



DIE UNIVERSITÀ DEGLI STUDI DI
TI. NA POLI FEDERICO II
DIPARTIMENTO DI INGEGNERIA ELETTRICA
E DELLE TECNOLOGIE DELL'INFORMAZIONE

A.A. 2021/2022

BASI DI DATI I

- Java DataBase Connectivity

Prof. Adriano Peron
Prof. Silvio Barra

COS'È JDBC ?

- E' un insieme di classi ed interfacce che forniscono un'API standard per operare con Basi di Dati Relazionali in pure JAVA .
- JDBC è un trademark SUN (dal 2010 Oracle)
 - Acronimo di Java Database Connectivity?
- Definito nel 1996 (JDBC 1.0),
 - Molto migliorato nel 1998 per Java 2 (JDBC 2.0),
 - Ottimizzato nel 2003 (JDBC 3.0).
- E' una base per altri package di più alto livello



ARCHITETTURE CLIENT-SERVER



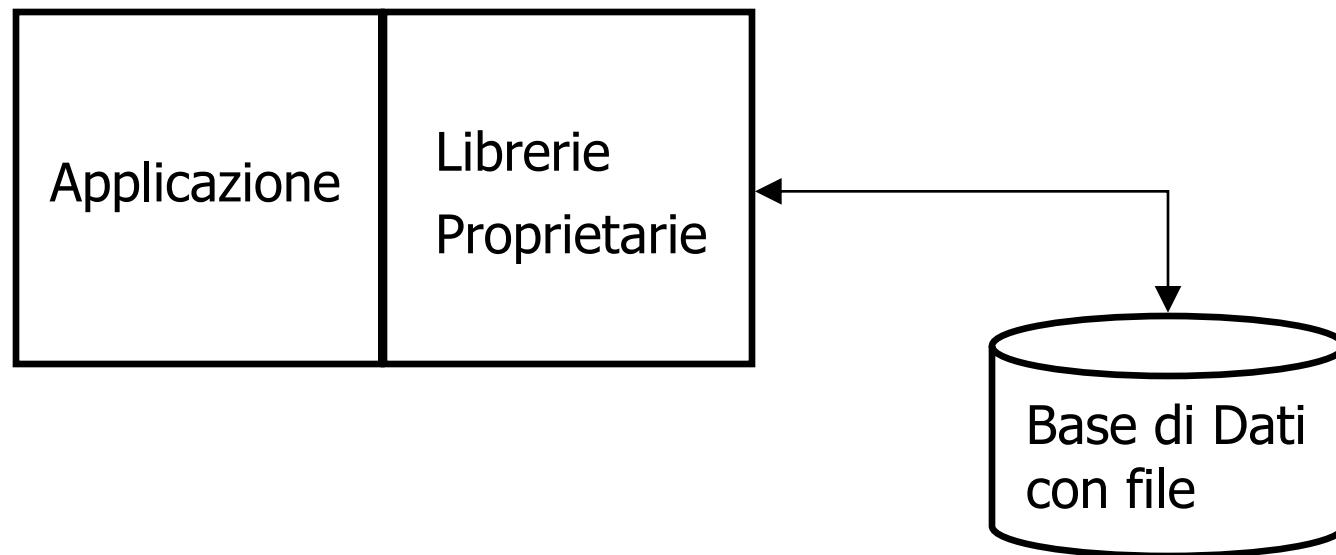
ACCESSO DIRETTO

- **Vantaggi**

- **Efficiente per soluzioni piccolissime**

- **Svantaggi**

- Sviluppo molto lento
- Non flessibile/scalabile
- Non condivisibile
- Distoglie dalla ‘business logic’



UTILIZZO DI DBMS

- Un DataBase Management System è un'applicazione general-purpose, la cui unica funzione è manipolare basi di dati
- Fornisce un modo per memorizzare informazioni in strutture dati efficienti, scalabili e flessibili,
- Permette ricerche da più ‘direzioni’

UTILIZZO DI DBMS (2)

- Permette la definizione di architetture client-server (o 2-tiered)
 - Il client conosce come gestire i dati
 - Il DBMS gestisce lo storage ed il retrieval dei dati
 - Lo sviluppatore scrive solo la “business logic”
- Esempi: Access, Oracle, PostgreSQL, SQL Server, Informix, ecc...

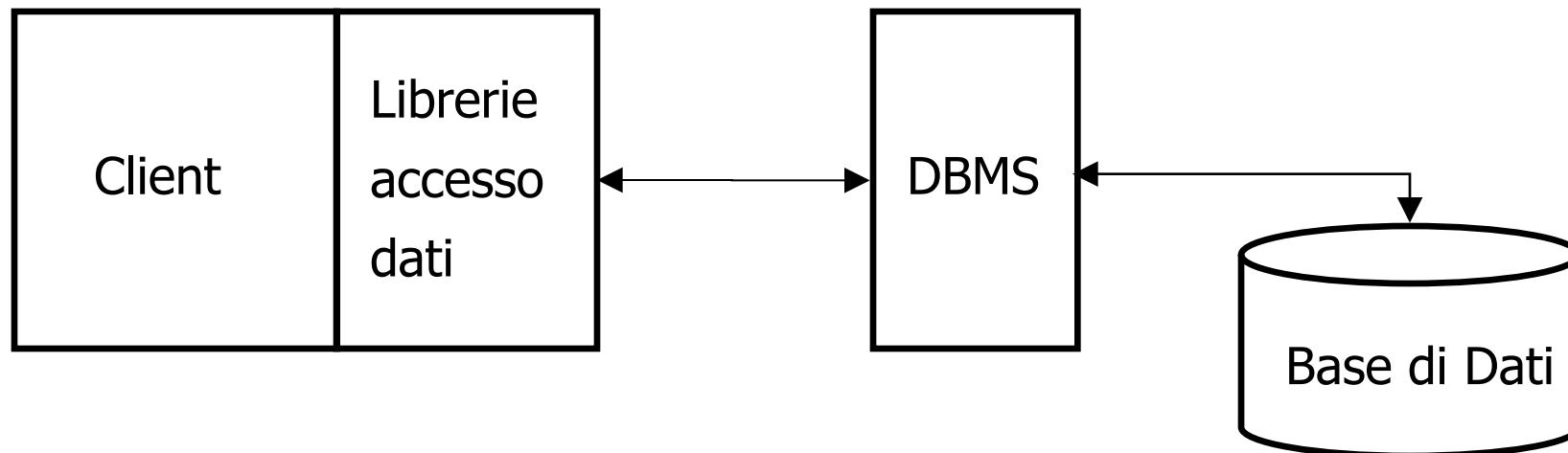
ARCHITETTURA 2-TIER

- Vantaggi

- Molto più efficiente
- Sviluppo rapido
- Scalabile/flessibile

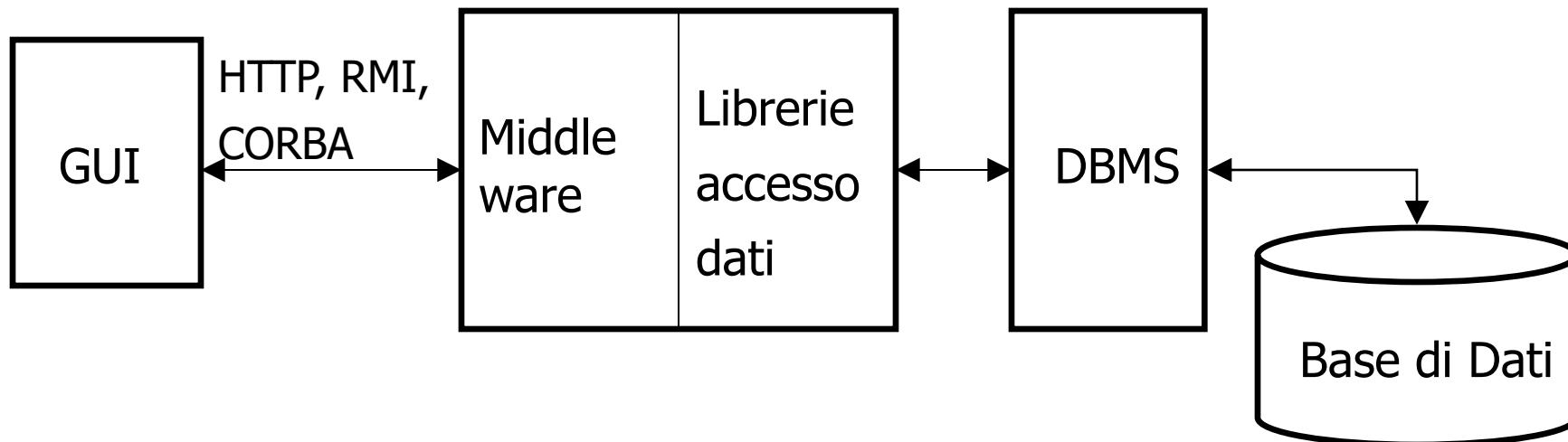
- Svantaggi

- Sicurezza
- # licenze per DBMS
- GUI legata a business logic
- Non utilizzabile sul Web



ARCHITETTURA 3-TIER

- Vantaggi
 - Policy sicurezza
 - GUI indipendente dai dati
 - Orientata al Web
- Svantaggi
 - Maggiore complessità



COME INTERAGIRE CON SQL

- Usare SQL interattivamente...

The screenshot shows a SQL interactive environment with the following components:

- Toolbar:** Includes icons for various database operations like Open, Save, Print, and Help.
- Menu Bar:** Contains "Centro comandi", "Interattivo", "Editare", "Strumenti", and "Aiuto".
- Tool Buttons:** A row of small buttons corresponding to the toolbar icons.
- Tab Bar:** Shows tabs for "Interattivo", "Script", "Risultati dell'interrogazione", and "Plan di accesso". The "Interattivo" tab is selected.
- Connection Information:** Displays "Collegamento database" and "CSITE61 - DB2 - SAMPLE".
- Command History:** Shows the history of commands entered.
- Current Command:** Displays the SQL query: `SELECT LastName, Salary, Job, Birthdate FROM Employee WHERE Salary > 30000 ORDER BY S...`.
- SQL Editor:** Shows the full query: `SELECT LastName, Salary, Job, Birthdate
FROM Employee
WHERE Salary > 30000
ORDER BY Salary DESC`. It includes buttons for "SQL Assist" and "Accodare allo script".
- Result Area:** Shows the results of the query in a grid format:

LASTNAME	SALARY	JOB	BIRTHDATE
HAAS	52750.00	PRES	1933-08-24
LUCCHESI	46500.00	SALESREP	1929-11-05
THOMPSON	41250.00	MANAGER	1948-02-02
GEYER	40175.00	MANAGER	1925-09-15
KWAN	38250.00	MANAGER	1941-05-11
PULASKI	36170.00	MANAGER	1953-05-26
STERN	32250.00	MANAGER	1945-07-07

Non è adatto ai nostri scopi!

COME INTERAGIRE CON SQL (2)

- Inserire SQL nel codice...

```
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();
```

Come gestire la connessione con il DBMS?

COME INTERAGIRE COL DBMS

- **Embedded SQL**
 - Il programma colloquia direttamente col DBMS
 - Gli statement SQL sono compilati utilizzando un precompilatore specifico del DBMS.
 - SQL diviene parte integrante del codice.
- **Call Level Interface**
 - Il programma accede al DBMS per mezzo di un'interfaccia standard
 - Gli statement SQL non sono compilati, ma inviati al DBMS a run-time.
 - Il DBMS può essere sostituito
 - La stessa interfaccia può interagire con più DBMS

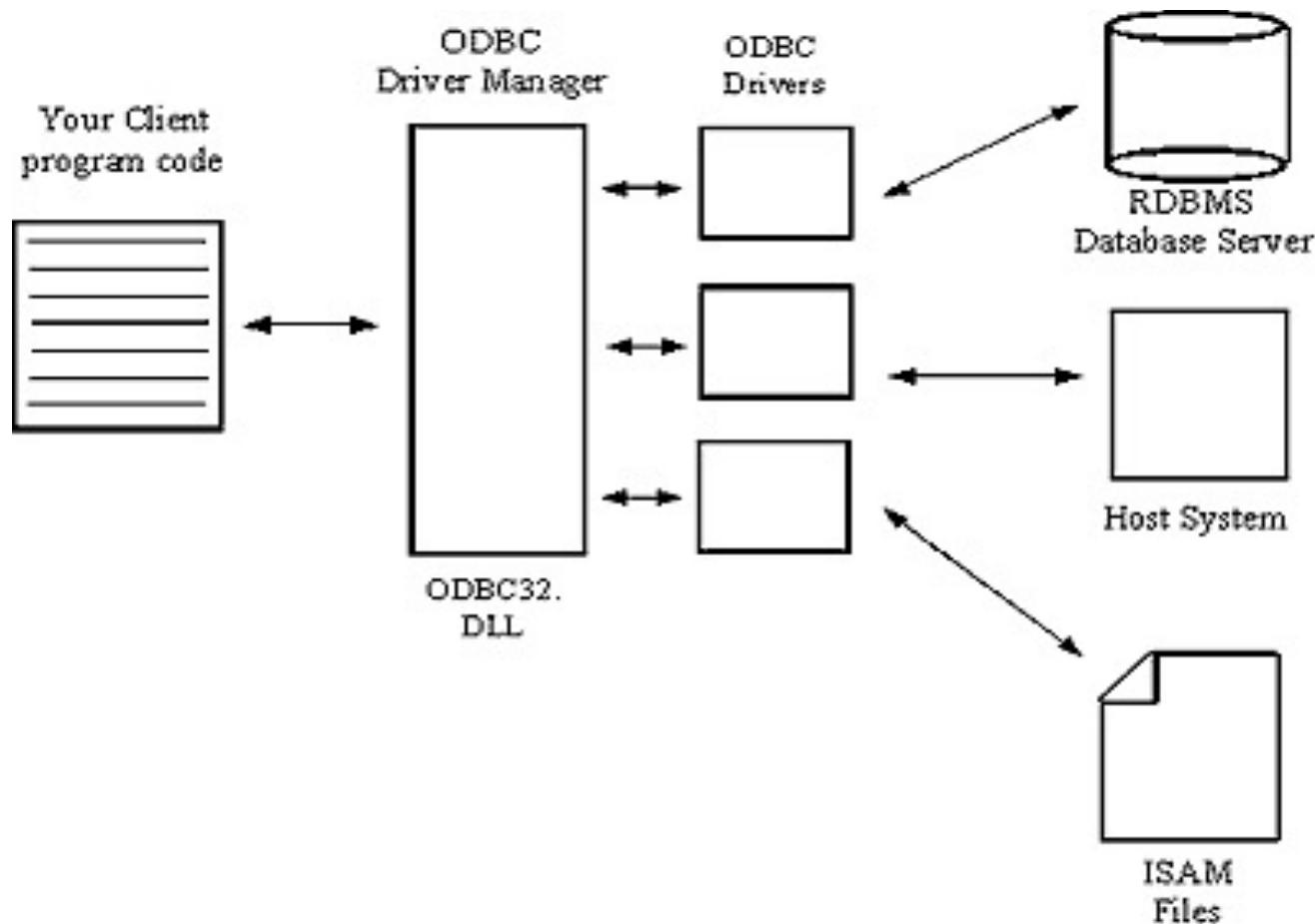
ESEMPI DI CLI

- **ODBC**
 - Open Database Connectivity
- **OLE-DB**
 - Object Linking and Embedding for Databases
- **ADO**
 - Active Data Objects
- **UDA**
 - Universal Data Access
- **JDBC**

ODBC

- API standard definita da Microsoft nel 1992
- La prima implementazione di un CLI
- Permette l'accesso a dati residenti in DBMS diversi (Access, MySQL, DB2, Oracle, ...)
 - Supportato da praticamente tutti i DBMS in commercio
- Gestisce richieste SQL convertendole in un formato comprensibile al particolare DBMS
 - Permette agli sviluppatori di formulare richieste SQL a DB distinti senza dover conoscere le interfacce di programmazione proprietarie di ogni singolo DB

ARCHITETTURA ODBC



◆ Esempio di sorgente dati in Windows

ODBC (SVANTAGGI)

- Utilizza interfacce C
- E' procedurale (NO O-O)
- Ha poche funzioni con molti parametri
- E' abbastanza ostico

JDBC

MOTIVAZIONI DI JDBC

- I Db sono il ‘core system’ di gran parte dei sistemi client-server
- 
- Quasi la totalità delle entità necessita di accedere ad un DB
 - Si vuole che tale accesso sia :
 1. Cross-plataform, cioè scritto in JAVA
 2. Capace di interagire con qualsiasi RDBMS

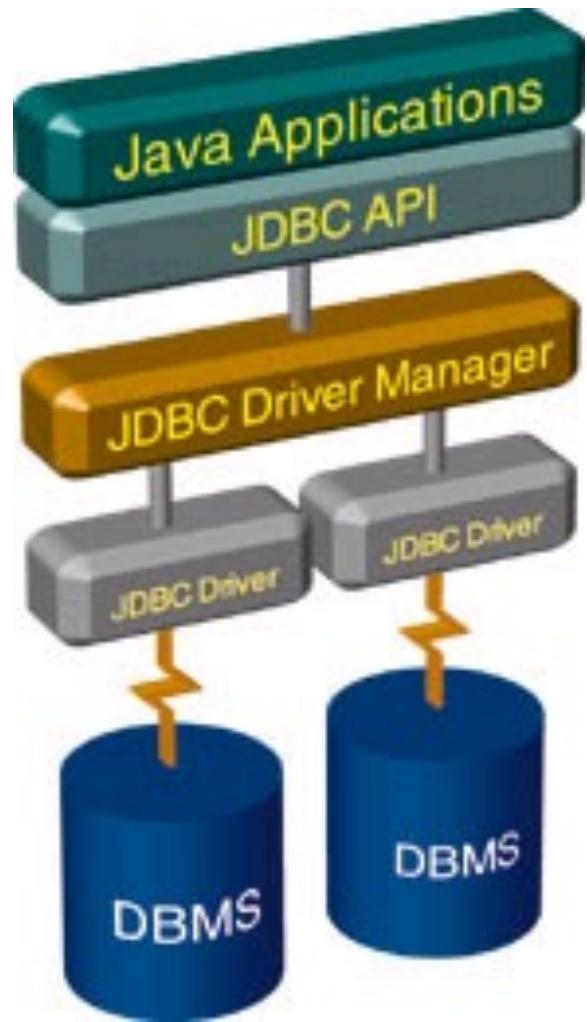
COSA FA JDBC

- JDBC permette ad oggetti JAVA di dialogare con database relazionali.
- Effettua tre operazioni fondamentali
 - Stabilisce una connessione con il database
 - Invia statements SQL
 - Processa i risultati

PUNTI CHIAVE DI JDBC

- Non necessita di Driver preinstallati (a differenza di ODBC)
- Utilizza la sintassi degli URL per la localizzazione di risorse
- Una classe speciale (il DriverManager) è responsabile di attivare il driver giusto per la connessione ad un RDBMS
- Mappa i tipi SQL in tipi JAVA

ARCHITETTURA DI JDBC



JDBC DRIVER

DRIVER TIPO I

Wrapper da JDBC ad ODBC

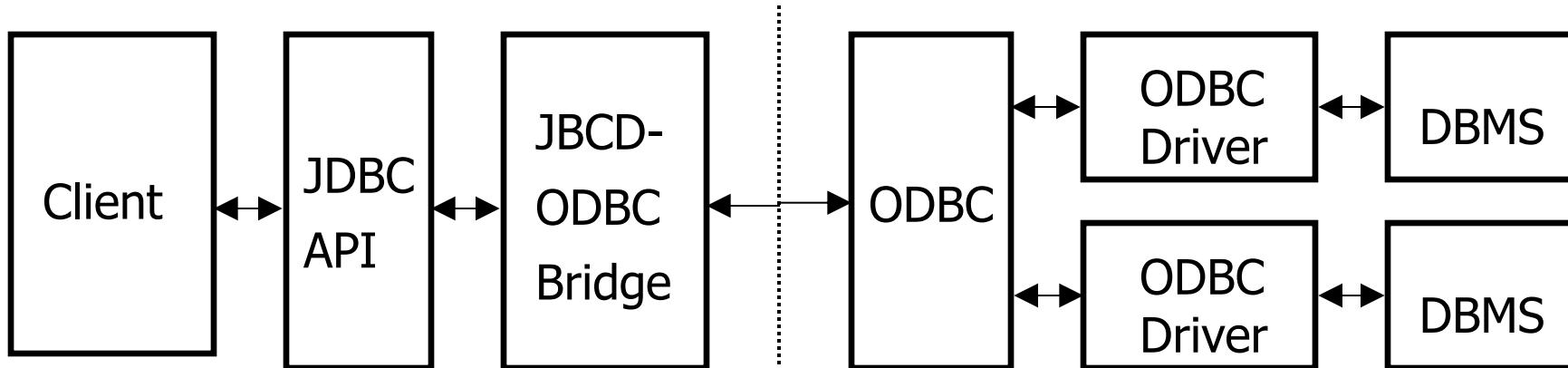
Traduce le JDBC calls in ODBC calls

◆ Vantaggi

- E' incluso nel JDK (gratis)
- Funziona con qualsiasi DBMS ODBC
- Facile da configurare

◆ Svantaggi

- Molto lento
- Utilizzabile solo per sviluppo codice



DRIVER TIPO II

Wrapper da JDBC a driver nativo DBMS

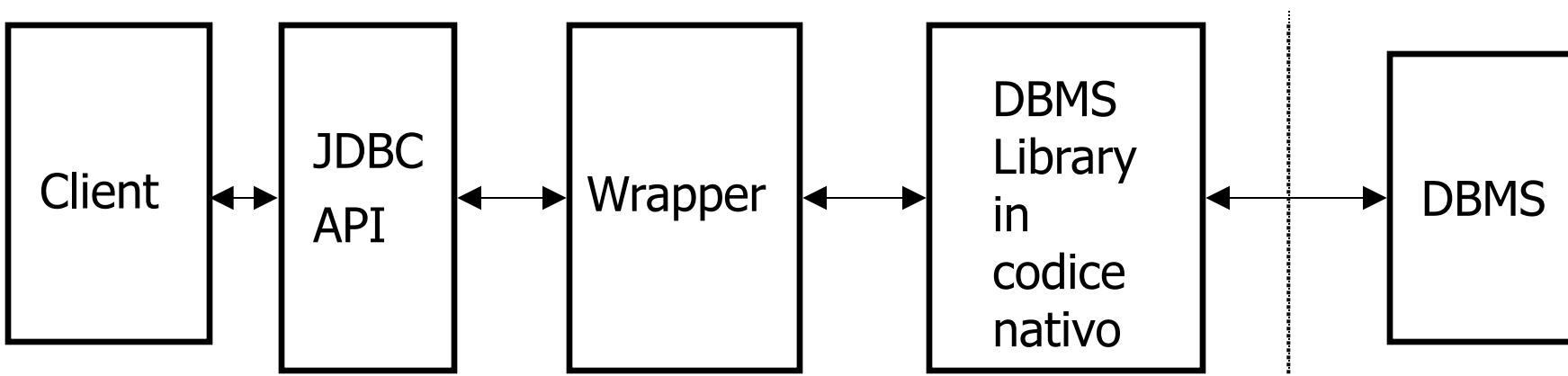
Traduce le JDBC calls in chiamate JNI al codice C/C++ del DBMS
Driver

◆ Vantaggi

- E' di facile realizzazione

◆ Svantaggi

- Un bug nel driver può portare a crash del sistema
- Difficile da configurare
- N.a. al Web



DRIVER TIPO III

Driver Pure Java per DB MiddleWare

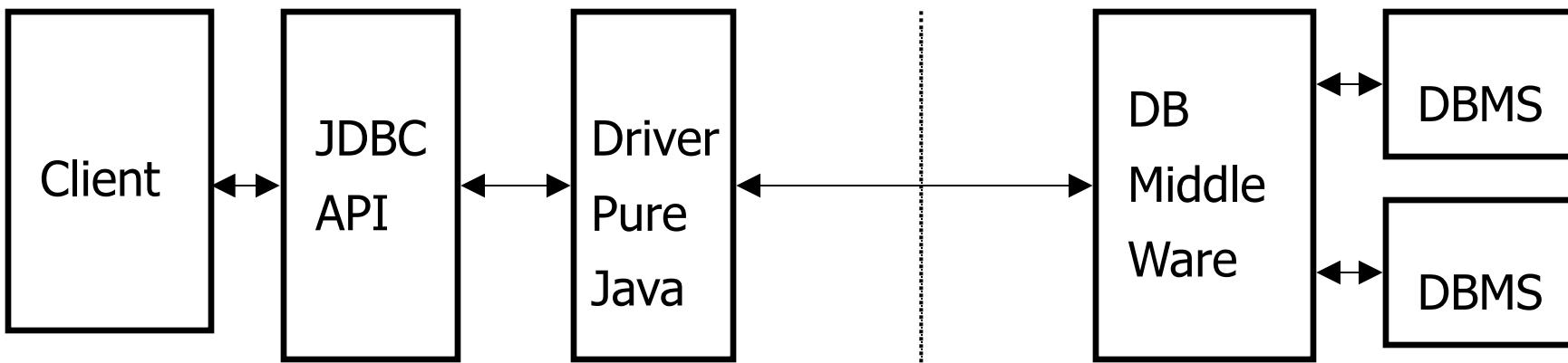
Traduce le JDBC calls in un net-protocol indipendente dal DBMS, convertito poi nel DBMS-protocol dal server

◆ Vantaggi

- Pure Java
- 1 driver client per N DBMS lato server

◆ Svantaggi

- Difficile da configurare
- I DBMS vendors non realizzano questo tipo di driver



DRIVER TIPO IV

Driver Pure Java per DBMS

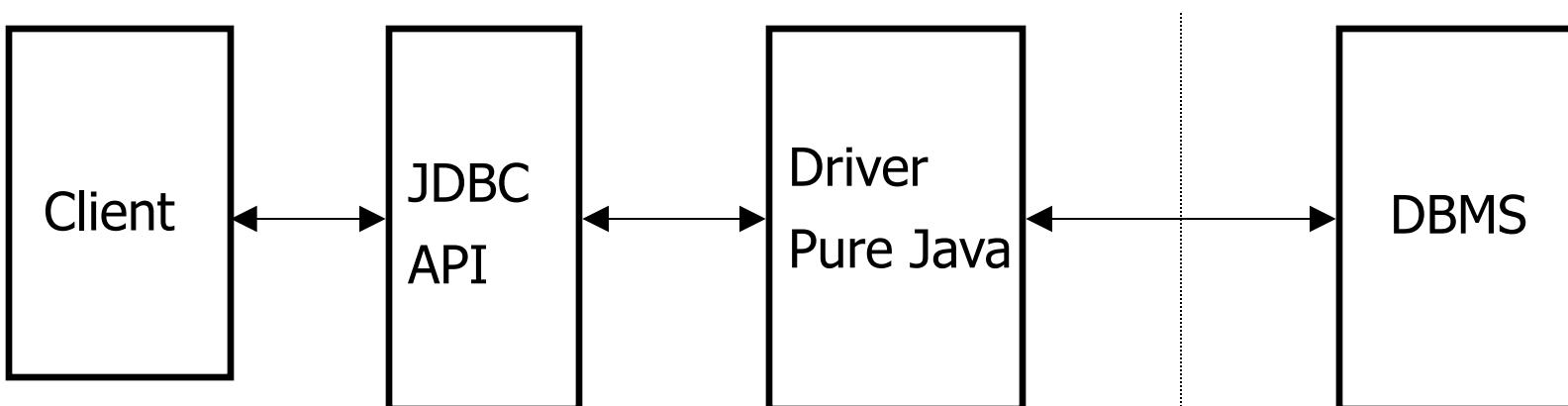
Traduce le JDBC calls in un net-protocol comprensibile dal DBMS

◆ Vantaggi

- Massima velocità
- Facili da configurare
- Pure Java

◆ Svantaggi

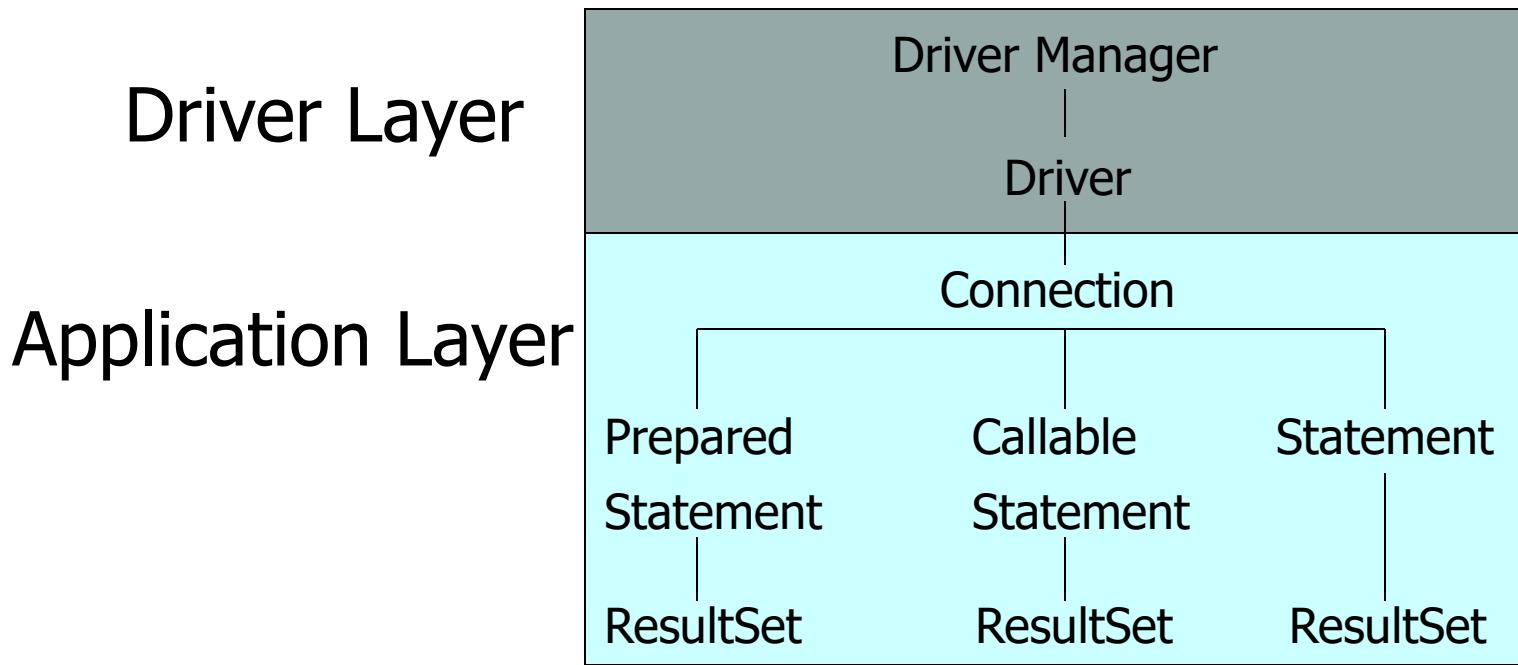
- Non supportato da tutti i DBMS
- Richiede 1 Driver per ogni DBMS, lato client



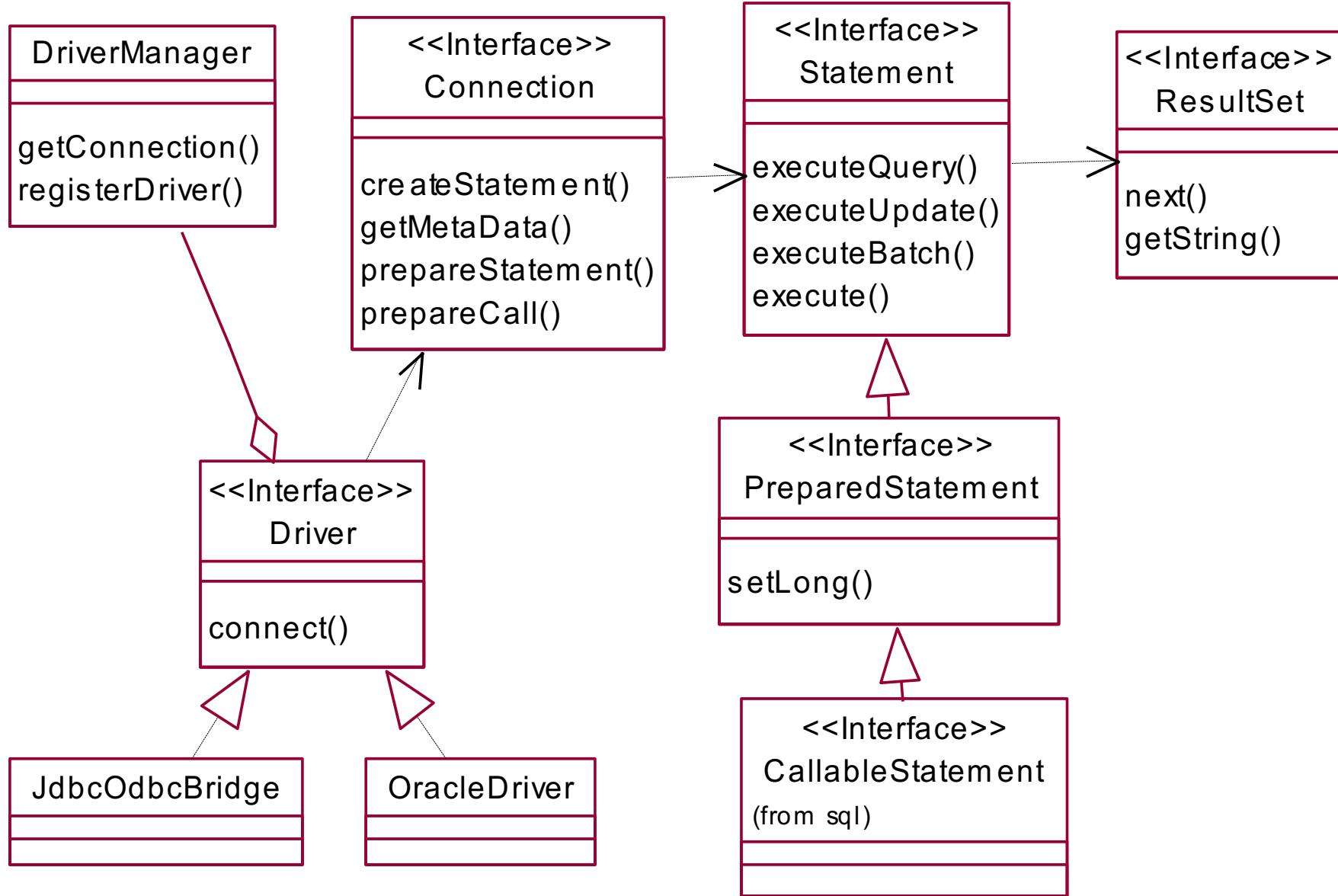
JDBC API

STRUTTURA DI JDBC

- JDBC è costituito da 2 'layer' principali:
 - JDBC driver (verso il DBMS)
 - JDBC API (verso l'applicativo)



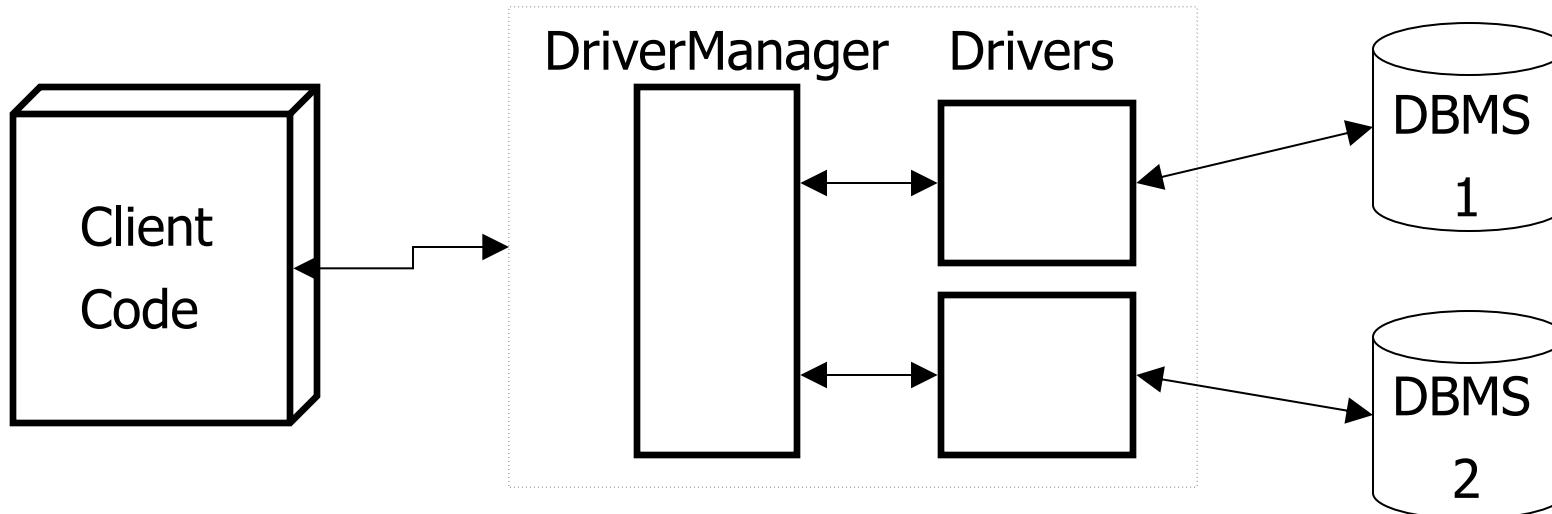
CLASS DIAGRAM IN UML



PUNTUALIZZAZIONE

- Tutte le classi fondamentali di JDBC sono interfacce
 - Non sono implementate in JDBC
 - Sono implementate dallo specifico driver

Esempio di Interface-Implementation pattern nell'O-O design.



JDBC API

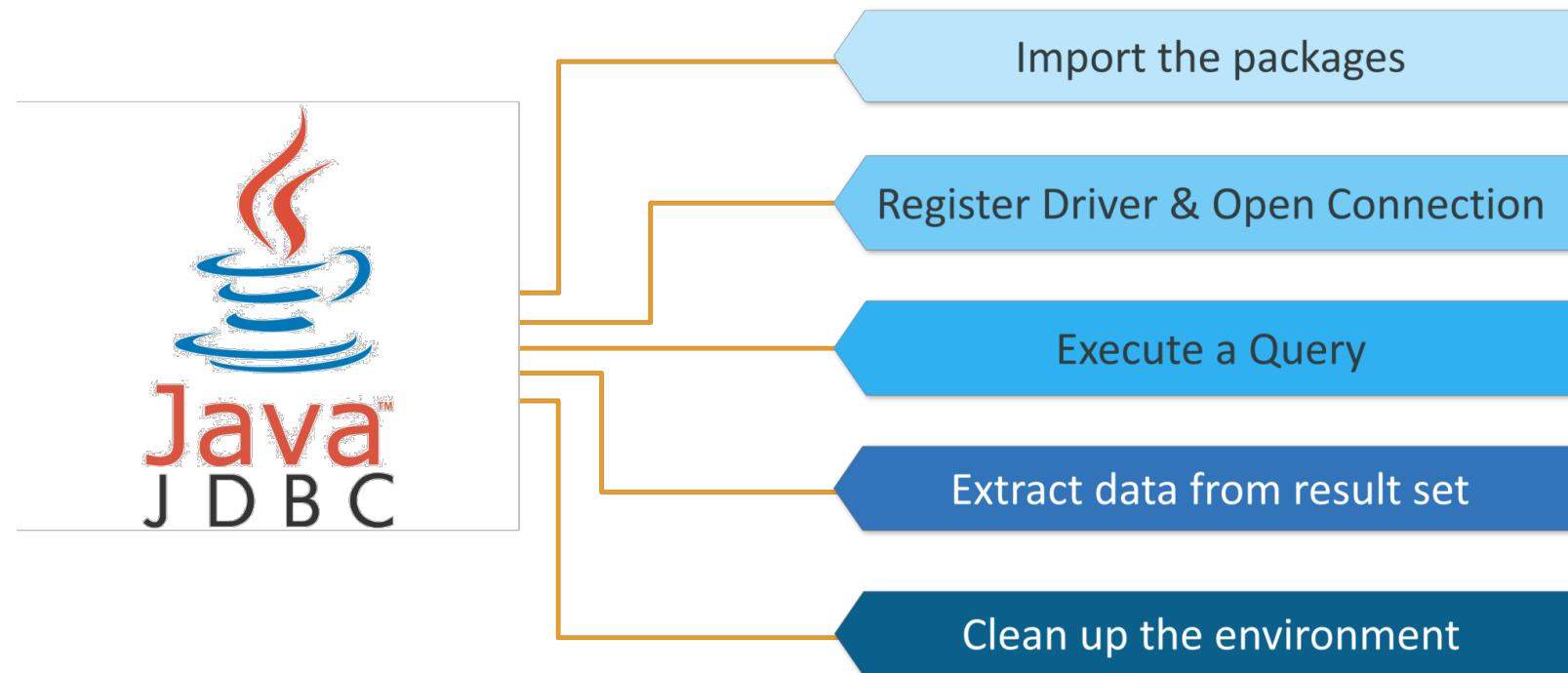
- Data Types
- API di base
 - 1. Caricare un driver
 - 2. Ottenere una connessione
 - 3. Eseguire uno Statement
 - 4. Gestire i risultati
 - 5. Rilasciare le risorse
- API avanzate
 - Prepared e Callable Statements
 - Eccezioni e Warning
 - Transazioni e Isolamento

DATA TYPES

JDBC definisce un insieme di tipi SQL nella classe `java.sql.Types`

JDBC Types Mapped to Java Types	
JDBC Type	Java Type
CHAR	String
VARCHAR	String
LONGVARCHAR	String
NUMERIC	<code>java.math.BigDecimal</code>
DECIMAL	<code>java.math.BigDecimal</code>
BIT	<code>boolean</code>
TINYINT	<code>byte</code>
SMALLINT	<code>short</code>
INTEGER	<code>int</code>
BIGINT	<code>long</code>
REAL	<code>float</code>
FLOAT	<code>double</code>
DOUBLE	<code>double</code>
BINARY	<code>byte[]</code>
VARBINARY	<code>byte[]</code>
LONGVARBINARY	<code>byte[]</code>
DATE	<code>java.sql.Date</code>
TIME	<code>java.sql.Time</code>
TIMESTAMP	<code>java.sql.Timestamp</code>

BASIC JDBC



CONNESSIONE.JAVA (POSTGRES)

```
import java.sql.*;

class Connessione {
    public static void main(String args[]) throws Exception {
        try {
            Class.forName("org.postgres.Driver");

            String url = "jdbc:postgresql://localhost:5432/nometabella";
            Connection conn = DriverManager.getConnection(url, "nomeutente", "password");
            System.out.println("Connessione OK \n");
            conn.close();
        }
        catch (ClassNotFoundException e) {
            System.out.println("DB driver not found \n");
            System.out.println(e);
        }
        catch(SqlException e) {
            System.out.println("Connessione Fallita \n");
            System.out.println(e);
        }
    }
}
```



CONNESSIONE.JAVA (MYSQL)

```
import java.sql.*;  
  
class Connessione {  
    public static void main(String args[]) throws Exception {  
        try {  
            Class.forName("com.mysql.cj.jdbc.Driver");  
  
            String url = "jdbc:mysql://localhost:3306/nometabella";  
            Connection con = DriverManager.getConnection(url, »nomeutente», »password»);  
            System.out.println("Connessione OK \n");  
            con.close();  
        }  
        catch (ClassNotFoundException e) {  
            System.out.println("DB driver not found \n");  
            System.out.println(e);  
        }  
        catch(SqlException e) {  
            System.out.println("Connessione Fallita \n");  
            System.out.println(e);  
        }  
    }  
}
```



PREPARAZIONE

- Importare le classi:

```
import java.sql.*;
```

- Usare i blocchi **try ... catch**:

- Il primo blocco **try** contiene il metodo `Class.forName`, dal package `java.Lang`. Questo metodo lancia un `ClassNotFoundException`, in maniera tale da permettere al blocco `catch` di gestire subito l'eccezione.
 - Può essere gestita una sola volta nel costruttore.
- Il secondo blocco **try** contiene i metodi JDBC, che lanciano tutti un'eccezione del tipo `SQLException`, così il blocco `catch` può gestire solamente oggetti di quel tipo.



STEP 1 – CARICARE IL DRIVER

```
import java.sql.*;  
.  
.  
try  
{  
    Class.forName(NOME_DRIVER) ;  
}  
catch(ClassNotFoundException)  
{// Driver non trovato !  
}
```

CARICAMENTO DEI DRIVER

```
// per un database MySQL
String driver = "com.mysql.cj.jdbc.Driver";

// se il database è Oracle
... driver = "oracle.jdbc.OracleDriver";

// se il driver è JDBC-ODBC (tipo 1)
... driver = "sun.jdbc.odbc.JdbcOdbcDriver";

// se il database è PostgreSQL
... driver = »org.postgresql.Driver»;

// il metodo forName forza il caricamento del driver
Class.forName(driver); // lancia una ClassNotFoundException
```



IL DRIVER MANAGER

- Il *DriverManager* mantiene una lista di classi *Driver* che registrano se stessi invocando il metodo *registerDriver()*.

CARICARE IL DRIVER

- L'inizializzatore statico del driver, chiamato dalla JVM al caricamento della classe, si registra nel Driver Manager

```
public class MyDriver
{
    static
    {
        new MyDriver();
    }
    public MyDriver()
    {
        java.sql.DriverManager.register(this);
    }
}
```

STEP 2 - OTTENERE UNA CONNESSIONE

- L'oggetto Connection rappresenta un canale di comunicazione con un DB.
- Una sessione di connessione include istruzioni SQL che vengono eseguite e processate ritornando i valori attraverso la connessione.

OTTENERE UNA CONNESSIONE

```
Connection con =  
    DriverManager.getConnection(  
        URL_MY_DATABASE);
```

- E' richiesta una connessione al Driver Manager, senza specificare quale particolare Driver debba istanziarla
 - E' l'unico modo per ottenere una completa indipendenza dal DBMS!

GLI URL IN JDBC

- E' una stringa formata da nodi, separati da ':' o '/'
- Formato:
 - protocollo:sottoprotocollo:databasename
- Il parametro "Databasename" non ha una particolare sintassi, ma è interpretato dal driver
- Esempi
 - jdbc:odbc:myDB
 - jdbc:oracle:@mywebsite:1521:myDB
 - jdbc:cloudscape:myDB
 - jdbc:cloudscape:rmi:myDB;create=true

QUALE DRIVER CREA LA CONNESSIONE?

- L'URL specifica un sottoprotocollo ed il data source-database system
 - Es. jdbc:odbc:MyDataSource
- Il Driver Manager trova il driver appropriato chiamando il metodo acceptURL (URL) di ogni driver caricato:
- Il primo driver che risponde true, stabilisce una connessione.
- E' possibile in questo modo scegliere il driver JDBC appropriato a run-time!

L'INTERFACCIA DRIVER

<<Interface>>

Driver

```
connect(url : String, info : java.util.Properties) : Connection  
acceptsURL(url : String) : boolean  
get PropertyInfo(url : String, info : java.util.Properties) : DriverPropertyInfo[]  
getMajorVersion() : int  
getMinorVersion() : int  
jdbcCompliant() : boolean
```

LA CLASSE DRIVERMANAGER

DriverManager

```
getConnection(url : String, info : java.util.Properties) : Connection  
getConnection(url : String, user : String, password : String) : Connection  
getConnection(url : String) : Connection  
getDriver(url : String) : Driver  
registerDriver(driver : java.sql.Driver) : void  
getDrivers() : java.util.Enumeration
```

L'INTERFACCIA CONNECTION

<<Interface>>

Connection

createStatement() : Statement

getMetaData() : DatabaseMetaData

prepareStatement(sql : String) : PreparedStatement

prepareCall(sql : String) : CallableStatement

STEP 3 - ESEGUIRE UNO STATEMENT

- Lo Statement è l'oggetto che ‘trasporta’ le istruzioni SQL sulla connection, verso il DB.
- È unico per tutta la durata della connessione.
 - Per fare una nuova query, semplicemente si cambia la stringa SQL al suo interno

ESEGUIRE UNO STATEMENT

```
try
{
    Statement st = con.createStatement();
    ResultSet rs = st.executeQuery("SELECT nome FROM studenti");
}
catch (SQLException sqe)
{
    // Problema
}
```

- Lo scopo principale della classe Statement è eseguire istruzioni SQL.

ESEGUIRE UNO STATEMENT

- Si utilizza `executeQuery()` per gli statement che restituiscono tuple
 - Comandi DQL
 - Restituisce un `ResultSet`
- Si utilizza `executeUpdate()` per gli statement che non restituiscono tuple
 - Comandi DDL/DML
 - Restituisce il numero di righe modificate
- JDBC 2.0 ha introdotto `executeBatch()` per eseguire più statement in sequenza (per motivi di efficienza)

ESEGUIRE UNO STATEMENT (2)

- Se non si conosce il tipo di query (es. L'utente la inserisce a run-time), si usa la execute()
 - Restituisce true se è disponibile un ResultSet
 - Si deve chiamare la getResult() per recuperare le informazioni
- Ad ogni Statement è associato un unico ResultSet

L'INTERFACCIA STATEMENT

<<Interface>>

Statement

```
executeQuery(sql : String) : ResultSet  
executeUpdate(sql : String) : int  
executeBatch()  
execute(sql : String) : boolean  
getWarnings() : SQLWarning  
getResultSet() : ResultSet  
getUpdateCount() : int  
getMoreResults() : boolean
```

PREPARED STATEMENT

- E' un SQL Statement precompilato (il DBMS la salva nella propria cache)
- Utilizzabile con query molto simili nella struttura, ma che cambiano spesso parametri
- Migliora le prestazioni se la query è eseguita molte volte

```
PreparedStatement updateEsami =  
    con.prepareStatement("UPDATE Studenti SET esami  
    = ? WHERE Matricola LIKE ?");  
updateEsami.setInt(1, 13);  
updateEsami.setString(2, "011/245389");  
updateEsami.executeUpdate();
```

CALLABLE STATEMENT

- E' la classe JDBC per supportare le stored procedures
 - Utilizzate per encapsulare un insieme di operazioni o query da eseguire sul database server
- L'utilizzo è richiesto solo per le stored procedure che restituiscono dei valori.

STEP 4 – GESTIRE I RISULTATI

- In genere l'esecuzione di una query porta alla restituzione di dati da parte del DBMS
- Un ResultSet contiene tutte le tuple che soddisfano la condizione nell'istruzione SQL inviata usando lo Statement
- I dati sono gestiti in un'apposita struttura dati
- Fornisce dei metodi per accedere ai dati che contiene.

GESTIRE I RISULTATI

```
while(rs.next())  
{  
    System.out.println("Nome: " +  
rs.getString("nome"));  
  
    System.out.println("Esami: " + rs.getInt("esami"));  
}
```

- I risultati di una query sono salvati in un oggetto RecordSet

RESULTSET

- Ha un ‘cursore’ che punta al record corrente
- Il cursore del ResulSet è posizionato prima della riga iniziale, dopo l’esecuzione di un metodo executeXXX()
- Dispone di molti metodi di navigazione (metodi per modificare la posizione del cursore)

RESULTSET (2)

- Permette di recuperare i valori tramite i metodi `getXXXX(colonna)`
- La colonna può essere specificata o con il nome o con il numero:
 - Il nome delle colonne è case-insensitive
 - La numerazione delle colonne parte da 1

```
rs.getString("nome");
```

```
rs.getString(1);
```

NAVIGAZIONE NEL RESULTSET

- Operazioni disponibili sul ResultSet
 - `first()`, `last()`, `next()`
 - `previous()`, `beforeFirst()`,
`afterLast()`
 - `absolute(int)`, `relative(int)`
 - JDBC 1.0 permetteva una navigazione solo in avanti

DATI NEL RESULTSET

- Il driver JDBC converte il tipo di dato del DB in quello richiesto con il metodo `getXXX()`.
 - Ad esempio se il tipo di dato nel DB è di tipo VARCHAR e la richiesta è di tipo String, JDBC effettua la conversione.
- Per determinare se un valore restituito è di tipo JDBC NULL, bisogna prima leggere la colonna e poi usare il metodo `ResultSet.wasNull` per scoprire se il valore ritornato è NULL.
 - Il metodo restituisce un valore che indica “null” a seconda del tipo di dato analizzato. Abbiamo :
 - Il valore null di Java per `getString`, `getDate`, `getTime` ed altri;
 - Il valore 0 (zero) per `getByte`, `getInt` ed altri numerici;
 - Il valore false per il tipo booleano.

MODIFICHE AL RESULTSET

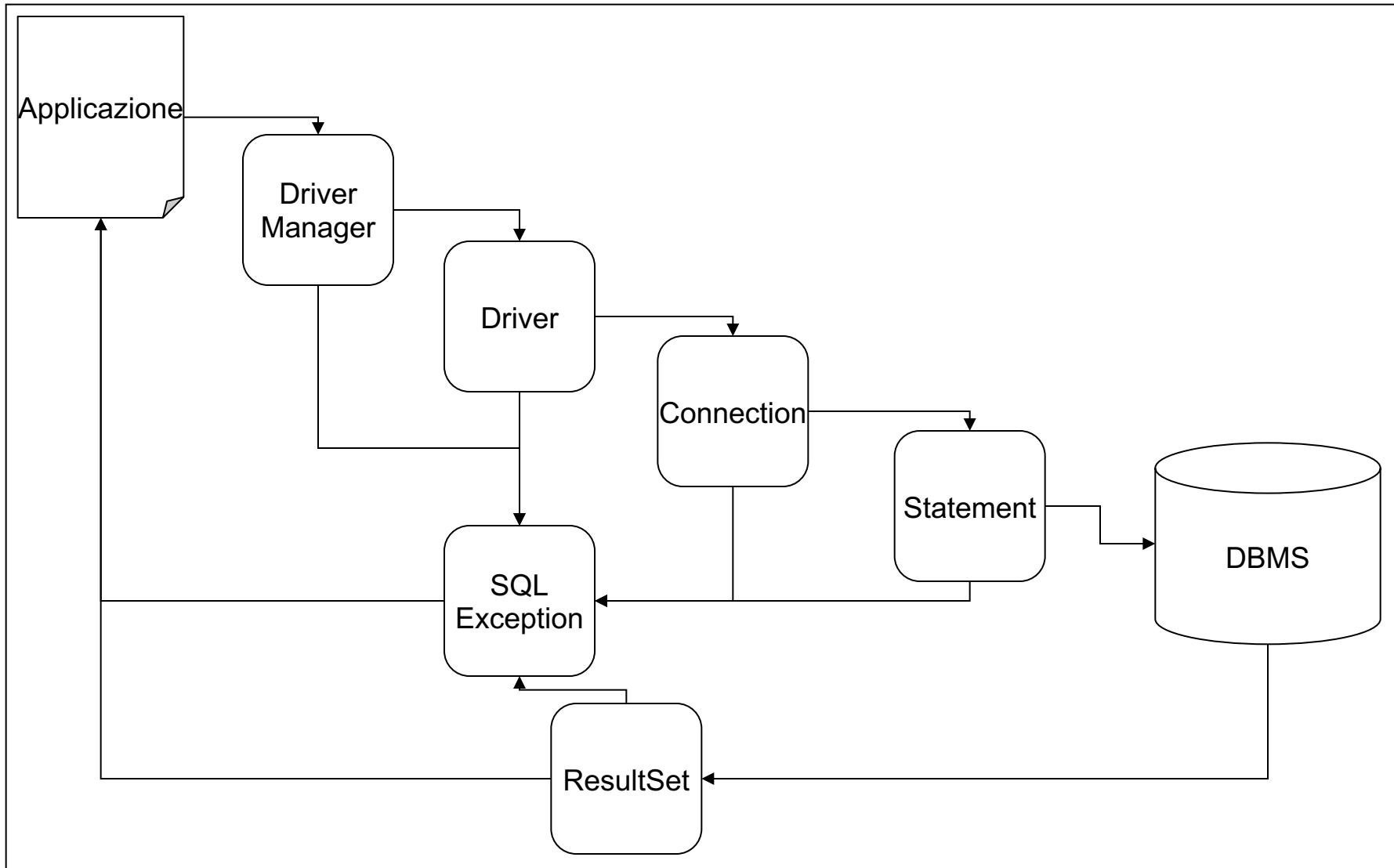
- Si possono aggiungere o modificare le righe di un ResultSet:
 - `rs.update(3, "new value");`
 - `rs.updateRow();`
- Si possono eliminare delle righe

STEP 5 – RILASCIARE LE RISORSE

```
rs.close();  
st.close();  
con.close();
```

ESEMPIO COMPLETO

```
import java.sql.*;  
  
...  
  
try  
    Class.forName(NOME_DRIVER);  
catch(ClassNotFoundException){/* Driver non trovato ! */}  
  
try  
{  
    Connection con = DriverManager.getConnection(URL_MY_DATABASE);  
    Statement st = con.createStatement();  
    ResultSet rs = st.executeQuery("SELECT nome FROM studenti");  
    while(rs.next())  
        System.out.println("Nome: " + rs.getString("nome"));  
  
    catch(SQLException sqe){/* Problema */}  
  
    rs.close();  
    st.close();  
    con.close();  
    ...
```



JDBC API AVANZATE

PROGRAMMI DINAMICI

- Non tutti i programmi sono a conoscenza della struttura del DB su cui operano.
- I ‘table viewer’, ad esempio, scoprono a run-time lo schema del database.
- Sono necessarie delle classi per accedere alla struttura del DB:
 - La classe `DatabaseMetaData`, istanziata da `Connection`, restituisce informazioni generiche sul database.
 - La classe `ResultSetMetaData`, istanziata da `ResultSet`, restituisce le informazioni sulla struttura dello specifico `ResultSet`

DATABASEMETADATA

- L'interfaccia DatabaseMetaData è utilizzata per reperire informazioni sulle sorgenti di dati.
- L'interfaccia mette a disposizione circa 150 metodi classificati nelle seguenti categorie :
 - informazioni generali circa la sorgente dati (Es. Nome del DBMS, ver. del Driver...);
 - aspetti e capacità supportate dalla sorgente dati;
 - limiti della sorgente dati;
 - cosa gli oggetti SQL contengono e gli attributi di questi oggetti (Es. Tutte le tabelle del DB);
 - transazioni offerte dalla sorgente dati.
- DatabaseMetaData md = con.getMetaData();

RESULTSETMETADATA

- Tra le informazioni troviamo:
 - Numero di colonne (getColumnCount())
 - Nome di una colonna (getColumnName())
 - Tipo di una colonna (getColumnTypeName())
- `ResultSetMetaData md = rs.getMetaData();`

LE TRANSAZIONI

- Sono insiemi di Statement raggruppati in un'unità logica inscindibile
- Ogni statement deve andare a buon fine (commit), altrimenti l'intera transazione viene annullata (roll back)
- Passi
 - Inizia transazione
 - esegui statements
 - commit o rollback della transazione

API JDBC PER LE TRANSAZIONI

- Di default ogni singola operazione è una transazione
- I metadati del database specificano il livello di isolamento supportato dal DBMS
- Per eseguire più statement in un'unica transazione si usa:

```
con.setAutoCommit(false);
// esegui gli statements
con.commit();
```

LIVELLI DI ISOLAMENTO

- Servono per gestire transazioni concorrenti
- Dipendono dall'oggetto Connection
 - `con.setTransactionIsolation(...)`
 - TRANSACTION_NONE
 - **Transazioni disabilitate o non supportate dal DBMS**
 - TRANSACTION_READ_UNCOMMITTED
 - Le altre transazioni possono vedere i risultati di una transazione non completata
 - Se una transazione fa il roll back, le altre applicazioni possono restare con dati non validi
 - I dati sono sempre disponibili per la lettura

LIVELLI DI ISOLAMENTO (2)

- TRANSACTION _ READ _ COMMITTED
 - Se un'altra transazione sta scrivendo dati, il reader resta in attesa
 - Le righe lette dal reader non sono bloccate
- TRANSACTION _ REPEATABLE _ READ
 - Se un'applicazione effettua una sequenza di read, otterrà gli stessi risultati
 - Se un'altra transazione sta scrivendo dati, il reader resta bloccato in attesa
 - Le righe lette dal reader sono bloccate finché non effettua una commit()
- TRANSACTION _ SERIALIZABLE
 - L'intera tabella è bloccata durante una lettura

ECCEZIONI E WARNING

PROBLEMI CON I DB

- Sebbene Java sia fortemente tipato, non è possibile effettuare, durante la compilazione, controlli sul corretto utilizzo dei valori provenienti da un DB
- Quasi tutte le istruzioni viste in precedenza possono portare a potenziali problemi (Es. Perdita connessione al db).
 - Alcuni problemi sono ‘recuperabili’, altri no.

ECCEZIONI E WARNING

- In JDBC, oltre alle classiche eccezioni Java, esistono anche i warning
 - Un'eccezione causa una brutale terminazione del metodo in corso da parte della JVM
 - Un warning viene concatenato all'oggetto che lo ha generato, permettendo la prosecuzione del metodo corrente
- È a discrezione dell'implementatore del driver stabilire quale problema generi un'eccezione e quale un warning

SQL EXCEPTION

- E' una sottoclasse di Java.lang.Exception
- Aggiunge informazioni sul tipo di errore proveniente dal database
- Più eccezioni possono essere concatenate

SQLEXCEPTION

SQLException

vendorCode : int

SQLException(reason : String, SQLState : String, vendorCode : int)

SQLException(reason : String, SQLState : String)

SQLException(reason : String)

SQLException()

getSQLState() : String

getErrorCode() : int

getNextException() : SQLException

setNextException(ex : SQLException) : void

SQL WARNING

- E' una sottoclasse di SQLException
- Non richiede il catch
- Utilizzato quando il problema non è tanto grave da richiedere un'eccezione
- Esaminabile con il metodo getWarnings() di Connection, Statement, ResultSet.

SQL WARNING

SQLWarning

```
SQLWarning(reason : String, SQLstate : String, vendorCode : int)
SQLWaming(reason : String, SQLstate : String)
SQLWaming(reason : String)
SQLWarning()
getNextWaming() : SQLWarning
setNextWarning(w : SQLWarning) : void
```

CONCLUSIONI

IL FUTURO

- ORDBMS o ODMG?
- SQL3 (Dicembre '99) aggiunge il supporto agli oggetti nei RDBMS
 - Nuovo Paradigma: Object Relational Database Management System (ORDBMS)
- Object Data Management Group è un consorzio di sviluppatori di DBMS.
 - Il binding ODMG per Java fornisce un supporto nativo alla memorizzazione di oggetti.

ODMG BINDING

- In JDBC lo sviluppatore ha la responsabilità di mappare gli oggetti Java in tabelle del DB e viceversa.
- I bindings ODMG permettono allo sviluppatore di rendere trasparente la persistenza di oggetti Java.

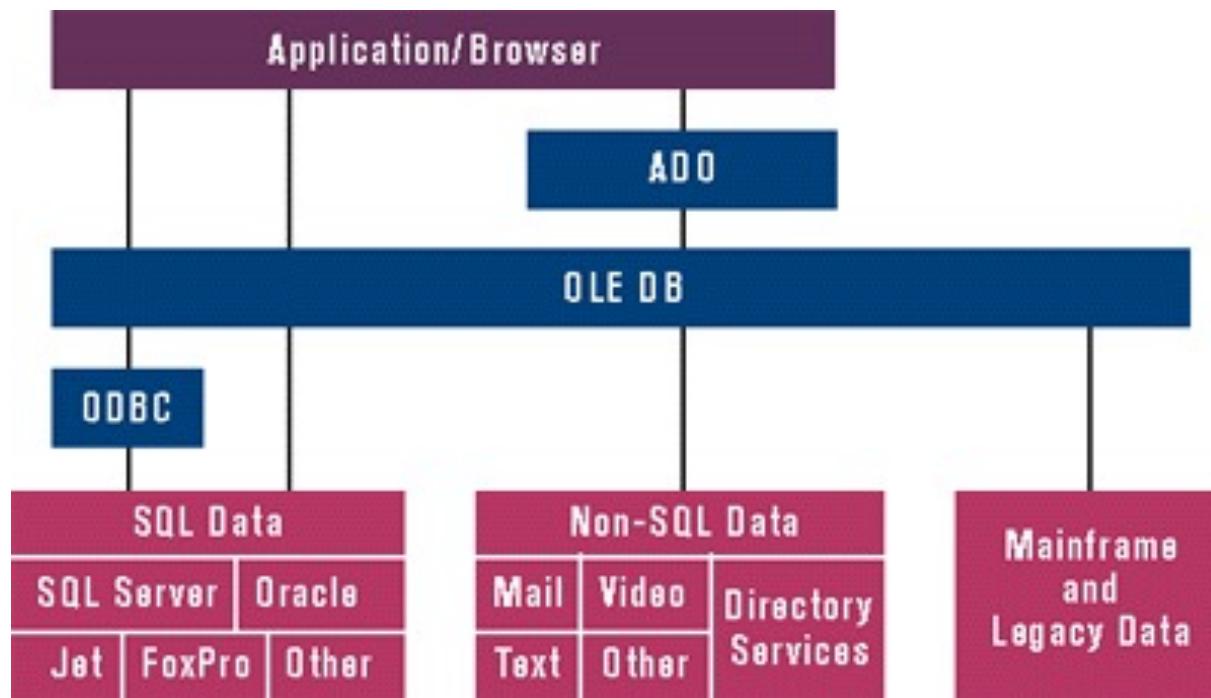
JAVA BLEND

- Il binding ODMG è implementato nel package “Java Blend”, costruito su JDBC.
- Java Blend crea a run-time oggetti da un database relazionale, che corrispondono ad oggetti Java
 - Le tuple di una tabella diventano oggetti
 - Le chiavi esterne diventano riferimenti...

IL FUTURO DELLE CLI

- ODBC non può supportare le nuove caratteristiche di SQL3
 - Nessun supporto agli oggetti
- OLE-DB, ADO e JDBC supportano le nuove caratteristiche
 - Sono basati sul modello ad oggetti
- Microsoft punta su UDA, che permette di accedere con la stessa interfaccia a qualsiasi fonte dati (non necessariamente RDBMS)

ARCHITETTURA UDA



UDA Architecture

RIFERIMENTI

- **JDBC**
 - <http://www.javasoft.com/products/jdbc/index.html>
 - <http://java.sun.com/docs/books/tutorial/jdbc>
 - <http://java.sun.com/products/jdbc/datasheet.html>
- **JDBC e Architetture**
 - <http://www.mokabyte.it>
- **UDA, ADO, OLE-DB**
 - <http://www.microsoft.com/Mind/0498/uda/UDA.htm>
- **DBMS e SQL**
 - Fundamentals of Database Systems
 - Elmasri – Navathe, Addison Wesley



FINE

Per eventuali domande: (in ordine di preferenza personale)

- Ora.
- Chat di Teams
- Mail: silvio.barra@unina.it

