JDBC

Accesso a DataBase con Java

A. Bechini 2004

Utilizzo di DB da applicazioni "esterne"

- Un DB contiene e gestisce dati, importanti per varie operazioni supportate da applicazioni software
- Come può un'applicazione connettersi e comunicare con un DB?
- Una tecnologia Java per questo scopo: JDBC
  - Connessione a DB con JDBC
  - Interrogazione e manipolazione di DB con JDBC

JDBC: generalità

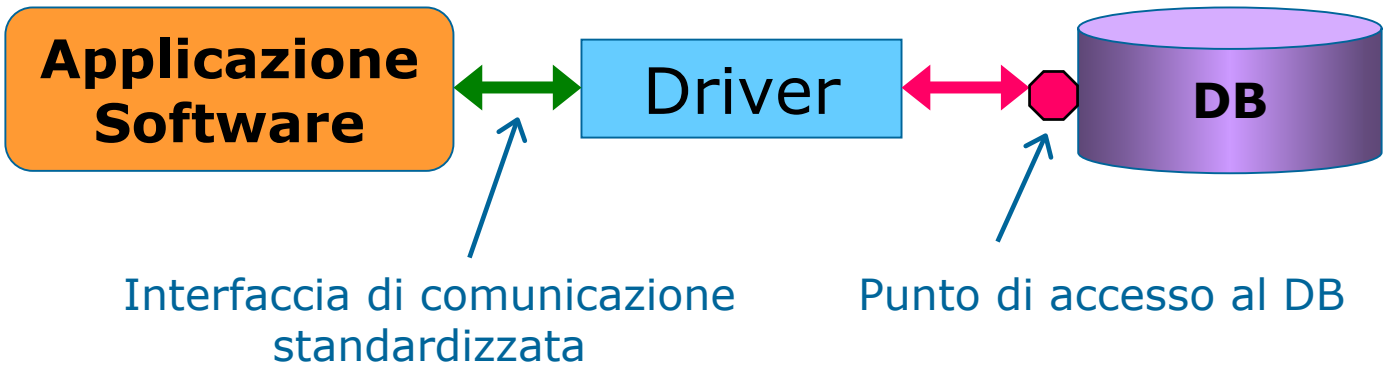
- JDBC: Java DataBase Connectivity kit
- Le sue funzionalità sono contenute nelle core API, nel package **`java.sql`**
- Due principali classi/interfacce:
  - **`DriverManager`**: classe usata per ottenere una *connessione* a DB
  - **`Connection`**: interfaccia per comunicare con un DB quando è stata stabilita una connessione
  - Altre classi/interfacce:
    `Statement, ResultSet, DatabaseMetaData, ResultSetMetaData, SQLException, ....`

Connessione a DB (I)

- Un DBMS tipicamente supporta le connessioni attraverso un modulo software detto "driver"
- Il driver accetta richieste da applicazioni esterne (formulate con una specifica sintassi), traducendole in comandi per il DBMS usato
- La standardizzazione della sintassi per le richieste permette alle applicazioni di ignorare dettagli implementativi del DB utilizzato

Connessione a DB (II)

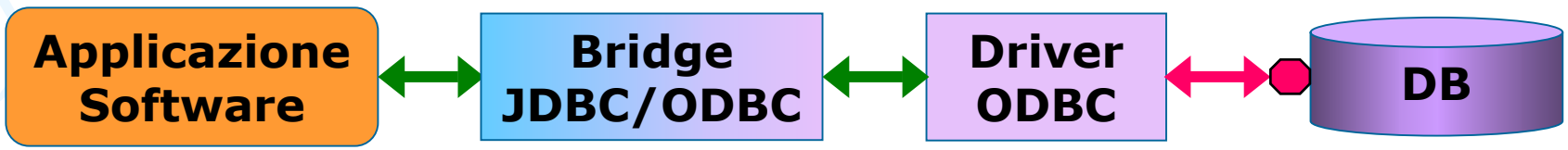
- Un'applicazione che vuole usare un DB, deve:
  - Avere a disposizione un apposito driver con cui è il grado di comunicare (fornito dal venditore del DB)
  - Sapere come *localizzare* il DB, tramite un suo "punto di accesso" (stabilito da chi amministra lo specifico DB)



Interfacce di comunicazione

- L'interfaccia di comunicazione più diffusa è ODBC (Open DataBase Connectivity, di Microsoft)
- In Java, si usa JDBC (analogo a ODBC, più semplice)

- I venditori di DB forniscono sempre driver ODBC (spesso anche JDBC) per accedere ai loro sistemi
- Si può eventualmente usare anche un driver "bridge JCBC/ODBC"

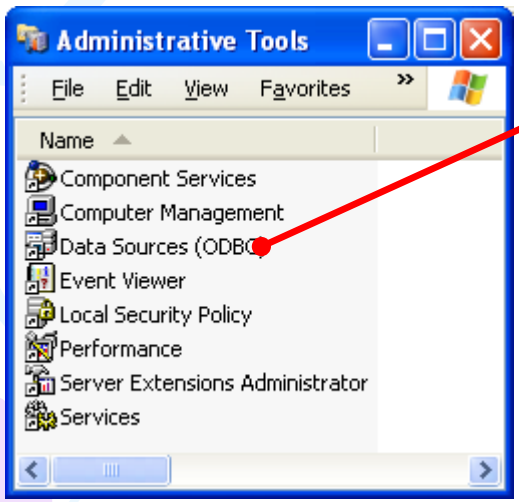


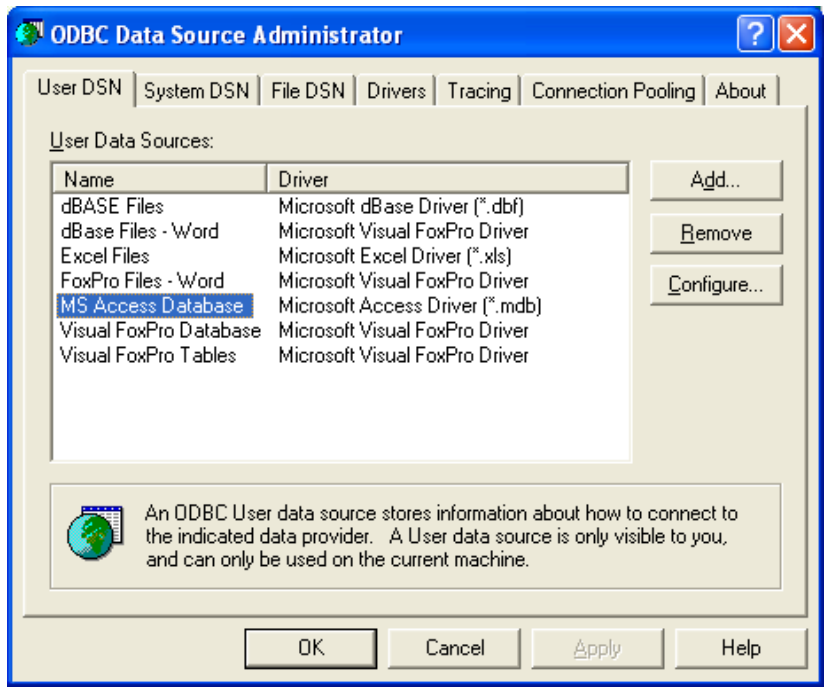
Connessione: Passi fondamentali

- Due azioni preliminari:
  - Rendere accessibile il DB dall'esterno (tramite indentificatore unico)
  - Registrazione del driver JDBC da utilizzare (su **`DriverManager`**)
- Instaurare la connessione

Connessione: Azioni preliminari (I)

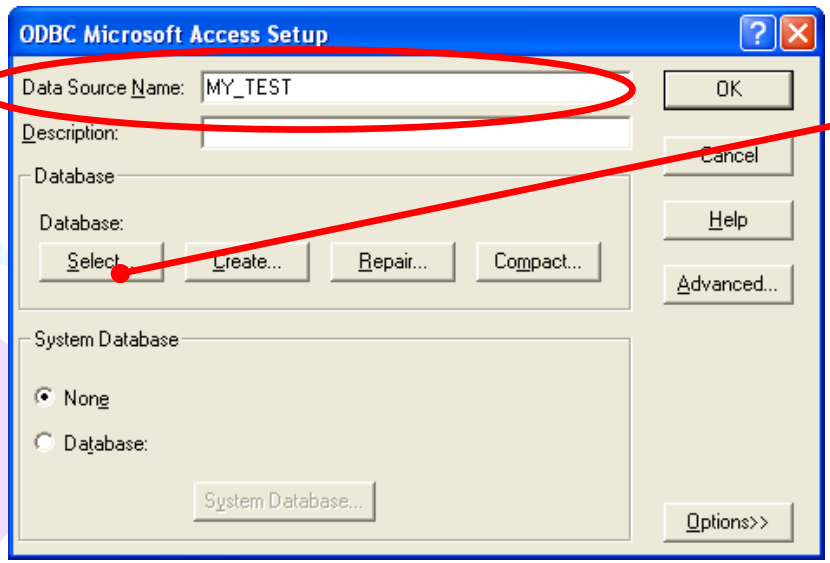
- Rendere accedibile il DB dall'esterno, p.es. sotto Windows registrando il file relativo con lo strumento "ODBC Data Source Administrator"

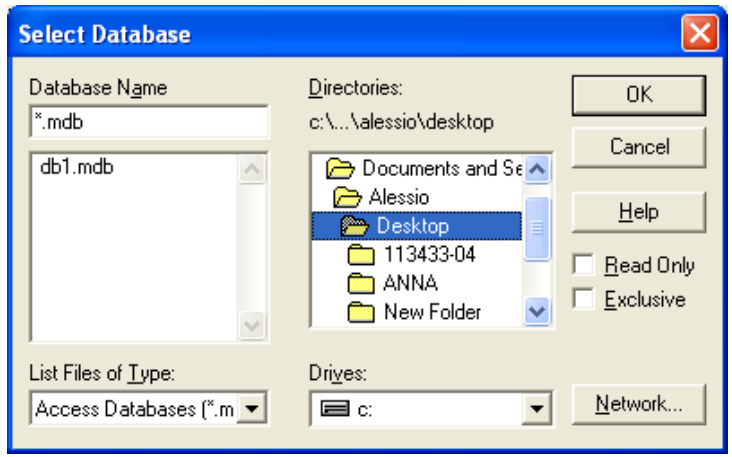




Connessione: Azioni preliminari (II)

- Con "ODBC Data Source Administrator", selezionare il tipo di driver desiderato (premendo il pulsante "**Add**").
- Selezionare p.es. "**Microsoft Access Driver**" (+ "**Finish**")
- Fornire un identificatore per la sorgente dati ODBC, e specificare il file del database (p. es. un .mdb)





Connessione: Azioni preliminari (III)

- In conseguenza delle azioni descritte, il file .mdb selezionato sarà individuato dall' "ODBC Data Source Administrator" tramite l'identificatore specificato (p.es. "MY_TEST").
- Ai fini JDBC, l'identificatore unico del DB sarà

```
jdbc:odbc:MY_TEST
```

Connessione: Azioni preliminari (IV)

- Registrazione del driver JDBC: 2 metodi alternativi

1. **`DriverManager`**, in fase di inzializzazione, carica tutte le classi specificate nella proprietà di sistema "**`jdbc.drivers`**": dunque, basterà fare:

```
System.setProperty("jdbc.drivers",
                   "sun.jdbc.odbc.JdbcOdbcDriver");
```

2. Caricare **esplicitamente** il driver:

```
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
```

Questa invocazione determina l'inizializzazione della classe in argomento (nell'es.: bridge jdbc/odbc)

Ottenere una connessione

- Si invoca un apposito metodo di **`DriverManager`** :

Oggetto per gestire la connessione

```
Connection con=
DriverManager.getConnection(
"jdbc:odbc:MY_TEST", "pippo", "x");
```

Localizzazione punto di accesso al DB (“URL”)

Username del DB

Password del DB

URL JDBC: struttura

- **`DriverManager`** può gestire più di un driver.
- Il particolare driver da usare viene specificato attraverso l'URL del DB
- Un URL JDBC: jdbc:<subprotocol>:<subname>
  - <subprotocol> si riferisce al driver
  - <subname> si riferisce alla sorgente dati (varie modalità di specifica, in dipendenza dal DBMS)
- Esempi:
  jdbc:odbc:db_1
  jdbc:oracle:thin:@host1:1532:mydb
  jdbc:mysql://a_firm.com/a_db

Eseguire query: lo statement

- Una volta ottenuta una connessione, si può operare sul DB con *query di selezione* e *query di comando* (UPDATE, DELETE, INSERT…)
- Per far questo si ricorre all'interfaccia **`java.sql.Statement`**
- Uno statement viene ottenuto dall'oggetto connessione:

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

Connessione ottenuta precedentemente

Es: una query SQL statica

- Si prepara una stringa che contiene la query:

```
String q = "SELECT Cognome FROM Tabella1";
```

- Si invoca il metodo **`executeQuery()`** sullo statement passando la stringa query come argomento:

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

- Si ottiene il risultato sotto forma di un oggetto di tipo (interfaccia) **`ResultSet`**

L'oggetto ResultSet

- L'oggetto restituito da `java.sql.Statement.executeQuery()` implementa l'interfaccia `java.sql.ResultSet`
- Esso rappresenta la tabella generata come risultato dell'esecuzione di una query
- Ci si può muovere tra le righe del `ResultSet` con un "*cursore*" manipolato p. es. dai metodi
  - `boolean next()` – si passa alla successiva
  - `boolean previous()` – alla precedente
  - `boolean absolute(int n)` – alla riga n
  - `boolean relative(int n)` – n righe più avanti

ResultSet: metodi getXXX()

- Per ottenere i dati dei campi di ciascuna riga, si usano i metodi
  `getXXX(String attr)`
  `getXXX(int index)`
  in cui **`XXX`** corrisponde al tipo di dato restituito
- L'argomento corrisponde, nelle due versioni:
  - al nome dell'attributo desiderato
  - alla posizione del campo (1=prima colonna)
- Es:

```
String nome1 = rs.getString("nome");
int e = rs.getInt("eta");
String nome2 = rs.getString(2);
```

Tipico uso di ResultSet

- Si può sfruttare il valore di ritorno dei metodi che modificano il cursore:

```
while (rs.next()) {
  for (int i = 1; i <= 2; i++)
      System.out.println(rs.getString(i));
  System.out.println("");
}
```

Es:

```
Mario
Rossi

Carlo
Verdi
```

Query di comando

- Si opera in modo analogo alle query di selezione:
- Si prepara una stringa che contiene la query:

```
String comando = "UPDATE MiaTabella ...";
```

- Si invoca il metodo **executeUpdate()** sullo statement passando la stringa query come argomento; restituisce il numero di righe coinvolte dall'esecuzione del comando:

```
int r = st.executeUpdate(comando);
```

Query parametriche (I)

- JDBC fornisce un meccanismo per formulare query di selezione/comando parametriche
- Si utilizza **`java.sql.PreparedStatement`**, una versione "specializzata" di **`java.sql.Statement`**
- Si formula la query SQL sostituendo il carattere '?' al valore dei parametri:

```
String pq = "SELECT x FROM Tab WHERE y > ? "
```

- Un PreparedStatement viene ottenuto dall'oggetto connessione:

```
PreparedStatement ps = con.prepareStatement(pq);
```

Query parametriche (II)

- Prima di poter eseguire la query, occorre specificare il valore dei parametri
- Questa operazione viene svolta dai metodi **`setXXX(int index, XXX val)`**
- **`index`** è il numero dell'occorrenza del '?' a cui ci si riferisce
- **`XXX`** corrisponde al tipo di dato trattato
- Es:

```
int a=5;
ps.setInt(1,a);
ResultSet rs =  ps.executeQuery();
```

Adesso:
SELECT x FROM Tab
WHERE y > 5

Nessun argomento

Eccezioni in JDBC

- Le varie attività JDBC possono sollevare eccezioni che devono essere gestite.
- L'eccezione più generale lanciata in questo ambito è **`java.sql.SQLException`** (p.es. da **`DriverManager.getConnection()`**, **`Statement.executeQuery()`**, etc.)
- Altre sotto-eccezioni: