserisci ID prodotto: 23

Inserisci nome prodotto: Sedile'); DELETE From Prodotti WHERE ('1'='1

Un utente potrebbe inserire codice malevolo sfruttando l'SQL injection!

Prepared Statement

L'utilizzo del **PreparedStatement** invece che del normale Statement previene l'SQL Injection.

Il PreparedStatement è un'estensione di Statement che consente di **precompilare** interrogazioni SQL con **parametri di input etichettati** con il simbolo '?' e **attualizzati** successivamente con metodi specifici prima dell'esecuzione effettiva.

Un oggetto PreparedStatement può essere creato con il metodo

```
PreparedStatement prep = Connection.prepareStatement(stringaSQL)
```

Prepared Statement

Nella query che vogliamo eseguire con il PreparedStatement indichiamo tutti i parametri con dei segnaposto '?'.
Tali segnaposto saranno sostituiti dai valori da utilizzare nella query con appositi metodi dell'oggetto PreparedStatement prima dell'esecuzione della query

Tale meccanismo consente di evitare che al posto del valore desiderato, venga passato codice malevolo

```
stringaSQL = "Insert Into Utente(nome, cognome)
              VALUES(?, ?)";
```

Prepared Statement

Prima di eseguire la query, devono essere sostituiti i valori al posto dei segnaposto.

Si utilizzano i metodi setXXX() dell'oggetto PreparedStatement.

```
stringaSQL = "Insert Into Utente(nome, cognome)
              VALUES(?, ?)";
```

```
PreparedStatement prep =
Connection.prepareStatement(stringaSQL);
```

```
prep.setString(1, "Mario");
prep.setString(2, "Rossi");
prep.executeUpdate();
```

Prepared Statement

```
try {
    Class.forName("org.postgresql.Driver");
    conn = DriverManager.getConnection(url,user,pwd);
    pstmt = conn.prepareStatement("INSERT INTO prodotti (PID,nome) values(?,?)");
    Scanner scanner = new Scanner(System.in);
    System.out.println("Inserisci ID prodotto: ");
    String idProdotto = scanner.nextLine();
    System.out.println("Inserisci nome prodotto: ");
    String nomeProdotto = scanner.nextLine();

    //Inserimento prodotto con prepared Statement
    pstmt.setString(1, idProdotto);
    pstmt.setString(2, nomeProdotto);
    int a = pstmt.executeUpdate();

    System.out.println("Inserito " + a + " nuovo prodotto");

    stmt = conn.createStatement();
    ResultSet rs = stmt.executeQuery("SELECT * FROM prodotti");
    //Utilizziamo il ResultSet per fare una stampa dei risultati della
    //query SELECT * FROM prodotti
    System.out.println("PID VS Nome VS Colore");
    while(rs.next()){
        System.out.println(rs.getString(1) + "\t" + rs.getString("nome") +
            "\t" + rs.getString("colore"));
    }

    rs.close();
    stmt.close();
    pstmt.close();
    conn.close();
} catch (ClassNotFoundException ex) {
```

5. Chiusura della connessione

Al termine delle operazioni effettuate su un ResultSet, è necessario chiudere il ResultSet per liberare risorse

```
rs.close();
```

Una volta terminate le operazioni con lo Statement, è necessario chiuderlo. Se un ResultSet è associato allo statement, viene chiuso.

```
stmt.close();
```

E infine chiudere la connessione con il database una volta completate tutte le operazioni.

```
conn.close();
```