

L'Accesso ai Dati Distribuiti

Dati Distribuiti

- Molto spesso i dati sono dislocati
 - su siti fisici diversi
 - su piattaforme HW/SW diverse

e organizzati in forme diverse (file, DBMS di vario tipo)

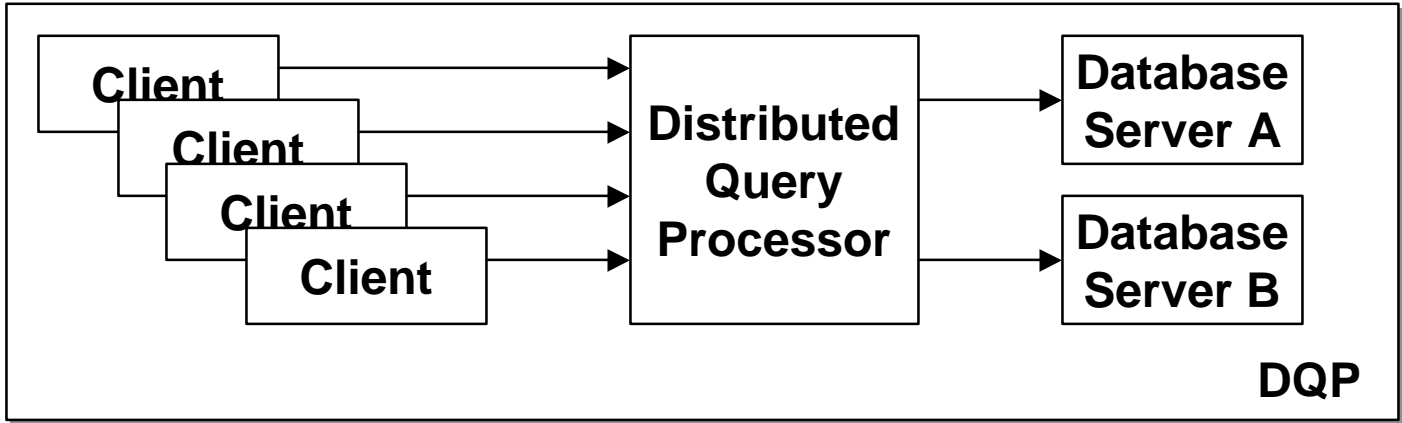
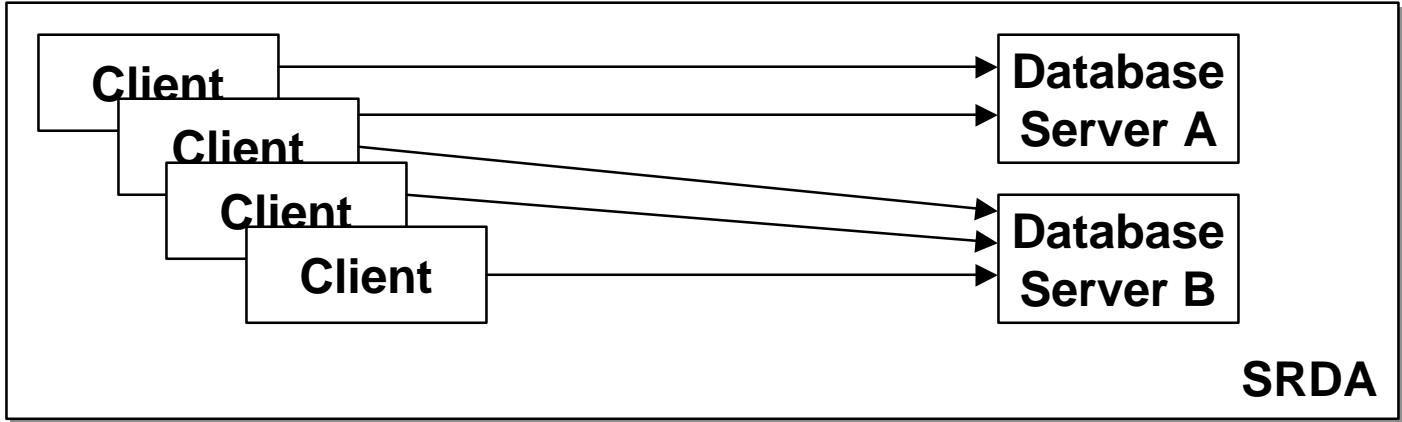
- La distribuzione dei dati può essere
 - senza replicazione
 - con replicazione

Il Middleware per l'Accesso ai Dati Distribuiti

- Deve fornire un accesso semplice ai dati distribuiti e/o replicati (come se fossero disponibili localmente e in copia unica)
- Deve risolvere e mascherare una molteplicità di problemi, anche complessi. I principali sono:
 - conversioni di formato/organizzazione dei dati
 - accesso concorrente ai dati sul singolo database e su database distribuiti
 - mantenimento della consistenza dei dati replicati
 - sicurezza
 - gestione dei failure e dei crash

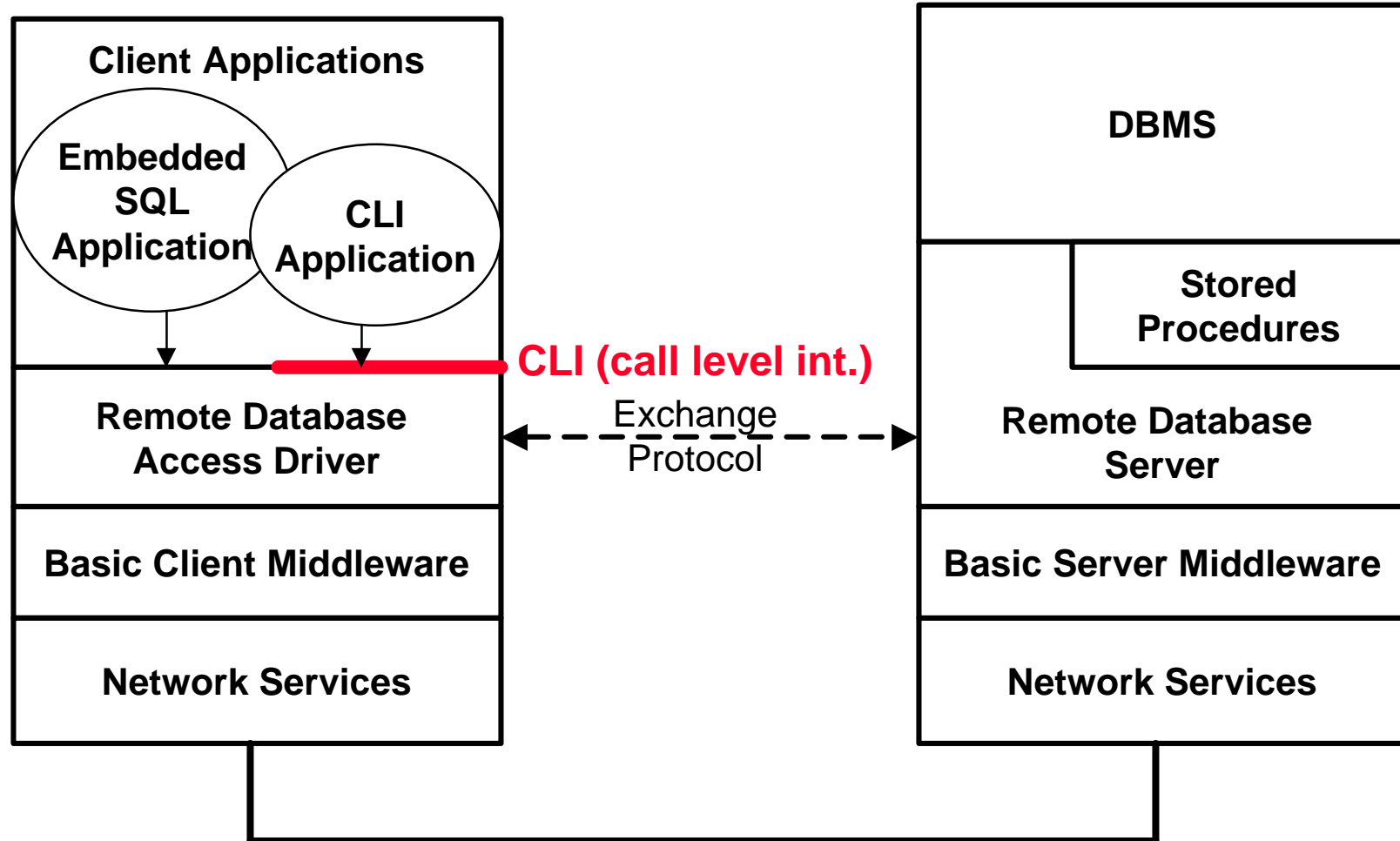
Classificazione delle Funzionalità

- Single-Site Remote Database Access (SRDA)
 - Accesso remoto a database singoli (tipicamente fornito come middleware di base da tutti i produttori di database)
- Distributed Query Processing (DQP)
 - Accesso remoto (read-only) a database multipli (query su tabelle dislocate su più database)
- Distributed Transaction Processing (DTP)
 - Transazioni distribuite (read/write) su database multipli con eventuale replicazione dei dati



interazione →

Tipica Organizzazione C/S di Middleware SRDA SQL



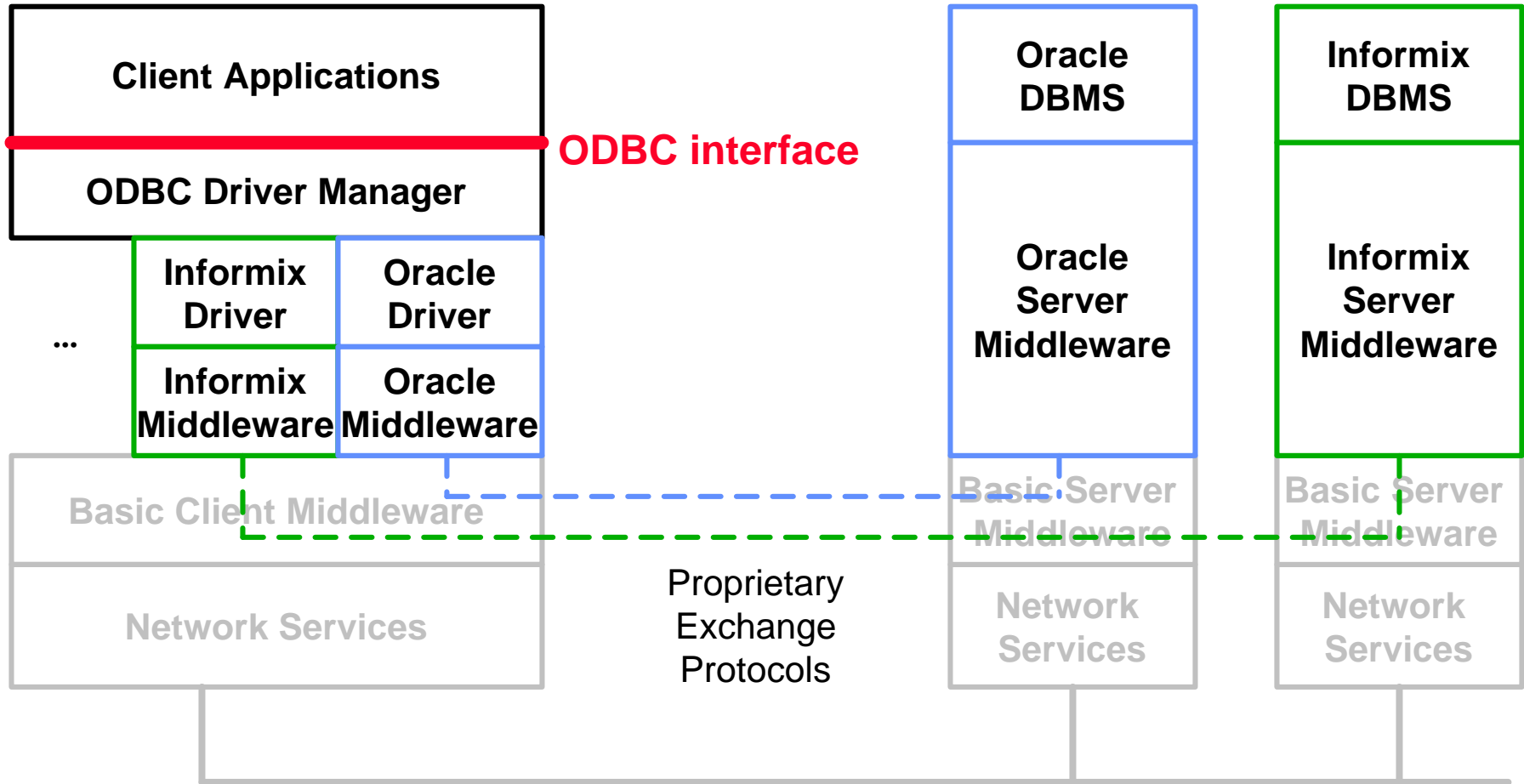
Standardizzazione

- Ogni produttore di database usa tipicamente Protocolli e API proprietari
- Sono però emersi alcuni standard:
 - API standard (permettono di utilizzare le stesse applicazioni client per accedere a DB di produttori diversi)
 - » **Esempi: ODBC (Microsoft), JDBC (Sun)**
 - Protocolli standard (permettono l'interoperabilità tra middleware client e server di produttori diversi)
 - » **Esempi: RDA (ISO), DRDA (IBM)**

ODBC: Open Database Connectivity

- API CLI (call level interface) per l'accesso a DB relazionali (locali o remoti) tramite SQL (specifiche X/Open e SAG SQL CAE)
- È una libreria di funzioni
 - in linguaggio C
 - che rappresentano le tipiche operazioni di accesso al DB in modo astratto (connessione, log on, query, ecc.)
 - che utilizzano rappresentazioni standard per i dati (e per i codici di errore)
- La API richiede un driver ODBC specifico per ciascun tipo di DB al quale si vuole accedere

Il Modello ODBC



Funzioni dei Componenti ODBC

- ODBC Driver Manager
 - Carica i driver secondo necessità
 - Gestisce i riferimenti ai database (nomi) mappandoli sui driver specifici
 - Controlla i parametri delle chiamate ODBC prima di inoltrarle
- ODBC Driver
 - Converte le chiamate (e i dati) ODBC in corrispondenti operazioni specifiche inoltrate al middleware
 - Viene fornito tipicamente dal produttore del DB

Driver ODBC Multiple Tier

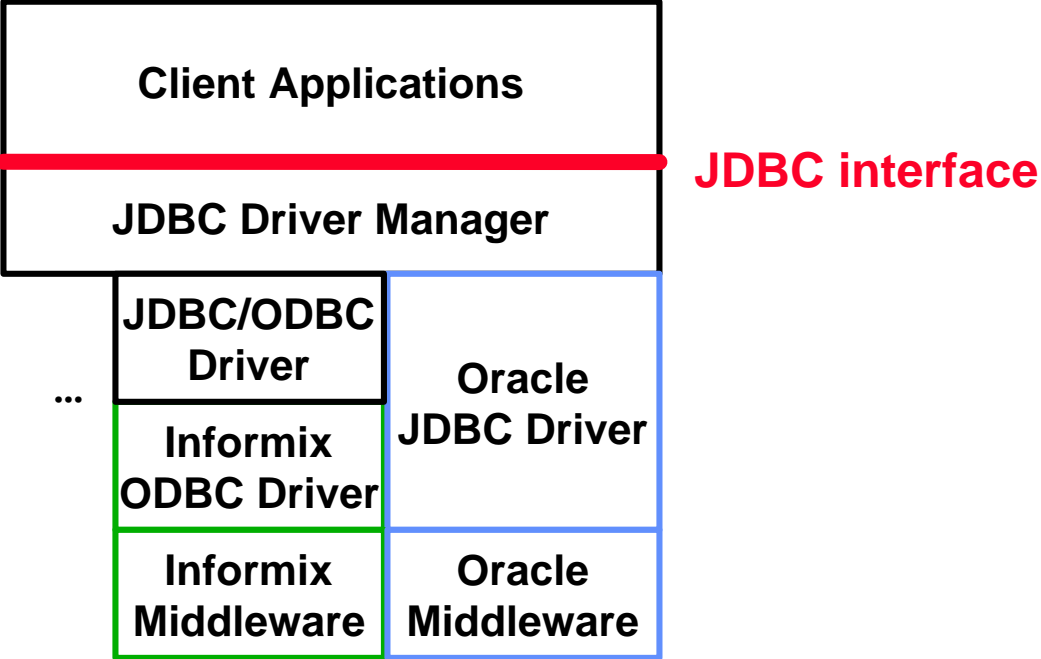
- Sono driver nei quali le funzioni di
 - interpretazione ODBC
 - accesso al driver specifico

sono separate in due moduli indipendenti

JDBC: Java Database Connectivity

- È la versione Java di ODBC
- Condivide con ODBC l'impostazione generale
 - offre in più indipendenza dalla piattaforma (Java)
- Esistono convertitori JDBC-ODBC
 - danno accesso a tutti i DB per i quali esistono driver ODBC
 - con un piccolo costo prestazionale
 - uno di questi (detto JDBC-ODBC bridge) viene fornito con la distribuzione java
- La piattaforma Java2 ha introdotto anche una versione 2 di JDBC (ancora non molto usata)

Client JDBC



JDBC e Applicazioni Web

- Normalmente JDBC viene usato sul lato server delle applicazioni web
- Un applet può accedere ad un DB remoto tramite JDBC, ma
 - resta la restrizione di potersi connettere solo al server da cui l'applet è stato scaricato, a meno che l'applet non sia firmato (DB e server Web devono essere sullo stesso host)

Le Librerie JDBC

- Funzioni base (core package)
 - Sono incluse nella libreria **java.sql**
- Funzioni avanzate (optional package, parte integrante di Java 2)
 - Sono incluse nella libreria **javax.sql**
- Versioni
 - JDBC 1
 - JDBC 2
 - JDBC 3 (comprende tutte le versioni precedenti e le estende)

Identificazione del DB

- Le informazioni di identificazione e modalità di accesso (driver, eventuali parametri, ecc.) vengono specificate in una stringa con formato stile URL:

`jdbc: subprotocol : nome_e_parametri_DB`

Specifica il driver da usare

Formato dipendente dal subprotocol

- Esempi:

`jdbc:odbc://www.polito.it:5000/myDB`

`jdbc:oracle:thin:@ORCL:1025`

Caricamento dei Driver

- Perché il Driver Manager possa attivare un driver jdbc, la sua classe deve essere stata caricata.
- Esistono 2 modalità di caricamento:
 - Chiamando esplicitamente il metodo `Class.forName()`:

```
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
```
 - Impostando la proprietà di sistema `jdbc.drivers` (stringa contenente l'elenco delle classi da caricare, separate da ':')
L'impostazione può essere fatta
 - » all'avvio dell'interprete

```
java -Djdbc.drivers=...
```
 - » importando un file di proprietà

Apertura di una Connessione

- Una connessione è rappresentata da un oggetto che implementa l'interfaccia Connection
- Per ottenere una connessione si usa il metodo statico getConnection della classe DriverManager:

```
url = "jdbc:odbc://www.polito.it:5000/myDB";  
user = "sisto";  
pass = "@%%@"  
Connection con =  
DriverManager.getConnection(url, user, pass);
```

Esecuzione di una Query

- Qualsiasi istruzione SQL sul DB viene eseguita tramite un oggetto che implementa l'interfaccia **Statement**
- I risultati di una Query sono rappresentati da un oggetto di tipo **ResultSet** (interfaccia)
- Esempio di esecuzione di una query:

```
Statement stmt = con.createStatement();  
ResultSet rs =  
    stmt.executeQuery("SELECT * FROM BOOKS");
```
- Lo stesso oggetto Statement può servire per più query

Ispezione di un ResultSet

- Il ResultSet corrisponde al record set di ODBC: esso contiene le righe del risultato
- Il record set può essere ispezionato sequenzialmente:

```
while (rs.next()) {  
    String s = rs.getString(1); // accesso alla riga  
}
```

- Il record set contiene un cursore che scorre le righe
- il cursore inizialmente punta prima della prima riga
- il trasferimento dei dati della riga dal server al client avviene solo alla chiamata di un metodo getXXXXX

I Metodi di Accesso

- Vi è un metodo get per ciascun tipo di dato SQL
- Il parametro può specificare
 - il numero d'ordine della colonna da leggere
 - il nome della colonna da leggere

- Esempi:

```
String s = rs.getString(1); // prima colonna
```

```
double d = getDouble("Price"); // colonna Price
```

Corrispondenza tra tipi SQL e Java

INTEGER o INT	int
SMALLINT	short
NUMERIC(n,m), DECIMAL(m,n) o DEC(m,n)	java.sql.Numeric
FLOAT(n)	double
REAL	float
DOUBLE	double
CHARACTER(n) o CHAR(n) o VARCHAR(n)	String
BOOLEAN	boolean
DATE	java.sql.Date
TIME	java.sql.Time
TIMESTAMP	java.sql.TimeStamp
BLOB	java.sql.Blob
CLOB	java.sql.Clob
ARRAY	java.sql.Array

Rilascio delle Risorse e Chiusura

- Connection, Statement e ResultSet hanno un metodo close()
- Il metodo close() va chiamato in ordine inverso rispetto alla creazione degli oggetti

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

Eccezioni

- I metodi della libreria JDBC possono sollevare una `SQLException`
- Questa prevede alcuni metodi di accesso al tipo di errore:
 - `getSQLState()` restituisce un identificatore `SQLState` (secondo le specifiche X/Open)
 - `getErrorCode()` restituisce il codice di errore fornito dal driver specifico del DB
 - `getNextException()` restituisce il prossimo errore (se ve ne è più di uno)
 - `setNextException()` permette di aggiungere un errore alla catena

Warning

- Gli oggetti `SQLWarnings` derivano sempre dalla classe `SQLException`, ma non sono vere eccezioni
 - non vengono lanciati, ma si ottengono chiamando il metodo `getWarnings()`, disponibile negli oggetti `Connection`, `Statement` e `ResultSet`
 - Hanno metodi `getNextWarning()` e `setNextWarning()` per gestire catene di warning

Aggiornamento dei Dati

- L'aggiornamento dei dati (comandi SQL UPDATE, INSERT, DELETE, e tutti i comandi DDL) avviene usando il metodo `executeUpdate`
- Il valore di ritorno è il numero di righe aggiornate
- Esempio:

```
String command = "UPDATE BOOKS "+  
    "SET PRICE=PRICE-5.00 "+"WHERE TITLE NOT LIKE "%HTML%";  
int n = stmt.executeUpdate(command);
```

Transazioni

- Per poter essere eseguite devono essere supportate (altrimenti viene generata un'eccezione)
- Per default, la connessione è in modalità autocommit (commit automatico dopo ogni statement SQL). Per disabilitare il commit automatico si usa il comando:

```
con.setAutoCommit(false);
```

- Per eseguire manualmente il commit (o il rollback) si usano i comandi:

```
con.commit()
```

```
con.rollback()
```

Tipica Transazione

```
try {  
    con.setAutoCommit(false);  
    Statement stmt = con.createStatement();  
    stmt.executeUpdate(command1);  
    stmt.executeUpdate(command2);  
    stmt.executeUpdate(command3);  
    ...  
    con.commit();  
} catch (Exception e) {  
    con.rollback();  
}
```

Concorrenza

- JDBC prevede i seguenti **livelli di isolamento** delle transazioni (in ordine di isolamento crescente):
 1. Nessun supporto per le transazioni (TRANSACTION_NONE)
 2. UnCommitted Read (TRANSACTION_READ_UNCOMMITTED)
 - » nessun livello di isolamento (dirty read, non-repeatable read e phantom read sono possibili)
 3. Committed Read (TRANSACTION_READ_COMMITTED)
 - » la lettura viene bloccata se i dati sono in transazione
 4. Repeatable Read (TRANSACTION_REPEATABLE_READ)
 - » vengono garantiti integrità dei dati e ripetibilità delle letture
 5. Serializable (TRANSACTION_SERIALIZABLE)
 - » risolve anche il problema delle phantom read (dati nuovi che compaiono in seguito ad un'altra transazione concorrente)

Concorrenza

- Il livello di isolamento viene manipolato con i metodi

```
int i = con.getTransactionIsolation()
```

```
con.setTransactionIsolation(i)
```

- **NB:**

- Un DB può supportare solo alcuni livelli di isolamento
- La connessione ha un livello di isolamento di default che può essere conosciuto chiamando il metodo statico

```
DatabaseMetaData.getDefaultTransactionIsolation();
```

- Le prestazioni diminuiscono all'aumentare del livello di isolamento (perché diminuisce la concorrenza)

Metadati

- Forniscono informazioni sulla struttura di un DB (nomi delle tabelle, nomi e tipi delle colonne, ecc.)
 - sono utili per scrivere applicazioni indipendenti dalla struttura del DB o in grado di funzionare con diverse strutture
- In JDBC vi si accede tramite un oggetto della classe `DatabaseMetaData`:

```
DatabaseMetaData md = con.getMetaData();  
ResultSet rs = md.getTables(null, null, null,  
    new String[] {"TABLE"});  
...
```

Metadati

- E' anche possibile ottenere (meta) informazioni su un record set, con la classe ResultSetMetaData:

```
ResultSetMetaData rsmd = rs.getMetaData();  
int nc = rsmd.getColumnCount();  
String name1 = rsmd洗getColumnName(1);  
...
```

Meccanismi per Migliorare le Prestazioni

- Statement Preparati
 - permettono di creare statement parametrizzati da eseguire con parametri attuali ogni volta diversi
- Stored Procedures
- Batch (JDBC 2)

Esempio di Prepared Statement

```
Connection connessione=null;
try{
    Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
    connessione=DriverManager.getConnection("jdbc:odbc:MokaDb");
    connessione.setAutoCommit(false);
    String sql=
        "update Lettori set nome = \'Topolino\' where nome = ?";
    System.out.println("Preparazione istruz. SQL precompilata: "+sql);
    PreparedStatement pst= connessione.prepareStatement(sql);
    System.out.println("Assegnazione parametro libero = Pippo");
    pst.setString(1,"Pippo");
    pst.execute();
    ...
    connessione.commit();
    connessione.close();
} catch(ClassNotFoundException cnfe){
    System.out.println("ERRORE "+cnfe.getMessage());
}
catch(SQLException sqle){
    System.out.println("ERRORE "+sqle.getMessage());
}
```

Il Package javax.sql

- Fornisce alcune funzionalità avanzate:
 - Interfaccia DataSource (per l'utilizzo di servizi di directory)
 - Supporto per Distributed Transactions
 - Connection Pooling (gestione di un connection manager che pre-alloca un certo numero di connessioni e le distribuisce)
 - RowSet (generalizzazione dei result set, non legati strettamente al DB)

Accesso Web ai Database Relazionali