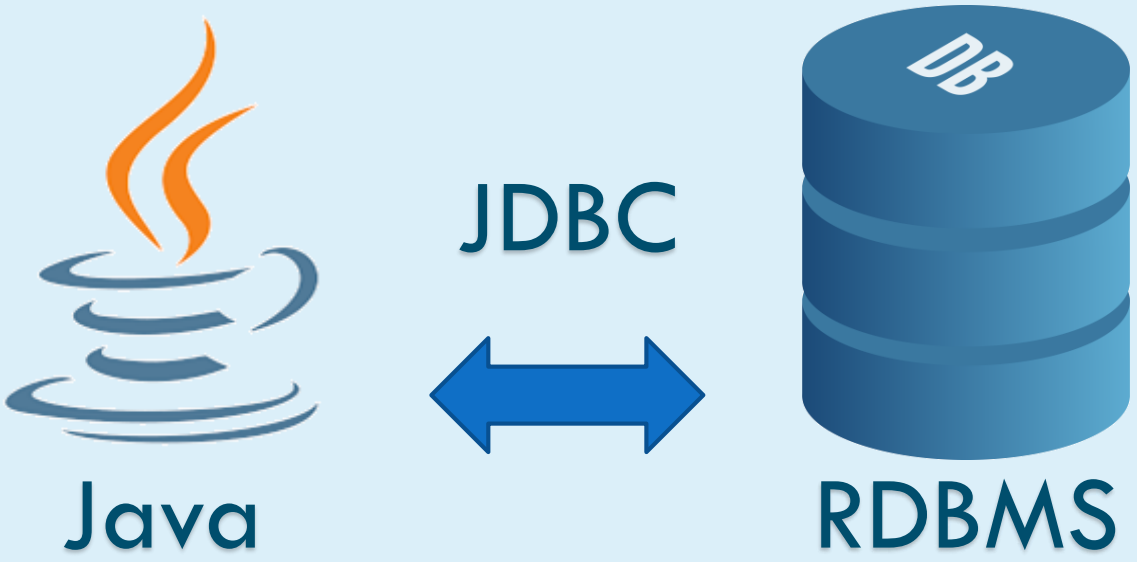


JDBC

Annalisa Franco

Università di Bologna

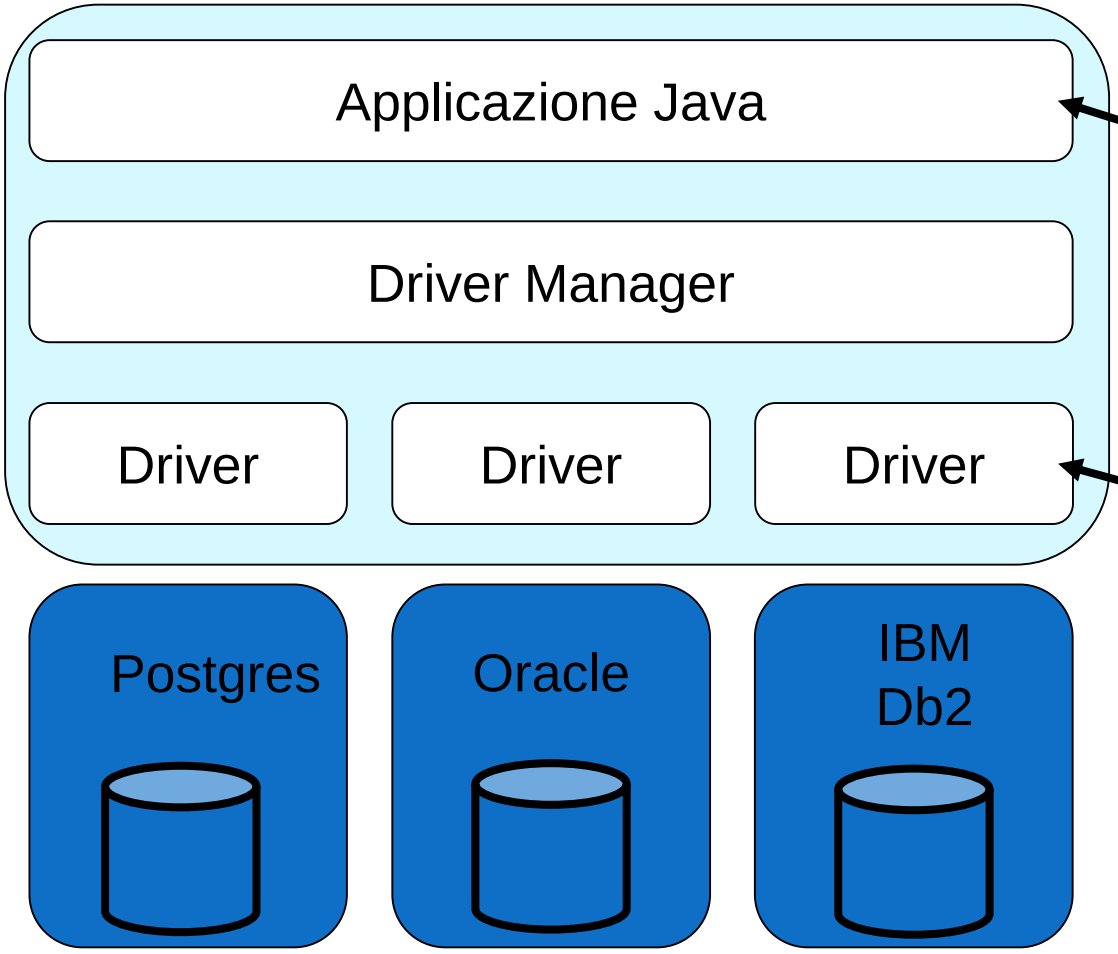
JDBC

- JDBC è una libreria di API per l'accesso **uniforme** a database relazionali da parte di applicazioni. Definisce classi per:
  - connessione a database
  - invio di query SQL
  - navigazione dei risultati delle query
  - accesso ai metadati del database/risultati

- Versioni
  - JDBC 1.0 -- JDK 1.1
  - JDBC 2.0 -- JDK 1.3
  - JDBC 3.0 -- JDK 1.4
  - JDBC 4.0 – Java SE 6
  - JDBC 4.1 – Java SE 7
  - JDBC 4.2 – Java SE 8
  - JDBC 4.3 – Java SE 9

- Packages Java
  - java.sql (classi fondamentali)
  - javax.sql (estensioni)

Architettura



utilizza
le interfacce
java.sql
e javax.sql

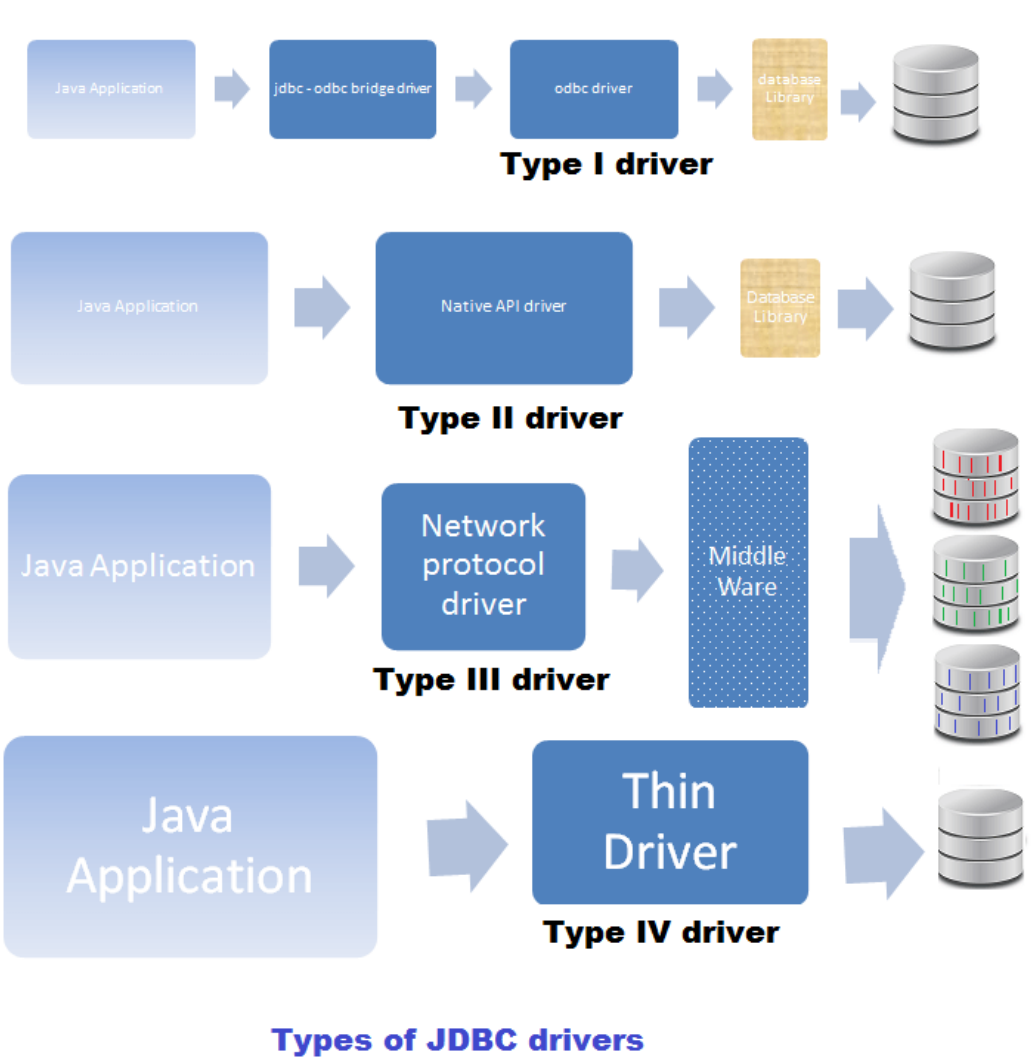
implementano
java.sql e,
in alcuni casi
javax.sql

JDBC vs ODBC

- Microsoft ha rilasciato ODBC (Open Database Connectivity)
  - dipendente dal sistema (scritto in C, non portabile)
  - utilizzabile da più linguaggi di programmazione
  - scambio dei dati complesso
- JDBC
  - indipendente dal sistema
  - utilizzabile solo con il linguaggio JAVA
  - minore complessità dell'interfaccia
- Entrambe le API permettono di richiamare funzionalità specifiche dei DBMS

Tipi di Driver

- Driver di tipo 1
  - traducono JDBC in ODBC
    - JDBC-ODBC Bridge
  - inefficienti
- Driver di tipo 2
  - le chiamate a JDBC vengono tradotte in chiamate all'API del DBMS prescelto
- Driver di tipo 3 (puro Java)
  - invia le chiamate verso il database passando per un componente di middleware.
  - la comunicazione verso il middleware avviene utilizzando un protocollo di rete.
- Driver di tipo 4 (puro Java)
  - comunica direttamente con un database utilizzando il protocollo nativo del DB stesso.

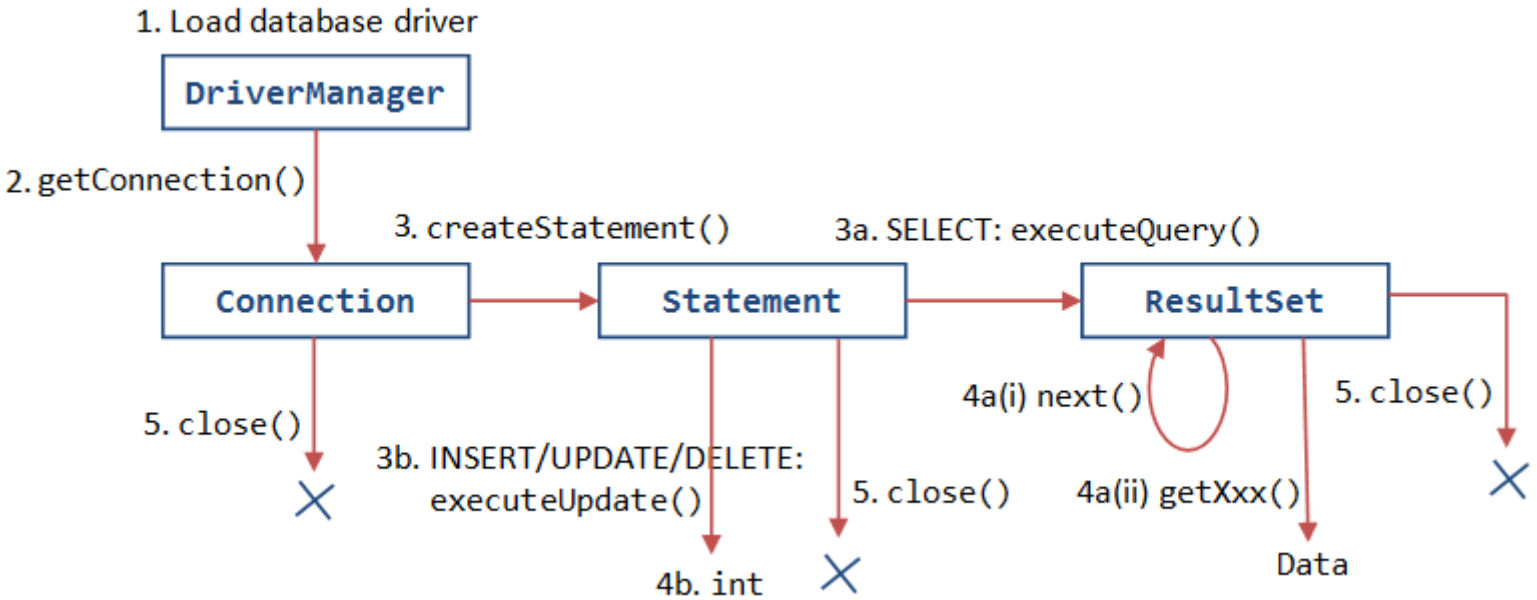


JDBC: Accesso ai dati

- L'API fornisce un meccanismo che carica dinamicamente i driver appropriati e li registra nel JDBC Driver Manager.
- Le connessioni JDBC supportano la creazione e l'esecuzione delle istruzioni (INSERT, UPDATE, DELETE, SELECT)
- I tipi di istruzioni supportati sono:
  - **Statement** - l'istruzione viene inviata al database di volta in volta;
  - **Prepared Statement** - l'istruzione viene compilata una sola volta, in modo che le chiamate successive siano più efficienti;
  - **Callable Statement** - usati per chiamare le stored procedure.
- I comandi di scrittura restituiscono un valore che indica il numero di righe modificate
- Le interrogazioni restituiscono un result set (classe ResultSet).

Step di accesso ai dati

1. Registrare il driver JDBC
2. Connettersi al DB
3. Definire una query SQL
4. Elaborare i risultati della query
5. Chiudere la connessione



Step di accesso ai dati 1-3

1. Il metodo Class.forName() carica la classe del driver JDBC.

```
Class.forName("com.mioDbms.mioDriver");
```

Attualmente il metodo è deprecato e il driver viene caricato automaticamente se si trova nel classpath.

2. Il metodo DriverManager.getConnection() crea una connessione.

```
Connection conn = DriverManager.getConnection(
"jdbc:mioDbms:altri dati utili per il driver",
"myLogin", "myPassword");
```

```
DriverManager.getConnection("jdbc:mysql://localhost/test?" +
                            "user=minty&password=greatsqldb");
```

La stringa da utilizzare dipende dal driver JDBC. Inizia sempre con "jdbc:", il resto varia a seconda del prodotto scelto.

3. Una volta stabilita la connessione, occorre passare una istruzione.

```
Statement stmt = conn.createStatement();
stmt.executeUpdate
("INSERT INTO myTab(myColumn) VALUES ( 'val')");
```

Step di accesso ai dati 4-5

4. I dati vengono prelevati dal database e restituiti come RecordSet da cui vengono elaborati.

```
Statement stmt = conn.createStatement();
ResultSet rs = stmt.executeQuery("SELECT * FROM myTable");

while (rs.next()) {
    int numCol = rs.getMetaData().getColumnCount();
    for ( int i = 1 ; i <= numCol ; i++ )

    { // I numeri di colonna iniziano da 1.
    System.out.println( "COL" + i + "=" +rs.getObject(i));

    }

}
```

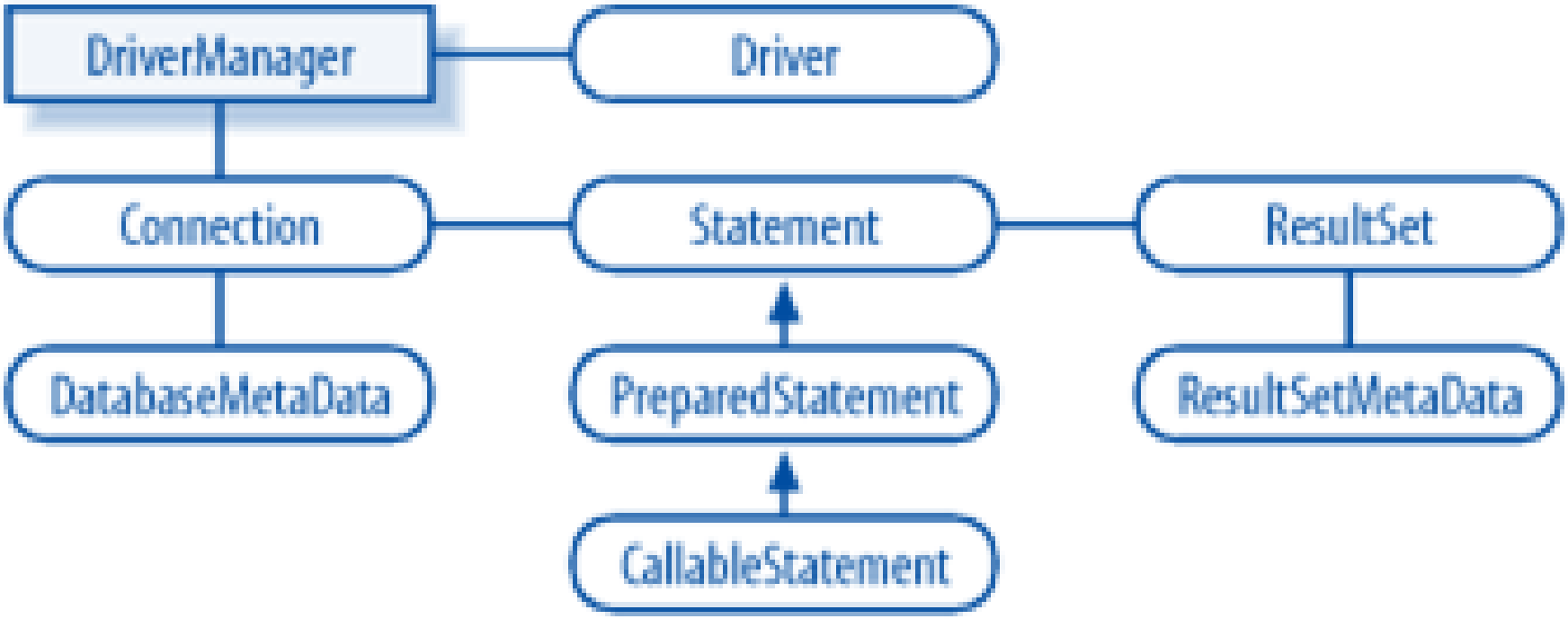
5. Chiusura connessione

```
    rs.close();
    stmt.close();
```

JDBC - classi e interfacce

- **java.sql.DriverManager:** ***classe***. Offre funzionalità di **gestione dei database driver**
- **java.sql.Driver:** ***interfaccia***. Astrae i dettagli del protocollo di **connessione al database**. I produttori di database implementano questa interfaccia
- **java.sql.Connection:** ***interfaccia***. Astrae dettagli di interazione con il database. Permette di **inviare statement SQL e di leggere i risultati** dell'esecuzione degli statement. I produttori di database implementano questa interfaccia

JDBC: Classi e interfacce Java



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
- È possibile ottenere un'istanza effettiva della classe Driver ricercando la classe con il metodo forName:

```
Driver d =
class.forName("com.mioDbms.mioDriver").newInstance();
```

- Attualmente il metodo sopra descritto è deprecato e il driver viene caricato automaticamente.

Classe java.sql.DriverManager

- Facilita la gestione di oggetti di tipo Driver
- Quando un oggetto Driver viene istanziato, esso viene automaticamente registrato nella classe DriverManager
- Ogni applicazione può registrare uno o più driver JDBC diversi tra loro
- Consente la connessione con il DBMS sottostante mediante il metodo statico getConnection
- Usa il driver appropriato tra quelli registrati

Interfaccia Connection

- Un oggetto di tipo Connection rappresenta una connessione attiva con il DB
- Il metodo getConnection di DriverManager, se non fallisce, restituisce un oggetto di tipo Connection
  - ```
    String url = "jdbc:odbc:dbname";
    Connection db = DiverManager.getConnection(url);
    ```
- L'interfaccia mette a disposizione una serie di metodi per preparare le query SQL da inviare:
  - Statement
    - ```
      Statement stmt = db.createStatement();
      ```
  - PreparedStatement
  - CallableStatement

Interfaccia Statement

- Gli oggetti di tipo Statement possono essere usati per inviare query SQL senza parametri al DBMS. La query può essere:
- di modifica (UPDATE, INSERT, CREATE)
  - (StatementObj).executeUpdate(stmt SQL)
  - Restituisce un intero rappresentante il numero di righe che sono state inserite/aggiornate/cancellate o 0 se comando DDL
- di selezione (SELECT)
  - (StatementObj).executeQuery(stmt SQL)
  - Per query di tipo SELECT il risultato è inserito in un oggetto ResultSet
- di tipologia non nota a priori:
  - (StatementObj).execute(stmt SQL)
  - Ritorna true se il risultato è un ResultSet, al quale si può accedere con il metodo(StatementObj).getResultSet()

Interfaccia Statement - Esempio

- Esempio **executeQuery**:
  - ```
    Statement s = db.createStatement();
    ResultSet r = s.executeQuery("Select * from Table");
    ```

- Esempio **execute**:
  - ```
    Statement s = db.createStatement();
    String query = "select * from emp";
    boolean status = s.execute(query);
    if(status){ // true se il risultato è un ResultSet
          //select query
          ResultSet rs = s.getResultSet();
          while(rs.next()){
              System.out.println(rs.getString(1)); }
          rs.close();
    } else {
         //update o altro
         int count = s.getUpdateCount();
    System.out.println("records updated: "+count);}
    ```

Interfaccia PreparedStatement

- Gli oggetti di tipo PreparedStatement possono essere usati per creare query SQL parametriche e precompilate ("prepared")
- Il valore di ciascun parametro non è specificato nel momento in cui lo statement SQL è definito, ma rimpiazzato dal carattere '?'
- Un oggetto PreparedStatement può essere creato con il metodo prepareStatement di Connection
  - ```
    String sSQL = "SELECT * FROM Table WHERE ID = ?";
    PreparedStatement pS = db.prepareStatement(sSQL);
    ```
- I parametri vengono poi settati mediante il metodo

Tipo di dato

(StatementObj.)setXXX(n, value)

Posizione parametro

```
pS.setInt(1,1);
ResultSet rs = pS.executeQuery(sSQL);
```
```

Interfaccia CallableStatement

- Gli oggetti di tipo CallableStatement possono essere usati per definire query parametriche con parametri di tipo IN, OUT e INOUT
- Permettono di eseguire una invocazione a una stored procedure memorizzata sul server DB
- Un oggetto CallableStatement può essere creato con il metodo prepareCall di Connection
  - ```
    String sql = "{call setSalary (?)}";
    CallableStatement cS = db.prepareCall(sql);
    cS.setInt (1,1000);
    cS.executeUpdate();
    ```
  - ```
    //ResultSet result = cS.executeQuery();
    ```

CallableStatement: Batch Updates

- Si possono raggruppare chiamate multiple a una stored procedure in un batch update.

```
String proc = "{call insertEMPLOYEE(?,?,?)}";
CallableStatement cS = db.prepareCall(proc);
cS.setInt(1,7);
cS.setString(2, "Mario Rossi");
cS.setInt(3,50000);
cS.addBatch();

cS.setInt(1,8);
cS.setString(2, "Giorgio Verdi");
cS.setInt(3,55000);
cS.addBatch();

cS.executeBatch();
```

CallableStatement: OUT Parameters

- Se una stored procedure restituisce dei parametri, questi sono accessibili dall'oggetto callebleStatement
- È necessario che ogni parametro di OUT e INOUT venga registrato prima dell'invocazione della stored procedure

```
String sql = "{call calculateStatistics(?)}";
CallableStatement cS = db.prepareCall(sql);
cS.setInt (1,123);
cS.registerOutParameter(1, java.sql.Types.VARCHAR);
cS.registerOutParameter(2, java.sql.Types.INT);
ResultSet result = cS.executeQuery();
String out1 = cS.getString(1);
int out2 = cS.getInt(2);
```

```
//while(result.next()) { ... }
```

Interfaccia ResultSet

- L'oggetto ResultSet è il risultato di una query di selezione
- Rappresenta una tabella composta da righe (gli elementi selezionati) e colonne (gli attributi richiesti)
- Per ottenere il valore del cursore alla riga corrente:
  - (ResultSetObj).getXXX(column-name)
  - (ResultSetObj).getXXX(column-number)
  - I metodi getInt, getString,... permettono la lettura degli attributi di una tabella
- 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)
  - ```
    while(result.next())
    { result.getString ("nome");
       result.getInt ("età"); // etc.
    }
    ```
    ```

Altri tipi di ResultSet

- È possibile creare dei ***ResultSet*** con maggiori funzionalità, passando degli opportuni parametri al momento della creazione dello ***Statement***:
  - ***scrollable ResultSet***:
    - le righe del risultato possono essere scandite in entrambe le direzioni
    - ci si può posizionare in maniera "assoluta" su una particolare riga
  - ***updatable ResultSet***:
    - mettono a disposizione dei metodi ***updateXXX*** analoghi ai metodi ***getXXX***, che consentono di modificare il valore di un campo sulla riga corrente
    - l'aggiornamento si riflette sulla corrispondente tabella nel database automaticamente

Tipi di ResultSet

- ResultSet.TYPE_FORWARD_ONLY – Il resultset è navigabile solo in avanti
- ResultSet.TYPE_SCROLL_INSENSITIVE – Il resultset è scrollabile. Modifiche fatte da altri sulle tabelle da cui è stato derivato il resultset non sono visibili
- ResultSet.TYPE_SCROLL_SENSITIVE – Il resultset è scrollabile. Modifiche fatte sulle tabelle da cui è stato derivato il resultset sono visibili nel resultset
- ResultSet.CONCUR_READ_ONLY – È un resultset di sola lettura (default)
- ResultSet.CONCUR_UPDATABLE – È un resultset che può essere aggiornato, e le modifiche riportate nel DB

Esempio di UPDATE

```
Statement st =
     db.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
     ResultSet.CONCUR_UPDATABLE);
ResultSet results =
     st.executeQuery("SELECT test_col FROM test_table");
// Muove il cursore alla 3a riga
results.absolute(3);
// Update della 3a riga
results.updateString("test_col",“nuovo_valore”);
results.updateRow();
// In caso di abort si usa:
//results.cancelRowUpdates();
results.absolute(5);
results.deleteRow();
// Aggiorna il resultSet
results.refreshRow();
```

Metodi di navigazione

- Boolean next(), previous(): Muove il puntatore alla riga successiva (precedente). Se è stata raggiunta l'ultima (prima) riga il metodo restituisce false e il puntatore punta immediatamente dopo l'ultima riga
- Boolean first(), last(): Muove il puntatore alla prima/ultima riga del resultSet.
- Boolean absolute(n): n>0: il puntatore si posiziona sulla riga n a partire dall'inizio (le righe si contano a partire da uno) n<0: come sopra ma a partire dalla fine
- Boolean relative(n): il puntatore si posiziona di n righe in avanti (o indietro se n<0) a partire dalla posizione attuale
- Void afterLast(), void beforeFirst()
- Boolean isFirst(), isBeforeFirst(), isLast(), isAfterLast(),
- Int getRow(): Restituisce il numero della riga corrente

JdbcRowSet

- L'oggetto JdbcRowSet gestisce dati tabellari in maniera più semplice che un ResultSet.
- Ci sono 5 diverse interfacce:
  - *JdbcRowSet:* sempre connesso al DB
  - *CachedRowSet:* lavora in cache
  - *WebRowSet:* può salvarsi in formato XML
  - *JoinRowSet:* crea delle connessioni (join) tra oggetti *RowSet* in memoria.
  - *FilteredRowSet:* filtra i dati in base a un predicato

JdbcRowSet: Esempio

```
import javax.sql.rowset.JdbcRowSet;
import com.sun.rowset.JdbcRowSetImpl;
//creo l'oggetto passandogli una connessione
JdbcRowSet jdbcRS = new JdbcRowSetImpl(conn);
jdbcRS.setType(ResultSet.TYPE_SCROLL_INSENSITIVE);
String sql = "SELECT * FROM customers";
jdbcRS.setCommand(sql);
//jdbcRs non contiene dati fino all'esecuzione di questa istruzione
jdbcRS.execute();
//scorre il JdbcRowSet
while (jdbcRS.next()) {
    System.out.println("id = " + jdbcRS.getString(1));
    System.out.println("name = " + jdbcRS.getString(2));
}
JdbcRowSet.first(); //spostamento random
…
JdbcRowSet.close(); //chiude il record set
```

Conversione tipi SQL-JAVA

JDBC Type	Java Type
BIT	boolean
TINYINT	byte
SMALLINT	short
INTEGER	int
BIGINT	long
REAL	float
FLOAT DOUBLE	double
BINARY VARBINARY LONGVARBINARY	byte[]
CHAR VARCHAR LONGVARCHAR	String

JDBC Type	Java Type
NUMERIC DECIMAL	BigDecimal
DATE	java.sql.Date
TIME TIMESTAMP	java.sql.Timestamp
CLOB	Clob*
BLOB	Blob*
ARRAY	Array*
DISTINCT	mapping of underlying type
STRUCT	Struct*
REF	Ref*
JAVA_OBJECT	underlying Java class

*SQL3 data type supported in JDBC 2.0

Conversione tipi SQL-JAVA

	INTEGER	REAL	CHAR	VARCHAR	DATE	CLOB
`getInt`	**X**	X	X	X		
`getFloat`	X	**X**	X	X		
`getString`	X	X	**X**	**X**	X	
`getDate`			X	X	**X**	
`getClob`						**X**
`getObject`	X	X	X	X	X	X

(1) Registrare il driver

- Significa caricare dinamicamente una classe Java
- sun.jdbc.odbc.JdbcOdbcDriver è il JDBC-ODBC Bridge fornito con JDK
- Altri driver:
  - SQL Server com.microsoft.jdbc.sqlserver.SQLServerDriver
  - IBM DB2 com.ibm.db2.jcc.DB2Driver
  - Oracle oracle.jdbc.OracleDriver
  - MySQL com.mysql.jdbc.Driver
- File Jar di SQL Server:
  - mssbase.jar
  - mssqlserver.jar
  - msutil.jar

(1) Caricare la classe

- Per caricare la classe occorre:
  - avviare il programma con –classpath <file_jar_del_driver>
  - o, modificare CLASSPATH con <file_jar_del_driver>
  - o, copiare <file_jar_del_driver> in jre/lib/ext

(2) Connessione al DB

- Per ottenere una connessione dal DriverManager occorre specificare
  - URL ("indirizzo" completo) della connessione (host, dbms, db)
  - utente e password
- Sintassi URL
  - jdbc:<sottoprotocollo>:<parametri>
- Esempi
  - jdbc:db2:nomeDB
  - jdbc:pointbase:nomeDB
  - jdbc:odbc:nomeDB
  - jdbc:mysql://localhost:3306/nomeDB
  - jdbc:oracle:thin:@localhost:1521:xe

(2) Connessione al DB

```
String url = "jdbc:microsoft:sqlserver:"+ // driver
             "SI-SQLSERVER;" +            // host
       "DatabaseName=NorthwindIT";  //database
String user = "student";
String password = "pass";
Connection conn =
  DriverManager.getConnection(url, user, password);
```

(3) Definire query SQL

```
// oggetto per la gestione di comandi SQL
Statement stmt = con.createStatement();

String query = "SELECT * FROM myTable";
// metodo per l'esecuzione di query SQL
ResultSet rs = stmt.executeQuery( query );

String update = "UPDATE myTable SET col1 = col1 + 1";
// metodo per l'esecuzione di update/insert/create
   table SQL
int affectedRows = stmt.executeUpdate(update );
```

(4) Elaborare i risultati

```
…
// ResultSet è un iteratore sulle righe
  while ( rs.next() )
  {
    // accesso ai valori dei campi per nome
    String nome = rs.getString("col1");
    int eta = rs.getInt("col2");

    //accesso ai valori dei campi per posizione (a
    partire da 1)
    int eta2 = rs.getInt( 2 );

    System.out.println(nome + " " + eta);
  }
…
```

(4) Scorrere il ResultSet

```
…
Statement stmt =
con.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
                    ResultSet.CONCUR_READ_ONLY);

String query = "SELECT col1,col2 FROM myTable";
ResultSet rs = stmt.executeQuery( query );

rs.previous();     // riga precedente
rs.relative(-5); // 5 righe indietro
rs.relative(7);  // 7 righe avanti
rs.absolute(100);// 100-esima riga
…
```

(4) ResultSet aggiornabili

```
…
Statement stmt =
con.createStatement(ResultSet.TYPE_FORWARD_ONLY,
                    ResultSet.CONCUR_UPDATABLE);

String query = "SELECT col1,col2 FROM myTable";
ResultSet rs = stmt.executeQuery( query );
…
while ( rs.next() )
{
    int eta = rs.getInt("col2");
    rs.updateInt("col2", eta+1);
    rs.updateRow();
}
rs.close();
```

(5) Chiusura Connessione

```
…

  // chiude la connessione al DB
  conn.close();

…
```

Oltre JDBC

- JDBC è una libreria completa, ma si tratta di un approccio "di base", che può richiedere la scrittura di molto codice.
- Librerie alternative (alcune basate su JDBC)
  - **Spring Framework e template JDBC**: *JdbcTemplate* svolgono in autonomia operazioni di apertura e chiusura di connessioni, preparazione degli Statement, la lettura di risultati e l'attivazione di transazioni;
  - **Hibernate e altri Object-Relational Mapping**: i framework OR/M, usano un approccio fondato sul mapping tra sistema informativo relazionale (incluso nel database) e modello a oggetti realizzato nel programma Java;
  - **iBatis**: un framework dedicato alla persistenza dei dati che facilita l'integrazione tra il programma e il database.
  - **jOOQ**: una libreria Java che fornisce un DSL (Domain Specific Language) per costruire queries tramite classi generate a partite da un database relazionale.

Riferimenti

- Sito JDBC
  - http://www.oracle.com/technetwork/java/javase/jdbc/index.html
- API Java
  - <directory java>/docs/guide/jdbc
- Tutorial on-line
  - http://docs.oracle.com/javase/tutorial/jdbc/
- Libro Atzeni, Ceri, Paraboschi, Torlone
  Basi di dati: modelli e linguaggi di interrogazione, McGraw-Hill Italia.
- Dalla documentazione JavaDoc
  - java.sql: classi fondamentali
  - javax.sql: estensioni

Domande?