RESEARCH CENTRE ON SOFTWARE TECHNOLOGY

JDBC

Usare i database da applicazioni Java

RESEARCH CENTRE ON SOFTWARE TECHNOLOGY

Istituti di Ricerca in Software Engineering in Europa

http://www.iese.fraunhofer.de/ - Kaisersautern, Germania

http://www.simula.no/, Fornebu, FInaldia

http://www.cwi.nl/ [Centrum foor Wiskunde en Informtica], Amsterdam Netherlands

http://www.esi.es/index.php?op=15.4.4 , Bilbao, Spain.

http://www.esi.es/index.php?op=15.4.4, Norway.

Outline

- Parte prima
  - Introduzione a JDBC
  - I driver
  - La connessione
  - Gli statement
  - Il ResultSet
- Parte seconda
  - Uso avanzato dei ResultSet
  - Esempio

RESEARCH CENTRE ON SOFTWARE TECHNOLOGY
Università degli Studi del Sannio

Fonti

La presente lezione è stata preparata con materiale reperibile sui siti java.sun.com e www.mysql.org

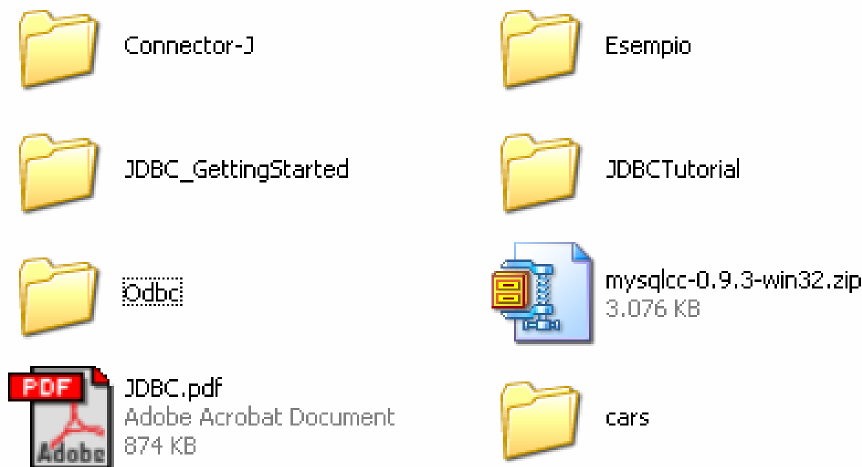
In particolare si rimanda al link:

java.sun.com/j2se/1.4.2/docs/guide/jdbc/getstart/intro.html

ed alla sezione su JDBC della documentazione di J2SDK 1.4 e 1.3

RESEARCH CENTRE ON SOFTWARE TECHNOLOGY
Università degli Studi del Sannio

Materiale di supporto



RESEARCH CENTRE ON SOFTWARE TECHNOLOGY
Università degli Studi del Sannio

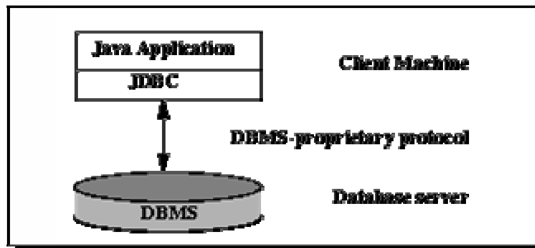
Cos'è JDBC

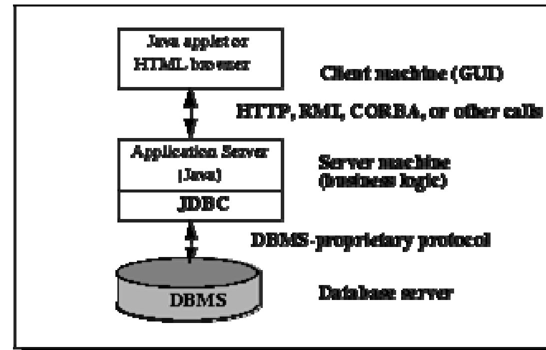
JDBC™ è un'API (application programming interface) che permette ad applicazioni Java di accedere a database ed altre fonti di dati.

E' definita nel package **`java.sql`** (**`javax.sql`** per le funzionalità server)

JDBC è un nome registrato ma spesso è interpretato come *Java DataBase Connectivity*

Cos'è JDBC





Altre API per i database

ODBC *Open DataBase Connectivity*: è uno standard di fatto definito da Microsoft per l'accesso a basi di dati da applicazioni in linguaggio **C**.

Tutti i principali DBMS possono essere acceduti tramite ODBC, il supporto ad ODBC è integrato nei sistemi operativi della famiglia Windows.

ADO (ActiveX Data Object) e ADO.NET: sono le API Object Oriented di Microsoft per l'accesso ai dati da applicazioni COM+ e .NET

OLE DB: è una API di basso livello usata dai tools.

JDBC e SQL

SQL, malgrado il suo nome, differisce largamente nelle diverse implementazioni.

Principali aree di diversità tra i vari DBMS sono:

- Il sistema dei tipi
- Le operazioni supportate (es. gli Outer Join)
- I formati per le date
- Le chiamate a stored procedures

NB Le API JDBC 3.0 sono conformi allo standard SQL99, tuttavia i driver dei DBMS possono limitarsi a ANSI SQL 92 Entry Level.

JDBC e SQL

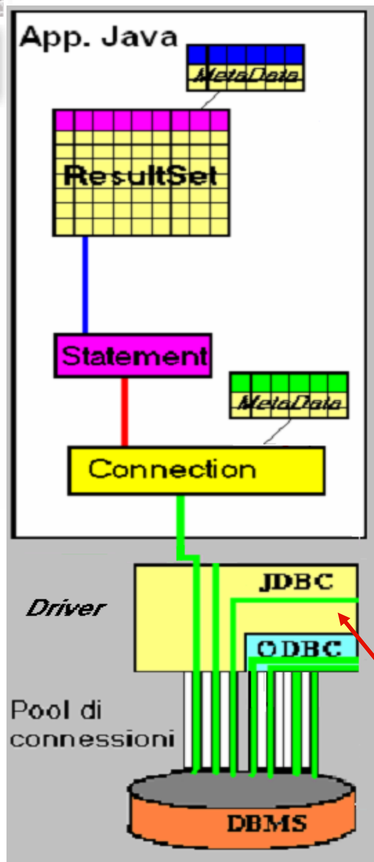
Per ovviare, parzialmente, a questi problemi in JBDC:

- E' possibile passare *qualsiasi* query al DBMS, lo sviluppatore deve accertarsi che questo sia in grado di eseguirla.
- E' disponibile una interfaccia (**java.sql.DatabaseMetaData**) tramite la quale ottenere informazioni sulle caratteristiche del DBMS.
- Nella classe **java.sql.Types** sono definiti gli identificatori per i tipi SQL più comuni.

RESEARCH CENTRE ON SOFTWARE TECHNOLOGY

Università degli Studi del Sannio

Elementi principali



Gli oggetti ResultSetMetaData contengono informazioni circa il ResulSet (numero e tipo delle colonne, possibilità di movimento, possibilità di scrittura)

Gli oggetti ResultSet contengono le tabelle risultanti dall'esecuzione dei comandi SELECT.

Gli oggetti Statement rappresentano i comandi che è possibile far eseguire al DBMS

Gli oggetti ConnectionMetaData permettono di accedere a informazioni circa il DB ed il DBMS (SQL supportato, possibilità, schemi ....)

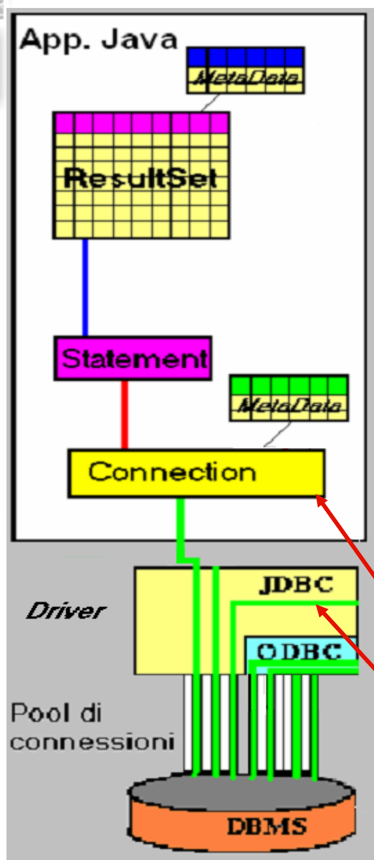
Gli oggetti Connection rappresentano i canali di comunicazione con il DBMS

I **driver** sono responsabili di tradurre i comandi dati tramite le API JDBC in opportune istruzioni al DBMS

RESEARCH CENTRE ON SOFTWARE TECHNOLOGY

Università degli Studi del Sannio

Elementi principali



Gli oggetti ResultSetMetaData contengono informazioni circa il ResulSet (numero e tipo delle colonne, possibilità di movimento, possibilità di scrittura)

Gli oggetti ResultSet contengono le tabelle risultanti dall'esecuzione dei comandi SELECT.

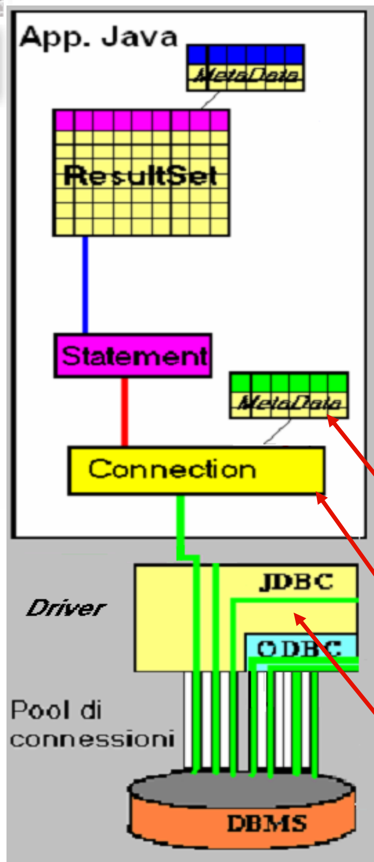
Gli oggetti Statement rappresentano i comandi che è possibile far eseguire al DBMS

Gli oggetti ConnectionMetaData permettono di accedere a informazioni circa il DB ed il DBMS (SQL supportato, possibilità, schemi ....)

Gli oggetti **Connection** rappresentano i canali di comunicazione con il DBMS

I driver sono responsabili di tradurre i comandi dati tramite le API JDBC in opportune istruzioni al DBMS

Elementi principali



Gli oggetti ResultSetMetaData contengono informazioni circa il ResulSet (numero e tipo delle colonne, possibilità di movimento, possibilità di scrittura)

Gli oggetti ResultSet contengono le tabelle risultanti dall'esecuzione dei comandi SELECT.

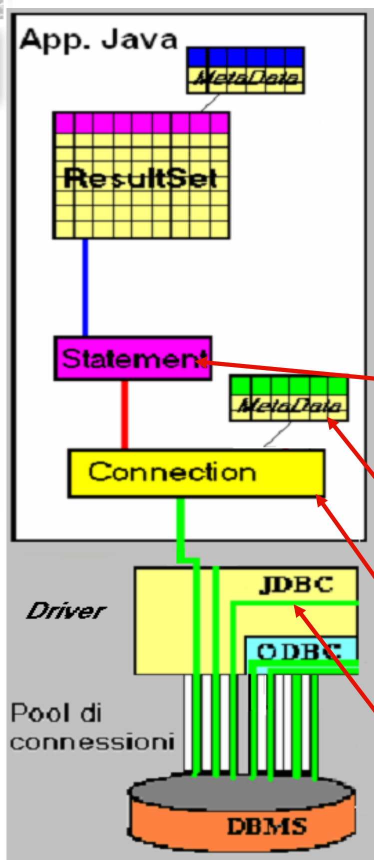
Gli oggetti Statement rappresentano i comandi che è possibile far eseguire al DBMS

Gli oggetti **ConnectionMetaData** permettono di accedere a informazioni circa il DB ed il DBMS (SQL supportato, possibilità, schemi ....)

Gli oggetti Connection rappresentano i canali comunicazione con il DBMS

I driver sono responsabili di tradurre i comandi dati tramite le API JDBC in opportune istruzioni al DBMS

Elementi principali



Gli oggetti ResultSetMetaData contengono informazioni circa il ResulSet (numero e tipo delle colonne, possibilità di movimento, possibilità di scrittura)

Gli oggetti ResultSet contengono le tabelle risultanti dall'esecuzione dei comandi SELECT.

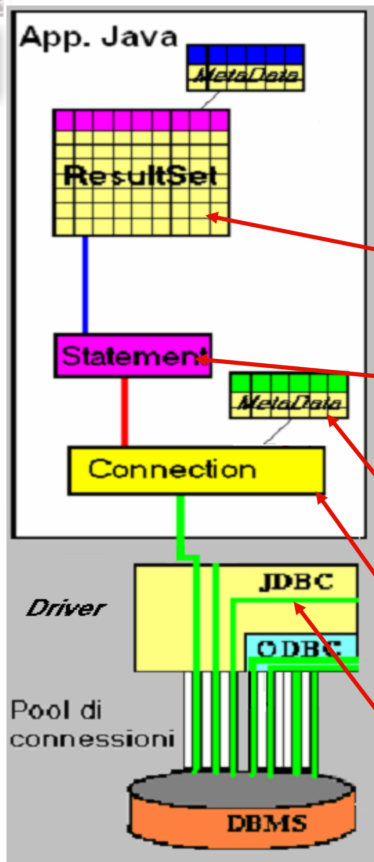
Gli oggetti **Statement** rappresentano i comandi che è possibile far eseguire al DBMS

Gli oggetti ConnectionMetaData permettono di accedere a informazioni circa il DB ed il DBMS (SQL supportato, possibilità, schemi ....)

Gli oggetti Connection rappresentano i canali di comunicazione con il DBMS

I driver sono responsabili di tradurre i comandi dati tramite le API JDBC in opportune istruzioni al DBMS

Elementi principali



Gli oggetti ResultSetMetaData contengono informazioni circa il ResulSet (numero e tipo delle colonne, possibilità di movimento, possibilità di scrittura)

Gli oggetti **ResultSet** contengono le tabelle risultanti dall'esecuzione dei comandi SELECT.

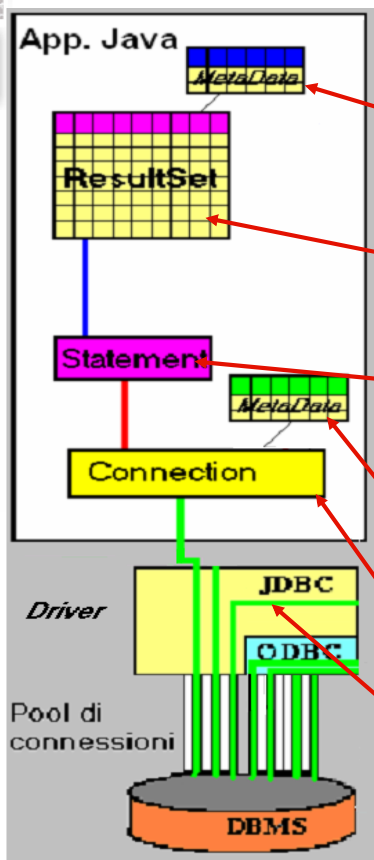
Gli oggetti Statement rappresentano i comandi che è possibile far eseguire al DBMS

Gli oggetti ConnectionMetaData permettono di accedere a informazioni circa il DB ed il DBMS (SQL supportato, possibilità, schemi ....)

Gli oggetti Connection rappresentano i canali di comunicazione con il DBMS

I driver sono responsabili di tradurre i comandi dati tramite le API JDBC in opportune istruzioni al DBMS

Elementi principali



Gli oggetti **ResultSetMetaData** contengono informazioni circa il ResulSet (numero e tipo delle colonne, possibilità di movimento, possibilità di scrittura)

Gli oggetti ResultSet contengono le tabelle risultanti dall'esecuzione dei comandi SELECT.

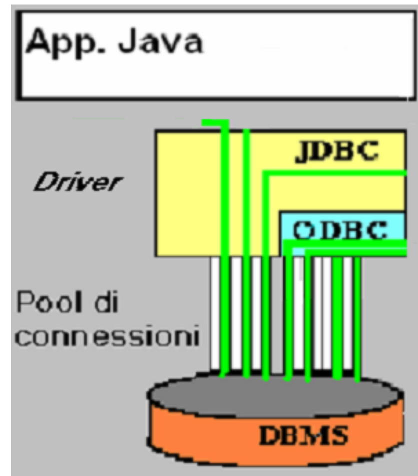
Gli oggetti Statement rappresentano i comandi che è possibile far eseguire al DBMS

Gli oggetti ConnectionMetaData permettono di accedere a informazioni circa il DB ed il DBMS (SQL supportato, possibilità, schemi ....)

Gli oggetti Connection rappresentano i canali di comunicazione con il DBMS

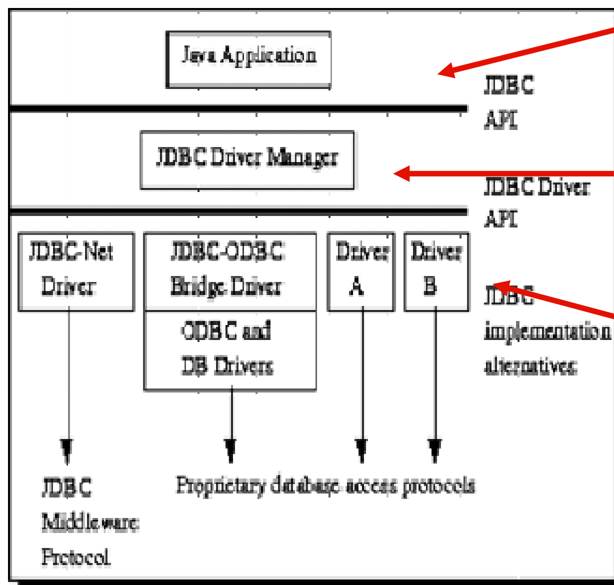
I driver sono responsabili di tradurre i comandi dati tramite le API JDBC in opportune istruzioni al DBMS

I driver



RESEARCH CENTRE ON SOFTWARE TECHNOLOGY
Università degli Studi del Sannio

JDBC uso dei driver



L'applicazione Java Esegue le chiamate tramite L'API JDBC

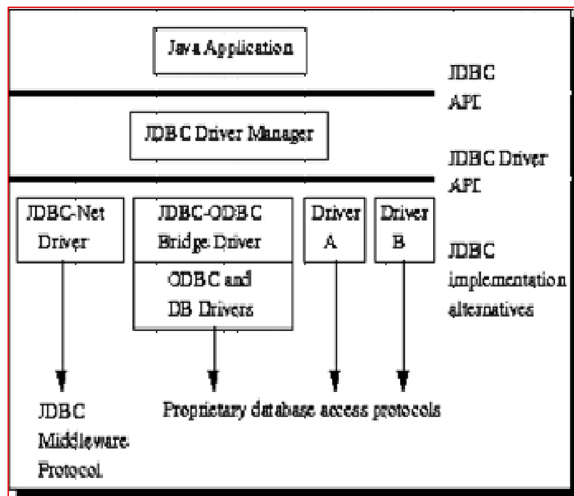
Il JDBC **DriverManager** connette l'applicazione con il driver specifico

I **Driver** eseguono il lavoro richiesto

RESEARCH CENTRE ON SOFTWARE TECHNOLOGY

Università degli Studi del Sannio

Driver **JDBC**



JDBC-Net : Il driver traduce le chiamate in un protocollo indipendente dal DBMS e le invia ad un server. Il server traduce le chiamate per lo specifico DBMS.

Bridge: Il driver JDBC converte le chiamate nelle API previste per il DBMS.

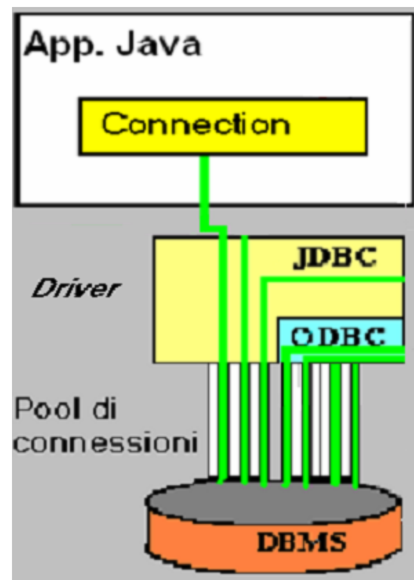
- Il driver **ODBC** è specifico per il DBMS ed il SO.
- La parte Java è standard nella API Java

***Native-protocol pure Java driver**:* Il driver, interamente scritto in Java, converte direttamente le chiamate nel protocollo di rete nativo del DBMS. **Specifico per il DBMS.**

RESEARCH CENTRE ON SOFTWARE TECHNOLOGY

Università degli Studi del Sannio

La Connessione



La Connessione

Un oggetto di tipo **java.sql.Connection** rappresenta una connessione (sessione) con un database. I comandi al DBMS ed i risultati sono inviati e ricevuti tramite la connessione.

RESEARCH CENTRE ON SOFTWARE TECHNOLOGY
Università degli Studi del Sannio

Ottenere una connessione

Per ottenere una connessione invocare il metodo **DriverManager.getConnection()**.

- Il metodo accetta tra i parametri una URL.
- Cerca tra le classi caricate un driver capace di comunicare con il database specificato nella URL.
- Attiva una connessione.

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

//Carica il driver bridge JDBC-ODBC

```
String url = "jdbc:odbc:Mydb";
Connection con = DriverManager.getConnection(url,
                    "Gabriele", "myPW");
```

//Apre una connessione con il DB 'Mydb' con username 'Gabriele' e password 'myPW'

Ottenere e rilasciare una connessione

Le connessioni occupano risorse importanti. Pertanto è importante accertarsi di rilasciarle appena possibile

```
Connection con;
con = DriverManager.getConnection(url, uname, upw);
    // . . . codice che esegue il lavoro
con.close();
```

RESEARCH CENTRE ON SOFTWARE TECHNOLOGY
Università degli Studi del Sannio

Nota sulle URL 1

Una URL (***U**niform **R**esource **L**ocator*) non è altro che un indirizzo di una risorsa in rete:

ftp://javasoft.com/docs/JDK-1_apidocs.zip

http://java.sun.com/products/JDK/CurrentRelease

file:/home/haroldw/docs/tutorial.html

La prima parte, sempre seguita dai due punti, (ftp:, http:, file:) indica il protocollo

Il seguito indica l'indirizzo della risorsa specificato in accordo con il protocollo

RESEARCH CENTRE ON SOFTWARE TECHNOLOGY

Nota sulle URL 2

La convenzione per le URL prevede tre parti

jdbc:<*subprotocol*>:<*subname*>

1. jdbc: caratterizza tutte le URL JDBC
2. <*subprotocol*>: è il nome del driver o del meccanismo di connettività usato (es odbc:)
3. <*subname*> indica la risorsa, il suo formato dipende dal <*subprotocol*>

jdbc:dbnet://wombat:356/fred

jdbc:odbc:fred

Ulteriori dettagli del contenuto di una URL JDBC sono determinati da chi scrive il driver.

RESEARCH CENTRE ON SOFTWARE TECHNOLOGY

Il subprotocol jdbc di mySQL

Il subprotocol mysql ha come formato :

jdbc:mysql://[host][,failoverhost...][:port]/[database][?propertyName1][=propertyValue1][&propertyName2][=propertyValue2]...

Se i nomi host sono assenti verrà contattato 127.0.0.1

Se è assente la porta sarà usata la 3306

Jdbc:mysql:///pluto connette al db *pluto* sulla macchina locale dalla porta 3306 (default di my sql)

Jdbc:mysql://db.rcost.unisannio.it:3388/personale connette al db *personale* sulla macchina db.rcost.unisannio.it dalla porta 3388

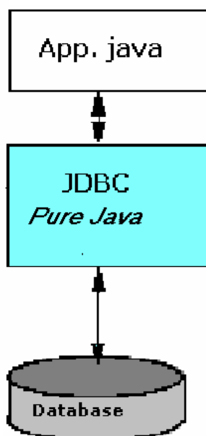
NB perché possa funzionare

La macchina locale deve poter risolvere il nome **db.rcost.unisannio.it** (es. tramite DNS)

Preparazione del sistema

RESEARCH CENTRE ON SOFTWARE TECHNOLOGY
Università degli Studi del Sannio

Preparazione del sistema

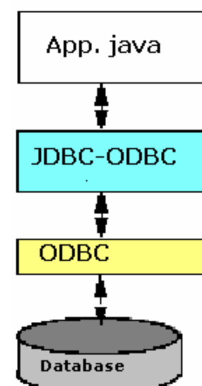


JDBC

- Posizionamento dei file su disco
- Settaggio della variabile CLASSPATH
- Connessione da Java tramite il driver

JDBC-ODBC

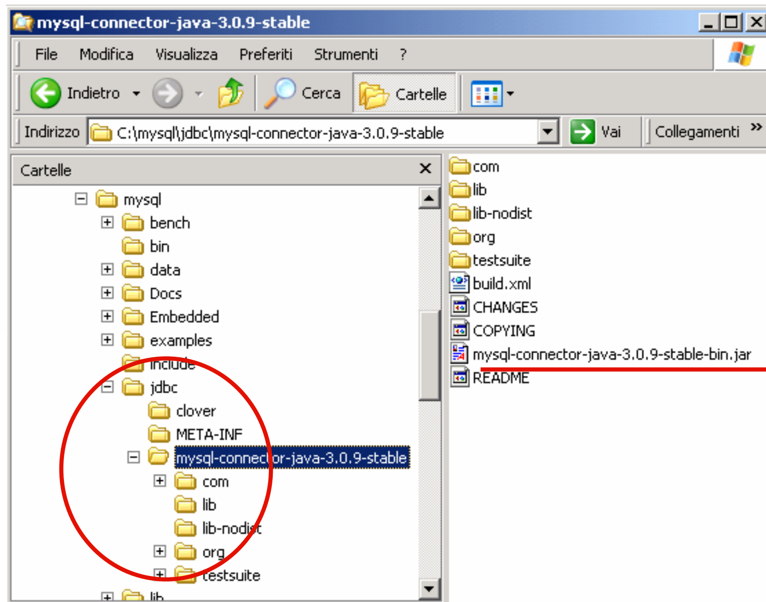
1. Installazione del driver ODBC di MySQL
2. Registrazione del database come fonte dati ODBC presso il sistema operativo
3. Connessione da Java tramite il bridge



I Driver JDBC

I Driver possono essere scaricati come file

Il file deve essere scompattato in una dir a scelta dell'utente.



File JAR che contiene i driver

Documentazione: http://www.mysql.com/documentation/connector-j/index.html

RESEARCH CENTRE ON SOFTWARE TECHNOLOGY
Università degli Studi del Sannio

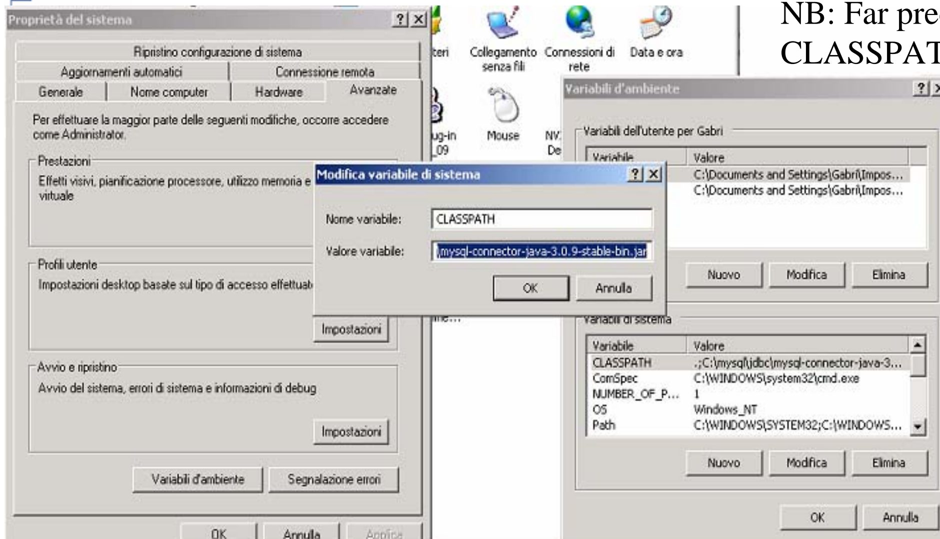
IL CLASSPATH

Le applicazioni Java trovano le classi che gli necessitano cercandole nei percorsi specificati nella variabile d'ambiente **CLASSPATH**

Pannello di controllo → Sistema –> Avanzate → Variabili d'ambiente.

Impostare il percorso completo di nome del file

NB: Far precedere il percorso da "**.;**"
CLASSPATH=.;C:\mysql……



Connessione da JDBC

```
import java.sql.*;
public class TestCon {
   public static void main(String[] argv) throws ClassNotFoundException,
        SQLException{
        Class.forName( "com.mysql.jdbc.Driver " ); //... Carica il Driver
        String url = "jdbc:mysql:///forum";
        Connection con= DriverManager.getConnection(url);
        System.out.println("Connessione riuscita!");
       //... Qui possiamo usare la connessione per effettuare il lavoro
        con.close();
   }
 };
```

RESEARCH CENTRE ON SOFTWARE TECHNOLOGY

Università degli Studi del Sannio

Installazione Driver ODBC

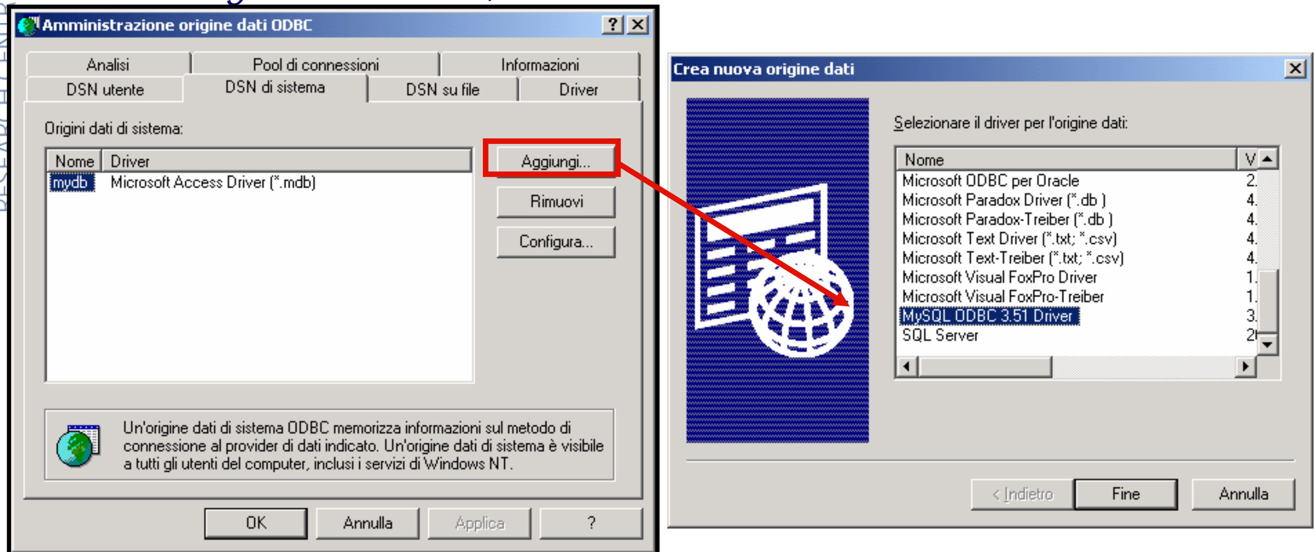
I driver ODBC per MySQL sotto Windows sono prelevati dal sito www.mysql.com nel file autoinstallante *MyODBC-3.51.06.exe*

MyODBC-3.51.06.exe

Registrazione del database

Registrazione del db come fonte dati ODBC

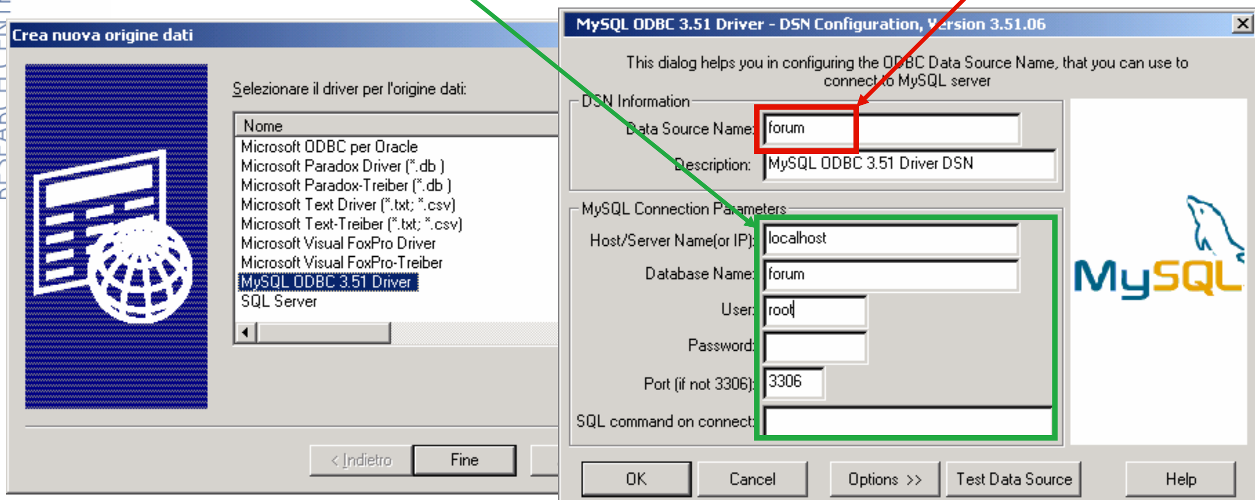
Da *Pannello di controllo → Strumenti di amministrazione → Origine dati ODBC* , inserire un nuovo DSN



Registrazione del database

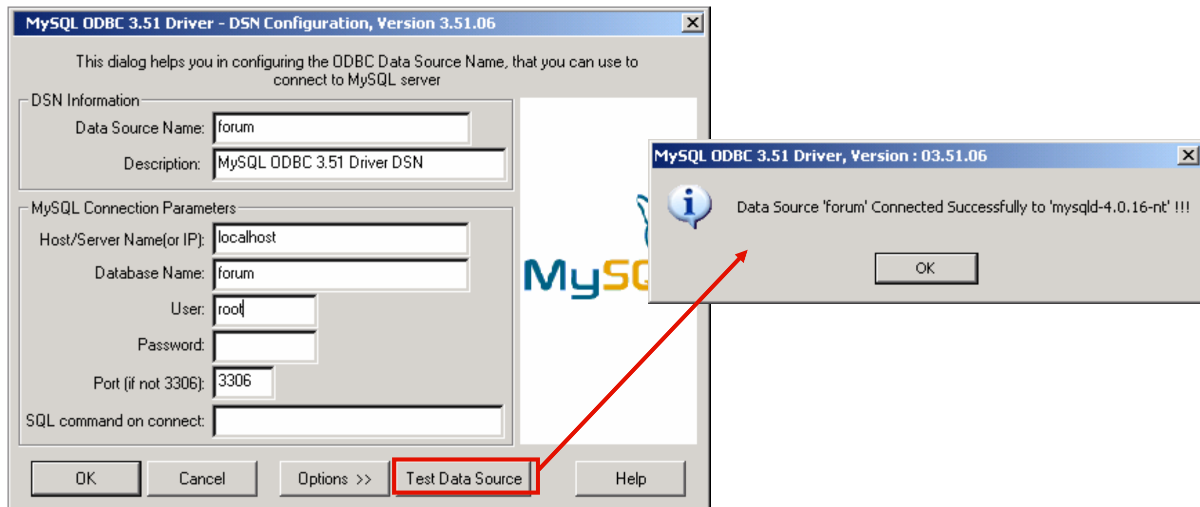
Inserire il nome con cui sarà identificata la sorgente dati da ODBC

e le informazioni riguardanti il DBMS ed il database



Registrazione del database

Testare la connessione

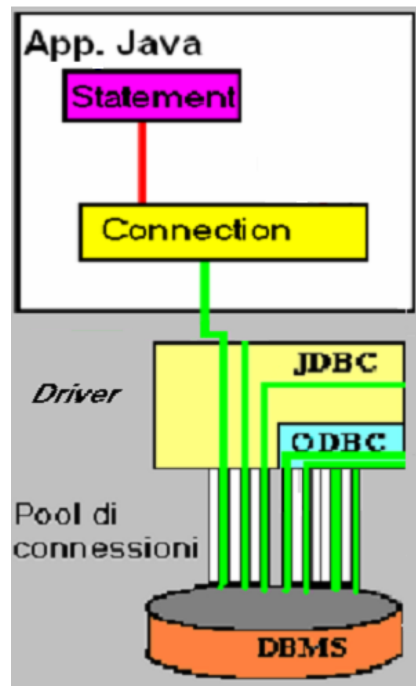


RESEARCH CENTRE ON SOFTWARE TECHNOLOGY
Università degli Studi del Sannio

Connessione da JDBC

```
import java.sql.*;
public class TestCon {
   public static void main(String[] argv) throws ClassNotFoundException, SQLException{
      Class.forName( "sun.jdbc.odbc.JdbcOdbcDriver“ );  //... Carica il Driver bridge
      String url = "jdbc:odbc:forum“ ;
      Connection con= DriverManager.getConnection(url);
      // qui possiamo usare la connessione
      System.out.println("Connessione riuscita!");
      con.close();
   }
}
```

L'invio di comandi



L'invio di comandi

Tramite una connessione è possibile inviare al database comandi modellati da tre diversi tipi di oggetti:

1. **Statement**: creati dal metodo **createStatement**, rappresentano comandi SQL senza parametri.
2. **PreparedStatement**: creati dal metodo **prepareStatement**, rappresentano comandi SQL che possono essere precompilati nel database ed accettare parametri (di IN) al momento dell'esecuzione
3. **CallableStatement**: creati dal metodo **prepareCall**, che possono essere usati per eseguire stored procedure ed accettano anche parametri di OUT e INOUT.

RESEARCH CENTRE ON SOFTWARE TECHNOLOGY

Università degli Studi del Sannio

L'invio di comandi

La creazione di un oggetto di tipo **Statement** è effettuata da un oggetto di tipo **Connection**:

```
Connection con = DriverManager.getConnection(...);
Statement stmt = con.createStatement();
```

Il particolare comando da eseguire può essere passato come parametro a uno dei metodi di esecuzione:

```
ResultSet rs = stmt.executeQuery("SELECT a, b FROM Table2;");
```

RESEARCH CENTRE ON SOFTWARE TECHNOLOGY

Università degli Studi del Sannio

Tipi di metodi di esecuzione

Sono disponibili tre tipi principali di metodi di esecuzione

executeQuery: Per statement SQL che producono un singolo set di risultati (es SELECT). Il metodo restituisce un **ResultSet**.

executeUpdate: Per statement DML quali INSERT, UPDATE, o DELETE e DDL come CREATE TABLE, DROP TABLE, and ALTER TABLE. Il metodo restituisce un intero (*update count*) che indica il numero di righe modificate per le istruzioni DML ed è sempre 0 per le istruzioni DDL.

execute: Per metodi che possono restituire più di un set di risultati.

PreparedStatement e CallableStatement hanno le proprie versioni di questi metodi

Esempio di esecuzione

Modifichiamo il livello dell'utente *Squitty*:

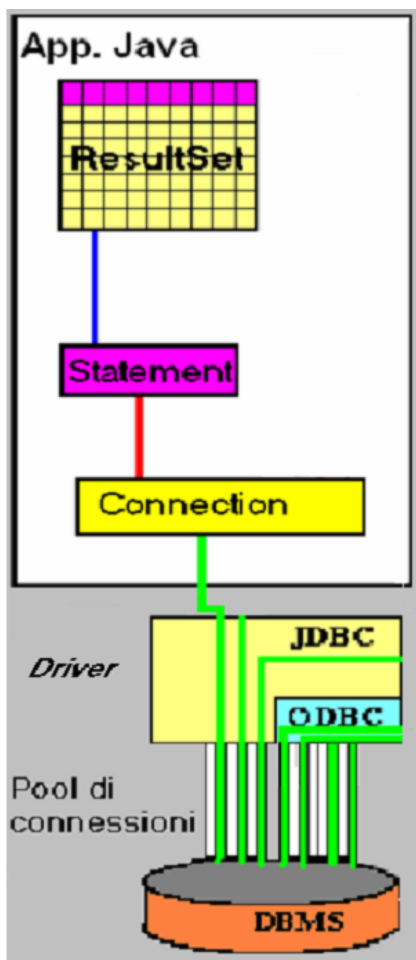
Versione 1

```
Statement stm =con.createStatement();
int affectedRows =stm.executeUpdate("UPDATE user SET
    U_LIVELLO=\"0\" WHERE U_USERNAME=\"Squitty\";");
```

Versione 2

```
String newLevel="0";
String uName="Squitty";
.............
Statement stm =con.createStatement();
int affectedRows = stm.executeUpdate("UPDATE user SET
    U_LIVELLO='"   +newLevel+"' WHERE
    U_USERNAME='"+uName+"';");
```

NB Cosa succede se alla variabile uName si assegna uName="D'accursio"?



Il ResultSet

Il ResultSet

I risultati di una query sono restituiti in un oggetto di tipo **`ResultSet`** che li contiene organizzati per riga e colonna

1 NOME	2 COGNOME	3 UNAME	4 PW	5 CF	6 LIVELLO	7 TEL	8 VIA	9 CITTA	10 CAP
				Dati					

- I metodi **`getTipo(String nomeColonna)`** o **`getTipo(int indiceColonna)`** permettono di prelevare i dati dalla riga corrente.
- Il **`ResultSet`** appena ottenuto è posizionato prima dell'inizio.
- Il metodo **`ResultSet.next()`** permette di avanzare riga per riga e restituisce false quando si giunge al termine.

NB: Il **ResultSet** rimane collegato allo **Statement** da cui è stato generato, **executeQuery** ed **executeUpdate** chiudono il **ResultSet** precedentemente creato dallo **Statement** su cui sono invocati.

RESEARCH CENTRE ON SOFTWARE TECHNOLOGY
Università degli Studi del Sannio

Esempio di esecuzione

Interroghiamo il database per ottenere il contenuto della tabella persone, vogliamo ottenere una tabella con username, nome, cognome, livello e data dell'ultima visita.

```
Statement stm =con.createStatement();
ResultSet rs=stm.executeQuery("SELECT U_USERNAME, U_NOME,
                U_COGNOME, U_LIVELLO, U_ULTIMA_VISITA FROM user");
while (rs.next()){
    System.out.print(rs.getString("U_USERNAME") + "; "
                    + rs.getString("U_NOME") + "; "
                    + rs.getString("U_COGNOME") + "; "
                    + rs.getString("U_LIVELLO") + "; "
                    + rs.getTimestamp("U_ULTIMA_VISITA"));
}
```

RESEARCH CENTRE ON SOFTWARE TECHNOLOGY
Università degli Studi del Sannio

Metodi di get

X - indica che il metodo è il metodo d'elezione per quel tipo SQL

x – indica che il metodo può essere usato per valori di quel tipo SQL

	TINYINT	SMALLINT	INTEGER	BIGINT	REAL	FLOAT	DOUBLE	DECIMAL	NUMERIC	BIT	BOOLEAN	CHAR	VARCHAR	LONGVARCHAR	BINARY	VARBINARY	LONGVARBINARY	DATE	TIME	TIMESTAMP	CLOB	BLOB	ARRAY	REF	STRUCT	DATALINK	JAVA OBJECT
getByte	X	x	x	x	x	x	x	x	x	x	x	x	x	x													
getShort	x	X	x	x	x	x	x	x	x	x	x	x	x	x													
getInt	x	x	X	x	x	x	x	x	x	x	x	x	x	x													
getLong	x	x	x	X	x	x	x	x	x	x	x	x	x	x													
getFloat	x	x	x	x	X	x	x	x	x	x	x	x	x	x													
getDouble	x	x	x	x	x	X	X	x	x	x	x	x	x	x													
getBigDecimal	x	x	x	x	x	x	x	X	X	x	x	x	x	x													
getBoolean	x	x	x	x	x	x	x	x	x	X	X	x	x	x													
getString	x	x	x	x	x	x	x	x	x	x		X	X	x	x	x	x	x	x	x						x	
getBytes															X	X	x										
getDate												x	x	x				X		x							
getTime												x	x	x					X	x							
getTimestamp												x	x	x				x	x	X							
getAsciiStream												x	x	X	x	x	x										
getUnicodeStream												x	x	X	x	x	x										
getBinaryStream															x	x	X										
getClob																					X						
getBlob																						X					
getArray																							X				
getRef																								X			
getCharacterStream												x	x	X	x	x	x										
getURL																										X	
getObject	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	X	x	X

Lettura di valori SQL NULL

Un valore **NULL** può essere convertito da JDBC in ***null***, ***0*** o ***false***

- **null** si ottiene da quei metodi che restituiscono oggetti java (getString, getBigDecimal, getBytes, getDate, getTime, getTime-stamp, getAsciiStrea …)
- **0** (zero) si ottiene da getByte, getShort, getInt, getLong, getFloat e getDouble
- **false** si ottiene da getBoolean

Tuttavia si può usare

boolean **wasNull**()

Il metodo **wasNull()** restituisce true se l'ultima lettura ha prodotto NULL

Il cursore

Ogni oggetto **ResultSet** mantiene un riferimento alla riga corrente (cursore).

Nei **ResulSet** di default (ed in tutti i **ResultSet** JDBC 1.0) il cursore può essere mosso solo in avanti con il metodo **next()**, I campi devono essere letti da sx verso dx ed una sola volta per campo.

- **ResultSet.TYPE_FORWARD_ONLY**

Da JDBC 2.0 sono disponibili **ResultSet** *scrollable* che possono essere esplorati con i metodi **previous()**, **first()**, **last()**, **absolute()**, **relative()**, **afterLast()** e **beforeFirst()**

- **ResultSet.TYPE_SCROLL_SENSITIVE**, permette di visuallizare cambiamenti dovuti ad altri utenti
- **ResultSet.TYPE_SCROLL_INSENSITIVE**, è insensibile ai cambiamenti dovuti ad altri utenti

Le caratteristiche di scrolling di un **ResultSet** possono essere accedute con il metodo **getType()**

Modificabilità del ResultSet

Un **ResultSet** può essere o meno modificabile. Tramite il metodo **ResulSet.getConcurrency()** si ottiene una delle seguenti costanti, definite sempre in **ResultSet**,

1. **CONCUR_READ_ONLY**: Il contenuto non può essere modificato. E' il solo tipo disponibile in JDBC 1.0 ed offre il maggior livello di concorrenza (acquisisce lock read-only)
2. **CONCUR_UPDATABLE**: Il contenuto può essere modificato

RESEARCH CENTRE ON SOFTWARE TECHNOLOGY

Università degli Studi del Sannio

Persistenza del ResultSet

I driver possono chiudere tutti i **ResultSet** coinvolti in una scrittura su DB al termine di questa.

- **ResultSet.HOLD_CURSORS_OVER_COMMIT**: I **ResultSet** aperti da questa connessione non sono chiusi dopo il commit.
- **ResultSet.CLOSE_CURSORS_AT_COMMIT**: I **ResultSet** creati da questa connessione sono chiusi dopo i commit.

RESEARCH CENTRE ON SOFTWARE TECHNOLOGY

Università degli Studi del Sannio

Ottenere ResultSet particolari

I **ResultSet** sono creati tramite l'invocazione di metodi di oggetti di tipo **Statement**.

Quando un oggetto di tipo **Statement** viene creato può essere determinato il tipo di **ResultSet** che questo produrrà

```
Connection con = DriverManager.getConnection(
                    "jdbc:my_subprotocol:my_subname");
Statement stmt = con.createStatement(
        ResultSet.TYPE_SCROLL_SENSITIVE,
        ResultSet.CONCUR_UPDATABLE),
        ResultSet.HOLD_CURSORS_OVER_COMMIT);
```

Lo Statement **stm** produrrà **ResultSet** *Scrollable*, sensibili ai cambiamenti del DB, modificabili e che rimangono disponibili dopo il commit.

Metodi di update

Per modificare i contenuti della riga corrente di un RecordSet sono disponibili opportuni metodi di update.

update*Type*(int col**,** *Type* value)
update*Type*(String colName**,** *Type* value)

Dove *Type* è il tipo java corrispondente (il driver si incarica di trasformare i valori secondo necessità),
col e colname sono rispettivamente l'indice o il nome della colonna*

```
int n = rs.getInt(3);
rs.updateInt(3, 91);
rs.updateRow();
```

```
int n = rs.getInt("SCORES");
rs.updateInt("SCORES", 88);
rs.updateRow();
```

NB Le modifiche sono trasferite al DB alla chiamata del metodo updateRow. Dato che questo trasferisce solo la riga corrente è fondamentale invocarlo quando ancora si è posizionati su questa

* L'indice della colonna nel ResultSet può essere diverso dallo stesso nella tabella

Inserire ed eliminare righe

```
rs.deleteRow(); // Elimina la riga corrente
```

La *insert row* è una riga associata ad un ResultSet ma non ancora parte di questo:

```
rs.moveToInsertRow();                  // sposta il cursore alla insert row
rs.updateObject(1, myArray);           // inserisce il primo elemento
rs.updateInt(2, 3857);                 // inserisce il secondo elemento
rs.updateString(3, "Mysteries");       // inserisce il terzo elemento
rs.insertRow();                        // inserisce la insert row nel ResultSet
rs.moveToCurrentRow();                 // si posiziona all'ultima riga visitata
```

NB

- Se si legge un valore dalla insert row prima di averlo assegnato il valore è indefinito
- I valori vengono immessi nel DB alla chiamata del metodo insertRow, (ciò può generare una SQLException)
- **moveToCurrentRow** riporta il cursore alla posizione precedente la chiamata di **moveToInsertRow**

RESEARCH CENTRE ON SOFTWARE TECHNOLOGY

Università degli Studi del Sannio

Una vista d'insieme

Per utilizzare JDBC

Sistema

Preparare il sistema con gli eventuali driver (JDBC, ODBC, ecc) e configurazioni

Applicazione

1. Caricare i driver necessari (una volta per applicazione)
2. Ottenere oggetti **Connection** (dal **DriverManager**)

   *Per ogni **Connection***

   1. Ottenere oggetti **Statement** (dalla **Connection**)

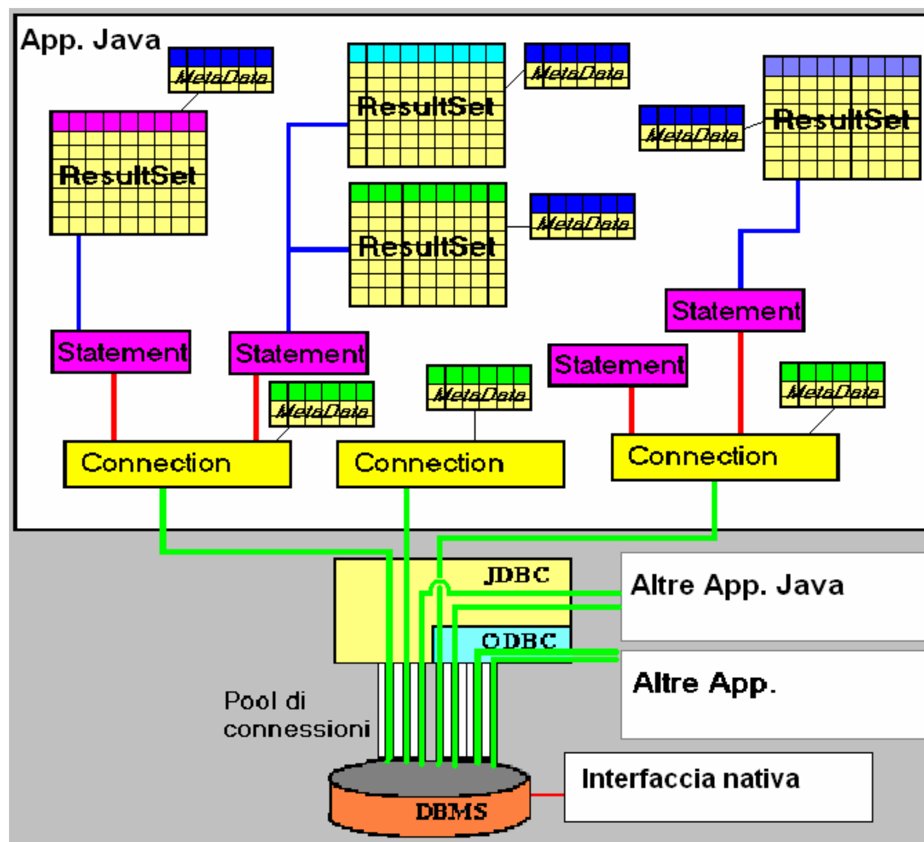
      *Per ogni **Statement***

      1. Preparare le istruzioni SQL necessarie
      2. Ottenere i risultati (eventualmente oggetti **ResultSet**) facendo eseguire le istruzioni SQL agli oggetti **Statement**
      3. Esplorare i risultati ed eventualmente modificarli
      4. Salvare i risultati modificati nel **ResultSet** con **updateRow** e **insertRow**

RESEARCH CENTRE ON SOFTWARE TECHNOLOGY

Università degli Studi del Sannio

Una vista d'insieme



I Driver ed il DriverManager

- Il DriverManager si occupa di cercare il driver opportuno per la connessione richiesta
- I Driver necessari devono essere accessibili dalla JVM e caricati una volta per ogni run dell'applicazione che ne necessita

RESEARCH CENTRE ON SOFTWARE TECHNOLOGY
Università degli Studi del Sannio

Gli oggetti Connection

- Modellano le connessioni di basso livello con il database
- Tramite gli oggetti di tipo Connection avviene la comunicazione con il db ed è possibile acquisire i metadati dello stesso
- Le connessioni di basso livello sono mantenute nel pool di connessioni dai driver JDBC (e ODBC)
- Quando un oggetto Connection viene creato una connessione è prelevata dal pool ed associata all'oggetto
- Quando l'oggetto è distrutto dal GC o viene invocato il metodo **close** la connessione è rilasciata al pool
- Le connessioni sono risorse scarse: E' consigliabile trattenerle per sé il meno possibile

NB. Se il driver effettua connection pooling ottenere e rilasciare una connessione è relativamente veloce, se il driver non effettua connection pooling ottenere una connessione è una operazione time consuming.

Gli oggetti Statement

- Modellano i comandi che è possibile eseguire sui DB
- Sono creati dalle connessioni
- Sono distrutti dal GC oppure chiusi con gli appositi metodi
- Al momento della loro creazione è determinato il tipo degli oggetti ResultSet che potranno produrre
- Dispongono di metodi per essere eseguiti

Gli oggetti ResultSet

- Modellano il contenuto delle tabelle del DB come risulta dalle query (e altri dati in forma tabellare)
- Possono essere acceduti riga per riga
- Dispongono di metadati che contengono le informazioni circa le colonne
- Sono creati dall'esecuzione di oggetti Statement
- Sono distrutti alla distruzione (e, normalmente, alla riesecuzione) dello Statement con cui sono stati creati
- Il loro tipo è determinato dall'oggetto Statement che li crea

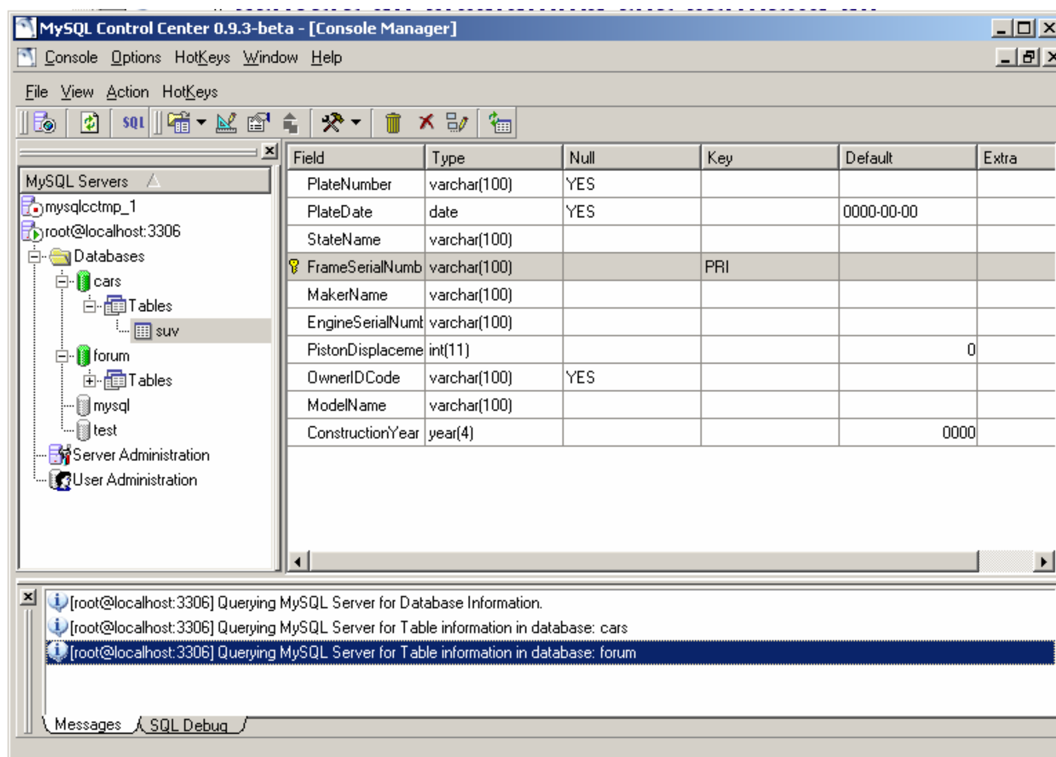
Esempio

Scrivere un programma che permetta di:

- Visualizzare una lista delle auto registrate nella tabella **suv** e che riporti, di ogni auto, FrameNumber, PlateNumber, OwnerID e PistonDisplacement.

- Accedere ai dati di un suv per mezzo del numero di telaio (chiave) e modificarne i dati.

Database



RESEARCH CENTRE ON SOFTWARE TECHNOLOGY

Università degli Studi del Sannio

Interfaccia utente

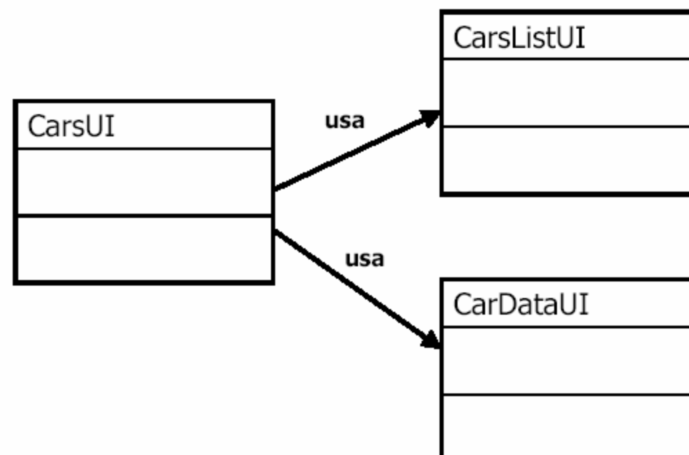
L'interfaccia utente deve svolgete tre compiti principali:

- Permettere di scegliere che funzionalità usare
- Visualizzare la lista
- Permettere di accedere ai dati dell'auto da modificare

RESEARCH CENTRE ON SOFTWARE TECHNOLOGY

Università degli Studi del Sannio

Struttura dell'interfaccia utente



RESEARCH CENTRE ON SOFTWARE TECHNOLOGY

La classe principale della UI

```
import java.io.*;
public class CarsUI {

// helper method that prints the main menu and collects the user choice
static int menu() throws NumberFormatException, IOException{
		System.out.println("****************************************");
		System.out.println("\t Options");
		System.out.println("\t0\t Quit the program");
		System.out.println("\t1\t Show car list");
		System.out.println("\t2\t Change car's data");
		System.out.println("****************************************");
		System.out.print("Insert your choice: ");
		return (Integer.parseInt(kbr.readLine()));
}

public static void main(String[] args) throws NumberFormatException, IOException, SQLException
{
		int choice;
		java.lang.Class.forName(driver); // loads the JDBC Driver
		do{		choice=menu();
				switch (choice){
						case 1:	(new CarsListUI(urldb)).list(); break;
						case 2:	System.out.print("Insert the Car's Frame Serial
										Number: ");
								(new CarDataUI(kbr.readLine(), urldb)).collectData();
				}
		}while (choice!=0);
}
static String driver ="com.mysql.jdbc.Driver" , urldb="jdbc:mysql:///cars";
// private static String driver ="sun.jdbc.odbc.JdbcOdbcDriver" , urldb="jdbc:odbc:forum";
static BufferedReader kbr=new BufferedReader(new InputStreamReader(System.in));
}
```

RESEARCH CENTRE ON SOFTWARE TECHNOLOGY

Visualizzazione lista

```
import java.util.*;
import java.sql.SQLException;
public class CarsListUI
{
List simpleCars; // Contains the SimpleCars from the DB

// Builds a new interface instance with a List obtained from the SimpleCar static method
public CarsListUI(String db) throws SQLException {
		simpleCars = SimpleCarDBImpl.SimpleCarList(db);
}

// Prints the List content
void list() throws IOException {
  if (simpleCars==null){System.out.println("List not availlable"); return;}
  Iterator iter=simpleCars.iterator();
  System.out.println("FRAME; PLATE; MAKER; OWNER ID; PISTON DISPLACEMENT ");
  while (iter.hasNext()){
		SimpleCar c= (SimpleCar) iter.next();
		System.out.println(c.getFrameSerialNumber()+"; "+ c.getPlateNumber()+"; "
							+c.getMakerName()+"; "+c.getOwnerID()+"; “
					+ c.getPistonDisplacement());
   }
  }
}
```

Modifica dati

```
import java.util.*;
import java.io.*;
import java.sql.SQLException;
public class CarDataUI {
Car c; // Car object to be used
// Creates a new instance of CarDataUI
public CarDataUI(String frameN, String dbUrl) throws SQLException{
        c=new CarDBImpl(frameN, dbUrl);
 }
// helper method that shows the Car's data
void show(){
 System.out.println("******* Car data ****************************************************");
 System.out.println("PLATE NUMBER=" + c.getPlateNumber() + " \t PLATE DATE=" +
 c.getPlateDate());
 System.out.println("STATE =" c.getStateName()+" \t FRAME
 NUMBER="+c.getFrameSerialNumber());
 System.out.println("MAKER ="+c.getMakerName()+" \t ENGINE
 NUM="+c.getEngineSerialNumber());
 System.out.println("PISTON DISP=" + c.getPistonDisplacement() + " \t OWNER=" + c.getOwnerID());
 System.out.println("MODEL=" + c.getModelName() + " \t CONST. YEAR=" +
 c.getConstructionYear());
 System.out.println("*********************************************************************");
 }
// Get Car's new data
public void collectData() throws IOException, NumberFormatException, SQLException {
………..
// Questo metodo interagisce con l'utente e salva i dati nell'oggetto di tipo Car
……….. } }
```

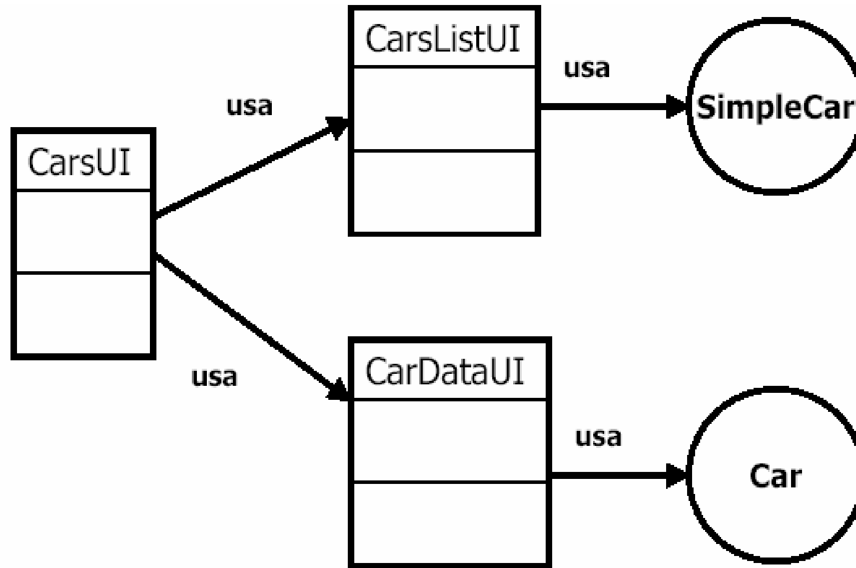
Modifica dati

```
public void collectData() throws IOException, NumberFormatException, SQLException {
 if (c==null){System.out.println("Car not availlable!"); return;}
 BufferedReader kbr=new BufferedReader(new InputStreamReader(System.in));
 String in;
 show();
 System.out.println("*** Insert new data (Empty string or incorrect data confirm current
 data)");
 System.out.print("PLATE NUMBER= "+ c.getPlateNumber()+" :");
        if ((in=kbr.readLine()).length()>0) c.setPlateNumber(in);
 System.out.print("PLATE DATE= "+c.getPlateDate()+" :");
        if ((in=kbr.readLine()).length()>0) c.setPlateDate(java.sql.Date.valueOf(in));
 System.out.print("STATE NAME= "+ c.getStateName()+" :");
        if ((in=kbr.readLine()).length()>0) c.setStateName(in);
 System.out.print("MAKER NAME= "+ c.getMakerName()+" :");
        if ((in=kbr.readLine()).length()>0) c.setMakerName(in);
 System.out.print("ENGINE SERIAL NUMBER= "+ c.getEngineSerialNumber()+" :");
        if ((in=kbr.readLine()).length()>0) c.setEngineSerialNumber(in);
 System.out.print("PISTON DISPLACEMENT= "+ c.getPistonDisplacement()+" :");
        if ((in=kbr.readLine()).length()>0) c.setPistonDisplacement(Integer.parseInt(in));
System.out.print("OWNER ID CODE= "+ c.getOwnerID()+" :");
        if ((in=kbr.readLine()).length()>0) c.setOwnerID(in);
System.out.print("MODEL NAME= "+ c.getModelName()+" :");
        if ((in=kbr.readLine()).length()>0) c.setModelName(in);
System.out.print("CONSTRUCTION YEAR= "+ c.getConstructionYear()+" :");
        if ((in=kbr.readLine()).length()>0) c.setConstructionYear(Integer.parseInt(in));
show();
char yn;
do {    System.out.print("Are these data correct? (Y/N)");
         yn=kbr.readLine().toUpperCase().charAt(0);
 }while (yn!='Y'&& yn!='N');
if (yn=='Y'){ c.save(); System.out.println("Data saved");
}
```

RESEARCH CENTRE ON SOFTWARE TECHNOLOGY

Università degli Studi del Sannio

Nuovi tipi usati



RESEARCH CENTRE ON SOFTWARE TECHNOLOGY

Università degli Studi del Sannio

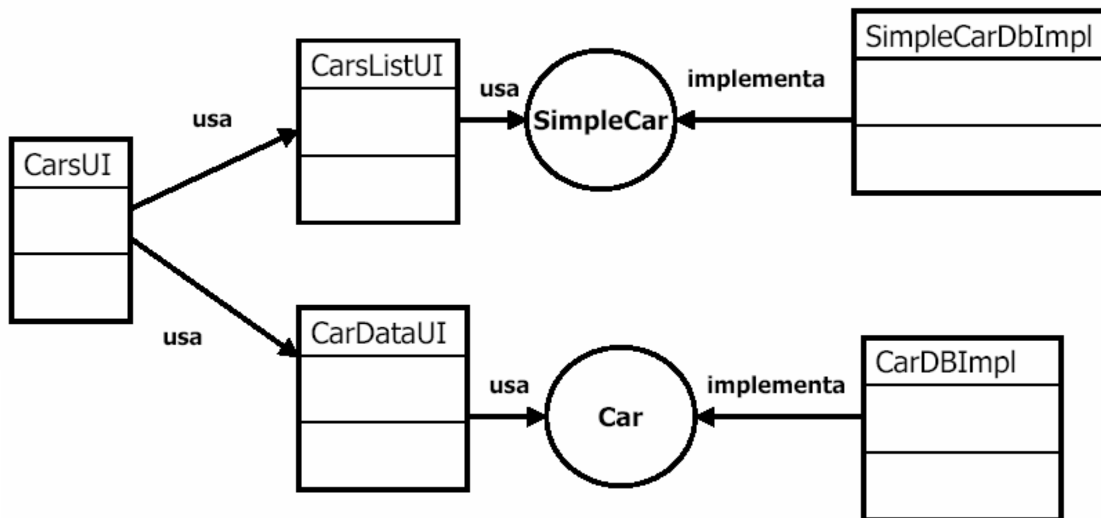
Interfaces

```
public interface SimpleCar {
  String
  getFrameSerialNumber();
  String getPlateNumber();
  String getMakerName();
  String getOwnerID();
  int getPistonDisplacement();
}
```

```
import java.sql.Date;
import java.sql.SQLException;
public interface Car{
  void save() throws SQLException;
  // Get methods
  Date getPlateDate();
  String getPlateNumber();
  String getStateName();
  String getFrameSerialNumber();
  String getMakerName();
  String getEngineSerialNumber();
  String getOwnerID();
  String getModelName();
  int getPistonDisplacement();
  int getConstructionYear();
  // Set methods
  void setPlateDate(Date plateD);
  void setPlateNumber(String plateN);
  void setStateName(String state);
  void setMakerName(String maker);
  void setEngineSerialNumber(String engineN);
  void setOwnerID(String ownerFC);
  void setPistonDisplacement(int PistonD);
  void setModelName(String modelName);
  void setConstructionYear(int year);
}
```

Università degli Studi del Sannio

Implementazioni delle interfaces



SimpleCarDBImpl

```
import java.sql.*;
import java.util.*;
public class SimpleCarDBImpl implements SimpleCar{
  // instace variables
  String frame, plate, maker, owner;
  int pistonDisp;

// get methods
  public String getFrameSerialNumber() {return frame;}
  public String getPlateNumber() {return plate;}
  public String getMakerName() {return maker;}
  public String getOwnerID() {return owner;}
  public int getPistonDisplacement() {return pistonDisp;}

  // Create a new instance of SimpleCar
SimpleCarDBImpl(String frame, String plate, String maker, String owner, int pistonDisp) {
    this.frame=frame; this.plate=plate; this.maker=maker;
    this.owner=owner; this.pistonDisp=pistonDisp;
  }

  // Create an ArrayList of SimpleCar objects
  public static List SimpleCarList(String db) throws SQLException{
      ArrayList myList= new ArrayList();
      ResultSet rs=DriverManager.getConnection(db).createStatement().executeQuery(
                              "SELECT FrameSerialNumber, PlateNumber, MakerName, OwnerIDCode,
                               PistonDisplacement FROM suv");
      while (rs.next()){
          myList.add(new SimpleCarDBImpl(rs.getString("FrameSerialNumber"),
                                               rs.getString("PlateNumber"), rs.getString("MakerName"),
                                               rs.getString("OwnerIDCode"),rs.getInt("PistonDisplacement")));
                }
  rs.close();
return myList;
}}
```

CarDBImpl

```
import java.sql.*;
public class CarDBImpl implements Car{
// instance variables
// Car data
  Date plateDate;
  String plateNumber, state, frameNumber, maker, engineNumber, ownerID, modelName;
  int pistonDisplacement, year;
  String dbUrl; // database
  // get methods
  public Date getPlateDate() {return plateDate;}
  public String getPlateNumber() {return plateNumber;}
  public String getStateName() {return state;}
  public String getFrameSerialNumber() {return frameNumber;}
  public String getMakerName() {return maker;}
  public String getEngineSerialNumber() {return engineNumber;}
  public String getOwnerID() {return ownerID;}
  public int getPistonDisplacement() {return pistonDisplacement;}
  public int getConstructionYear() {return year;}
  public String getModelName() {return modelName;}
  // Set methods
  public void setPlateDate(Date plateD) {plateDate=plateD;}
  public void setPlateNumber(String plateN) {plateNumber=plateN;}
  public void setStateName(String state) {this.state=state;}
  public void setMakerName(String maker) {this.maker=maker;}
  public void setEngineSerialNumber(String engineN) {engineNumber=engineN;}
  public void setOwnerID(String ownerID) {this.ownerID=ownerID;}
  public void setPistonDisplacement(int pistonD) {pistonDisplacement=pistonD;}
  public void setConstructionYear(int year) {this.year=year;}
  public void setModelName(String modelName) {this.modelName=modelName;}
 …………..
```

CarDBImpl

```
…………….
// Builds a Car object taking the data from the row with the passed frame number;
public CarDBImpl(String frameN, String dbUrl) throws SQLException{
  this.dbUrl=dbUrl;
  String query = "SELECT * FROM suv WHERE FrameSerialNumber='"+frameN+"';";
  ResultSet myRow=DriverManager.getConnection(dbUrl).createStatement().executeQuery(query);
  myRow.next();
  plateNumber=myRow.getString("PlateNumber");
  plateDate=myRow.getDate("PlateDate");
  state=myRow.getString("StateName");
  frameNumber=myRow.getString("FrameSerialNumber");
  maker=myRow.getString("MakerName");
  engineNumber=myRow.getString("EngineSerialNumber");
  pistonDisplacement=myRow.getInt("PistonDisplacement");
  ownerID=myRow.getString("OwnerIDCode");
  modelName=myRow.getString("ModelName");
  year=myRow.getInt("ConstructionYear");
  myRow.close();
}
```

CarDBImpl

```
…………..
// Saves the contained data in the corresponding row
public void save() throws SQLException{
  String sqlquery = "SELECT * FROM suv WHERE FrameSerialNumber='“ + frameNumber + "';";
  ResultSet myRow=DriverManager.getConnection(dbUrl).createStatement(
                                        ResultSet.TYPE_SCROLL_INSENSITIVE,
                                        ResultSet.CONCUR_UPDATABLE)).executeQuery(sqlquery);
  myRow.next();
  myRow.updateString("PlateNumber", plateNumber);
  myRow.updateDate("PlateDate", plateDate);
  myRow.updateString("StateName", state);
  myRow.updateString("FrameSerialNumber", frameNumber);
  myRow.updateString("MakerName", maker);
  myRow.updateString("EngineSerialNumber", engineNumber);
  myRow.updateInt("PistonDisplacement", pistonDisplacement);
  myRow.updateString("OwnerIDCode", ownerID);
  myRow.updateString("ModelName", modelName);
  myRow.updateInt("ConstructionYear", year);
  myRow.updateRow();
  myRow.close();
}
……….
```

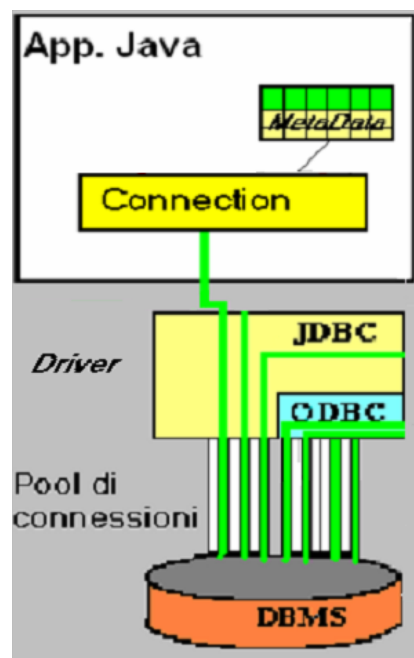
CarDBImpl

```
…………..
// Saves the contained data in the corresponding row
public void save() throws SQLException{

 String sqlquery = “UPDATE suv  PlateNumber='“ + plateNumber
                           + “' PlateDate= '“ + plateDate
                           +  “' StateName='“ + state
                           + “' FrameSerialNumber='“ + frameNumber
                           +  “' MakerName='“ + maker
                           + “' EngineSerialNumber='“ + engineNumber
                           +  “' PistonDisplacement =“ + pistonDisplacement
                           +  “ OwnerIDCode='“+ ownerID
                           +  “' ModelName='“ + modelName
                           + “ ConstructionYear=" year
                           +  “ WHERE FrameSerialNumber='“ + frameNumber + "';";

  int n=DriverManager.getConnection(dbUrl)
                                        .createStatement()
                                        .executeUpdate(sqlquery);
}
……….
```

Ottenere informazioni sul DB e sulle tabelle (i metadati)

RESEARCH CENTRE ON SOFTWARE TECHNOLOGY
Università degli Studi del Sannio

Ottenere informazioni sul DB



RESEARCH CENTRE ON SOFTWARE TECHNOLOGY

Università degli Studi del Sannio

Ottenere informazioni

Usando il metodo Connection.getMetaData possiamo ottenere un oggetto di tipo DatabaseMetaData che ci permette di accedere ai metadati.

Ad esempio con DatabaseMetaData.getTables possiamo ottenere informazioni circa le tabelle presenti nel database

RESEARCH CENTRE ON SOFTWARE TECHNOLOGY

Università degli Studi del Sannio

Ottenere informazioni

```
public ResultSet getTables(String catalog, String schemaPattern,
                           String tableNamePattern, String[] types)
                           throws SQLException
```

Recupera informazioni sulle tabelle, per ogni tabella, ad esempio:

- **TABLE_CAT** String => Catalogo di appartenenza (può essere null)
- **TABLE_NAME** String => Nome della tabella
- **TABLE_TYPE** String => Tipo della tabella (es.: "TABLE", "VIEW", "LOCAL TEMPORARY",...)
- **REMARKS** String => Commenti

Parametri:

catalog – Il nome del catalogo da esplorare *

schemaPattern – Lo schema da esplorare *

tableNamePattern – Pattern dei nomi di tabella (valgono le convenzioni SQL circa "_" e "%")

types – I tipi di tabelle da includere (null significa *tutti*) *

* "" significa informazione assente, null significa non considerare nella selezione

RESEARCH CENTRE ON SOFTWARE TECHNOLOGY

Università degli Studi del Sannio

Ottenere informazioni

I risultati sono restituiti in un oggetto di tipo **ResultSet** che li contiene organizzati per riga e colonna

1	2	3	4	5	6	7	8	9	10
TABLE_CAT	TABLE_SCHEM	TABLE_NAME	TABLE_TYPE	REMARKS	TYPE_CAT	TYPE_SCHEM	TYPE_NAME	SELF_REFERENCING_COL_NAME	REF_GENERATION
					Dati				

NB: Non tutti i DBMS e i driver usano tutti i campi, MySQL usa da 1 a 5

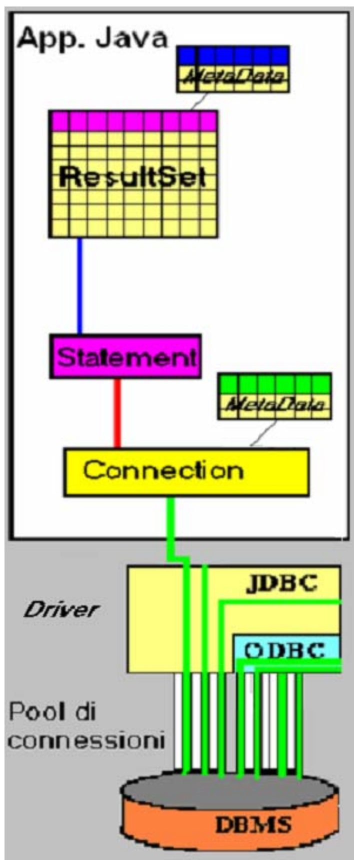
RESEARCH CENTRE ON SOFTWARE TECHNOLOGY

Università degli Studi del Sannio

Ottenere informazioni

```
………….

DatabaseMetaData dbMD = con.getMetaData();
ResultSet rs=dbMD.getTables(null ,null, "%",null);
System.out.println("CATALOGO; TABELLA; TIPO; NOTE");
while (rs.next()){
        System.out.print(rs.getString("TABLE_CAT")+"; ");
        System.out.print(rs.getString("TABLE_NAME")+"; ");
        System.out.print(rs.getString("TABLE_TYPE")+"; ");
        System.out.println(rs.getString("REMARKS"));
}
……….
```

```
CATALOGO; TABELLA; TIPO; NOTE
forum; categoria; TABLE; MySQL table
forum; forum; TABLE; MySQL table
forum; post; TABLE; MySQL table
forum; user; TABLE; MySQL table
```



Esplorazione del ResultSet

RESEARCH CENTRE ON SOFTWARE TECHNOLOGY

Esplorazione del ResultSet

L'oggetto ResultSet dispone di un ricco insieme di metadati (ResultSetMetaData) che ci permette di ottenere informazioni circa il suo contenuto:

Ad esempio possiamo voler conoscere il numero, il nome delle colonne contenute ...

```
............
Statement stm =con.createStatement();
ResultSet rs=stm.executeQuery("SELECT * FROM user");
ResultSetMetaData rsmd= rs.getMetaData();
int colcount = rsmd.getColumnCount();
for (int i=1; i=<colcount; i++)
     System.out.print(rsmd.getColumnName(i)+"; ");
System.out.println();
.............
```

Esplorazione del ResultSet

... ed esplorare le singole righe ed estrarre i dati in accordo con il loro tipo (a seconda dei tipi definiti in java.sql.Types)

```
while (rs.next()){
    for (int i=1; i=<colcount; i++){
        switch (rsmd.getColumnType(i)){
            case Types.BIGINT: case Types.INTEGER:
            case Types.SMALLINT: case  Types.TINYINT:
                System.out.print(rs.getInt(i)); break;
            case Types.DECIMAL: case Types.DOUBLE:
            case Types.FLOAT: case Types.NUMERIC:
            case Types.REAL:
                System.out.print(rs.getDouble(i)); break;
            case Types.DATE:
                System.out.print(rs.getDate(i)); break;
            case Types.TIMESTAMP:
                System.out.print(rs.getTimestamp(i)); break;
            case Types.TIME:
                System.out.print(rs.getTime(i)); break;
            case Types.BOOLEAN:
                System.out.print(rs.getBoolean(i)); break;
                case Types.CHAR: case Types.VARCHAR:
            case Types.LONGVARCHAR:
                System.out.print(rs.getString(i)); break;
                default:;
    }System.out.print("; ");  } System.out.println();    }
```

RESEARCH CENTRE ON SOFTWARE TECHNOLOGY
Università degli Studi del Sannio

INSERT su tabelle con campi generati dal DB

RESEARCH CENTRE ON SOFTWARE TECHNOLOGY
Università degli Studi del Sannio

RESEARCH CENTRE ON SOFTWARE TECHNOLOGY
Università degli Studi del Sannio

Esecuzione di Insert e campi generati dal db

Alcuni campi sono generati automaticamente dal DBMS quando vengono inserite tuple (contatori, identificatori, ...).

L'accesso a questi campi può avvenire in due passi:

1. Notificare al driver che deve rendere disponibili i valori generati automaticamente (al momento dell'escuzione dello statement con i metodi executeUpdate e execute). Le versioni overloaded di questi metodi accettano un secondo parametro che può essere:
   1. Una costante (Statement.RETURN_GENERATED_KEYS o Statement.NO_GENERATED_KEYS) che indica se rendere disponibili i valori
   2. Un array di interi: le posizioni delle colonne da rendere disponibili
   3. Un array di stringhe con i nomi delle colonne da rendere disponibili
2. Dopo l'esecuzione può essere invocato il metodo getGeneratedKeys che restituisce un ResultSet con i valori generati.

NB I PreparedStatement ricevono l'indicazione sui campi generati automaticamente al momento della creazione.

RESEARCH CENTRE ON SOFTWARE TECHNOLOGY
Università degli Studi del Sannio

Esecuzione di Insert e campi generati dal db

```
String sql = "INSERT INTO AUTHORS (LAST, FIRST, HOME) VALUES " +
        "'PARKER', 'DOROTHY', 'USA'";
int rows = stmt.executeUpdate(sql, Statement.RETURN_GENERATED_KEYS);
ResultSet rs = stmt.getGeneratedKeys();
if (rs.next()) {
        ResultSetMetaData rsmd = rs.getMetaData();
        int colCount = rsmd.getColumnCount();
        do {
            for (int i = 1; i <= colCount; i++) {
                    String key = rs.getString(i);
                    System.out.println("key " + i + "is " + key);
            }
        } while (rs.next());
} else { System.out.println("There are no generated keys."); }
```

NB Non pienamente supportato da MySQL