JDBC

Query semplici

e

Query parametriche

Creare applicazioni indipendenti dal DBMS

- Le API JDBC consentono alle applicazioni Java di inviare query SQL al server di database e di gestirne i risultati
- I DBMS in commercio non sempre rispettano in pieno lo standard ANSI
  - “per difetto”: es. non forniscono Stored Procedure, o OUTER JOIN
  - “per eccesso”: aderiscono allo standard ma estendono SQL con funzionalità proprie

Creare applicazioni indipendenti dal DBMS

- Ogni istruzione può arrivare al DBMS sottostante senza che JDBC effettui alcun controllo, sarà il DBMS a segnalare situazione di errore
- Affinchè un driver JDBC possa essere definito *JDBC-compliant* è necessario che supporti lo standard ANSI-SQL 2 Entry Level 1992
- Se si utilizzano solo query ANSI-SQL 2, l'applicazione funzionerà con tutti i driver JDBC-compliant e quindi con i più diffusi DBMS

Statement SQL con JDBC

Query con JDBC

- Tre diverse classi per inviare le query:
  - Statement
    - Un oggetto Statement può essere creato con il metodo *createStatement* di Connection
    - E' di solito usato per inviare query semplici che non utilizzano parametri
  - PreparedStatement
    - Un oggetto PreparedStatement può essere creato con il metodo *prepareStatement* di Connection
    - Estende le potenzialità dell'interfaccia Statement da cui deriva
      - Consente di specificare query parametriche (IN)
      - E' più efficiente: la query sarà pre-compilata per usi futuri

Query con JDBC (2)

- CallableStatement
  - Un oggetto CallableStatement può essere creato con il metodo *prepareCall* di Connection
  - Viene utilizzato per invocare le Stored Procedure che risiedono sul server di database
  - Possono essere utilizzati parametri IN, OUT e INOUT

Interfaccia Statement

- Stabilita la connessione, si può creare un oggetto Statement:

  Statement *Connection.*createStatement();

- Tre diversi metodi per inviare uno statement SQL:
  - *executeUpdate*
  - *executeQuery*
  - *execute*

executeUpdate

- `int executeUpdate(String sql);`
- Consente di inviare query SQL di tipo DDL e DML
  - CREATE TABLE, DROP TABLE, INSERT, UPDATE; DELETE
- Il nome deriva dal fatto che è utilizzato con query di aggiornamento
  - Restituisce il numero di righe aggiornate
  - Per le istruzioni DDL che non operano su righe (CREATE TABLE) restituisce 0
- Con un insieme di chiamate a *executeUpdate* possiamo costruire lo schema di un DB, tabelle e vincoli d'integrità.
- E' possibile inviare statement per assegnare diritti amministrativi

Esempio:creazione tabella

```
import java.sql.*;
public class ExecuteQuery {
   //Oggetto Connection
   Connection con = null;
   Statement st = null;
   public static void main(String[] args) {
        ExecuteQuery eq = new ExecuteQuery();
        // Connessione al database
        try {
             Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
             String url = "jdbc:odbc:DSNTEST";
             eq.con = Drivermanager.getConnection(url);
             eq.st = con.createStatement();
             System.out.println("Connessione OK");
       }
        catch (Exception e) {
          System.out.println("Connessione Fallita");
          e.printStackTrace();
          System.exit(1);
         }
       //Creazione della tabella Articolo
       eq.createTableArticolo();
```

Esempio:creazione tabella

```
  private void createTableArticolo() {
    try {

      String sqlCommand =
        "Create Table articolo " +
        "(codice CHAR(15) CONSTRAINT v PRIMARY KEY," +
       " descrizione CHAR(50), " +
        " genere      CHAR(20), " +
       " prezzo      INTEGER)";
     st.executeUpdate(sqlCommand);
    }
    catch (SQLException e) {
     System.out.println("Non posso creare tab articolo");
        e.printStackTrace();
        System.exit(1);
     }
}
```

Esempio:aggiungere righe

```
//esecuzione di più query con lo stesso Statement
private void addArticolo() {
  try {

     String sqlCommand =
     "INSERT INTO articolo VALUES ( " +
     /* Codice */    "'566-78-09'," +
     /* Descrizione*/ "'TV Color Sony 16p'," +
     /* Genere */      "'TV Color'," +
     /* Prezzo  */      "400"+
     ")";
     st.executeUpdate(sqlCommand);
     sqlCommand =
     "INSERT INTO articolo VALUES ( " +
     /* Codice */    "'565-79-91'," +
     /* Descrizione*/ "'Eriksson GF 768'," +
     /* Genere */      "'Telefono'," +
     /* Prezzo  */      "200"+
     ")";
     st.executeUpdate(sqlCommand);
}
  catch (SQLException e) {
     System.out.println("Errore nell'inserimento");
     e.printStackTrace();
     System.exit(1);
   }
}
```

Esempio:modificare righe

```
//aumenta del 20% il prezzo degli articoli con codice che
// inizia per 566
private void updateArticolo() {
  try {

    String sqlCommand =

      "UPDATE articolo " +
      "SET prezzo=prezzo*1.2 " +
      "WHERE codice LIKE '566%'";

    int num = st.executeUpdate(sqlCommand);
    System.out.println("Articoli Aggiornati: " + num);
    }
  catch (SQLException e) {
    System.out.println("Non posso aggiornare la
tabella");
    e.printStackTrace();
    System.exit(1);
   }
}
```

Esempio: eliminare righe

```
//elimina dalla tabella gli articoli di genere 'TV
color'
private void deleteArticolo() {
  try {

    String sqlCommand =

      "DELETE FROM articolo " +
      "WHERE genere = 'TV color'";

    int num = st.executeUpdate(sqlCommand);
    System.ou.println("Articoli Eliminati: " + num);
    }
  catch (SQLException e) {
    System.out.println("Non posso eliminare le righe");
    e.printStackTrace();
    System.exit(1);
   }
}
```

executeQuery

- Nell'interfaccia Connection è definito il metodo

```
Resultset executeQuery(String sql);
```

- Consente di inviare query di selezione che restituiscono un insieme di righe
  - SELECT, JOIN
- Vincolo: solo query costanti, non possono essere specificati parametri
- L'istruzione SQL è specificata dall'argomento `String sql`.
- Il risultato sarà inserito in un oggetto Resultset, e potrà essere gestito nel modo che l'applicazione vorrà: potrà essere visualizzato, inserito in una tabella, inviate via rete ad un altro sistema, ecc

Esempio: recuperare righe

```
//recupera dalla tabella Articolo tutte le righe il cui
// genere sia 'TV Color' e il prezzo inferiore a 200 euro
private void selectArticolo() {
  try {
     String sqlCommand =

       "SELECT * FROM articolo WHERE" +
       "genere='TV Color' AND " +
       "prezzo<200";

     Resultset rs = st.executeQuery(sqlCommand);

     while(rs.next()) {
       System.out.println(
             rs.getString("codice") + " " +
             rs.getString("descrizione") + " " +
             rs.getString("prezzo"));
     }
  }
   catch (SQLException e) {
     e.printStackTrace();
     System.exit(1);
    }
}
```

Ulteriori metodi di Statement: Rilasciare le risorse

- `void close();`
  - consente di rilasciare le risorse allocate all'oggetto Statement,
  - quando ci si rende conto che un serve più all'applicazione senza dovere aspettare l'attivazione del Garbage Collector
  - ogni successivo tentativo di utilizzare un oggetto Statement fallirà

Ulteriori metodi di Statement:

- `void setMaxRows(int max);`
  - consente di limitare il numero di righe restituite da una query di selezione
  - Esempio di necessità: seleziona gli abbonati Esposito a Napoli ➔ crash del S.O., applicazione “ out of memory”, comunque non utile all’utente
  - Utilizzando questo metodo l’applicazione segnalerà all’utente il caso in cui la query retituisce un numero di righe superiore a quello consentito ➔ l’utente potrà riformulare la query
  - `int getMaxRows();`

  Consente di conoscere il valore max impostato da setMaxRows, (per default max = 0 → No limite)

Esempio: limitare le righe

```
/*il metodo prende come parametri un oggetto Connection e una query di
selezione, se la risposterà raggiungerà il limite di 30 occorrenze sarà
visualizzato un messaggio */
void executeLimitedQuery(Connection con, String sql) {
  try {
      Statement st = con.createStatement();
      st.setMaxRows(30);

      Resultset rs = st.executeQuery(sql);
      int i=1;

      for(i=1;rs.next(); i++) {
               System.out.println(
               rs.getString("codice") + " " +
               rs.getString("descrizione") + " " +
               rs.getString("prezzo"));

      if (i==30)
       System.out.println("Attenzione Raggiunto il Limite");
       }
      }
      catch (SQLException e) {
      e.printStackTrace();
      System.exit(1);
    }
}
```

Ulteriori metodi di Statement:

- `void setQueryTimeout(int seconds);`
  - consente di limitare il periodo di tempo che intercorre tra l'invio della query e la risposta del DBMS
  - Esempio di necessità: evitare lunghe attese quando la rete ha malfunzionamento o la query è troppo complessa
  - `int getQueryTimeout ();`

  Consente di conoscere il valore max impostato da setQueryTimeout, per default seconds = 0 → non esistono limiti di tempo

Query Parametriche

Query parametriche: l'interfaccia PreparedStatement

- L'interfaccia PreparedStatement è lo strumento che consente la gestione di query parametriche
- Rappresenta estensione di Statement ➔ ne eredita tutte le funzionalità.
- Differenze con l'interfaccia Statement:
  - Le istruzioni SQL contenute in un oggetto PreparedStatement possono contenere parametri input, perché l'interfaccia fornisce un insieme di metodi per assegnare valori reali a tali parametri
  - Le istanze di PreparedStatement contengono query SQL precompilate
    - prestazioni più elevate, soprattutto per eseguire la query più volte (Access non supporta query precompilate!)

Query parametriche

```
SELECT codice, descrizione, prezzo
FROM articolo
WHERE genere = ? AND prezzo <?
```

- I valori rappresentati da ? non sono noti ma l'applicazioni li fornirà prima di inviare la query
- Gli oggetti PreparedStatement forniscono metodi per ricevere una query parametrica e sostituire i ? con valori reali
- Ciò non accade con gli oggetti Statement, ma è possibile simulare tale comportamento
  - costruendo run-time il comando SQL quando si hanno a disposizione i dati completi

Esempio: Query parametriche

- Supponiamo di avere una maschera per la ricerca degli articoli:
  - Utilizzando la Combo-Box possiamo selezionare il genere dell'articolo
  - Utilizzando un Text-Field possiamo inserire il prezzo max
- Nella lista dovrà apparire l'insieme di tutti gli articoli del tipo scelto e con un prezzo minore di quello inserito.
  - Se non è stato inserito alcun prezzo o un valore non consentito, saranno visualizzati tutti gli articoli di quel genere
- Quando l'utente preme Cerca l'applicazione avrà i dati necessari per creare la stringa e inviare la query.

Esempio: query parametriche

```
private void inviaQuery() {

   // Composizione del comando SQL

   int prezzo = 0;
   try {
      prezzo = Integer.parseInt(TxtPrezzo.getText());
    //TxtPrezzo rappresenta l'istanza del Text-Field
      contenente il prezzo
    }
   catch (Exception e) {
     txtPrezzo.setText(" ");
    }
   String articolo = cmbArticolo.getSelectedItem();
   //cmbArticolo rappresenta l'istanza del Combo-Box
   realtiva al genere
   StringsqlSelect =
       "SELECT codice, descrizione, prezzo " +
       "FROM articolo";
```

Esempio: query parametriche

```
String sqlWhere = “”;
if (prezzo >0)
   sqlWhere =
        “WHERE prezzo<“ + prezzo +
        “ AND genere=‘” + articolo +
“’”;
else
   sqlWhere =
   “WHERE genere =‘” + articolo + “’”,
String sqlCommand = sqlSelect + “ “ +
sqlWhere;
```

Esempio: query parametriche

```
// Invio Query
Statement st = null;
try {
    st = con.createStatement();
    Resultset rs = st.executeQuery(sqlCommand);
      }

// Visualizzazione dei Risultati

lstArticolo.removeAll();
while(rs.next()) {
     String strRiga =(
             rs.getString("codice") + " " +
             rs.getString("descrizione") + " " +
             rs.getInt("prezzo"));

     lstArticolo.add(strRiga);
}

 catch (SQLException e) {
     e.printStackTrace();
     Return;
  }
}
```

Interfaccia PreparedStatement

- Riscriviamo l'esempio precedente utilizzando al posto di Statement l'interfaccia PreparedStatement
  - L'applicazione costruisce una query parametrica concatenando il comando SQL con i parametri provenienti dall'interfaccia utente
  - Il codice rimane invariato tranne per il metodo *inviaQuery*

Query parametriche: l'interfaccia PreparedStatement

- Ogni oggetto di tipo PreparedStatement è associato ad un'unica query parametrica
  - L'associazione avviene nel momento in cui l'oggetto è costruito
  - Poi sarà possibile impostare i parametri ed eseguire la query più volte
- Creare un oggetto PreparedStatement:
  - Si utilizza il metodo prepareStatement dell'interfaccia Connection che prende in input la stringa che rappresenta la query (Selezione o DML)
- Esempio:

```
PrepareStatement pst =
  con.prepareStatement(
      "SELECT * " +
      "FROM articolo " +
      "WHERE genere =? AND prezzo < ?"
  );
```

Passaggio dei parametri

- L'interfaccia *PreparedStatement* mette a disposizione un insieme di metodi *setXXX* per assegnare alla query i valori per gli attributi (sostituendo i vari ? con i valori effettivi)
  - *setXXX* per ogni tipo base di Java, per la classe String e per le date
  - Tutti i metodi *setXXX* hanno in input
    - la posizione ordinale del parametro all'interno del comando SQL e
    - il valore da assegnare

Esempio: setInt e setString

```
// Preparazione  della Query
PreparedStatement st = null;

String querySQL =
    "SELECT codice, descrizione, prezzo " +
    "FROM articolo" +
    "WHERE genere = ? AND prezzo < ? ";
```

/* assegnazione dei valori ai parametri tramite i metodi setString e SetInt di PreparedStatement*/

```
try {
   st = con.prepareStatement(querySQL);
   st.setString(1, genere);
   st.setInt(2, prezzo);
```

Interfaccia PreparedStatement

- L'esecuzione di una query "preparata" sarà delegata ai metodi della classe *executeXXX*
  - È tecnicamente possibile usare i metodi dell'interfaccia Statement:

```
int executeUpdate(String sql);
ResultSet executeQuery(String sql);
Boolean execute(String sql);
```

  - Non si utlizzano perché non ha senso specificare di nuovo la stringa *sql* visto che l'oggetto è già associato ad una query. I metodi utilizzati sono sempre tre e non vogliono parametri:

```
int executeUpdate();  /*query di aggiornamento*/
ResultSet executeQuery();/*  query di selezione*/
Boolean execute();  /*per quelle che restituiscono più di un ResultSet*/
```

Esempio: query parametriche

```
private void inviaQuery() {

   /* Trasferimento del calore di genere e prezzo dalla GUI alle
   variabili locali genere e prezzo */

   int prezzo = 0;

   try {
      prezzo = Integer.parseInt(TxtPrezzo.getText());
    /*TxtPrezzo rappresenta l'istanza del Text-Field contenente il prezzo*/
    }
   catch (Exception e) {
     txtPrezzo.setText(" ");
     prezzo = 999999;
*/ se la conversione del prezzo non va a buon fine, prezzo è forzato a un valore molto alto*/
    }

   String genere = cmbArticolo.getSelectedItem();
   /*cmbArticolo rappresenta l'istanza del Combo-Box relativa al genere*/
```

Esempio: query parametriche

```
// Preparazione  della Query
PreparedStatement st = null;

String querySQL =
    "SELECT codice, descrizione, prezzo " +
    "FROM articolo" +
    "WHERE genere = ? AND prezzo < ? ";
```

/* assegnazione dei valori ai parametri tramite i metodi setString e SetInt di PreparedStatement*/

```
try {
   st = con.prepareStatement(querySQL);
   st.setString(1, genere);
   st.setInt(2,prezzo);
```

/* Invocazione del metodo executeQuery*/

```
ResultSet rs = st.executeQuery();
}
```

Esempio: query parametriche

```
/* Visualizzazione dei Risultati (uguale al caso
Statement)*/

lstArticolo.removeAll();
while(rs.next()) {
    String strRiga =(
          rs.getString("codice") + " " +
          rs.getString("descrizione") + " " +
          rs.getInt("prezzo"));

    lstArticolo.add(strRiga);
}

 catch (SQLException e) {
    e.printStackTrace();
    Return;
  }
}
```

Query precompilate

- Per come è scritto il codice di *inviaQuery* l'applicazione non trae vantaggio dal fatto che la query è precompilata:
  - Il metodo è invocato ad ogni pressione del tasto "Cerca" e ad ogni chiamata sarà costruito un oggetto *PreparedStatement* diverso
  - Ciò è dovuto al fatto che la creazione dell'oggetto *PreparedStatement* e l'esecuzione della query avvengono nello stesso metodo
  - Per ovviare:
    - l'istanza di *PreparedStatement* (*st*) può essere dichiarato come attributo privato della classe e non del metodo ➔ l'oggetto *PreparedStatement* sarà costruito assieme alla classe (nel costruttore) e non ad ogni chiamata di *inviaQuery*

Query parametriche: l'interfaccia PreparedStatement

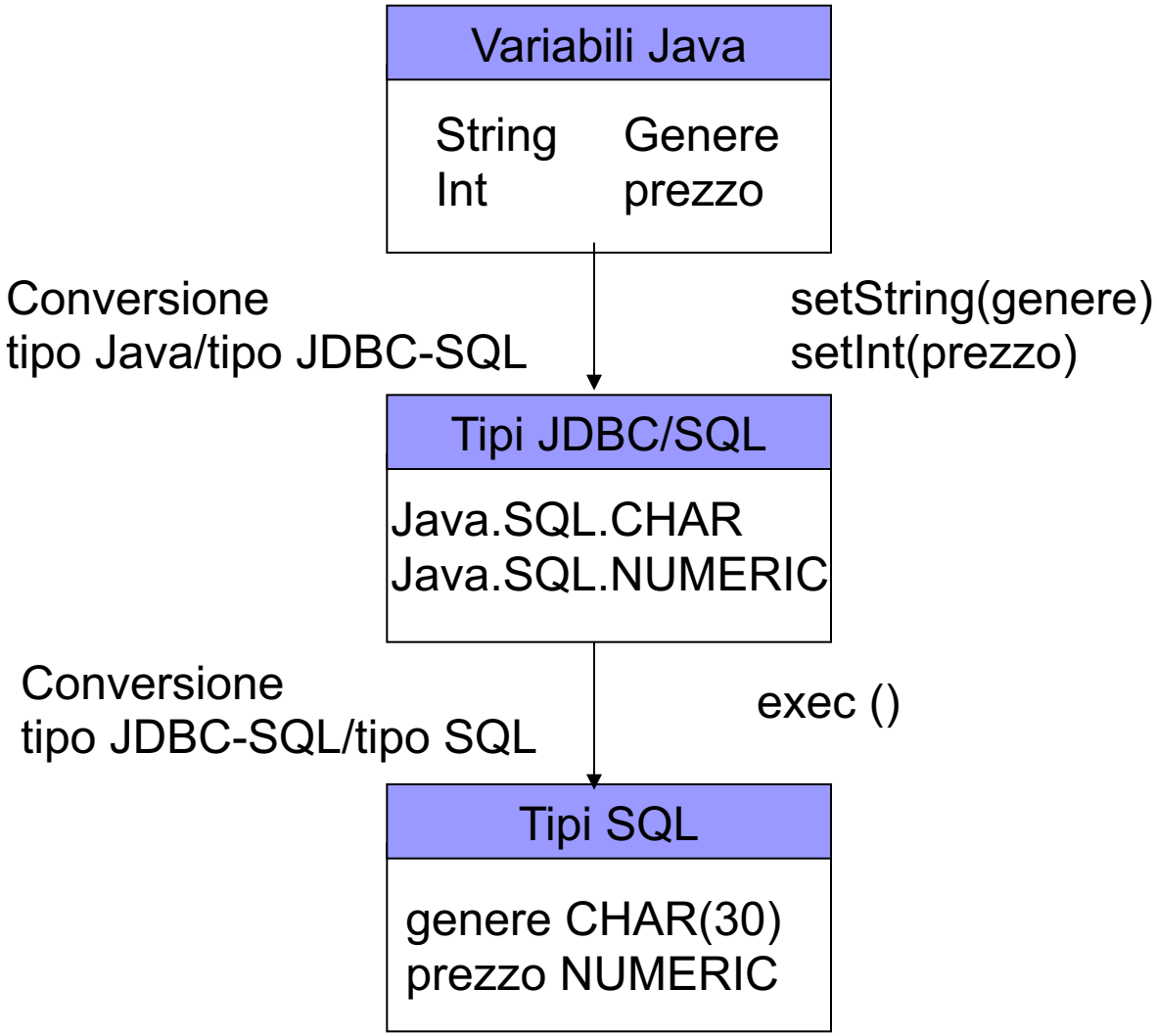
- Esempio Query DML:

```
PrepareStatement pst =
  con.prepareStatement(
     "UPDATE articolo " +
     "SET prezzo = prezzo*1.2 " +
     "WHERE genere =?
  );
```

Conversione da tipo Java a tipo SQL

- Una variabile Java passata come argomento a uno dei metodi della famiglia *setXXX* deve subire una serie di operazioni preliminari per poter essere correttamente interpretato da un DBMS:
  - I metodi setXXX a partire dalle variabili effettuano una conversione tra il tipo Java e il tipo JDBC/SQL
  - I tipi JDBC/SQL definiti nella classe java.sql.Types fanno da ponte verso i tipi SQL veri e propri. Ciò è necessario per garantire l'indipendenza dai tipi SQL definiti dai vari DBMS
  - Il driver JDBC converte i valori JDBC/SQL nei corrispondenti tipi SQL del DBMS usato

Conversione da tipo Java a tipo SQL



SetObject

- I metodi *setXXX* di *PreparedStatement* presuppongono che il tipo di un attributo sia noto al momento della compilazione
- In alcune circostanze il tipo è noto solo a tempo di esecuzione
- Come ovviare?
  - Utilizzando il metodo setObject

```
void setObject (int parameterIndex, Object x);
```

consente di impostare il parametro x come oggetto generico.

Successivamente il driver JDBC convertirà x nel tipo che il DBMS si aspetta.

Si verificherà un ClassCastException nel caso x non sia di un tipo compatibile con quello del relativo attributo della tabella

Esempio: setObject

```
void query(Object prezzo) throws SQLException {

String sqlcommand =
    "SELECT * FROM  articolo" +
    "WHERE prezzo > ? ";
PreparedStatement st =
con.prepareStatement(sqlCommand);
st.setObject(1, prezzo);
```

/*prima di inviare la query, JDBC cercherà di convertire l'oggetto generico prezzo nel tipo SQL NUMERIC, se questo non è possibile si verificherà un'eccezione*/

```
st.execute();

}
```

setObject

- Al metodo setObject non è possibile passare argomenti appartenenti a tipi java fondamentali
  - Errore: `st.setObject(1, 34000);`
  - 34000 è di tipo fondamentale (int)
  - Per ovviare usare le classi wrapper:

```
st.setObject(1, new Integer (34000));
```

SetObject: altra versione del metodo

- Il parametro aggiuntivo è l'indicazione esplicita del tipo JDBC/SQL che si intende utilizzare per la conversione

```
void setObject (int parameterIndex, Object x, intsqlType);
```

- Esempio

```
void query(Object prezzo) throws SQLException {

   String sqlcommand =
        "SELECT * FROM  articolo" +
        "WHERE prezzo > ? ";
   PreparedStatement st = con.prepareStatement(sqlCommand);
   st.setObject(1, prezzo, Types.Numeric);
/*si comunica al driver che si intende convertire il parametro prezzo nel tipo
   NUMERIC*/
   st.execute();
}
```

SetNull

- Serve ad impostare su Null il valore dei parametri

```
void setNull (int parameterIndex, intsqlType);
```

- Esempio:

/*recuperare dalla tabella articolo tutti gli articoli per i quali non è stato ancora impostato il prezzo*/

```
void query(Object prezzo) throws SQLException {

   String sqlcommand =
        "SELECT * FROM  articolo" +
        "WHERE prezzo > ? ";
   PreparedStatement st = con.prepareStatement(sqlCommand);
   st.setNull(1, Types.Numeric);
```

/*è obbligatorio specificare il tipo JDBC/SQL*/

```
   st.execute();
}
```

SetBinaryStream: Input da Stream

- Serve ad assegnare un valore ad un parametro prelevandolo da file
- Utile soprattutto quando la quantità di informazione da trasferire è elevata
- Signature:

```
void setBinaryStream (int
  parameterIndex, inputStream x, int
  length);
```

  - Dove x è il file da cui prelevare
  - Occorre specificare la lunghezza perché alcuni server di database devono conoscere a priori la quantità esatta di caratteri prima di eseguire la query

SetBinaryStream: Input da Stream

- Esempio:

/*le informazioni assegnate al parametro *stuff* provengono dal file “tmp/data” e dovranno essere compatibili con il tipo SQL di *stuff**/

```
File file = new File(“/tmp/data”);
Int fileLength = file.length();
InputStream fin = new FileInputStream(file);

PreparedStatement st = con.prepareStatement(
   UPDATE Table5 SET stuff = ? WHERE index = 4”);

   st.setBinaryStream(1, file, fileLength);
   st.executeUpdate();
}
```

- Se lo stream è di tipo ASCII o UNICODE è possibile usare getAsciiStream o getUnicodeStream