La JDBC è un API di Java che accede ai dati presenti nel database, la JDBC consente di creare applicazioni di Java, creare query, recupera ed elabora i risultati della query.  
  
Per utilizzare JDBC con Java, devo utilizzare una libreria specifica, con MySQL usiamo il MySQL connector - J che consente di avere connettere fisicamente il DB a Java.  
  
Scrivendo il codice abbiamo   
Creazione di una istanza, se è null crea istanza di connectionfactory, se la connessione è nulla crea una connessione che si prendere il driver da un localhost e return connection, se la connessione esiste invece crea metodo di chiusura connessione.  
  
Abbiamo un metodo insertPreparedStatementet che mi consente di inserire uno statement, che è una forma arcaina della scrittura di una query e non si utilizzano quasi più per una questiore di sicurezza, e può causare SQL injection, vulnerabilità di sicurezza, manipolando una query, perché magari un hacker potrebbe scrivere in quella statement un drop table e ciao, quindi milioni di danni.

Ricorda quando abbiamo una queryString = “INSERTI INTO passeggero values (?,?,?,?)”;  
  
quando faccio un preparedStatement.setString(0, passeggero.getNome) quindi zero la posizione del primo punto di domanda.  
  
poi se updateRows > 0 inserimento avvenuto con successo, altrimenti nessun dato inserito.  
  
Poi stiamo vedendo i resultset, e rappresenta il risultato di una query, nelle slide vediamo il if (resultSet.next()) {  
Quel next  
  
Lo statement però è una cosa, lo statement consente di creare query statiche ed è meno efficiente rispetto agli altri.  
  
Invece il preparedStatement è un altro, ed è più avanzato e supporta più parametri

Gli **Statement** in JDBC sono oggetti utilizzati per eseguire comandi SQL direttamente su un database. Gli **Statement** rappresentano la forma più basilare per inviare query al database, e permettono di eseguire comandi SQL come SELECT, INSERT, UPDATE e DELETE.

**Tipologie di Statement in JDBC**

Ci sono tre principali tipi di statement in JDBC:

1. **Statement**:
   * Esegue comandi SQL direttamente, senza parametri.
   * Esempio: Statement statement = connection.createStatement();
   * Adatto per query semplici ma **meno sicuro** rispetto agli altri tipi, poiché può essere vulnerabile a **SQL Injection** se le query contengono dati esterni non controllati.
2. **PreparedStatement**:
   * Usato per query con parametri.
   * Previene vulnerabilità come la SQL Injection, rendendo il codice più sicuro.
   * Esempio:

java

Copia codice

PreparedStatement preparedStatement = connection.prepareStatement("INSERT INTO utenti (nome, email) VALUES (?, ?)");

preparedStatement.setString(1, "Mario");

preparedStatement.setString(2, "mario@example.com");

1. **CallableStatement**:
   * Utilizzato per chiamare **stored procedure** (funzioni definite nel database).
   * Utile per operazioni complesse o frequenti che è più efficiente eseguire lato database.

**Differenze Principali**

| **Tipo** | **Parametri** | **Sicurezza SQL Injection** | **Quando usarlo** |
| --- | --- | --- | --- |
| Statement | No | Bassa | Per query semplici, senza input esterno |
| PreparedStatement | Sì | Alta | Per query con parametri variabili |
| CallableStatement | Sì | Alta | Per eseguire stored procedure |

**Esempio di Statement**

Ecco un esempio di utilizzo di uno **Statement** in JDBC per eseguire una query SELECT:

java

Copia codice

Statement statement = connection.createStatement();

ResultSet resultSet = statement.executeQuery("SELECT \* FROM utenti");

while (resultSet.next()) {

String nome = resultSet.getString("nome");

String email = resultSet.getString("email");

System.out.println("Nome: " + nome + ", Email: " + email);

}

In sintesi:

* **Statement** è il tipo più semplice, ma presenta rischi di sicurezza.
* Per query sicure e dinamiche, si preferisce **PreparedStatement**.

4o

**Hai detto:**

Lo statement però è una cosa, lo statement consente di creare query statiche ed è meno efficiente rispetto agli altri. Invece il preparedStatement è un altro, ed è più avanzato e supporta più parametri

**ChatGPT ha detto:**

ChatGPT

Esatto! Ecco un riassunto chiaro e organizzato delle differenze tra **Statement** e **PreparedStatement** in JDBC:

**Statement**

* **Cos'è**: Uno strumento di base in JDBC per eseguire query statiche sul database.
* **Caratteristiche**:
  + Esegue query che **non cambiano** (sono statiche).
  + Non supporta parametri, quindi richiede che l'intera query sia già scritta nel codice.
  + **Meno efficiente** rispetto a PreparedStatement, poiché non ottimizza query ripetute.
  + **Vulnerabile a SQL Injection**, perché non consente una separazione sicura tra query e valori dei parametri.
* **Quando Usarlo**: Adatto solo per query semplici, non dinamiche (senza input variabile).

**Esempio di Statement**:

java

Copia codice

Statement statement = connection.createStatement();

ResultSet resultSet = statement.executeQuery("SELECT \* FROM utenti");

**PreparedStatement**

* **Cos'è**: Un’estensione avanzata di Statement, progettata per gestire query dinamiche in modo più sicuro ed efficiente.
* **Caratteristiche**:
  + **Supporta parametri**: usa ? come segnaposto, permettendo l’inserimento di valori in modo sicuro.
  + **Più efficiente** per query ripetitive, poiché può riutilizzare la stessa query con valori diversi.
  + Riduce il rischio di **SQL Injection** poiché gestisce separatamente i parametri dai comandi SQL.
  + **Più performante**: il database può ottimizzare le query preparate se vengono eseguite ripetutamente con valori diversi.
* **Quando Usarlo**: Preferito per query dinamiche con input variabili, specialmente se questi provengono da fonti esterne.

**Esempio di PreparedStatement**:

java

Copia codice

PreparedStatement preparedStatement = connection.prepareStatement("INSERT INTO utenti (nome, email) VALUES (?, ?)");

preparedStatement.setString(1, "Mario"); // Imposta il primo parametro

preparedStatement.setString(2, "mario@example.com"); // Imposta il secondo parametro

preparedStatement.executeUpdate();

**Differenze Chiave**

| **Caratteristica** | **Statement** | **PreparedStatement** |
| --- | --- | --- |
| **Tipo di Query** | Statica | Dinamica |
| **Efficienza** | Meno efficiente | Più efficiente per query ripetute |
| **Sicurezza (SQL Injection)** | Bassa | Alta, grazie alla separazione dei parametri |
| **Supporto ai Parametri** | No | Sì |

**In sintesi**:

* **Usa Statement** per query semplici, statiche e senza input dinamico.
* **Usa PreparedStatement** per query più sicure e dinamiche con input variabile, migliorando anche l’efficienza e prevenendo vulnerabilità di sicurezza come SQL Injecti