Tecnologie di Sviluppo per il Web

Programmazione su Basi di Dati: JDBC

versione 3.1

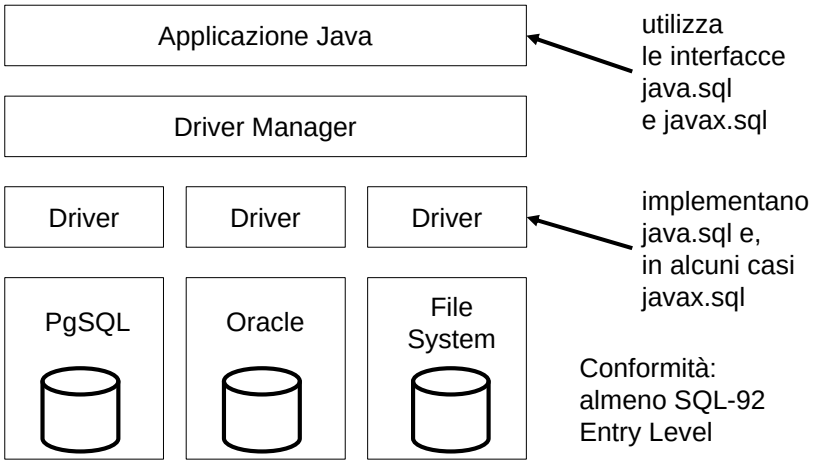
G. Mecca – mecca@unibas.it – Università della Basilicata

Sommario

Introduzione

- JDBC
  - ⇨NON è l'acronimo di Java DataBase Connectivity
  - ⇨esempio di CLI orientata agli oggetti
  - ⇨API standard di Java 2 SDK
  - ⇨arrivata alla versione 3.0 (da J2SE1.4.0)
- Package
  - ⇨java.sql: classi fondamentali
  - ⇨javax.sql: estensioni

Architettura



Introduzione

- I componenti fondamentali di JDBC
  - Interfaccia java.sql.Driver
  - Classe java.sql.DriverManager
  - Interfaccia java.sql.Connection
  - Interfaccia java.sql.Statement
  - Interfaccia java.sql.ResultSet
  - Eccezione java.sql.SQLException

Introduzione

- Nel seguito
  - descriveremo le seguenti operazioni
  - Operazione n.1: Caricare il driver
  - Operazione n.2: Aprire una connessione
  - Operazione n.3: Definire l'istruzione SQL
  - Operazione n.4: Gestire il risultato
  - Operazione n.5: Chiudere la connessione

Driver

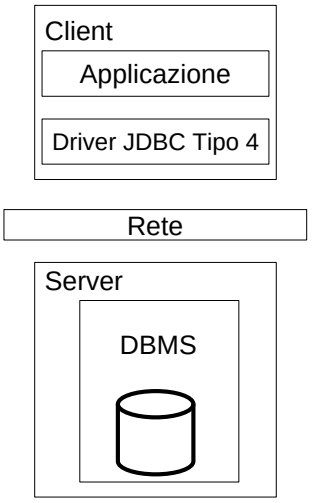
- Per la scrittura di un'applicazione JDBC
  - ⇨è necessario avere a disposizione un driver JDBC
- Esempi
  - ⇨driver JDBC di PgSQL – package org.postgresql
  - ⇨driver jdbc-odbc fornito a corredo dell'SDK dalla Sun

Driver

- Esistono vari tipi di driver
  - ⇨Tipo 1 - "JDBC-ODBC Bridge"
  - ⇨Tipo 2 - Parte Java, Parte API Nativa
  - ⇨Tipo 3 - Driver di Rete Tutto in Java
  - ⇨Tipo 4 - Driver Nativo Tutto in Java
- Nei DBMS moderni
  - ⇨tipicamente driver di tipo 4, tutti scritti in Java
  - ⇨per comodità faremo riferimento anche al driver di tipo 1, fornito a corredo dell'SDK

Driver di Tipo 4

- Driver Nativo Tutto in Java
  - ⇨il driver comunica direttamente con il DBMS
  - ⇨indipendente dalla piattaforma
  - ⇨buone prestazioni
  - ⇨buon compromesso tra prestazioni e semplicità
  - ⇨es: driver di PgSQL

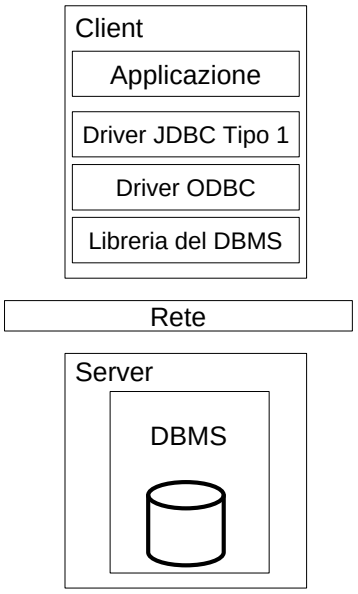


Driver

- Esempio: il driver di PostgreSQL
  - ⇨open source,
  - ⇨disponibile sul sito jdbc.postgresql.org sotto forma di file jar
  - ⇨diverse versioni con nomi diversi
  - ⇨a seconda della versione di PostgreSQL utilizzata
  - ⇨a seconda della versione di JDBC (1.x, 2.x o 3.x) utilizzata

Driver di Tipo 1

- Ponte JDBC-ODBC
  - ⇨usa un driver ODBC
  - ⇨traduce da JDBC in chiamate ODBC
  - ⇨prestazioni scadenti
  - ⇨dip. dalla piattaforma
  - ⇨fornito con l'SDK
  - ⇨classe JdbcOdbcDriver nel package sun.jdbc.odbc



Operazione n.1: Caricare il Driver

- Creare un oggetto della classe Driver
  - ⇨java.sql.Driver d = new org.postgresql.Driver();
- Registrare il driver sul DriverManager
  - ⇨java.sql.DriverManager.registerDriver(d)
- A questo punto
  - ⇨il driver è utilizzabile per ottenere connessioni al DBMS

Caricare il Driver

- Molto spesso, per creare il driver
  - ⇨al posto delle istruzioni precedenti, viene usato il metodo Class.forName
    es: Class.forName(“org.postgresql.Driver”);
- Metodo forName()
  - ⇨metodo statico della classe Class
  - ⇨forza il caricamento della classe specificata come parametro da parte del ClassLoader

Caricare il Driver

- Normalmente, in un Driver
  - ⇨c’è un blocco di inizializzazione statico
  - ⇨il blocco crea il driver e lo registra
- Nota
  - ⇨creando esplicitamente l’oggetto Driver, il blocco viene eseguito comunque
  - ⇨vengono registrati due Driver diversi
  - ⇨il garbage collector ne elimina uno

Caricare il Driver

- Esempio: il driver di PostgreSQL

```
package org.postgresq;
public class Driver implements java.sql.Driver {
    public static final int DEBUG = 0;
    public static final int INFO = 1;
    public static final int WARN = 2;
    public static final int ERROR = 3;
    public static final int FATAL = 4;
    private static int logLevel = FATAL;

    static {
      try {
        java.sql.DriverManager.registerDriver(new Driver());
      } catch (SQLException e)  {e.printStackTrace(); }
    }
   ...
```


Caricare il Driver

- Vantaggi di Class.ForName
  - ⇨posso specificare la classe corrispondente al Driver come parametro al metodo che lo registra
  - ⇨vantaggio: è possibile cambiare il driver senza dover ricompilare il codice
  - ⇨es: usando un file di configurazione

Caricare il Driver

- Attenzione
  - ⇨in questo caso può verificari una ulteriore eccezione: ClassNotFoundException

```
private void creaDriver (String nomeClasseDriver)
                throws SQLException  {
        try {
                 Class.forName(nomeClasseDriver);
        } catch (ClassNotFoundException e) {
            System.out.println(e);
        }
}
```


Operazione n.2: Connessione

- Connessione
  - ⇨oggetto dell'interfaccia java.sql.Connection
  - ⇨viene richiesto al DriverManager
  - ⇨per farlo è necessario specificare a quale server, dbms e base di dati
  - ⇨quale utente e quale password
- URI della connessione
  - ⇨specifica server, porta e base di dati

Connessione

- Sintassi
  - ⇨jdbc:<sottoprotocollo>:<parametri>
- Struttura dell'URI
  - ⇨protocollo principale: jdbc
  - ⇨sottoprotocollo: dipende dal DBMS; serve al driver manager per capire con quale driver si vuole lavorare; es: postgresql, odbc
  - ⇨parametri: servono a fornire le informazioni sulla connessione (host, porta, base di dati)

Connessione

- URI per PostgreSQL
  - ⇨jdbc:postgresql:<baseDati>
  - ⇨jdbc:postgresql://<host>/<baseDati>
  - ⇨jdbc:postgresql://<host>:<porta>/<baseDati>
- Esempi
  - ⇨jdbc:postgresql:automobili
  - ⇨jdbc:postgresql://193.204.22.253/automobili
  - ⇨jdbc:postgresql://127.0.0.1:765/automobili

Connessione

- Creazione della connessione
  - ⇨metodo getConnection() di DriverManager
  - ⇨vari parametri: URI, nomeUtente, password

```
java.sql.Connection conn =
DriverManager.getConnection(
  “jdbc:postgresql:automobili”, “pguser”, “pguser”);
```


Connessione

- Attenzione
  - ⇨la gestione della connessione è uno degli aspetti più delicati della programmazione sul DBMS (>>)
- In questa unità
  - ⇨per semplicità viene creata una nuova connessione per ogni operazione sulla base di dati
  - ⇨ma non si tratta della soluzione più efficiente

Connessione

>> DataSource.java

- La classe DataSource
  - ⇨nell'esempio le connessioni al DBMS vengono ottenute da una classe DataSource
  - ⇨il suo unico compito è servire connessioni alle altre classi che ne hanno bisogno
  - ⇨fornisce un metodo getConnection() per ottenere nuove connessioni e metodi per chiuderle

Operazione n.3: Istruzione SQL

- Oggetto dell'interfaccia Statement
  - ⇨metodo createStatement di Connection
  - ⇨es: Statement st = conn.createStatement();
- Metodi per l'esecuzione dell'istruzione
  - ⇨diversi a seconda che sia necessario effettuare un aggiornamento o una interrogazione

Istruzione SQL

- Istruzione SQL da eseguire
  - ⇨una stringa che può essere costruita dinamicamente
- Aggiornamento
  - ⇨metodo int executeUpdate(String sql);
- Interrogazione
  - ⇨ResultSet executeQuery(String sql);

Istruzione SQL

- Struttura tipica dei metodi di aggiornam.
  - ⇨richiesta di una connessione a DataSource
  - ⇨creazione di un oggetto Statement
  - ⇨preparazione dell'istruzione SQL
  - ⇨esecuzione attraverso lo statement
  - ⇨chiusura dello statement e della connessione – metodo DataSource.close() >>

>> DAOProprietario.java

Istruzione SQL

- Attenzione
  - nella costruzione della stringa SQL è molto facile commettere errori
  - a causa della concatenazione tra stringhe costanti e valori variabili e dei diversi separatori (virgolette, apici ecc.)
  - è opportuno utilizzare regolarmente il logging per verificare che i comandi SQL prodotti siano corretti e interpretare l'eventuale causa delle eccezioni sollevate dal DBMS

Operazione n.4: Gestire il Risultato

- Oggetto dell'interfaccia ResultSet
  - rappresenta la collezione di ennuple restituita da una SELECT
  - metodo boolean next(); per scorrere le ennuple (muove il cursore)
  - metodi getXXX(String attributo) per acquisire i valori degli attributi
  - esempio: int getInt(String attributo);
  - esempio: String getString(String attributo);

Gestire il Risultato

- Ricerca di un proprietario per nome
  - ⇨viene eseguita una SELECT con la condizione che il nome sia quello cercato
  - ⇨viene restituito un ResultSet
  - ⇨il ResultSet viene scandito con il metodo next()
  - ⇨vengono prelevate le informazioni sui proprietari da ciascuna ennupla con getString() e getInt()

>> DAOProprietario.java

Operazione n.5: Chiusura

- Chiusura
  - ⇨è necessario chiudere gli oggetti ResultSet, Statement e Connection per limitare l'occupazione di risorse di rete e del DBMS
- Attenzione
  - ⇨le istruzioni SQL possono generare eccezioni
  - ⇨eccezione SQLException
  - ⇨la chiusura deve essere fatta sia che tutto vada bene, sia che si generino eccezioni

Chiusura

>> DAOProprietario.java
>> DataSource.java

- Di conseguenza
  - ⇨le operazioni sul DBMS vengono racchiuse in un blocco try-catch
  - ⇨alla fine – utilizzando il blocco finally – vengono chiusi tutti gli oggetti
  - ⇨metodi close() di DataSource
- Nota
  - ⇨l'event. eccezione viene catturata e rilanciata per segnalare al controllo il problema

Prepared Statement

- Una tipologia particolare di statement
  - ⇨gli statement preparati
- Prepared statement
  - ⇨si tratta di statement che vengono precompilati una volta per tutte dal DBMS
  - ⇨successivamente è possibile eseguirli più volte specificando parametri diversi
  - ⇨in questo modo le prestazioni dovrebbero migliorare a causa della precompilazione

Prepared Statement

- In JDBC
  - ⇨interfaccia PreparedStatement che estende Statement
- Utilizzo dei prepared statement
  - ⇨due fasi
  - ⇨I fase: creazione e preparazione dello statement
  - ⇨II fase: utilizzo dello statement

Prepared Statement

- I fase: Creazione e preparazione
  - ⇨usando il metodo public PreparedStatement prepareStatement(String query) della classe Connection
  - ⇨bisogna specificare il codice SQL
- Attenzione
  - ⇨lo statement è fatto per essere usato in condizioni diverse e con parametri diversi
  - ⇨per cui il codice della query è “parametrico”

Prepared Statement

- Parametri per lo statement preparato
  - vengono indicati nella query con “?”
  - es: **`“select * from utenti where nomeutente = ?”`**
  - viene utilizzata una notazione posizionale con base 1
  - il primo punto interrogativo è il parametro 1, il secondo il parametro 2 ecc.

Prepared Statement

- II fase: esecuzione
  - prevede che vengano per prima cosa forniti i valori dei parametri e successivamente che venga eseguita la query
- Assegnazione dei parametri
  - metodi setXXX (<numPar>, <valore>) di PreparedStatement
  - es: setString(1, “pinco”)
  - metodo setNull() per i valori nulli

Prepared Statement

- Esecuzione dello statement preparato
  - ⇨metodo int executeUpdate()
  - ⇨metodo ResultSet executeQuery()
- Attenzione alla differenza
  - ⇨a differenza dei metodi di Statement, i due metodi non hanno argomenti
  - ⇨perchè la query SQL è già nota e non deve essere ulteriormente specificata

Prepared Statement

>> DaoUtente

- Esempio: DAOUtente
  - ⇨la ricerca dell'utente viene effettuata utilizzando uno statement preparato

```
try {
    connection = dataSource.getConnection();
    statement = connection.prepareStatement("select * from utenti where nomeutente = ?");
    statement.setString(1, nomeUtente);
    resultSet = statement.executeQuery();
    if (resultSet.next()){
            utente = new Utente();
            utente.setNomeUtente(resultSet.getString("nomeutente"));
            ...
    }
}
```

Prepared Statement

>> it.unibas.prestazioni

- Vantaggi concreti nell'utilizzo dei p. s.
  - dipende molto dal DBMS
  - cosa vuol dire "precompilare" ?
- Un test di prestazioni
  - Core Servlet and Java Server Pages
  - la stessa query ripetuta 40 volte con e senza prepared statement
  - su Oracle in rete: miglioramento del 30%
- Un altro test di prestazioni
  - it.unibas.prestazioni

Prepared Statement

- Attenzione
  - l'istruzione deve essere preparata in anticipo
  - ha senso solo se viene eseguita ripetutamente
- Esempio: ricerca di proprietario per nome
  - preparo l'istruzione una volta per tutte all'inizio dell'applicazione
  - all'occorrenza specifico il valore del parametro con il metodo ed eseguo la query

Prepared Statement

- Ma...
  - ⇨gli statement sono collegati alle connessioni
  - ⇨se le connessioni vengono chiuse ogni volta NON c'è nessun vantaggio significativo
- Di conseguenza
  - ⇨l'utilizzo degli statement preparati a scopo di prestazioni ha senso solo nel caso in cui venga utilizzato un pool di connessioni
  - ⇨ma ci sono altri utilizzi (>>)

Riassumendo

- Introduzione
- Driver
- Caricare il Driver (DriverManager)
- Connessione (Connection)
- Istruzione SQL (Statement)
- Gestire il Risultato (ResultSet)
- Chiusura
- Prepared Statement

Un Esempio

- La base di dati

```
create table utenti (
  nomeUtente char(10)
     not null primary key,
  nome varchar(50),
  password varchar(10),
  ruolo varchar(50)
)
```

```
create table proprietari (
  codiceFiscale char(16)
          not null primary key,
  nome varchar(50) not null,
  cittaDiResidenza varchar(50),
  annoPatente integer
)

create table automobili (
  targa char(7) not null
                primary key,
  modello varchar(50),
  cilindrata integer,
  proprietario char(16)
          not null references
        Proprietari(codiceFiscale)
)
```


Termini della Licenza