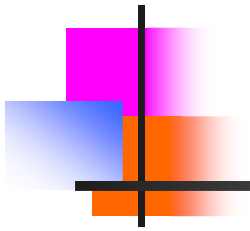


Interfaccia JDBC

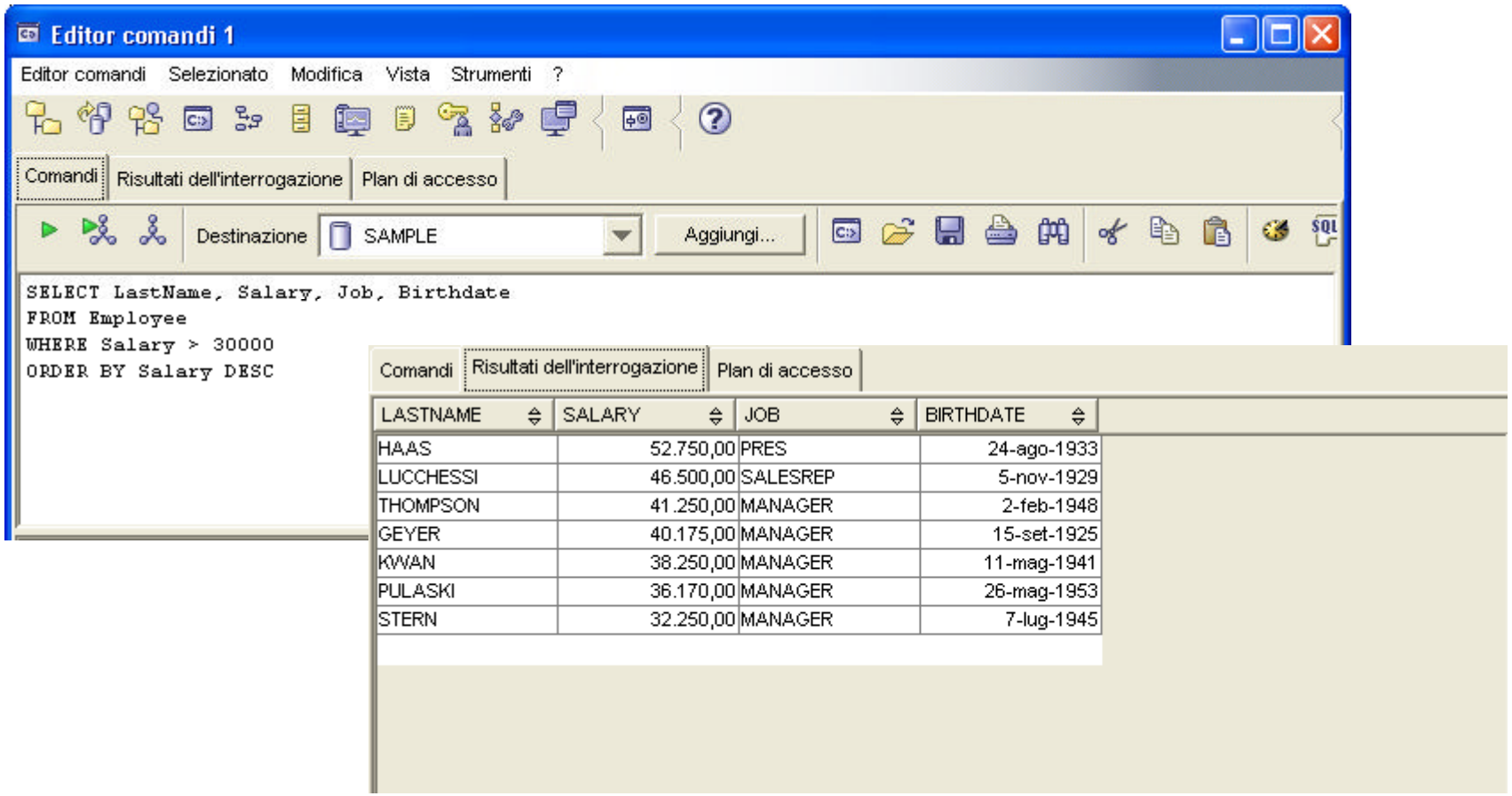
Sistemi Informativi L-A

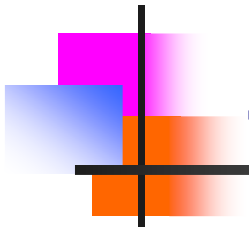
Home Page del corso:
http://www-db.deis.unibo.it/courses/SIL-A/

Versione elettronica: JDBC.pdf

…come usare SQL (1)

- Le istruzioni SQL possono essere eseguite interattivamente…



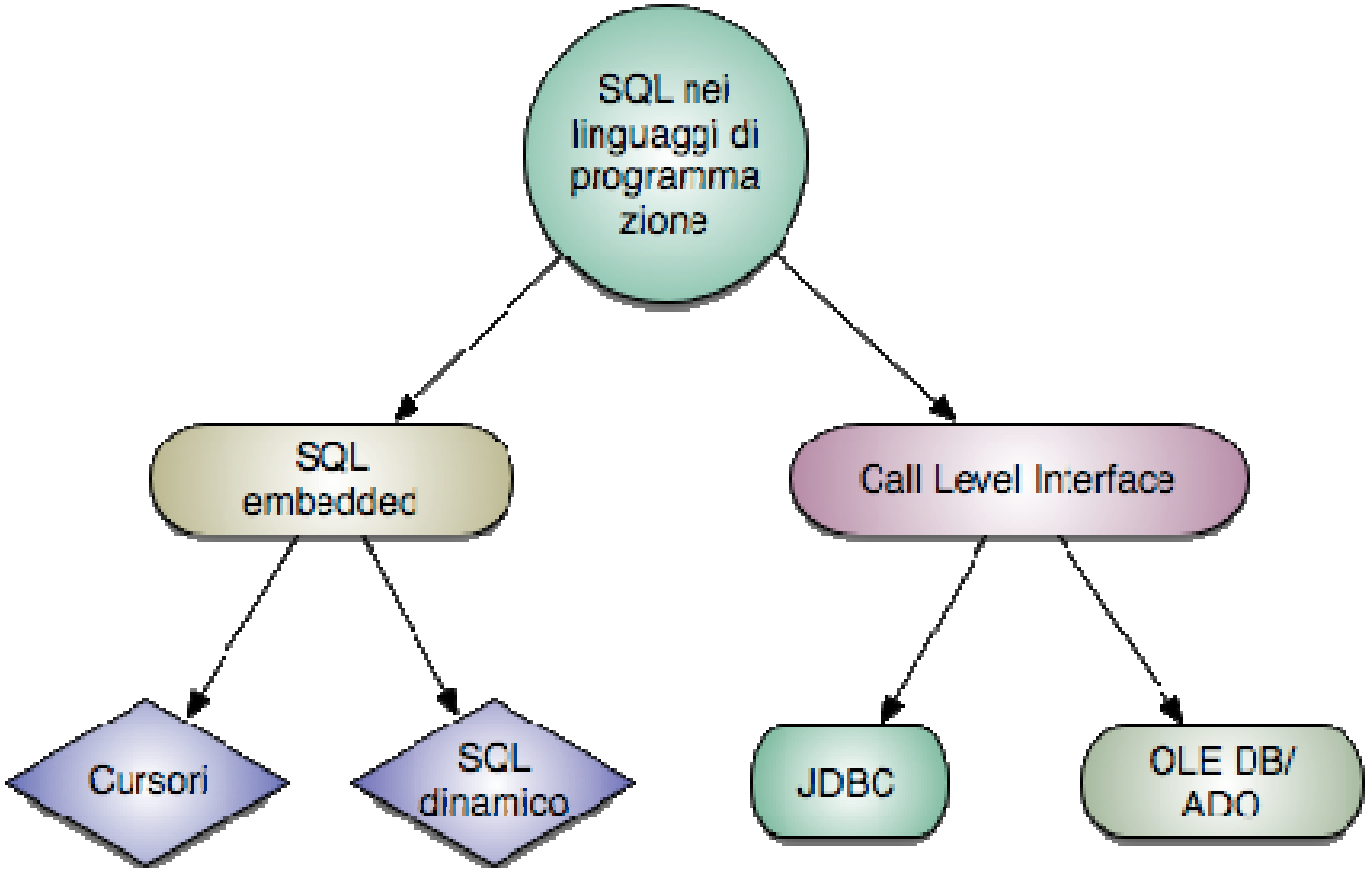


…come usare SQL (2)

- … o inserendole nel codice di un'applicazione scritta in un linguaggio di programmazione "ospite" (ad es. Java)

```
System.out.println("Retrieve some data from the database...");
Statement stmt = con.createStatement();
ResultSet rs = stmt.executeQuery("SELECT * FROM EMPLOYEE");
// display the result set
while (rs.next()) {
            String a = rs.getString(1);
            String str = rs.getString(2);
            System.out.print(" empno= " + a);
            System.out.print(" firstname= " + str);
            System.out.print("\n");
          }
          rs.close();
          stmt.close();
```

SQL nei linguaggi di programmazione

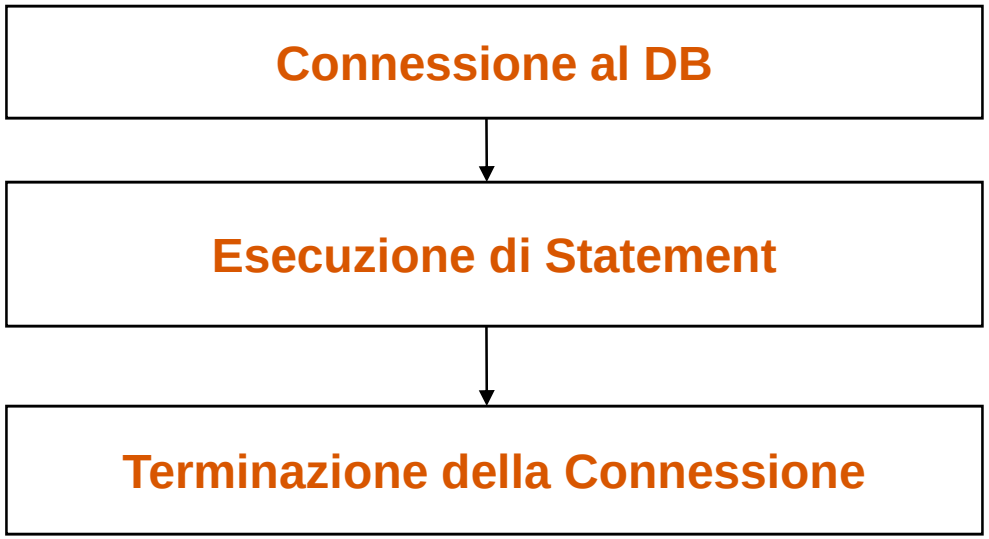


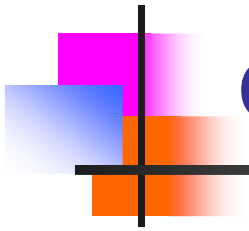
Diverse soluzioni a confronto

Nome	Descrizione	Pro :)	Contro :(
Sql embedded	Le istruzione SQL sono introdotte direttamente all'interno del listato programma, distinte da un separatore (exec sql <istruzione> ;)	Cursori per gestire risultati composti da più tuple Sql dinamico per ottenere flessibilità	Necessità di un preprocessore e di un supporto DBMS-piattaforma-linguaggio-compilatore
OLE DB/ ADO	Soluzione proprietaria Microsoft che consente, grazie all'uso di driver specifici, di interfacciare il linguaggio di programmazione con il DBMS	Integrato in Windows Driver offerti dai maggiori produttori di DBMS Interfacciamento con altri tipi di dato (documenti, mailbox ecc...)	Dialetto SQL ristretto Proprietario Utilizzato solo su linguag-gi e su piattaforme Microsoft
JDBC	Interfaccia fra il mondo java ed i diversi DBMS. Utilizza dei driver specifici ma offre anche un ponte con ODBC (non è vero l'inverso)	Java, multipiattaforma, codice aperto. Disponibile per qualsiasi DBMS grazie anche al ponte ODBC	Come ODBC richiede la disponibilità di un driver offerto dal DBMS

DB2 e sviluppo software di base

- DB2 mette a disposizione interfacce di programmazione (application program interface, API) per i principali linguaggi di programmazione, quali Java, C/C++, VB…
- Un esempio d'uso di base di DB2 può essere quello di costruire applicazioni client che, conoscendo la struttura delle tabelle del database (DB) residente sul server e utilizzando un API per connettersi al DB stesso, interrogano e/o aggiornano il contenuto delle tabelle mediante statement SQL

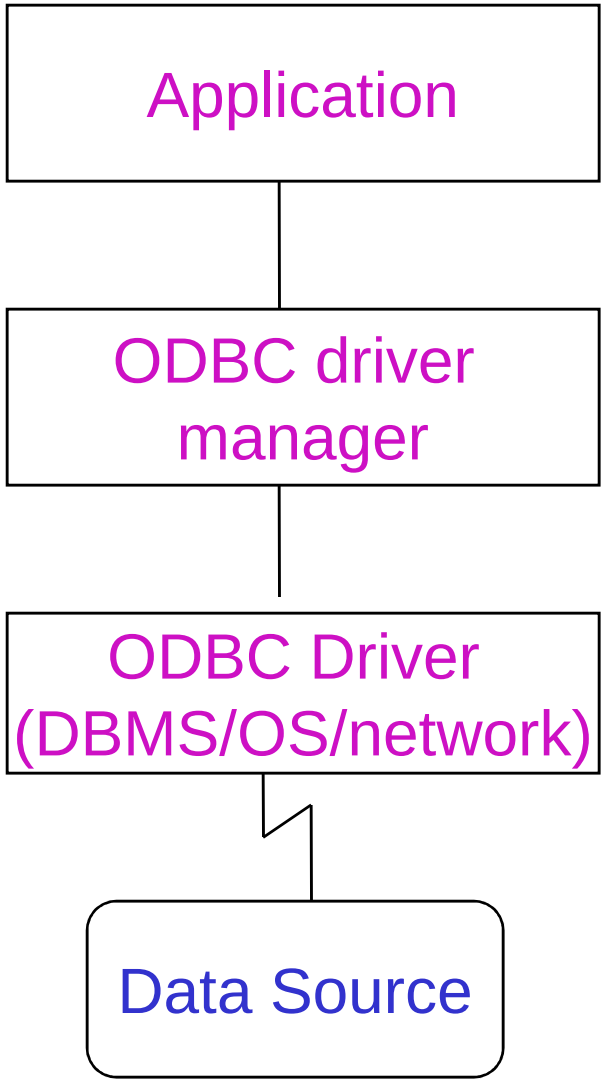


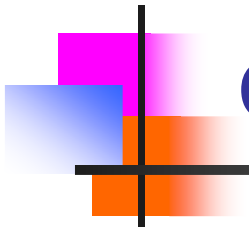


Cos'è ODBC?

- Acronimo di Open Database Connectivity
- API standard definito da Microsoft nel 1992
- Permette l'accesso a dati residenti in DBMS diversi (Access, MySQL, DB2, Oracle, …)
- Permette ai programmatori di formulare richieste SQL che accederanno a dati relativi a DB distinti senza dover conoscere le interfacce proprietarie di ogni singolo DB
- Gestisce richieste SQL convertendole in un formato comprensibile al particolare DBMS

Architettura ODBC

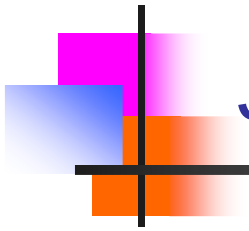




Cos'è JDBC?

- Cos'è?
  - API standard definita da Sun Microsystems (…Sun assicura che JDBC NON è acronimo di Java Database Connectivity… ma è un semplice marchio registrato!! ☺)
  - Rappresenta la controparte Java di ODBC
  - È un API Java di connessione a dati residenti in DB relazionali
  - Consiste di un insieme di classi e interfacce scritte nel linguaggio di programmazione Java (`package java.sql`)
  - Fornisce ai programmatori uno strumento di sviluppo di tool/DB

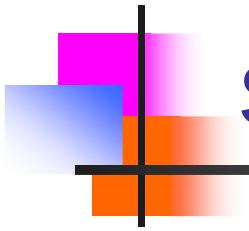
- Cosa fa?
  - Stabilisce una connessione a un DB
  - Invia istruzioni SQL
  - Processa i risultati



JDBC vs ODBC

- ODBC non è appropriato per un uso diretto dal linguaggio Java perché usa interfacce scritte in linguaggio C
- Una traduzione da API C ODBC a API Java non è raccomandata
- Una API Java come JDBC è necessaria per permettere una soluzione Java “pura”

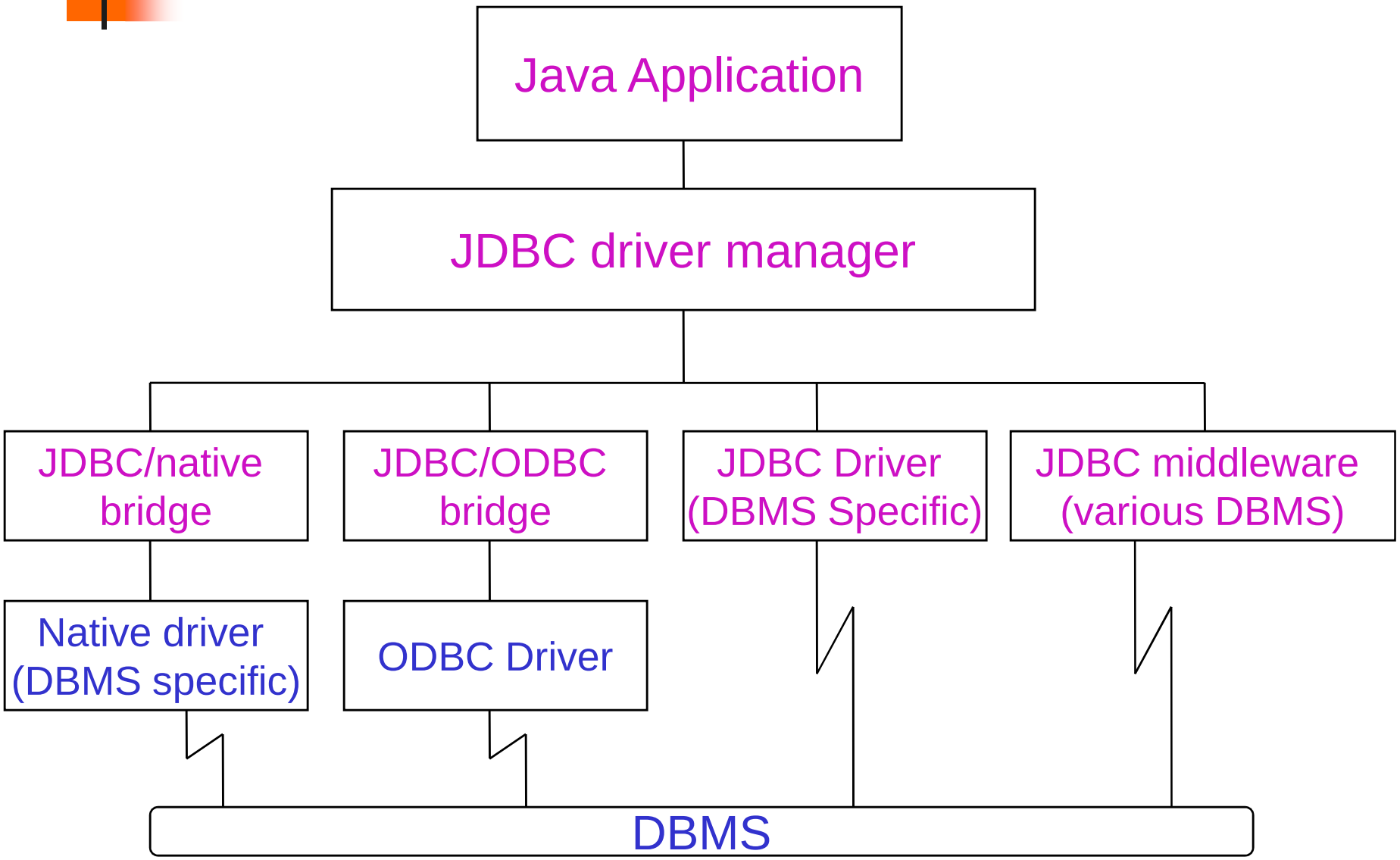
- ODBC è solitamente usato per applicazioni eterogenee
- JDBC è normalmente utilizzato da programmatori Java per connettersi a DB relazionali
- Attraverso un piccolo programma “bridge” è possibile usare l’interfaccia JDBC per accedere a DB accessibili via ODBC

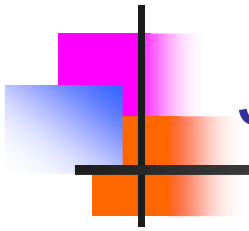


Storia di JDBC

- Prima distribuzione (jdbc.sql)
- JDBC fa parte del pacchetto software JDK a partire dalla versione 1.1 (pakage java.sql)
- Con Java 2, è stato introdotto JDBC 2.0:
  - Migliorate le funzionalità e i tipi di dato disponibili
  - Offerto come package opzionale per funzionalità estese
- Java 2, versione 1.5, include JDBC 3.0

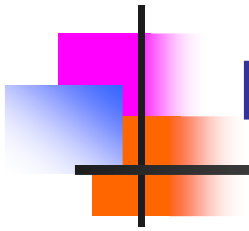
Architettura JDBC





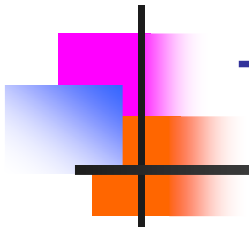
JDBC Driver Manager

- Rappresenta il livello di gestione di JDBC e opera tra l'utente e i driver
- Tiene traccia dei driver disponibili e gestisce la creazione di una connessione tra un DB e il driver appropriato



Driver JDBC

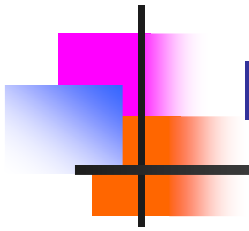
- Sono i driver che realizzano la vera comunicazione con il DB
- Permettono di:
  - Stabilire una connessione con una sorgente di dati
  - Inviare istruzioni di interrogazione e aggiornamento alla sorgente di dati
  - Processare i risultati



Tipi di driver JDBC

- Modello three-tier: affinché l'applicazione possa interagire con il DB occorre che le chiamate JDBC siano convertite in chiamate API native (caso JDBC/native bridge) o in chiamate ODBC (caso JDBC/ODBC bridge)
  - L'utente può utilizzare una API di alto livello (di più semplice utilizzo) che viene tradotta dal driver in chiamate di basso livello
  - Non è realmente portabile in quanto richiede l'utilizzo di componenti nativi (specifici dell'ambiente in cui vengono eseguiti)

- Modello two-tier: l'applicazione interagisce direttamente con il DB mediante un opportuno protocollo di rete, per es. TCP/IP (caso driver JDBC middleware, detto anche Net-Driver), oppure mediante un protocollo di rete proprietario (caso driver JDBC Driver, detto anche Driver "Java puro")
  - Si appoggia su un ambiente completamente Java



Interfacce e Classi JDBC (1)

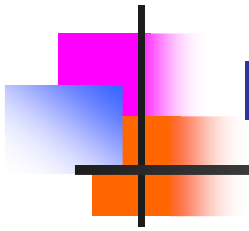
- Interfaccia Driver: rappresenta il punto di partenza per ottenere una connessione a un DBMS. I produttori di driver JDBC implementano l'interfaccia Driver (mediante opportuna classe) affinché possa funzionare con un tipo particolare di DBMS
  - Avendo a disposizione un oggetto Driver è possibile ottenere la connessione al database. Ogni driver JDBC ha una stringa di connessione che riconosce nella forma:

    `jdbc:product_name:database_alias`

    in cui **`database_alias`** specifica il DB a cui connettersi

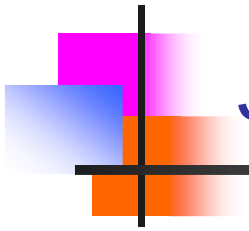
    (…nel nostro caso: **`jdbc:db2:[sample|studenti]`**)

- Classe DriverManager: facilita la gestione di oggetti di tipo Driver e consente la connessione con il DBMS sottostante. Nel momento in cui un oggetto Driver viene istanziato viene automaticamente registrato nella classe DriverManager



Interfacce e Classi JDBC (2)

- Interfaccia Connection: un oggetto di tipo Connection rappresenta una connessione attiva a un DB. L'interfaccia mette a disposizione un insieme di metodi che permettono, tra le altre cose, di:
  - Formulare query SQL da inviare al DBMS tramite gli oggetti Statement, PreparedStatement o CallableStatement
- Interfaccia Statement: un oggetto di tipo Statement viene utilizzato per inviare query SQL semplici, ovvero che non fanno uso di parametri, verso il DBMS (una query può comprendere: UPDATE, INSERT, CREATE o SELECT)
- Interfaccia PreparedStatement: un oggetto di tipo PreparedStatement viene utilizzato per creare query parametriche precompilate ("prepared"). Il valore di ciascun parametro non è specificato nel momento in cui lo statement SQL è definito, ma rimpiazzato dal carattere '?'
- Interfaccia CallableStatement: un oggetto di tipo CallableStatement viene usato per costruire query parametriche con parametri di input e output. Consente di eseguire una stored procedure memorizzata sul server
- Interfaccia ResultSet: un oggetto ResultSet è il risultato di una query di selezione (di fatto una tabella composta da righe e colonne)



JDBC e DB2

- DB2 prevede 2 driver JDBC di tipo two-tier:
  1. Net-Driver **`COM.ibm.db2.jdbc.net.DB2Driver`**
  1. Driver Java puro **`COM.ibm.db2.jdbc.app.DB2Driver`**

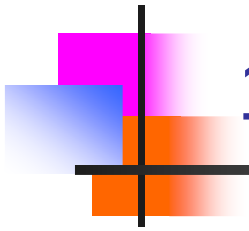
  entrambi sono contenuti nel file **`db2java.zip`** (direttorio **`\sqllib\java`**)

- Il Net-Driver serve per connettersi via rete (ad esempio, mediante il protocollo di rete TCP/IP) a server remoti, a condizione che questi abbiano attivo il servizio DB2 Jdbc Applet Server
- Il Driver Java puro permette di connettersi ad istanze DB2 residenti sulla macchina locale o catalogate localmente mediante un protocollo di rete proprietario DB2

Programmare un'applicazione JDBC

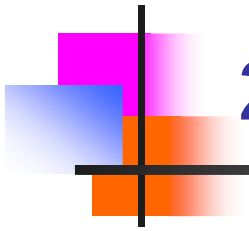
- Passi principali:

1. Importazione package
2. Registrazione driver JDBC
3. Apertura connessione al DB (Connection)
4. Creazione oggetto Statement
5. Esecuzione query e restituzione oggetto ResultSet
6. Utilizzo risultati
7. Chiusura oggetto/i ResultSet e oggetto Statement
8. Chiusura connessione



1: Importazione package

```
// Questo programma mostra un semplice esempio di
// applicazione Java (Esempio.java) in grado di eseguire
// interrogazioni/aggiornamenti sul database DB2 SAMPLE
// utilizzando JDBC


   //importazione package
   import java.sql.*;       //package JDBC
```

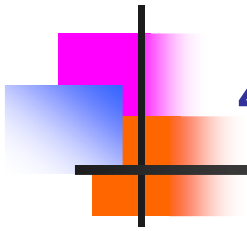


2: Registrazione driver JDBC

```
class Esempio {

   public static void main(String argv[]) {

   try {
   // caricamento e registrazione driver
    Class.forName("COM.ibm.db2.jdbc.app.DB2Driver").newInstance();
```

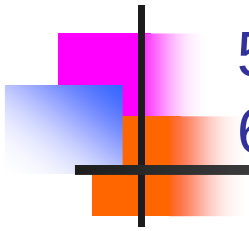
3: Apertura connessione DB

```
Connection con = null;
// URL jdbc:db2:database_alias
String url = "jdbc:db2:sample";
 if (argv.length == 2) {
        String userid = argv[0];
        String passwd = argv[1];

        // connessione con id/password forniti dall'utente
        con = DriverManager.getConnection(url, userid,
                                          passwd);
 }
  else {
        System.out.println("\nUsage: java Esempio username
                                    password\n");
        System.exit(0);
  }
```



4. Creazione oggetto Statement

```
// interrogazione table EMPLOYEE
 System.out.println("Retrieve some data from the database...");

 Statement stmt = con.createStatement();
```

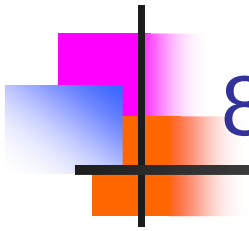


5. Esecuzione query e restituzione oggetto ResultSet
6. Utilizzo oggetto ResultSet

```
// esegue la query
ResultSet rs = stmt.executeQuery
                          ("SELECT * FROM DB2ADMIN.EMPLOYEE");
System.out.println("Received results:");
// mostra i risultati
// rs.next() = “false” se non ci sono più righe risultato
     while (rs.next()) {
         String a = rs.getString(1);
         String str = rs.getString(2);

         System.out.print(" empno= " + a);
         System.out.print(" firstname= " + str);
         System.out.print("\n");
      }
```

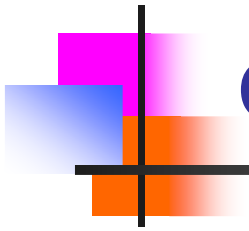
7. Chiusura oggetti ResultSet e Statement

```
// chiude ResultSet e Statement
  rs.close();
  stmt.close();


// …Esecuzione di altre istruzioni SQL
// aggiorna il database… prova ad aggiornarlo! ☺
   System.out.println("\n\nUpdate the database... ");
    stmt = con.createStatement();
    int rowsUpdated = stmt.executeUpdate("UPDATE DB2ADMIN.EMPLOYEE
                  SET firstname = 'SHILI' WHERE empno = '000010'");
    System.out.print("Changed "+rowsUpdated);
    if (1 == rowsUpdated)
        System.out.println(" row.");
    else
        System.out.println(" rows.");
    stmt.close(); // chiude Statement
```



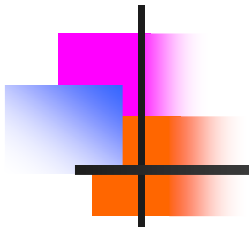
8. Chiusura connessione

```
    con.close(); // chiude Connection
   } // try
   catch( Exception e ) {
   e.printStackTrace();
   }
  } // main
} // classe
```



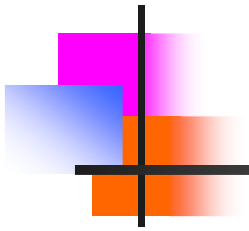
Oggetto Statement

- Un oggetto Statement fornisce tre metodi per eseguire una query SQL:
  - (StatementObj.) executeQuery(query), per statement che generano un unico result set (SELECT)
  - (StatementObj.) executeUpdate(stmt), per statement di modifica
  - (StatementObj.) execute(stmt), se il risultato di uno statement può includere più di un risultato o più di un contatore di aggiornamento



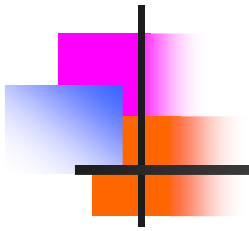
executeQuery

- Usato tipicamente per query di tipo SELECT
- Restituisce un oggetto ResultSet



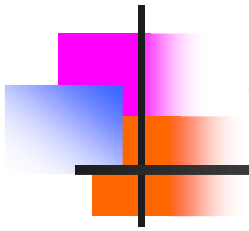
executeUpdate

- Usato per query di tipo INSERT, UPDATE o DELETE e per statement di tipo DDL quali CREATE TABLE e DROP TABLE
- Restituisce un intero rappresentante il numero di righe che sono state inserite/aggiornate/cancellate (contatore di aggiornamento). In caso di statement di tipo DDL, restituisce sempre il valore 0



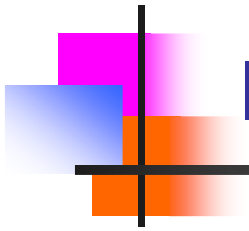
execute

- Usato quando la query restituisce più di un risultato o più di un contatore di aggiornamento
- Utilizza i seguenti metodi:
  - (StatementObj.) getResultSet() per ottenere il result set successivo
  - *(StatementObj.)* getUpdateCount() per ottenere il contatore di aggiornamento successivo
  - *(StatementObj.)* getMoreResults() per sapere se ci sono altri result set o contatori di aggiornamento
- Restituisce true se il risultato corrente è di tipo ResultSet; false se il risultato è di tipo Count o non ci sono più risultati



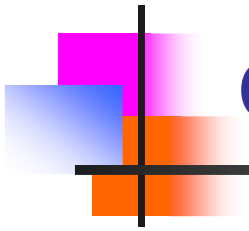
Oggetto ResultSet

- Un oggetto ResultSet contiene il risultato di una query SQL (cioè una tabella)
- Un oggetto ResultSet mantiene un cursore alla riga corrente
- Per ottenere un valorerelativo alla riga corrente:
  - (ResultSetObj.) getXXX(*column-name*)
  - (ResultSetObj.) getXXX(*column-number*)
- Per spostare il cursore dalla riga corrente a quella successiva:
  - (ResultSetObj.) next() (restituisce true in caso di successo; false se non ci sono più righe nell'insieme risultato)



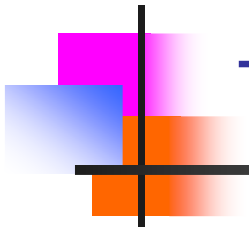
I metodi get*xxx*

- getByte
- getShort
- getInt
- getLong
- getFloat
- getDouble
- getBigDecimal
- getBoolean
- getString
- getBytes
- getDate
- getTime
- getTimestamp
- getAsciiStream
- getUnicodeStream
- getBinaryStream
- getObject



Controllo sui valori NULL

- I valori NULL SQL sono convertiti in null, 0, o false, dipendentemente dal tipo di metodo get*XXX*
- Per determinare se un particolare valore di un risultato corrisponde a NULL in JDBC:
  - Si legge la colonna
  - Si usa il metodo (ResultSetObject.) wasNull()



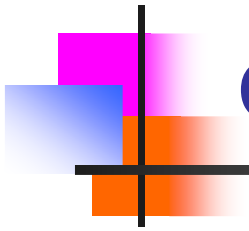
Tipi di dato: SQL2Java (1)

- Alcuni tipi di dato specifici di SQL devono essere mappati in corrispondenti tipi di dato Java per poter essere utilizzati
- La conversione riguarda tre categorie:
  - Alcuni tipi di dato SQL hanno i diretti equivalenti in Java e possono essere letti direttamente nei tipi Java (esempio: il tipo **`INTEGER`** SQL è equivalente al tipo **`int`** di Java)
  - Alcuni tipi di dato SQL possono essere convertiti negli equivalenti tipi Java (esempio: i tipi SQL **`CHAR`** e **`VARCHAR`** possono essere convertiti nel tipo **`String`** di Java)
  - Una minoranza di tipi di dato SQL sono unici e necessitano della creazione di uno speciale oggetto Java, relativo a una classe dato, per ottenere l'equivalente SQL (esempio: il tipo SQL **`DATE`** si converte nell'oggetto **`Date`** definito dall'omonima classe Java)

Tipi di dato: SQL2Java (2)

SQL type	Java Type
CHAR	String
VARCHAR	String
LONGVAR CHAR	String
NUMERIC	java.math.Big Decimal
DECIMAL	java.math.Big Decimal
BIT	boolean
TINYINT	byte
SMALLINT	short
INTEGER	int

SQL Type	Java Type
BIGINT	long
REAL	float
FLOAT	double
DOUBLE	double
BINARY	byte[]
VARBINARY	byte[]
LONGVAR BINARY	byte[]
DATE	java.sql.Date
TIME	java.sql.Time
TIMESTAMP	java.sql.Time stamp

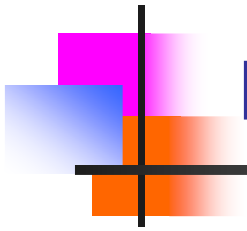


Oggetto PreparedStatement

- Usato quando la query SQL prende uno o più parametri come input, o quando una query semplice deve essere eseguita più volte
- L'interfaccia PreparedStatement estende l'interfaccia Statement ereditandone tutte le funzionalità. In più sono presenti metodi per la gestione dei parametri
- L'oggetto viene creato con l'istruzione

  Connection.prepareStatement(stmt)
- I parametri vengono poi settati mediante il metodo

  (StatementObj.) set*XXX*(*n*,*value*)
- La query pre-compilata viene eseguita mediante i metodi executeQuery(), executeUpdate() o execute() senza bisogno di passare alcun parametro!!



PreparedStatement: Esempio

- I parametri sono specificati con "?"

  Esempio:

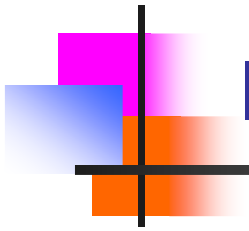
```
PreparedStatement ps = con.prepareStatement(
                    "UPDATE NameTable SET a = ? WHERE b = ?");
```

- Per settare i parametri (necessariamente prima dell'esecuzione della query):

```
ps.setInt(1, 20);
ps.setInt(2,100);
```

- Per eseguire lo statement:

```
int res = ps.executeUpdate();
```



I metodi set*xxx*

- setByte
- setShort
- setInt
- setLong
- setFloat
- setDouble
- setBigDecimal
- setBoolean
- setNull
- setString
- setBytes
- setDate
- setTime
- setTimestamp
- setAsciiStream
- setUnicodeStream
- setBinaryStream
- setObject

Tipi di dato: Java2SQL

Java Type	SQL type
String	VARCHAR or LONG-VARCHAR
java.math. BigDecimal	NUMERIC
boolean	BIT
byte	TINYINT
short	SMALLINT
int	INTEGER
long	BIGINT

Java Type	SQL type
float	REAL
double	DOUBLE
byte[]	VARBINARY or LONGVARBINARY
java.sql.Date	DATE
java.sql.Time	TIME
java.sql.Timestamp	TIMESTAMP

Il problema dell'SQL injection

- Cosa succede se, data una query con parametri inseriti dall'utente (es. tramite interfaccia Web), questi ha la possibilità di agire direttamente sul valore dell'input di tipo stringa (oggetto String), aggiungendo, ad esempio, apici e altre istruzioni di controllo??
  - Può inserire istruzioni arbitrarie che verranno eseguite dal DBMS!!! ☹

Esempio:

```
Statement =
    "SELECT * FROM users WHERE name = '" + userName + "';"
```

con la variabile `userName` assegnata al valore: `a';DROP TABLES users;`

- Questo tipo di vulnerabilità viene detta "*SQL injection*", in quanto l'utente può "iniettare" statement SQL arbitrari con risultati catastrofici, come la divulgazione di dati sensibili o l' esecuzione di codice
- A prevenzione del problema, l'interfaccia PreparedStatement permette di gestire in modo corretto anche l'inserimenti di dati "ostili"

La storia del piccolo “Bobby Tables”

(From the comic strip xkcd)

School: "Hi, this is your son's school. We're having some computer trouble."

Mom: "Oh, dear -- Did he break something?"

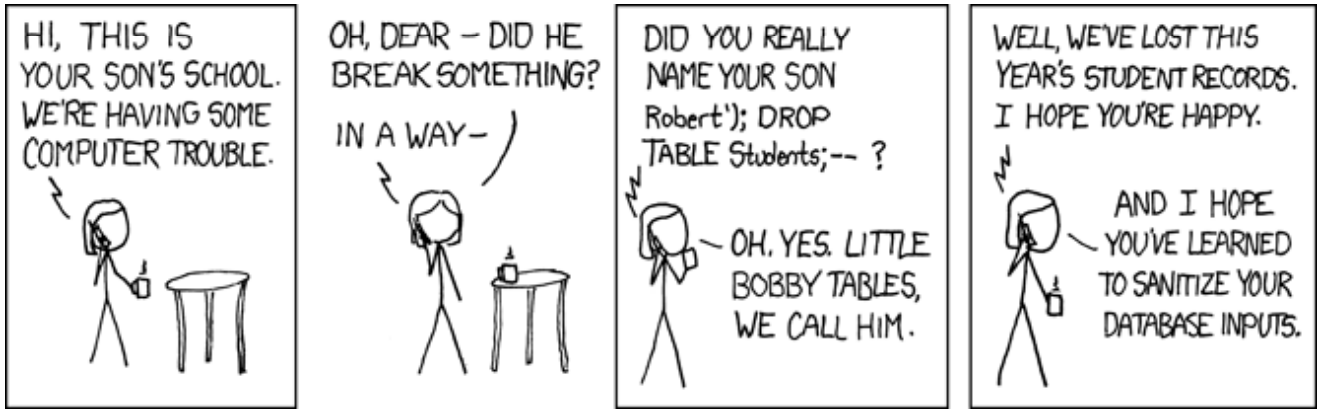
School: "In a way. Did you really name your son Robert'); DROP TABLE Students;-- ?"

Mom: "Oh. Yes. Little Bobby Tables we call him."

School: "Well, we've lost this year's student records. I hope you're happy."

Mom: "And I hope you've learned to sanitize your database inputs."

(Alt-text: "Her daughter is named Help I'm trapped in a driver's license factory.")



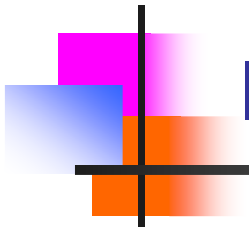
Fonte: http://bobby-tables.com/

- C’è solo un modo per evitare attacchi di tipo Bobby Tables
  - Non creare mai statement SQL che includono dati esterni
  - Usare sempre chiamate SQL parametrizzate (PreparedStatement)

Metodo **ReadEntry** (per completezza)

```
    //  Il method “readEntry” permette di leggere una stringa
    //  dal prompt dei comandi e di restituirla
public static String readEntry (String prompt)
 {
     try{
       StringBuffer buffer = new StringBuffer ();
       System.out.print (prompt);
       System.out.flush ();
       int c = System.in.read ();
       while (c != '\n' && c != -1){
    buffer.append ((char)c);
    c = System.in.read ();
       }
       return buffer.toString ().trim ();
     }
     catch (IOException e){
     return "";
     }
 }
```

```
// Nella classe main

…

String user;

String password;


user=readEntry(“username: ”);


Password=readEntry(“password: ”);
```



Informazioni Utili

- Per la documentazione relativa al pakage java.sql fare riferimento a Java2Docs:
  - **`http://java.sun.com/j2se/1.5.0/docs/api/index.html`**
- Per saperne di più:
  - **`http://java.sun.com/products/jdbc/`**
  - **`http://java.sun.com/j2se/1.5.0/docs/guide/jdbc/index.html`**