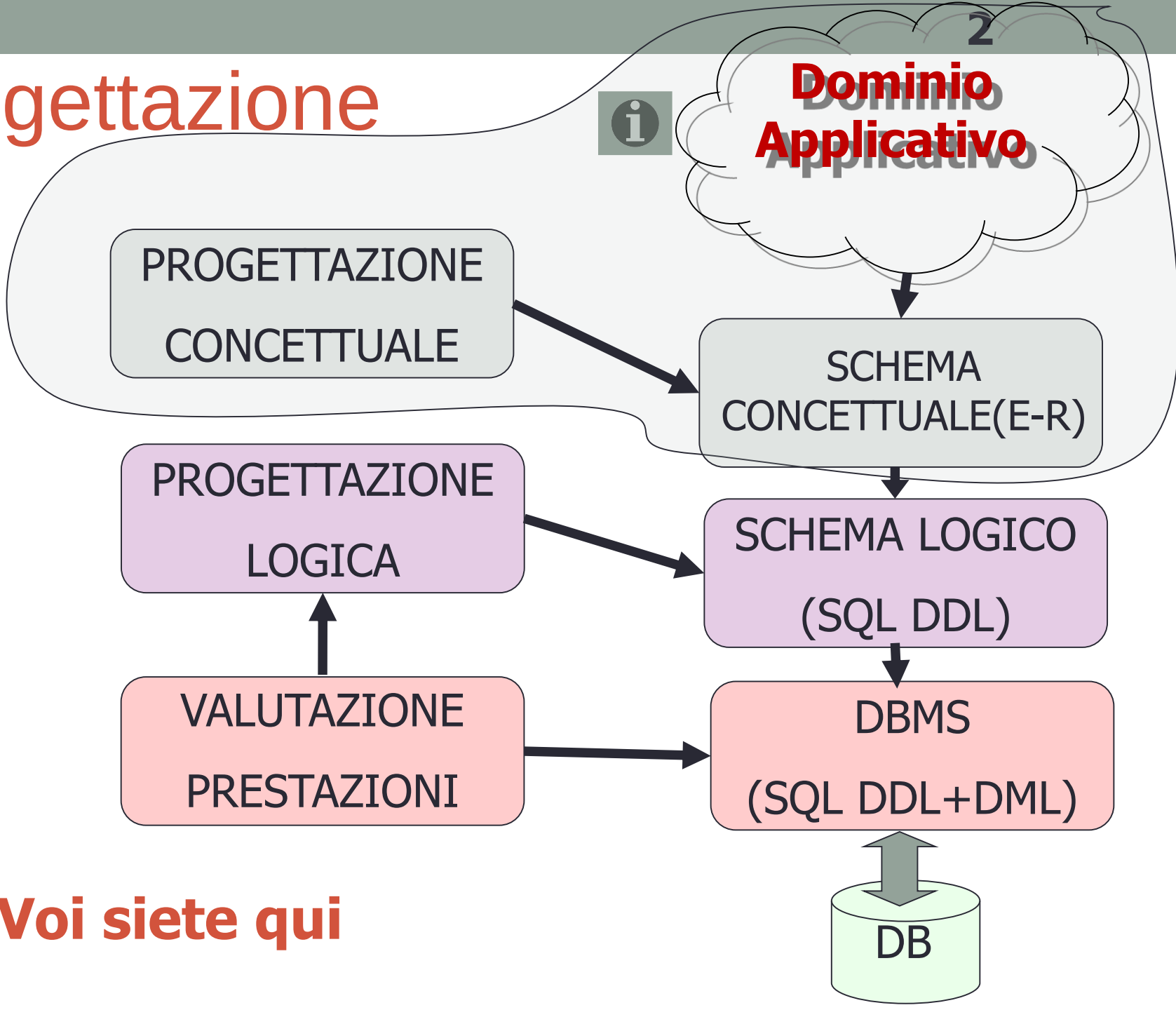
DBMS ED APPLICAZIONI (CAPITOLO 6)

Progettazione di Applicazioni che interagiscono con i DB

- SQL incluso in codice applicativo
- Embedded SQL
- Cursori
- Dynamic SQL
- JDBC
- Stored procedures: triggers

Progettazione

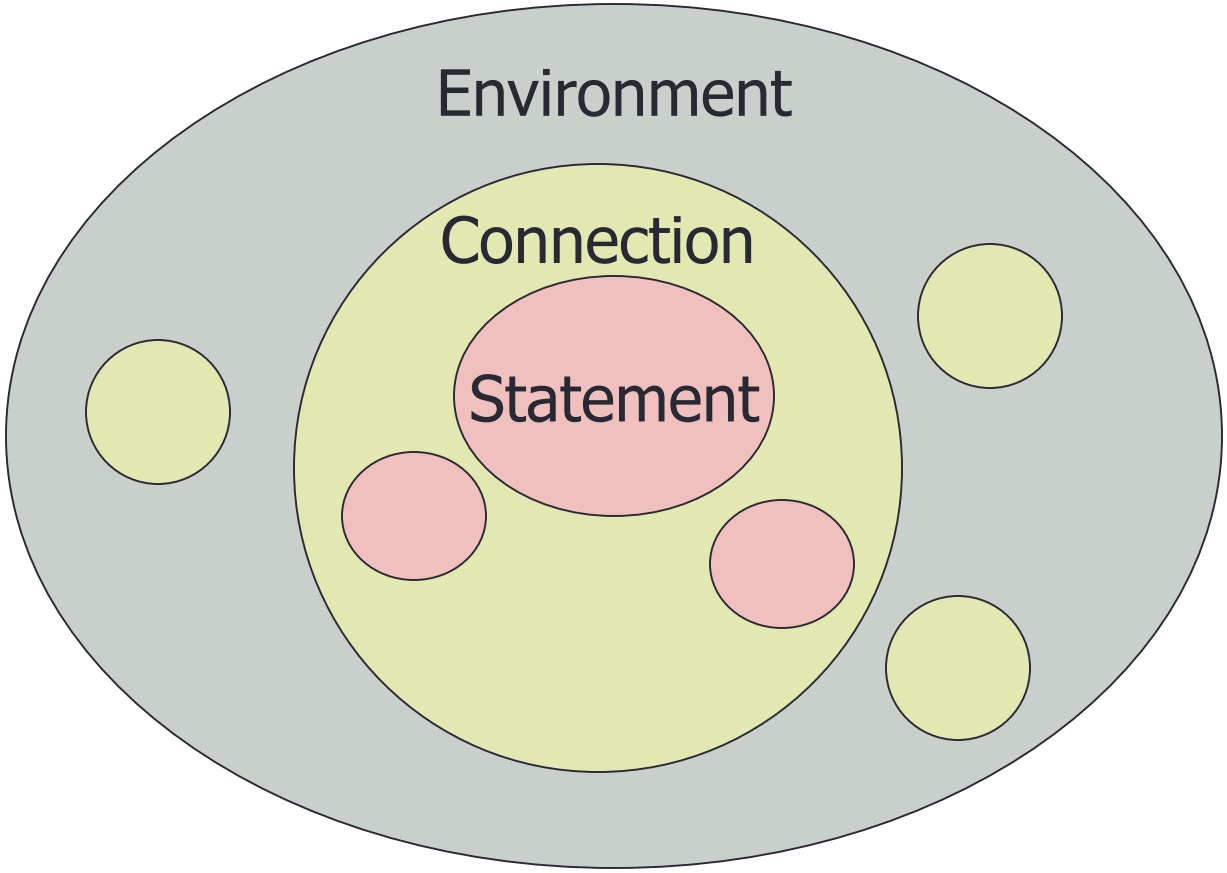


Voi siete qui

SQL e codice applicativo

- APPROCCI:
  - **Embedded SQL** : permette di accedere al databse includendo comandi SQL nel codice di un programma in linguaggio host (ad esempio il C, Cobol, Java)
  - Utilizzare **API specifiche** per richiamare comandi SQL (es. JDBC)

Visione Logica della interazione tra DB e Applicazioni



Data Structures

- L’host language si connette al DB attraverso astrazioni (Strutture dati) dei tipi seguenti:
  1. *Environments* : che descrivono la installazione del DBMS disponibile.
  2. *Connections* : le azioni di login verso il database.
  3. *Statements* : I comando SQL che passano atraverso una connessione.
  4. *Descriptions* : I record che riguardano le tuple relative ad una query o i parametri di un comando.

Embedded SQL

- Il programmatore semplicemente include comandi SQL all'interno di altri linguaggi di programmazione (es.: COBOL, C, ecc.) utilizzando appositi marcatori.

- 1. Si scrive normalmente il codice nel linguaggio ospite marcando opportunamente le linee dei comandi SQL

- 2. Si utilizza un preprocessore (proprietario) che converte le chiamate dell'embedded SQL in chiamate a librerie di database.

- 3. Il programma può essere compilato e sottoposto al linker.

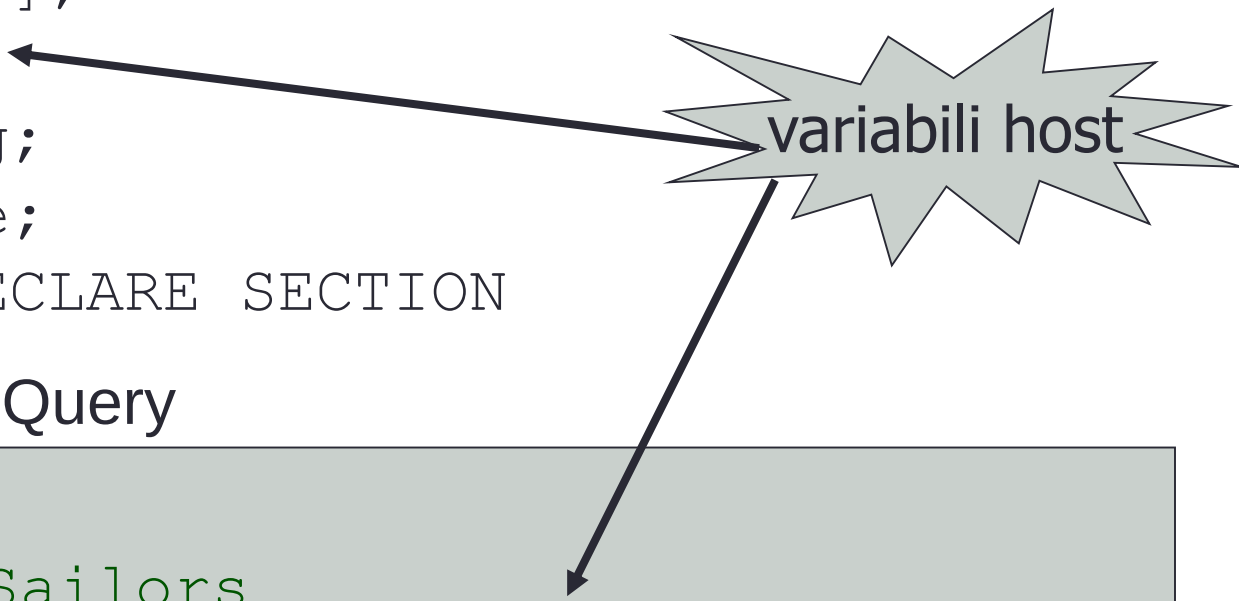
Embedded SQL
- EXEC SQL -

```
/* Per connettersi a runtime con un database*/
EXEC SQL CONNECT TO <nome_Database> AS <nome_Database>
USER <user> WITH <PASSWORD.password>

 /* Per spostarsi su di un'altra connessione: */
 EXEC SQL SET CONNECTION <nome_Database>
 /* Per disconnettersi al termine si usa
    il comando DISCONNECT */

 /* Dichiarazione di variabili host */
EXEC SQL BEGIN (END) DECLARE SECTION

  /* Statements */
EXEC SQL Statement;
```

Embedded SQL-Variabili host

- Dichiarazione delle variabili di “interscambio” dati tra comandi SQL ed linguaggio host (ad esempio il linguaggio C)

```
EXEC SQL BEGIN DECLARE SECTION
char c_sname[20];
long c_sid;
short  c_rating;
float     c_age;
EXEC SQL END DECLARE SECTION
```



- Embedded Insert Query

```
EXEC SQL
INSERT INTO  Sailors
VALUES(:c_sname, :c_sid, :c.rating,:c_age);
```

Embedded SQL- Indipendance mismatch

- Casting per le **variabili SQL specifiche per la gestione degli errori** – 'mantenute' nella SQL Communications Area (SQLCA) :
  - SQLCODE ⟹ long
  - codice (negativo) che identifica l'errore
  - SQLSTATE ⟹ char[6]
  - codici predefiniti per lo stato del comando

- **Impendance mismatch** – Le relazioni SQL sono **(multi-)set di record**. Questa struttura non e' presente nella maggior parte dei linguaggi host. Per gestirle e' necessario utilizzare un costrutto dell'SQL chiamato **cursore**.

Embedded SQL - I cursori (1)

Costrutto del linguaggio SQL, che mi permette di rappresentare i (multi-) set e per il quale l'SQL standard offre metodi di gestione delle tuple in esso rappresentate

```
DECLARE nome_cursore [INSENSITIVE][SCROLL]
  CURSOR [WITH HOLD]
FOR  query
[ORDER BY lista_elementi_ord]
[FOR READ ONLY | FOR UPDATE]
```

Operazioni sui cursori:

- OPEN
- FETCH
- CLOSE

Embedded SQL - I cursori (2)

```
EXEC SQL DECLARE CursorePersona CURSOR FOR
   SELECT COGNOME, NOME, ANNI
                FROM ANAGRAFE
                WHERE COGNOME = "Rossi";
EXEC SQL OPEN CursorePersona;
WHILE (SQLSTATE != '02000') {
EXEC SQL FETCH CursorePersona INTO :c_cognome,
   :c_nome, :c_anni;
  printf("%s Rossi ha %d anni\n",c_nome,c_anni);
}
EXEC SQL CLOSE Cursore Persona;
```

Dopo aver eseguito l'OPEN del cursore, utilizzando ripetutamente l'operatore di FETCH, possiamo leggere le tuple della relazione

Embedded SQL - I cursori (3)

Se l'operazione di selezione produce una relazione composta da una sola tupla non ho bisogno di utilizzare i cursori!

```
EXEC SQL BEGIN DECLARE SECTION
char  c_nome[20];
char  c_cognome[20];
long  c_codfisc;
float       c_anni;
EXEC SQL END DECLARE SECTION

EXEC SQL SELECT COGNOME, NOME, ANNI
        INTO :c_nome,:c_cognome,:c_anni
        FROM ANAGRAFE
        WHERE CODFISCALE =:c_codfisc;
```

Embedded SQL +/-

- Vantaggi:
  - utilizza pura sintassi SQL
- Svantaggi
  - Il precompilatore SQL può entrare in conflitto con gli eventuali altri precompilatori del linguaggio ospitante.
  - Se si accede a più database si può avere a che fare con più API e quindi più precompilatori che a loro volta potrebbero avere una sintassi molto simile;
  - Il passaggio di dati tra SQL e il linguaggio ospitante potrebbe troncare dei valori (errore nel casting dei valori).

Query SQL Dinamiche

- Costruzione di query dinamicamente a run time. Utente 'prepara' la query a run-time e l'applicazione genera lo statement SQL appropriato per richiedere dati al DBMS

```
char c_SQLstring[]=
{"DELETE  FROM ANAGRAFE WHERE eta'<18"};
EXEC SQL PREPARE cancellaMin FROM :c_SQLstring;
EXEC SQL EXECUTE cancellaMin;
```

- **Vantaggi**
  Maggiore flessibilita'

- **Svantaggi**
  Overhead a run-time per la preparazione delle query

ODBC e JDBC

- **Open DataBase Connectivity(ODBC)** introdotto dalla Microsoft per interoperare con diversi DBMS. E' attualmente il più diffuso standard per l'utilizzo di DBMS relazionali.
- **Java DataBase Connectivity(JDBC)** della SUN, fornisce API standard per interrogazione di DBMS

- Il Driver traduce le richieste formulate in JDBC o ODBC in chiamate allo specifico DBMS. Il driver viene caricato dinamicamente a run-time quando viene richiesto ad un *gestore dei driver*, il **Drive Manager**

  - Definizione di un livello standard di astrazione **API (Application Programming Interface)** per accedere alle *capabilities* di DBMS eterogenei.
  - **Portabilita' a livello di eseguibile**: permettono di definire un singolo eseguibile per accedere a diversi DBMS (senza dover ricompilare il programma)

ODBC e JDBC

- Un'applicazione che vuole interrogare un data source tramite JDBC o ODBC deve:
  - selezionare il ***data source*** da interrogare
  - caricare dinamicamente il **driver** corrispondente (tramite il Drive Manager)
  - stabilire una **connessione** con il data source
- NB: parliamo di *data source* e non di DBMS perché tramite JDBC e ODBC raggiungiamo un grado di astrazione che ci permette di ottenere i dati tramite query SQL indipendentemente dal tipo di DBMS (sorgente dati) sottostante.

JDBC Caratteristiche

- **API** (Application Programming Interface) Java
- Uno **standard**
  - Può essere utilizzata da diverse componenti (es: Applet, Applicazioni, EJB, Servlet)
- E' (in generale) **indipendente dal DBMS** (caricato a *run-time*)

- **FUNZIONALITA'**:
  - **Esecuzione di comandi SQL** (DML e DDL)
  - **Manipolazione di result set** (insieme di tuple) con cursori
  - **Gestione dei metadati** (ogg.DatabaseMetaData)
  - **Gestione delle transazioni**
  - Definizione di ***stored procedure***

Architettura JDBC

- **Applicazione**: Inizia e termina la connessione ad un data source
- **Driver Manager** :si occupa di caricare i driver JDBC e di passare le chiamate JDBC dell'applicazione al driver specifico
- **Driver** : Stabilisce la connessione con il data source
- **Data Source** : processa i comandi provenienti dal driver e restituisce i dati richiesti

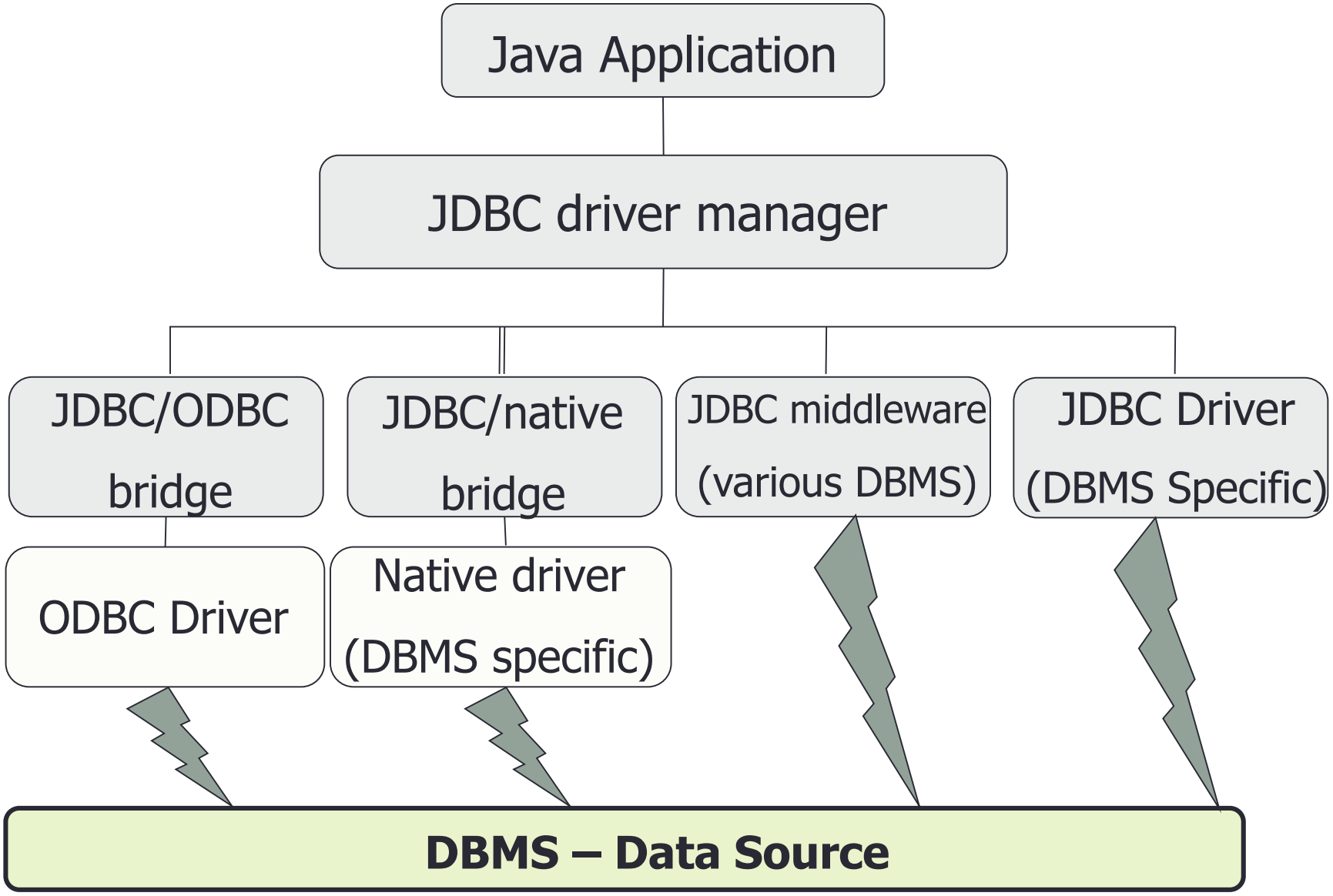
Classi di driver JDBC

http://industry.java.sun.com/products/jdbc/drivers

Vengono distinte quattro diverse tipologie di driver JDBC:

- **Classe1-Bridges**: traduce comandi SQL in API non-native (es: JDBC-ODBC Bridge)
- **Classe2-Traduzione diretta ad API native** (no Java)
  - Traduce le query SQL in API native del data source (richiede installazione di sw su client)
- **Classe3-Network Bridges**:
  - spedisce comandi in rete ad un middleware server che dialoga con il data source (non richiede installazione di sw su client)
- **Classe4-Traduzione diretta ad API native** (Java based):
  - converte le chiamate JDBC direttamente nel protocollo di rete utilizzato dal DBMS. Richiede un driver java su ogni client

Architettura JDBC



Applicazione Java connDB (1)

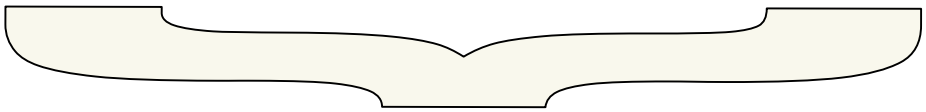
import java.sql.*;

1 Caricare il driver JDBC:

- `Class.forName("com.mysql.cj.jdbc.Driver");`
- `Class.forName("com.mysql.jdbc.Driver").newInstance();`

2 Definire l'URL della connessione al Data Base:

- `url="jdbc:mysql://localhost/musicians";`



"jdbc:*connectionType://host:port/database*"

3 Stabilire la connessione:

```
String user = "nomeutente"; pwD = "password";
Connection con = DriverManager.getConnection(url, user, pwD);
```

4 Creare un oggetto statement.

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

Applicazione Java connDB (2)

5 Eseguire una query : (ad es. INSERT,SELECT,DELETE)

```
String query = "SELECT col1, col2, col3 FROM table";
ResultSet results = statement.executeQuery(query);
```

6 Analizzare/Calcolare i risultati:analisi del risultato contenuto nella classe ResultSet

```
while (results.next()) {
        String a = results.getString(1);
        Integer eta = results.getInt(2);
        System.out.print("NOME= " + a);
        System.out.print("ETA'= " + eta.toString());
        System.out.print("\n");}
```

7 Chiudere/Rilasciare la connessione e lo statement

```
con.close();
statement.close();
```

Java e SQL Data Type Matching

SQL Type	Java Classes	ResultSet get method
BIT	Boolean	getBoolan()
CHAR	String	getString()
VARCHAR	String	getString()
DOUBLE	Double	getDouble()
FLOAT	Double	getDouble()
INTEGER	Integer	getInt()
REAL	Double	getFloat()
DATE	java.sql.Date	getDate()
TIME	java.sql.Time	getTime()
TIMESTAMP	java.sql.TimeStamp	getTimestamp()

Esempio: DB Negozi

- Vedi demo

Alcune considerazioni

- ODBC non è appropriato per un uso diretto dal linguaggio Java perché utilizza interfacce scritte in linguaggio C
- Una traduzione da API C ODBC a API Java non è raccomandata
- Una API Java come JDBC è necessaria per permettere una soluzione Java “pura”
- ODBC è solitamente usato per applicazioni eterogenee
- JDBC è normalmente utilizzato da programmatori Java per connettersi a DB relazionali
- Attraverso un piccolo programma “bridge” è possibile usare l’interfaccia JDBC per accedere a DB accessibili via ODBC

Stored Procedure(s)

- **Stored procedure**: programma residente ed eseguito (tramite un singolo statement SQL) tra i processi del database server

- Vantaggi:
  - Differenti utenti possono riutilizzare e condividere logica applicativa.
  - Software facile da mantenere perche' centralizzato.
- Svantaggi:
  - Legato al DBMS
  - Sovraccarico delle attivita' del database server.

Stored Procedure: Esempi

```
CREATE PROCEDURE ShowNumR
SELECT S.sid,S.nome,COUNT(*)
FROM Sailors S, Reserves R
WHERE S.sid=R.sid
GROUP BY S.sid
```

Le Stored Procedure possono avere parametri:

IN, OUT, INOUT

```
CREATE PROCEDURE AddRating (
IN sailor_sid INTEGER,
    IN inc INTEGER)
UPDATE Sailors
SET rating=rating+inc
WHERE sid=sailor_sid
```

Stored Procedure: Esempi

Le **Stored Procedure** possono essere scritte in java

```
CREATE PROCEDURE TOPSailors ( IN num INTEGER)
LANGUAGE JAVA
EXTERNAL NAME "file:///c:storedProcs/rank.jar"
```

Richiamare **Stored Procedure**

```
EXEC SQL BEGIN DECLARE SECTION
int sid;
int rating;
EXEC SQL END DECLARE SECTION
EXEC CALL AddRating(:sid,:rating);
```

Stored Procedure: Esempi

Le **Stored Procedure** possono essere scritte in java

```
CREATE PROCEDURE TOPSailors ( IN num INTEGER)
LANGUAGE JAVA
EXTERNAL NAME ”file:///c:storedProcs/rank.jar”
```

Richiamare **Stored Procedure in JDBC**

```
CalableStatement callst =
            con.prepareCall(“call TOPSailors(?)”);
callst.setInt(1,5);
ResultSet rs= callst.executeUpdate();
while(rs.next()) …
```

PL/SQL (Oracle Persistent Stored Modules)

Linguaggio Oracle per sviluppare stored procedure

Parte di dichiarazione della **Stored Procedure:**

```
CREATE PROCEDURE nomeproc (p1,p2,...,p3)
dichiarazione delle variabili locali
codice della procedura;
```

```
CREATE FUNCTION nomefun (p1,p2,...,p3)
RETURNS sqlDataType
dichiarazione delle variabili locali
codice della funzione;
```

Esternamente al blocco PL/SQL (BEGIN/END) ho una sezione di DICHIARAZIONE ed eventualmente di **EXCEPTION.**

PL/SQL

```
Declare
Nome sname.Sailors%type;
Contatore Number(2) := 1;
Begin
  SELECT sname
  INTO Nome
  FROM Sailors S
  WHERE S.sid=7302;
  EXCEPTION
     Statement;
END;
```

Esempio PL/SQL

```
DECLARE
contatore number(4) := 7000;
CURSOR query_clienti IS
Select nominativo, codice From clienti;
BEGIN
For record IN query_clienti
LOOP
IF record.codice < 8000 THEN
Contatore := contatore + 1;
DELETE Clienti WHERE CURRENT OF query_clienti;
ELSE
UPDATE Clienti SET Codice = codice + 100
WHERE CURRENT OF query_clienti;
END IF;
EXIT WHEN query_clienti%ROWCOUNT > 1000;
END LOOP;
END;
```

Conclusioni

- **Embedded SQL** permette di eseguire query statiche (parametriche) incluse in un linguaggio host
- ( registrate ed ottimizzate ;  poca flessibilita')
- **Dinamic SQL** permette di eseguire query definite a run-time
- ( alta flessibilita';  overhead)
- **Cursore** componente per la gestione tupla per tupla delle relazioni, elimina l'incompatibilita' tra la gestione SQL dei dati come (multi-) set e i linguaggi host
- **APIs** ad esempio il JDBC per la definizione di un livello intermedio di astrazione tra applicazione e DBMS.
- **Stored Procedures** eseguono logica applicativa internamente al DBMS
- **SQL/PSM Standard** per lo sviluppo di *stored procedure* (es: PL/SQL)