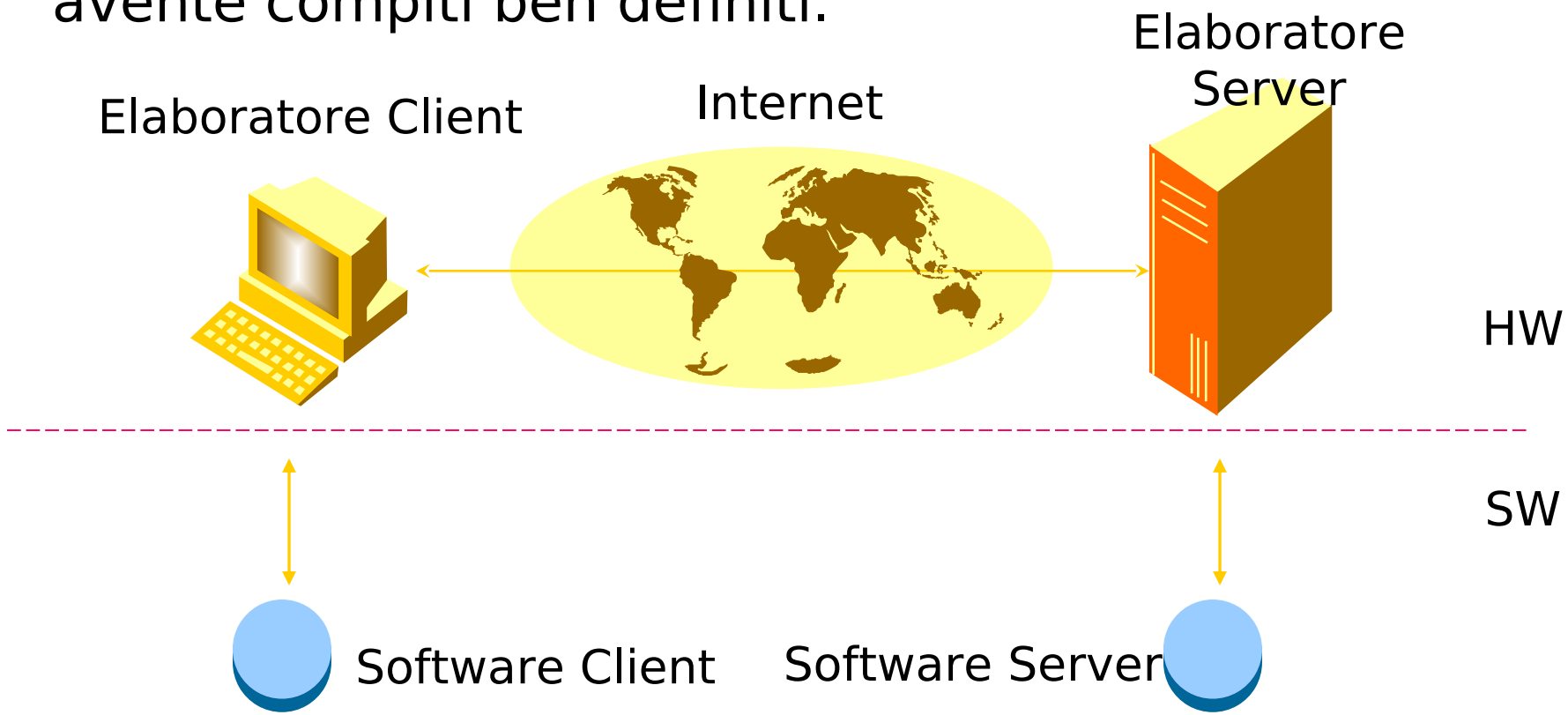
DBMS & Web

L’architettura client-server del WWW

Il Web è una architettura software di tipo *client-server*, nella quale sono previste due tipologie di componenti software: il *client* e il *server*, ciascuno avente compiti ben definiti.



L’architettura client-server del WWW

Il *client* è lo strumento a disposizione dell'utente che gli permette l'accesso e la navigazione nell'ipertesto del Web. Esso ha varie competenze tra le quali:

1) Trasmettere all'opportuno server le richieste di reperimento dati che derivano dalle azioni dell'utente

2) Ricevere dal server le informazioni richieste

3) Visualizzare il contenuto della pagina Web richiesta dall'utente, gestendo tutte le tipologie di informazioni in esse contenute

I client vengono comunemente chiamati *browser* (sfogliatori) (es: Netscape Navigator; Internet Explorer).

L’architettura client-server del WWW

Il *server* è tipicamente un processo in esecuzione su un elaboratore. Esso è sempre in esecuzione ed ha delle incombenze molto semplici:

1) rimanere in ascolto di richieste da parte dei client

2) fare del suo meglio per soddisfare ogni richiesta che arriva:

- se possibile, consegnare il documento richiesto

- altrimenti, spedire un messaggio di notifica di errore (documento non esistente, documento protetto, ecc.)

I server vengono comunemente chiamati *demoni-HTTP* o *web server*

Web & basi di dati

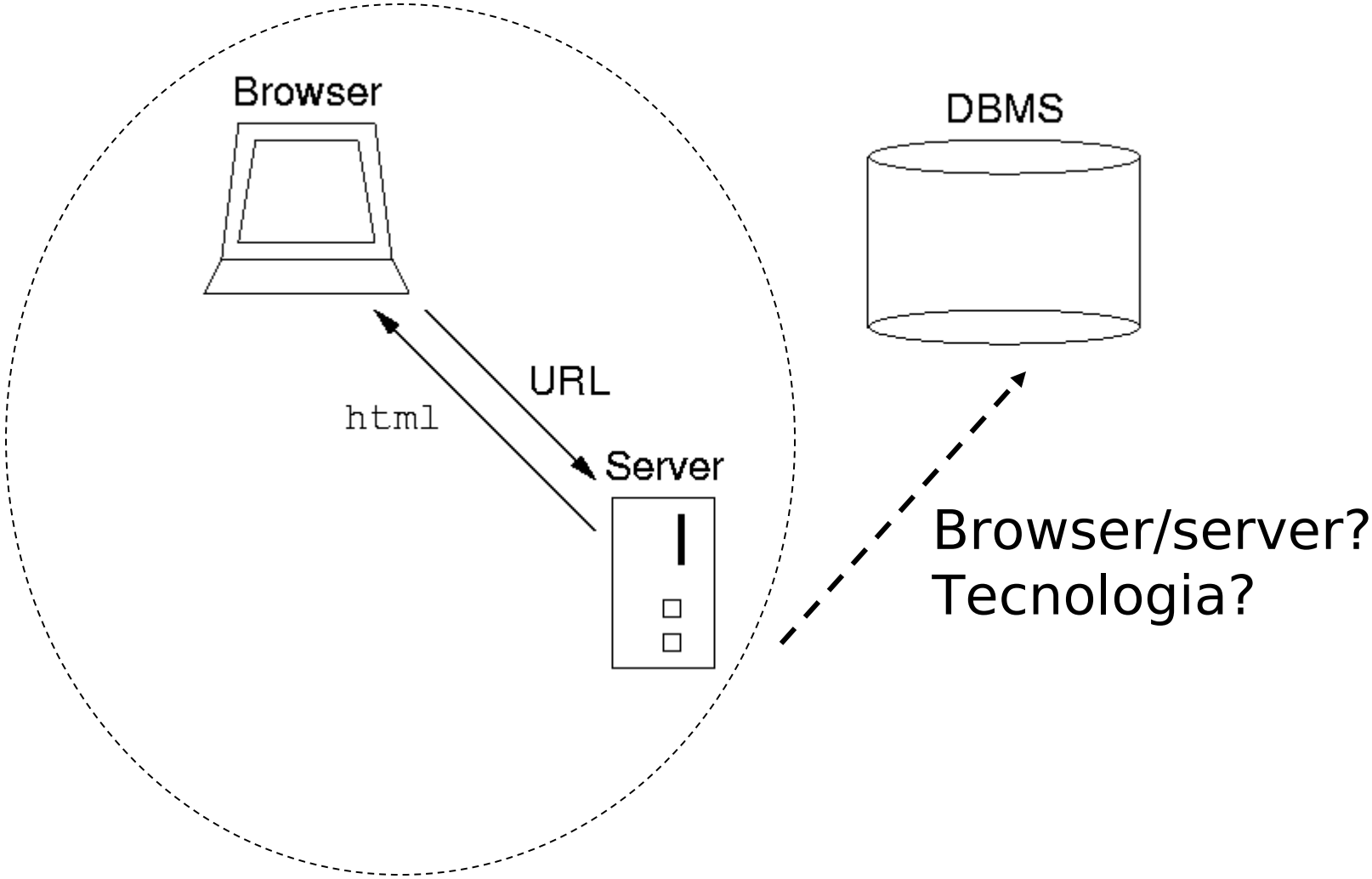
- Obiettivi: :
  - ottenere la generazione dinamica di pagine a partire da dati contenuti in una base di dati
  - sfruttare i pregi di Web e basi di dati, aggirandone i difetti

Pregi e difetti di basi di dati e Web

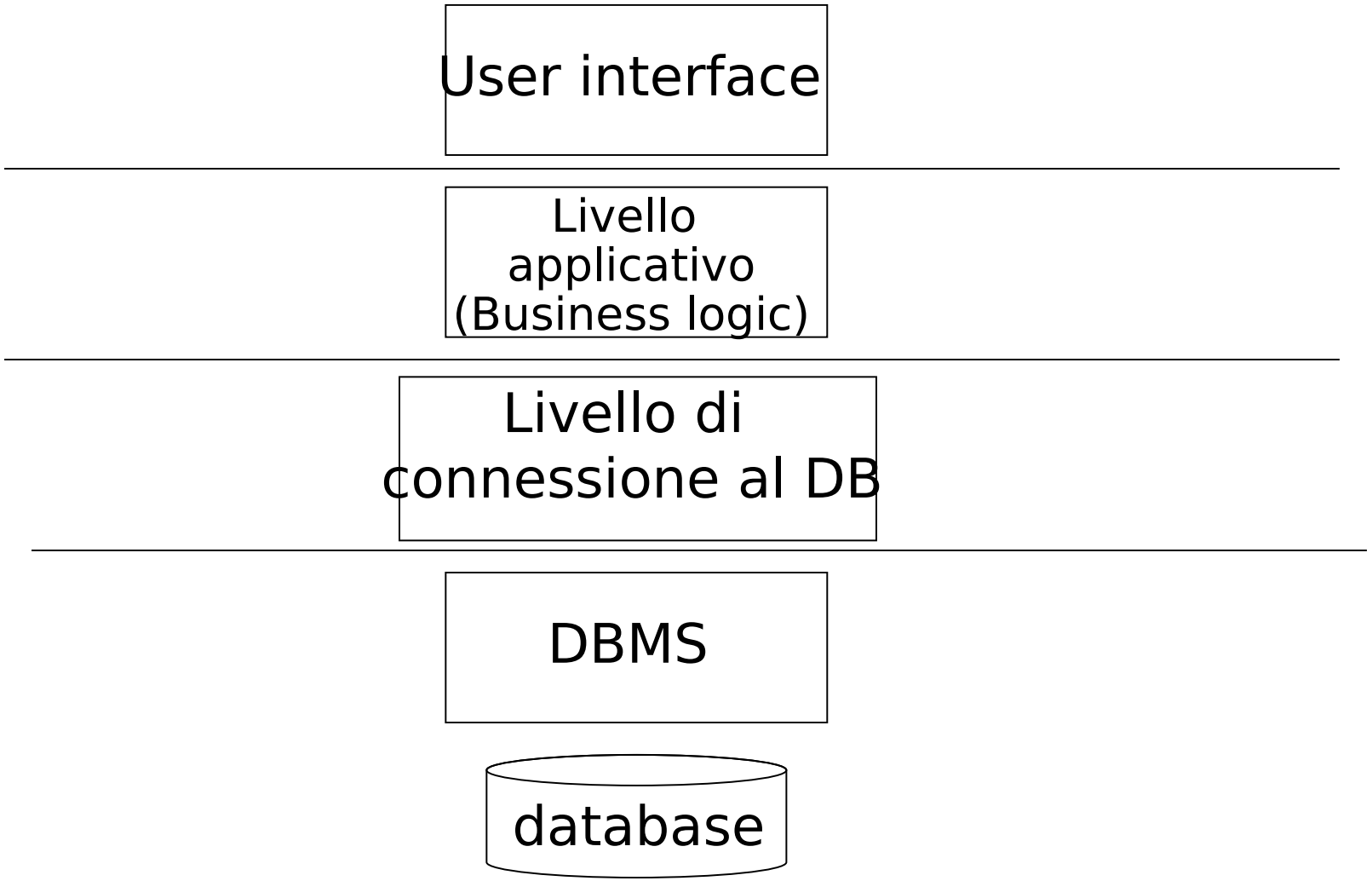
- **Web**
  - Pro
    - semplice
    - portabile
    - a basso costo
    - indipendente dalle interfacce
    - ipermediale
  - Contro
    - basato su file
    - statico

- **Basi di dati**
  - Pro
    - modelli dei dati
    - linguaggi di interrogazione
    - funzioni di amministrazione
  - Contro
    - complesse
    - proprietarie
    - navigazione e presentazone assenti

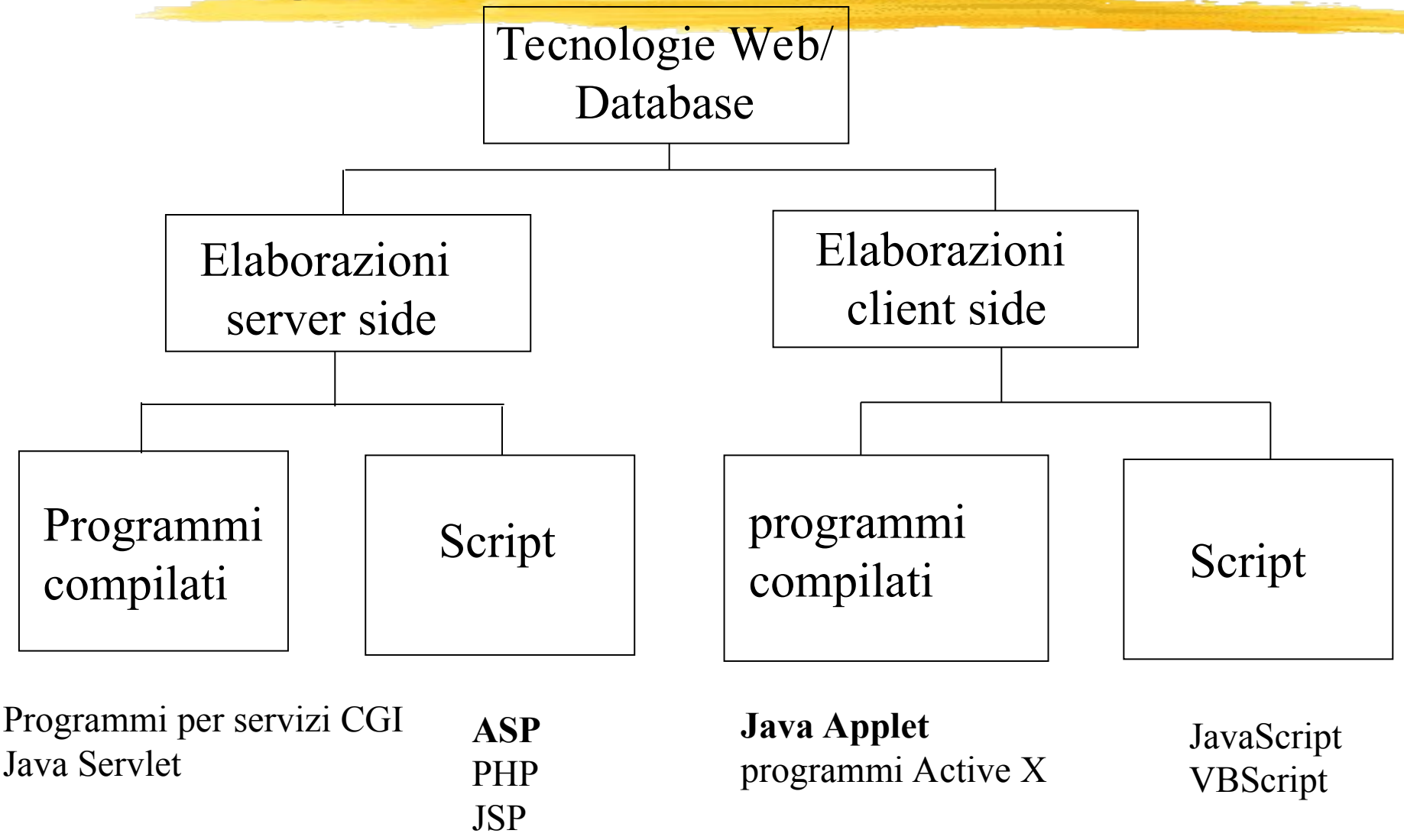
Problema



Architettura generale a livelli



Una gerarchia di soluzioni



Apriamo una parentesi …

- Il problema principale nelle architetture client-server è l'esistenza di client eterogenei:
  - Backend: database server
  - Frontend: gui/forms/interfaccie web
- Nasce il problema di capire come le applicazioni client possano comunicare con il server:
  - come descrivere le query
  - come descrivere i risultati delle query
- Prima soluzione: uso SQL
  - Svantaggi: cambiando server potrebbe essere necessario cambiare il client, in quanto lo specifico dialetto SQL potrebbe cambiare
- Necessità di accedere i dati in modo interoperabile, cioè indipendente dallo specifico server considerato
  - se cambia il server non cambia l'applicazione
  - i tempi di sviluppo applicazioni client si riducono

Introduzione

- L’interoperabilità rappresenta il problema principale nello sviluppo di applicazioni eterogenee per sistemi distribuiti
- Richiede la disponibilità di funzioni di adattabilità e di conversione che rendano possibile lo scambio di informazione tra sistemi, reti ed applicazioni, anche se eterogenei
- In riferimento ai sistemi di gestione dei dati, l’interoperabilità ha richiesto lo sviluppo di standard adeguati, nella forma di API (Application Program Interface)
  - ODBC
  - JDBC

Introduzione

- Per comprendere la necessità di comunicare attraverso API con un DBMS, consideriamo come sia possibile in generale realizzare questa comunicazione:
  - embedded SQL
  - moduli SQL
  - call-level interface (CLI)

Introduzione

- Embedded SQL:
  - SQL “inserito” in un linguaggio di programmazione
  - statement processati da uno speciale precompilatore
  - può essere:
    - statico (statement noti a compile-time)
    - dinamico (statement generati a run-time)
  - meccanismo di interazione standard ma il codice dipende dal DBMS prescelto
    - cambiando DBMS l’applicazione deve essere nuovamente compilata
  - Esempio: per Java SQLJ

Introduzione

- Moduli
  - Ogni modulo racchiude un insieme di statement SQL
  - un modulo può essere interpretato come un oggetto di libreria legato al codice applicativo
  - questo collegamento dipende dall'implementazione:
    - le procedure possono essere compilate e linkate al codice applicativo
    - possono essere compilate e memorizzate nel DBMS e chiamate dal codice applicativo
    - possono essere interpretate
  - chiara separazione tra statement SQL e linguaggio di programmazione
  - meccanismo di interazione standard ma il codice dipende dal DBMS prescelto
    - cambiando DBMS l'applicazione deve essere nuovamente compilata
  - Esempio: SQL Server Stored procedures

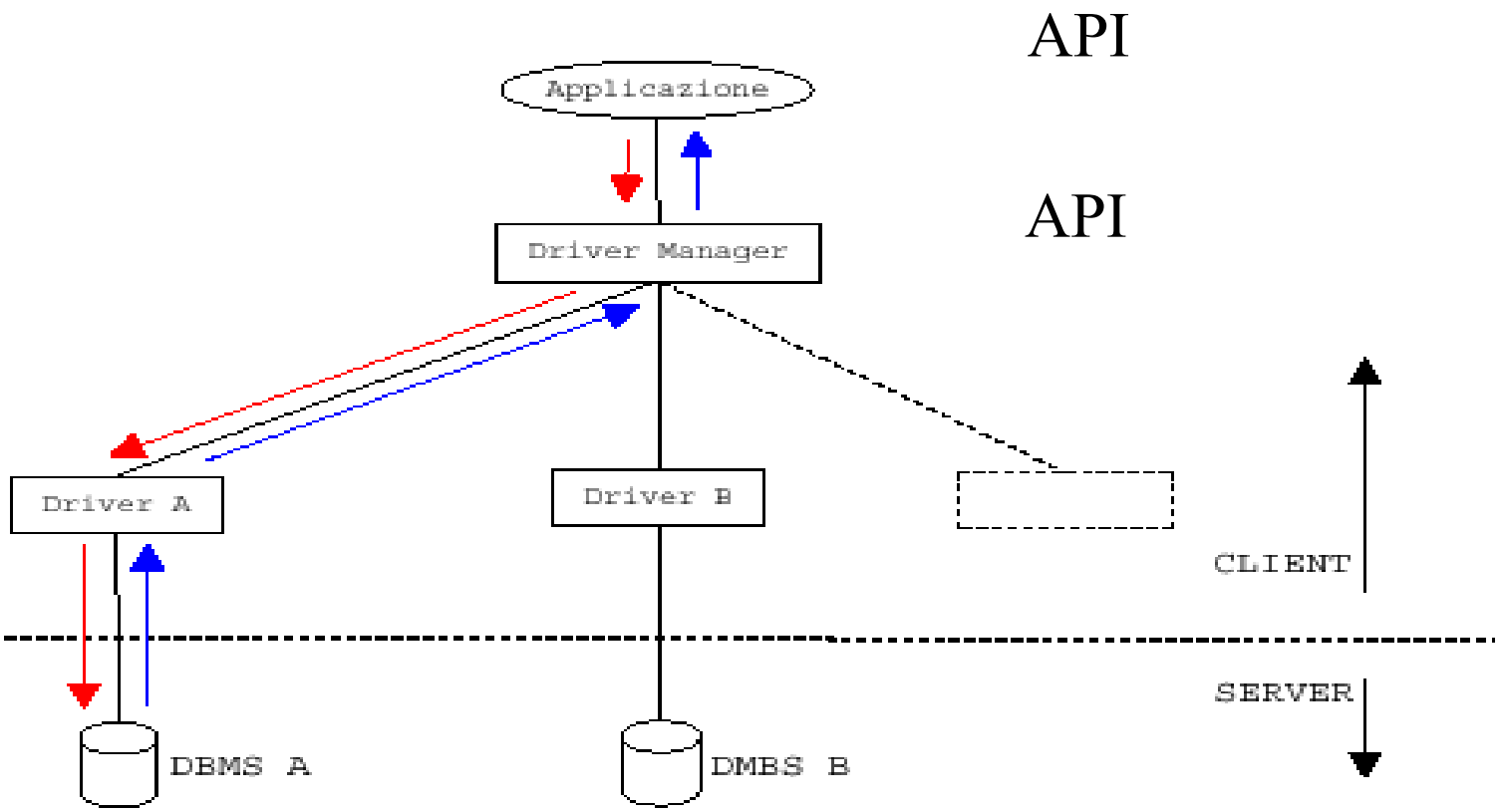
Introduzione

- Call-level interface
  - libreria di funzioni del DBMS che possono essere chiamate dai programmi applicativi
  - simile alle tipiche librerie C
  - passi tipici:
    - l'applicazione effettua una chiamata ad una funzione CLI per connettersi al DBMS
    - l'applicazione crea uno statement SQL e lo inserisce in un buffer, quindi chiama una o più funzioni CLI per inviare lo statement al DBMS per l'esecuzione
    - se lo statement è di tipo SELECT, la funzione restituisce le tuple del risultato
    - se è di tipo INSERT, DELETE, UPDATE, la funzione restituisce il numero di tuple modificate
    - se è di tipo DDL, non viene restituito alcun valore significativo
    - l'applicazione effettua una chiamata ad una funzione CLI per disconnettersi dal DBMS

Introduzione

- Tra i tre livelli di interazione precedenti, le CLI sembrano il meccanismo più adeguato per garantire interoperabilità:
  - indipendenti dal codice client
  - l'applicazione cambia interazione con il server chiamando in modo opportuno le funzioni di libreria
  - chiara distinzione tra interfaccia ed implementazione
  - una stessa applicazione binaria può funzionare considerando diversi DBMS
- Necessità di standardizzazione
  - le applicazioni devono potere accedere DBMS diversi usando lo stesso codice sorgente
  - le applicazioni devono potere accedere DBMS diversi simultaneamente
  - ODBC, JDBC si possono vedere come CLI ottenute come risultato del processo di standardizzazione
    - ODBC: libreria C
    - JDBC: libreria Java

Architettura di riferimento



Applicazione

- Un’applicazione è un programma che chiama specifiche funzioni API per accedere ai dati gestiti da un DBMS
- può anche essere un’applicazione Web
- Flusso tipico:
  - selezione sorgente dati (DBMS e specifico database) e connessione
  - sottomissione statement SQL per l’esecuzione
  - recupero risultati e processamento errori
  - commit o rollback della transazione che include lo statement SQL
  - disconnessione

Driver Manager

- È una libreria che gestisce la comunicazione tra applicazione e driver
- risolve problematiche comuni a tutte le applicazioni
  - quale driver caricare, basandosi sulle informazione fornite dall'applicazione
  - caricamento driver
  - chiamate alle funzioni dei driver
- l'applicazione interagisce solo con il driver manager

Driver

- Sono librerie dinamicamente connesse alle applicazioni che implementano le funzioni API
- ciascuna libreria è specifica per un particolare DBMS
  - driver Oracle è diverso dal driver Informix
- traducono le varie funzioni API nel dialetto SQL utilizzato dal DBMS considerato (o nell'API supportata dal DBMS)
- il driver maschera le differenze di interazione dovute al DBMS usato, il sistema operativo e il protocollo di rete
- Si occupano in particolare di:
  - iniziare transazioni
  - sottomettere statement SQL
  - inviano dati e recuperano dati
  - gestiscono errori

DBMS

- Il DBMS sostanzialmente rimane inalterato nel suo funzionamento
- riceve sempre e solo richieste nel linguaggio supportato
- esegue lo statement SQL ricevuto dal driver e invia i risultati

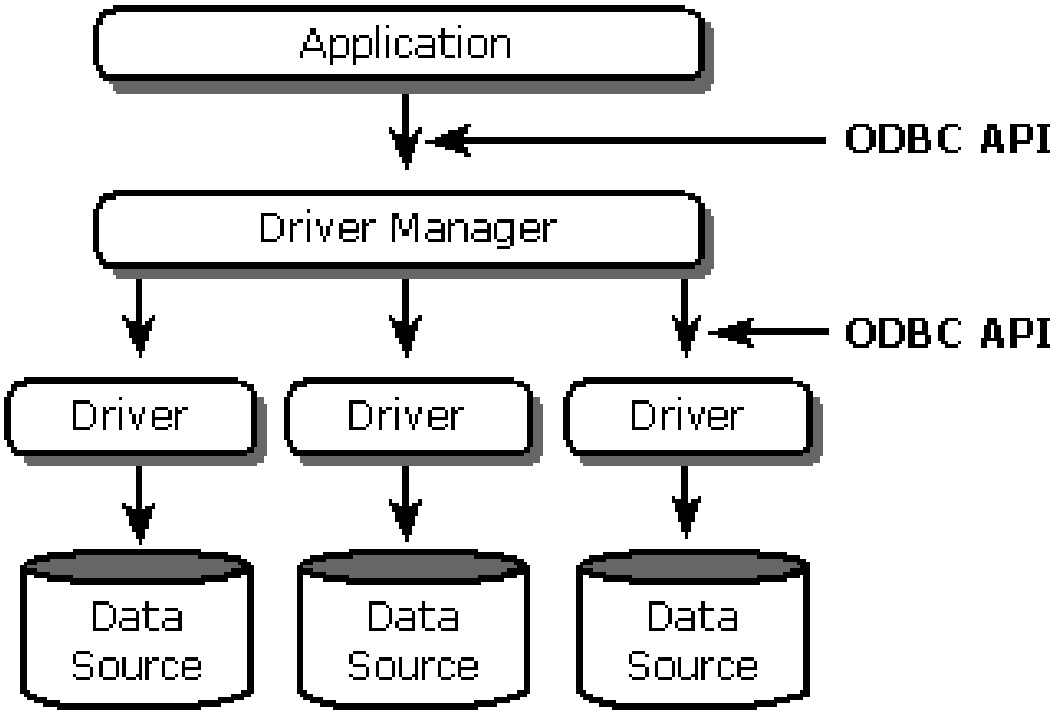
ODBC (Open DataBase Connectivity)

- Standard proposto da Microsoft nel 1991
- Supportata da praticamente tutti i sistemi di gestione dati relazionali
- Offre all'applicazione un'interfaccia che consente l'accesso ad una base di dati non preoccupandosi di:
  - il particolare dialetto di SQL
  - il protocollo di comunicazione da usare con il DBMS
  - la posizione del DBMS (locale o remoto)
- è possibile connettersi ad un particolare DB tramite una DSN (Data Source Name), che contiene tutti i parametri necessari alla connessione con il DB:
  - protocollo di comunicazione
  - tipo di sorgente dati (es: Oracle DBMS)
  - specifico database

ODBC

- Il linguaggio supportato da ODBC è un SQL ristretto, caratterizzato da un insieme minimale di istruzioni
- Questa è una scelta obbligata se si vuole permettere un cambio di DBMS lasciando inalterati i client
- Il linguaggio permette di eseguire query statiche per applicazioni di tipo tradizionale o dinamiche, per applicazioni interattive
- Nel primo caso eventuali errori di SQL sono riportati al momento stesso della compilazione
- Architettura conforme a quando descritto in generale
- Funzioni implementate in C, quindi non completamente portabili

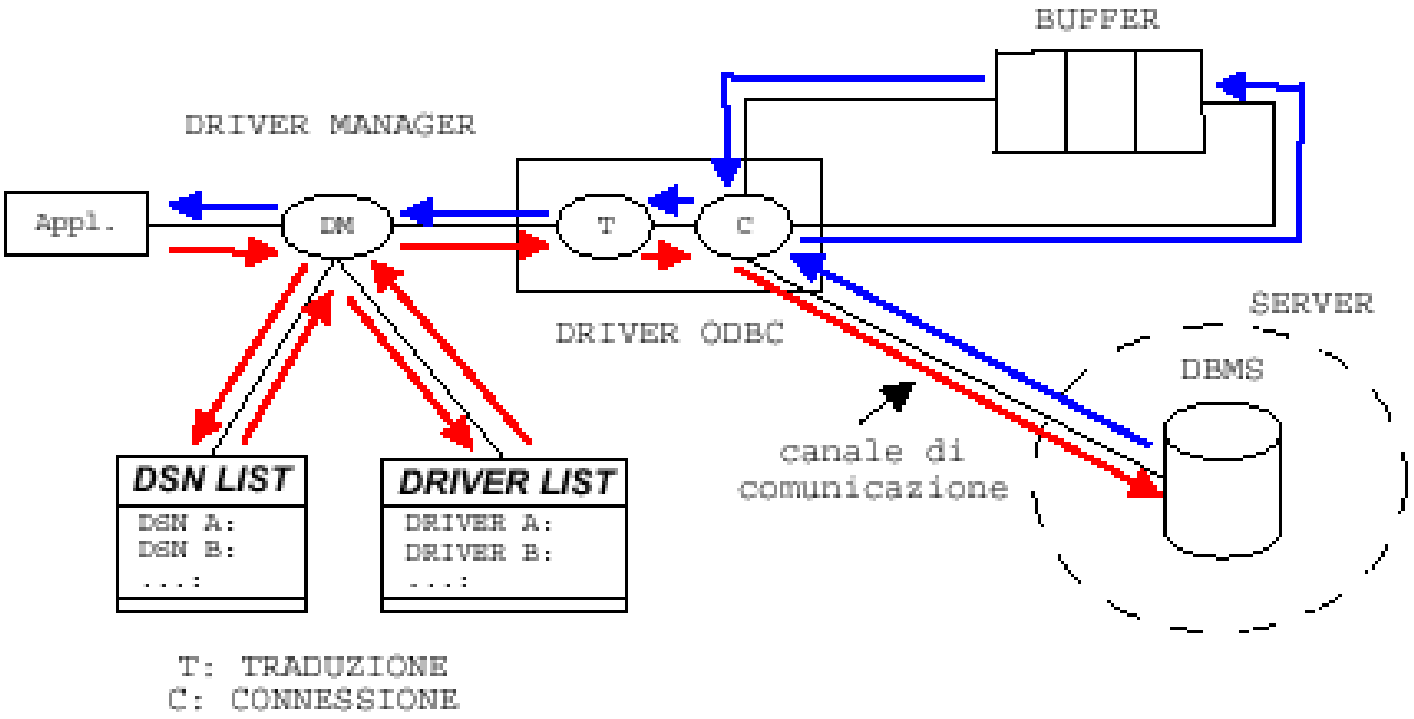
Architettura



Architettura

- Driver Manager:
  - su Windows, fornito con il sistema operativo
  - con Linux, deve essere installato
- Driver:
  - forniti con il DBMS a cui si riferiscono
  - spesso installati e registrati a livello Driver Manager quando si installa il DBMS client

Funzionamento di una query ODBC



Come registrare una DSN?

- I driver ODBC mettono in genere a disposizione tool di amministrazione che permettono di specificare tutti i dettagli relativi ad un DSN (Data Source Name)
- DSN: Data Source Name, è la stringa che identifica:
  - tipo di driver
  - tipo di comunicazione
  - tipo di database
  - nome database
- Dopo la registrazione, queste informazioni sono a disposizione del Driver Manager

Vantaggi e svantaggi

- Vantaggi
  - astrazione (almeno teorica) rispetto al DBMS utilizzato
  - diffusione: supportato almeno parzialmente da molti DBMS commerciali e non
  - non modifica il server
- Svantaggi
  - per ogni coppia (piattaforma client, server) serve un driver specifico (strategia commerciale Microsoft)
  - gran parte del lavoro è demandata al driver, che diventa quindi cruciale per il funzionamento

Universal Data Access

- ODBC rientra nella strategia Microsoft Universal Data Access, che ha come obiettivo l'accesso ai dati (non solo alle basi di dati) indipendentemente dal linguaggio e dagli strumenti utilizzati
- Universal Data Access include l seguenti elementi:
  - ODBC
  - OLE DB: insieme di interfaccie di programmazione di basso livello per l'accesso a dati eterogenei
  - ADO (ActiveX Data Objects): interfaccie ad alto livello per interagire con ADO (ADO, RDS, ADOX, ADO MD)
    - ADO è forse la più conosciuta (utilizzata per interagire con DBMS da pagine ASP)

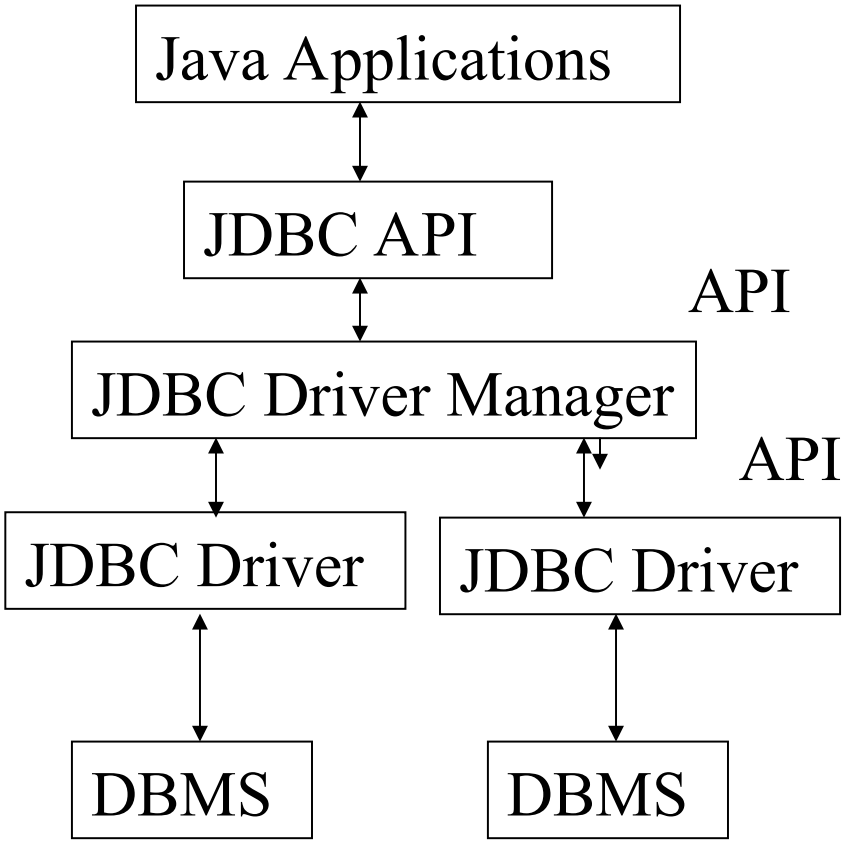
JDBC (Java Database Connectivity)

- ODBC è un'API sviluppata in C che richiede API intermedie per essere utilizzata con altri linguaggi
- Inoltre è difficile da capire: alcuni concetti vengono portati a livello interfaccia ma sarebbe meglio mantenerli nascosti
- JDBC (che non è solo un acronimo ma un trademark della SUN) è stato sviluppato nel 1996 dalla Sun per superare questi problemi
- è compatibile ed estende X/open SQl CLI e ISO SQL/CLI

JDBC

- Rappresenta una API standard per interagire con basi di dati da Java
- cerca di essere il più semplice possibile rimanendo al contempo flessibile
- permette di ottenere una soluzione “pure Java” per l’interazione con DBMS
  - indipendenza dalla piattaforma

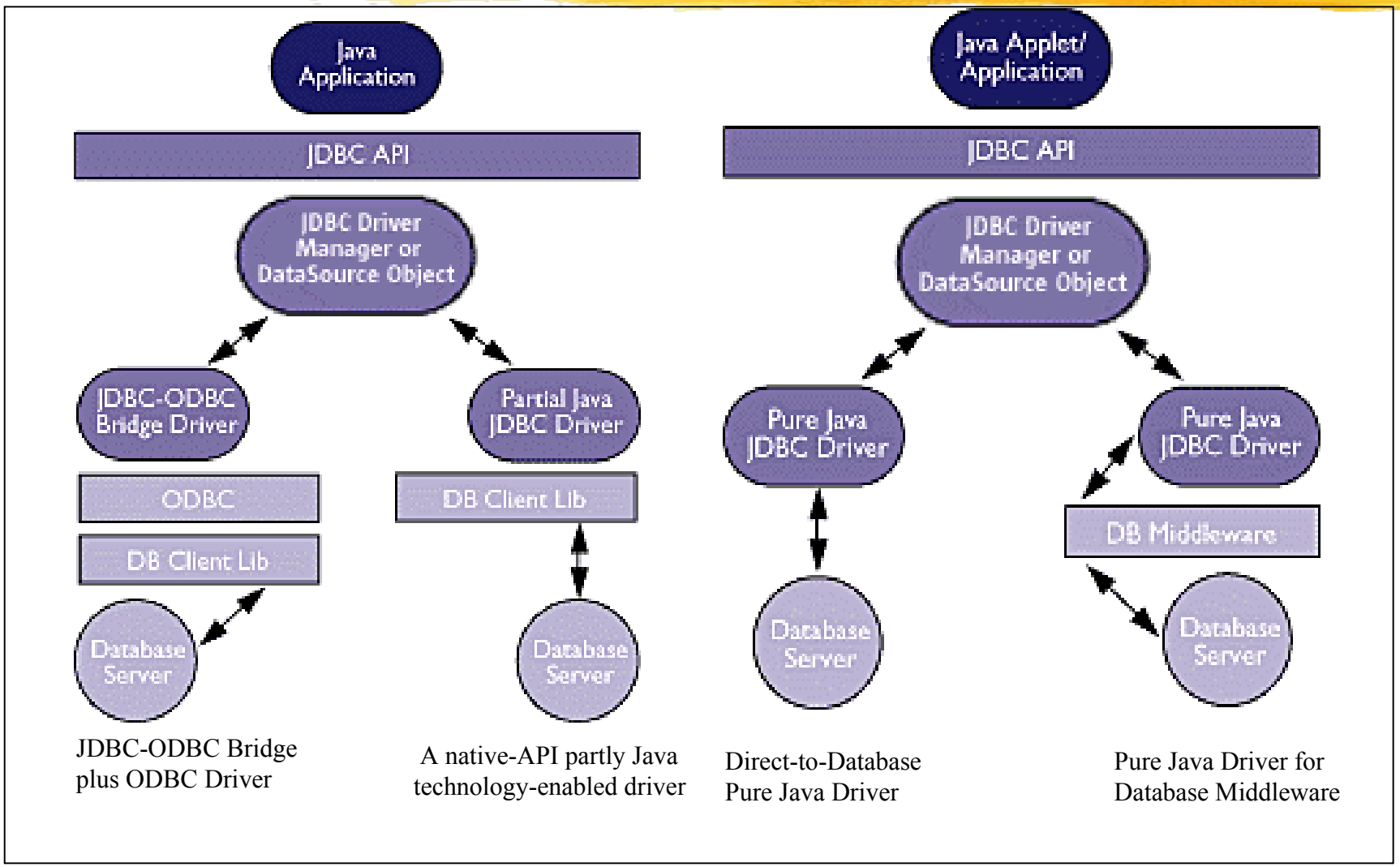
Architettura generale



Tipi di driver

- Esistono quattro tipi di driver:
  - JDBC-ODBC Bridge + ODBC Driver
  - A native-API partly Java technology-enabled driver
  - Pure Java Driver for Database Middleware
  - Direct-to-Database Pure Java Driver.

Tipi di driver



JDBC-ODBC Bridge plus ODBC Driver

A native-API partly Java technology-enabled driver

Direct-to-Database Pure Java Driver

Pure Java Driver for Database Middleware

1 2 4 3

Driver di tipo 1

- Accesso a JDBC tramite un driver ODBC
- Uso di JDBC-ODBC bridge
- il codice ODBC binario e il codice del DBMS client, deve essere caricato su ogni macchina client che usa JDBC-ODBC bridge
- si consiglia di utilizzare questa strategia quando non esistono soluzioni alternative ...
- è necessario installare le librerie sul client perché non sono trasportabili su HTTP
- compromette “write once, run anywhere”

Driver di tipo 2

- Le chiamate a JDBC vengono tradotte in chiamate all'API del DBMS prescelto (es. OCI Oracle)
- Il driver contiene codice Java che chiama metodi scritti in C/C++
- non utilizzabile per applet/servlet
- anche in questo caso, codice binario deve essere caricato sul client
- compromette "write once, run anywhere"

Driver di tipo 3

- Basato completamente su Java (utilizzabile con Applet/servlet)
- converte le chiamate JDBC nel protocollo di una applicazione middleware che traduce quindi la richiesta del client nel protocollo proprietario del DBMS
- l'applicazione middleware può garantire l'accesso a diversi DBMS
- il protocollo middleware dipende dal DBSM sottostante

Driver di tipo 4

- Basato completamente su Java (utilizzabile con Applet/servlet)
- converte le chiamate JDBC nel protocollo di rete usato direttamente dal DBMS
- permette quindi un accesso diretto dalla macchina client alla macchina server
- è la soluzione più pura in termini Java

Caratteristiche e benefici

Benefici:

- Flessibilità
- semplice ed economico
- non è richiesta alcuna configurazione a livello client

Caratteristiche:

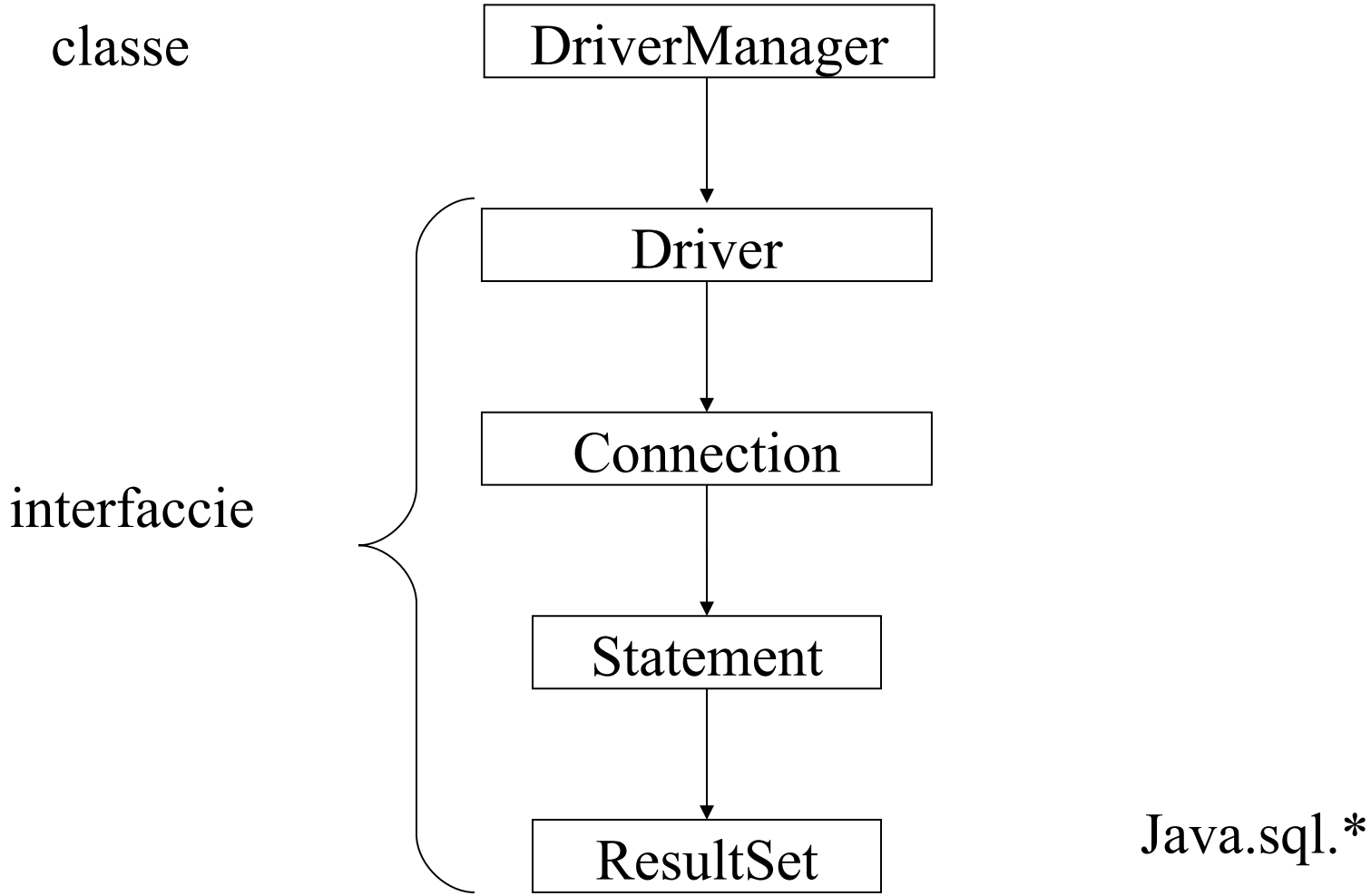
- installazione semplice
- facile accesso ai metadati (non li vediamo)
- possibilità di accedere database tramite URL

Tipi di dato

- In maniera analoga ad ODBC, JDBC definisce un insieme di tipi SQL, che vengono poi mappati in tipi Java
- Gli identificatori sono definiti nella classe `java.sql.Types`

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

Interfaccia JDBC (semplificata - solo le classi che vedremo)



Eccezioni

- La classe `java.sql.SQLException` estende la classe `java.lang.Exception` in modo da fornire informazioni ulteriori in caso di errore di accesso al database, tra cui:
  - la stringa SQLState che rappresenta la codifica dell'errore in base allo standard X/Open
    - `getSQLState()`
  - il codice di errore specifico al DBMS
    - `getErrorCode()`
  - una descrizione dell'errore
    - `getMessage()`

Passo 1: Caricamento driver

- Il driver manager mantiene una lista di classi che implementano l'interfaccia java.sql.Driver
- l'implementazione del Driver deve essere registrata in qualche modo nel DriverManager
- JDBC richiede che, quando una classe Driver viene caricata, crei un'instanza ed effettui anche la registrazione

Passo 1: caricamento driver

- Due modi:
  - `Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");`
  - `newinstance()` si può omettere se viene direttamente effettuata l'inizializzazione
  - `DriverManager.registerDriver(new sun.jdbc.odbc.JdbcOdbcDriver());`
- Il nome della classe da usare viene fornito con la documentazione relativa al driver

Esempio di caricamento driver

```
import java.sql.*;

class JdbcDriver
{
   public static void main (String args []) {
      try{
      Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
    } catch(java.lang.ClassNotFoundException e) {
    System.err.print("ClassNotFoundException: ");
    System.err.println(e.getMessage());
    }
   }
}
```

Passo 2: Connessione

- Per realizzare la connessione vengono utilizzate le seguenti classi ed interfaccie:
  - classe `java.sql.DriverManager`: gestisce la registrazione dei driver
  - interfaccia `java.sql.Driver`: non viene esplicitamente utilizzata a livello applicativo
  - interfaccia `java.sql.Connection`: permette di inviare una serie di richieste SQL al DBMS
- È possibile connettersi a qualunque database, locale e remoto, specificandone l'URL
- in JDBC, l'URL è formato da tre parti:

  jdbc: <subprotocol>: <subname>
  - <subprotocol> identifica il driver o il meccanismo di connessione al database
  - <subname> dipende da subprotocol ed identifica lo specifico database

Connessione

- se si usa JDBC-ODBC driver:
  jdbc:odbc:subname
  dove subname è il nome della DSN ODBC

- Esempio: jdbc:odbc: giuntiDSN

Connessione

- La connessione avviene chiamando il metodo `getConnection` della classe `DriverManager`, che restituisce un oggetto di tipo `Connection`

```
Connection con =
   DriverManager.getConnection(“jdbc:odbc:giuntiDSN”,
   "myLogin", "myPassword");
```

- Se uno dei driver caricati riconosce l’URL fornito dal metodo, il driver stabilisce la connessione

Esempio di connessione

```
import java.sql.*;

class JdbcConn
{
  static String ARS_URL = "jdbc:odbc:giuntiDB";

  public static void main (String args [])
   {
        try{
   Class.forName ("sun.jdbc.odbc.JdbcOdbcDriver");
   } catch(java.lang.ClassNotFoundException e) {
   System.err.print("ClassNotFoundException: ");
   System.err.println(e.getMessage());
   }
  } }
```

Passo 3: creazione ed esecuzione statement

- Per creare ed eseguire uno statement vengono utilizzate le seguenti classi:
  - `java.sql.Connection`: per creare lo statement
  - `java.sql.Statement`: permette di eseguire gli statement
  - `java.sql.ResultSet`: per analizzare i risultati delle query
- un oggetto di tipo Statement viene creato a partire da un oggetto di tipo Connection e permette di inviare comandi SQL al DBMS :

```
Connection con;
…
Statement stmt = con.createStatement();
```

Esecuzione statement

- Come in ODBC, è nenessario distinguere tra statement che rappresentano query e statement di aggiornamento
- Per eseguire una query:

```
stmt.executeQuery(”SELECT * FROM IMPIEGATI");
```

- Per eseguire una operazione di aggiornamento, inclusi gli statement DDL:

```
stmt.executeUpdate(”INSERT INTO IMPIEGATI VALUES
‘AB34’,‘Gianni’, ‘Rossi’,’GT67’,1500");

stmt.executeUpdate("CREATE TABLE PROVA (CAMPO1   NUMBER)”);
```

- il terminatore dello statement (es. ‘;’) viene inserito direttamente dal driver prima di sottomettere lo statement al DBMS per l’esecuzione

Prepared Statement

- Come in ODBC, è possibile preparare gli statement prima dell'esecuzione
- Questo garantisce
  - statement precompilato
  - riduce i tempi di esecuzione
  - usate per statement utilizzati molte volte

```
PreparedStatement queryImp = con.prepareStatement(
                              ”SELECT * FROM
IMPIEGATI");

queryImp.executeQuery():
```

Uso di parametri

- È possibile specificare che la stringa che rappresenta lo statement SQL deve essere completata con parametri (eventualmente letti da input) identificati da ‘?’

```
PreparedStatement queryImp = con.prepareStatement(
                ”SELECT * FROM IMPIEGATI WHERE Nome = ? AND
Dip# = ?");

queryImp.setString(1, ‘Rossi’);
queryImp.executeQuery();
```

- Questo è possibile anche in ODBC (non visto)
- Si noti l’uso di “ e ‘ (devono essere alternati)
- setXXX, dove XXX è il tipo Java del valore del parametro

Passo 4: Elaborazione risultato

- JDBC restituisce i risultati di esecuzione di una query in un result set, come ODBC

```
String query = " SELECT * FROM IMPIEGATI ";
ResultSet rs = stmt.executeQuery(query);
```

- Come in ODBC, il result set è costruito solo per query e non per INSERT, DELETE, UPDATE
- In questo caso viene restituito un intero, che rappresenta il numero di tuple modificate
- Per muoversi su un result set si deve utilizzare il metodo next:

```
while (rs.next()) {
    String s = rs.getString(”Cognome");
    float n = rs.getFloat(”Stipendio");
    System.out.println(s + "   " + n);
}
```

Metodi per accedere i valori associati agli attributi

- Il metodo `getXXX`, di un `ResultSet`, permette di recuperare il valore associato ad un certo attributo, puntato correntemente dal cursore. XXX è il tipo Java nel quale il valore deve essere convertito

```
String s = rs.getString(”Cognome");
```

- Gli attributi possono anche essere acceduti tramite la notazione posizionali:

```
String s = rs.getString(2);
int n = rs.getInt(5);
```

- Usare `getInt` per valori numerici, `getString` per char, varchar

Metodi per accedere i valori associati agli attributi

- Il metodo next() permette di spostarsi nel result set (cursore):

```
while (rs.next()) {/* get current row */}
```

- inizialmente il cursore è posizionato prima della prima tupla
- il metodo diventa falso quando non ci sono più tuple

Esempio esecuzione diretta

- Vedi JdbcQueryDirect.java

Esempio prepared statement

- Vedi JdbcQueryPrep.java

Passo 5: Transazioni

- Per default, JDBC esegue il commit di ogni statement SQL inviato al DBMS (auto-commit)
- è possibile disattivare l'autocommit tramite il metodo `setAutoCommit`

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

- a questo punto ogni connessione diventa una transazione indipendente per la quale si può richiedere il commit o l'abort tramite:

```
con.commit();
con.rollback();
```

Esempio transazioni

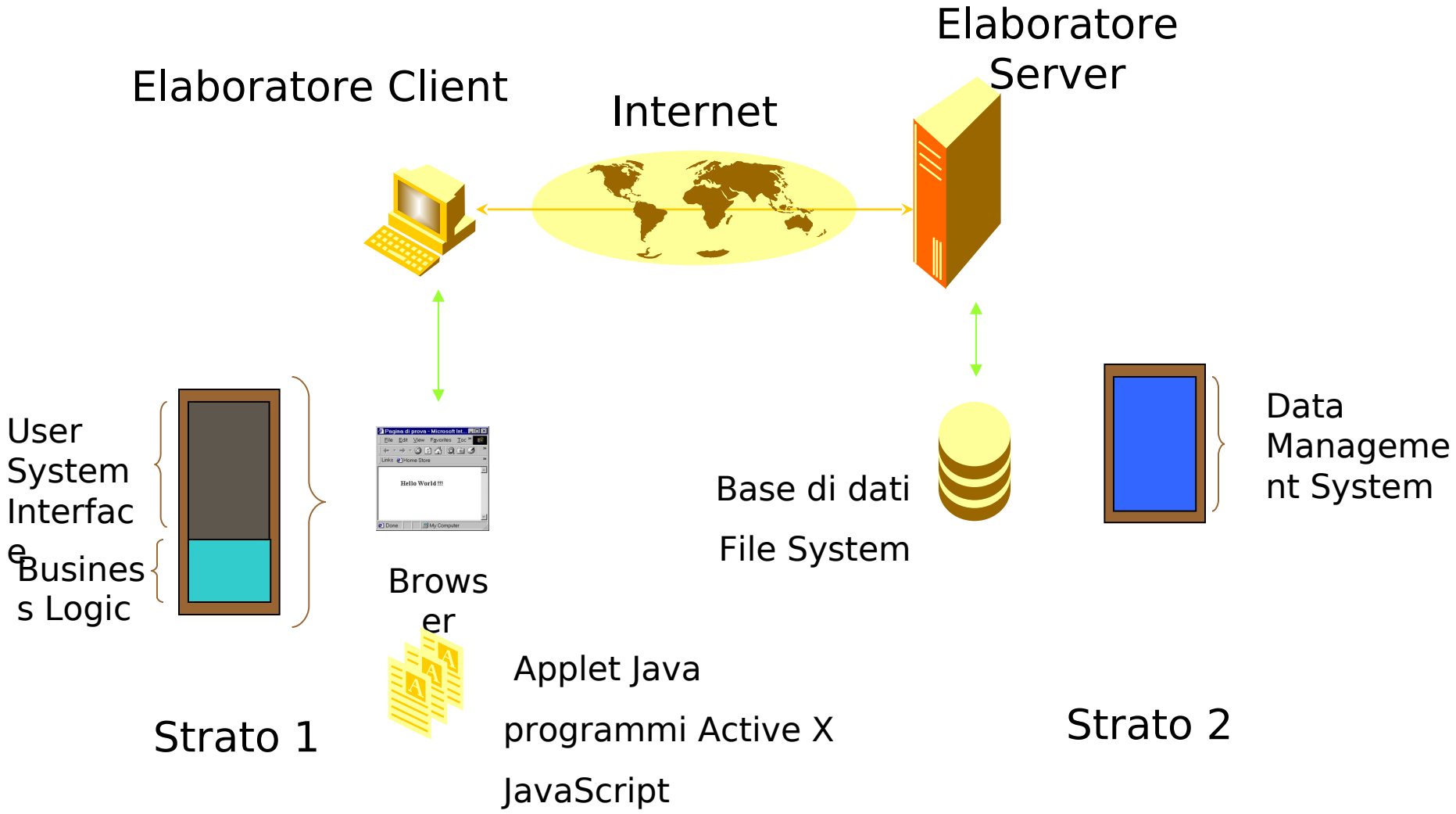
- Vedi JdbcTrans.java

Passo 6: Disconnessione (chiusura)

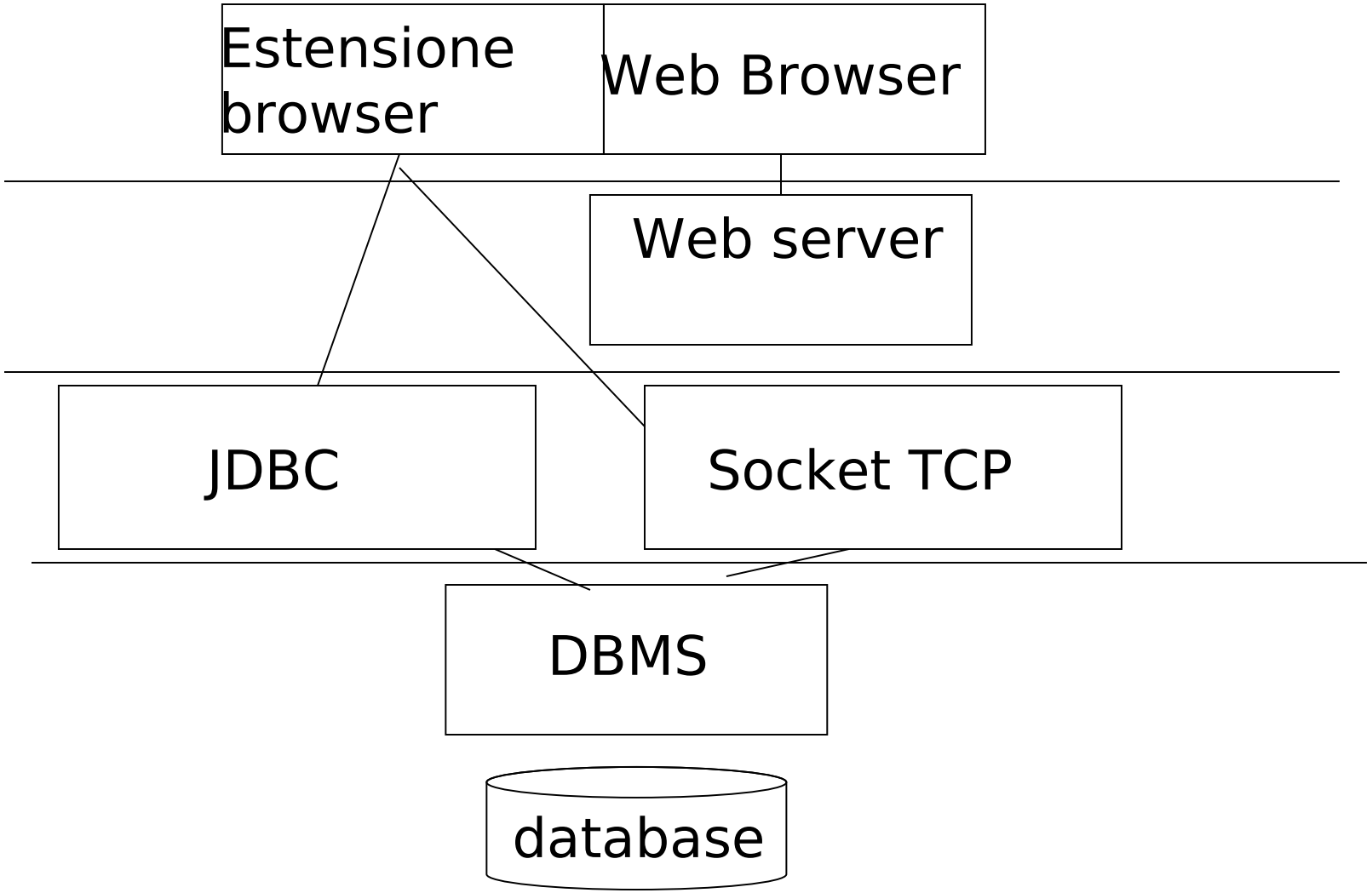
- Per risparmiare risorse, può essere utile chiudere gli oggetti di classe `Connection`, `Statement`, `ResultSet` quando non vengono più utilizzati
- metodo `close()`
- la chiusura di un oggetto di tipo `Connection` chiude tutti gli `Statement` associati mentre la chiusura di uno `Statement` chiude `ResultSet` associati

Soluzioni lato client

Soluzioni lato client: sistema classico a 2-strati



Livelli


Estensione browser
Web Browser
Web server
JDBC
Socket TCP
DBMS
database

Estensione al browser

- Le tecnologie lato client si possono tutte vedere come approcci che estendono le funzionalità del browser:
  - tramite applicazioni Java
    - Applet
  - tramite script
    - JavaScript
    - VBScript
- In entrambi i casi, è possibile comunicare con il DBMS tramite protocolli di comunicazione (JDBC da Applet)

Vantaggi e svantaggi

- Vantaggi
  - permette in modo semplice di manipolare i dati inseriti dall'utente
  - facile da usare
  - flessibile
- Svantaggi
  - carica di lavoro il client
  - problematiche di sicurezza
  - tipicamente usata per gestire l'interfaccia e non per gestire la logica applicativa
  - meno usate e flessibili rispetto alla tecnologia lato server

Applet e DBMS

- In quanto applicazione Java, l'applet può comunicare con un DBMS tramite JDBC
  - Unico problema: il DBMS deve risiedere sull'host dal quale proviene l'applet

- L'applet può anche invocare servlet o programmi CGI, che risiedono sull'host dal quale proviene l'applet, tramite socket
  - questi programmi possono a loro volta connettersi ad un DBMS remoto
  - applet: gestione interfaccia

    servlet: gestione logica applicativa
  - soluzione che permette di distribuire il carico di lavoro tra client e server

Esercitazione proposta

- Estendere l'applet sviluppata durante le lezioni di Java per:
  - leggere tutti (o alcuni) dati di un corso (sicuramente tutti quelli che non possono essere nulli)
  - inserire I dati del corso nella tabella corrispondente
  - nella finestra di dialogo visualizzare:
    - "Inserimento effettuato", se l'inserimento è andato a buon fine
    - "Errore in inserimento", se si verifica qualche errore
- Estendere l'applet sviluppata durante le lezioni di Java per:
  - chiedere all'utente una certa materia
  - visualizzare tutti I corsi relativi a quella materia

Suggerimento

- Includere java.sql.*
- introdurre nuovi textfield e label, una per ogni informazione da inserire
- associare al bottone submit la comunicazione con il DBMS:
  - connessione
  - costruzione statement SQL (conviene farlo preparato)
  - esecuzione statement
  - analisi risultato
  - impostazione finestra di dialogo (eventuale)

Osservazione

- Se viene effettuata la connessione o la preparazione degli statement nel codice associato al tasto SUBMIT, tali operazioni vengono eseguite ogni volta che si preme il bottone
- se l'utente usa il tasto più volte, vengono generate inutilmente tante connessioni e tanti statement preparati
- Soluzione:
  - associare queste operazioni a metodi particolari

Ciclo di vita

- Un applet si definisce creando una sottoclasse Java della classe predefinita `Applet`

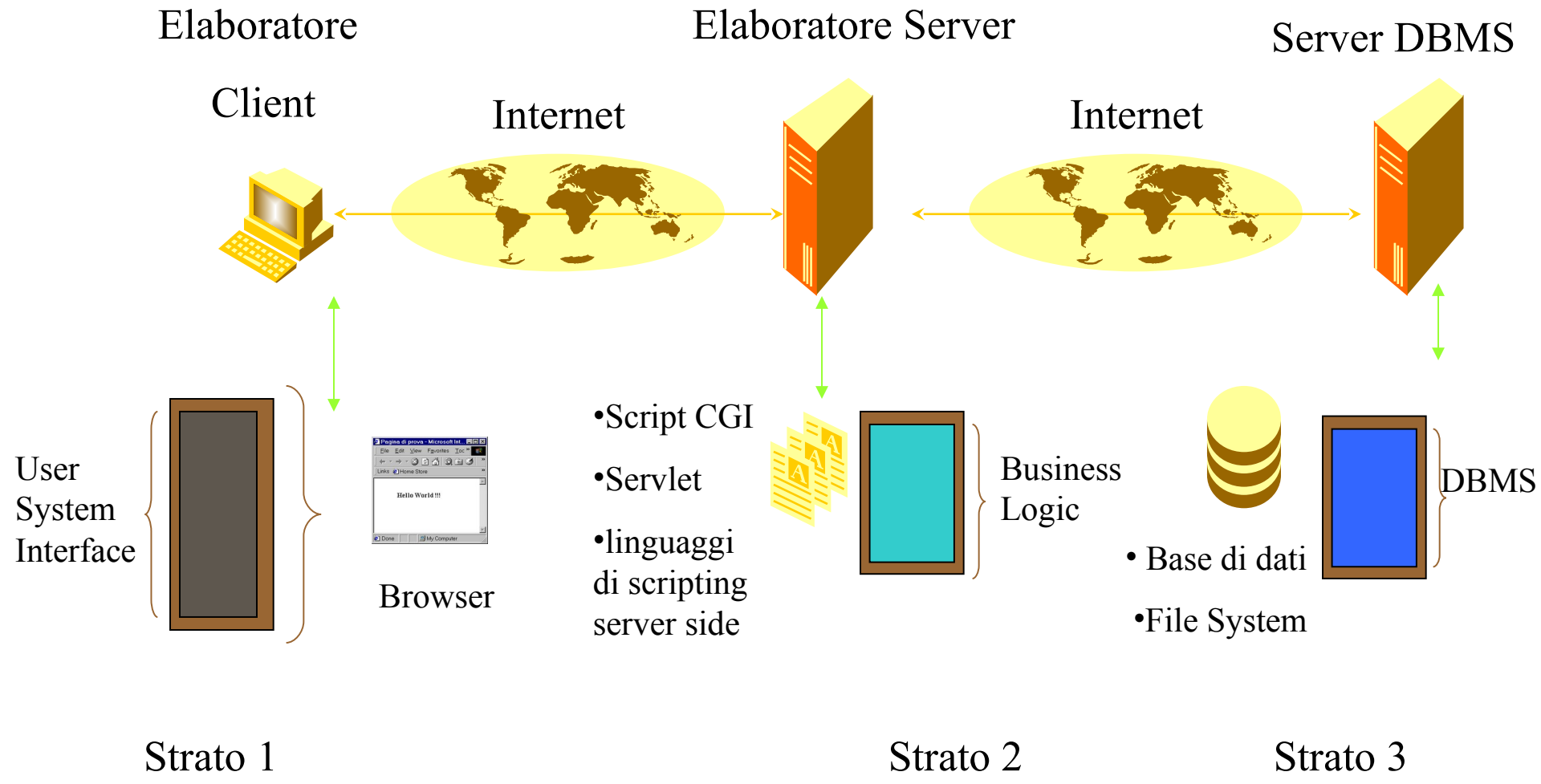
- In genere un applet ridefinisce i seguenti metodi:
  - `init()`: inizializza l'applet ogni volta che viene caricata
  - `start()`: codice eseguito al momento del caricamento dell'applet e ogni volta che si visita la pagina ad essa collegata
  - `stop()`: termina l'esecuzione dell'applet, quando si lascia la pagina collegata all'applet
  - `destroy()`: pulizia prima di scaricare l'applet

Quindi …

- L’applet potrebbe essere organizzata nel modo seguente:
  - init()
    - connessione + prepared statement
  - destroy()
    - disconnessione
- oppure
  - start()
    - connessione + prepared statement
  - stop()
    - disconnessione

Soluzioni lato server

Soluzioni lato server: sistema classico a 3-strati



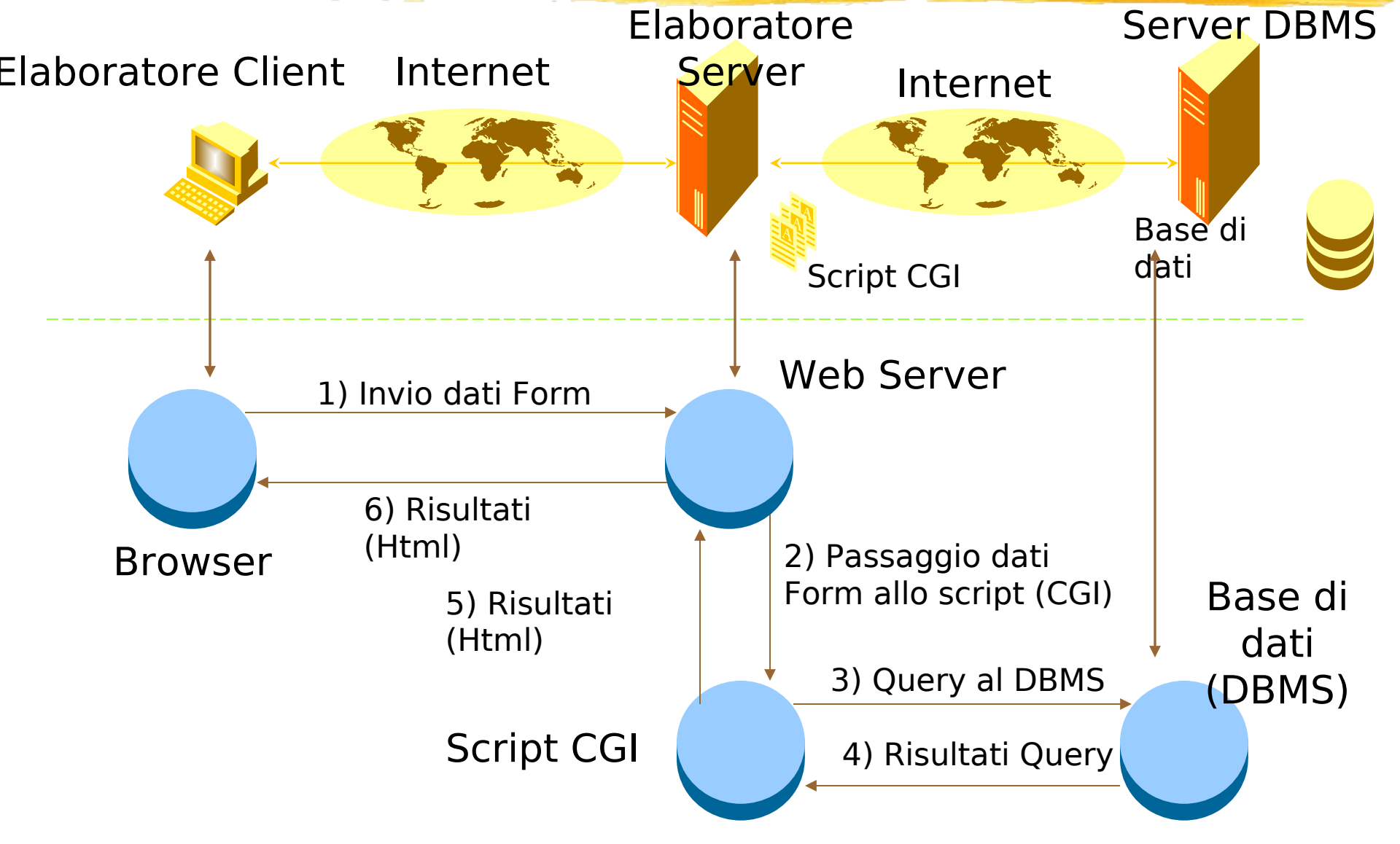
Approcci

- Programmi compilati
  - CGI
  - Java Servlet
- Linguaggi di scripting
  - ASP
  - PHP
  - JSP

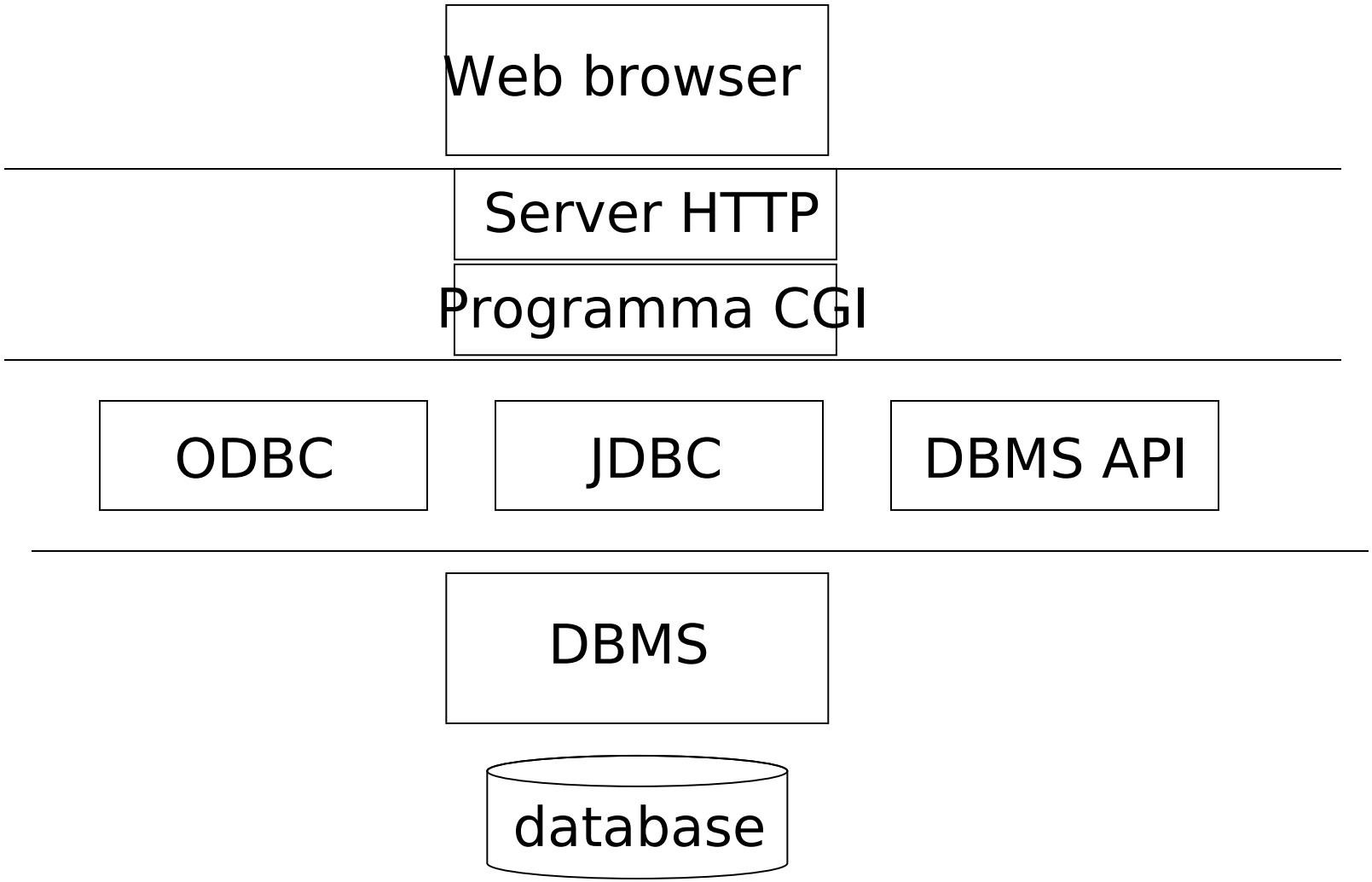
CGI

- Protocollo che consente al Web Server di eseguire applicazioni esterne in grado di creare pagine dinamicamente
- Definisce un insieme di variabili di ambiente utili alla applicazione (ad esempio, parametri inviati dal client tramite form)
- L'applicazione può essere scritta in un qualsiasi linguaggio (C, Java, linguaggi per script) o usare i comandi di una shell
- Gli eseguibili devono essere inseriti in una directory gestita dal Web Administrator (/cgi-bin)
- Il Web Server deve permettere la gestione delle variabili di ambiente

CGI: interazione con base di dati



CGI: livelli


Web browser
Server HTTP
Programma CGI
ODBC
JDBC
DBMS API
DBMS
database

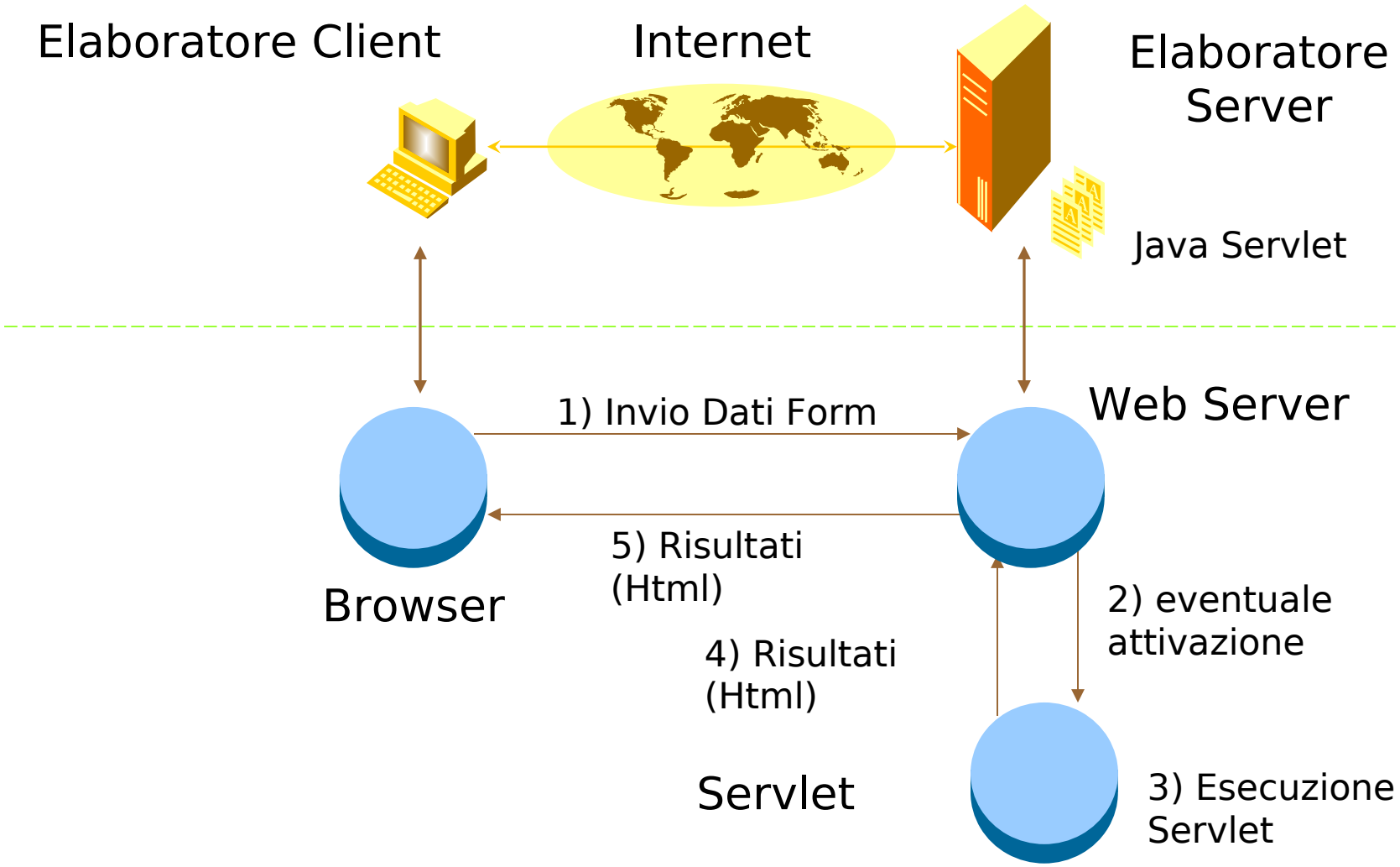
Java servlet

- Java Servlet si riferisce ad un particolare package: `java.servlet`
- con Java Servlet intendiamo un'applicazione Java in esecuzione su una JVM residente sul server
- fornisce una serie di servizi in base al paradigma "richiesta-risposta":
  - client invoca la servlet nel contesto di una FORM HTML (come CGI)
  - la servlet esegue le operazioni richieste (come CGI)
  - la servlet ridirige l'output al client in forma di pagina HTML (come CGI)
- Differenza rispetto a CGI: la servlet corrisponde ad un processo che viene caricato solo una volta anche se viene utilizzato per eseguire più operazioni distinte

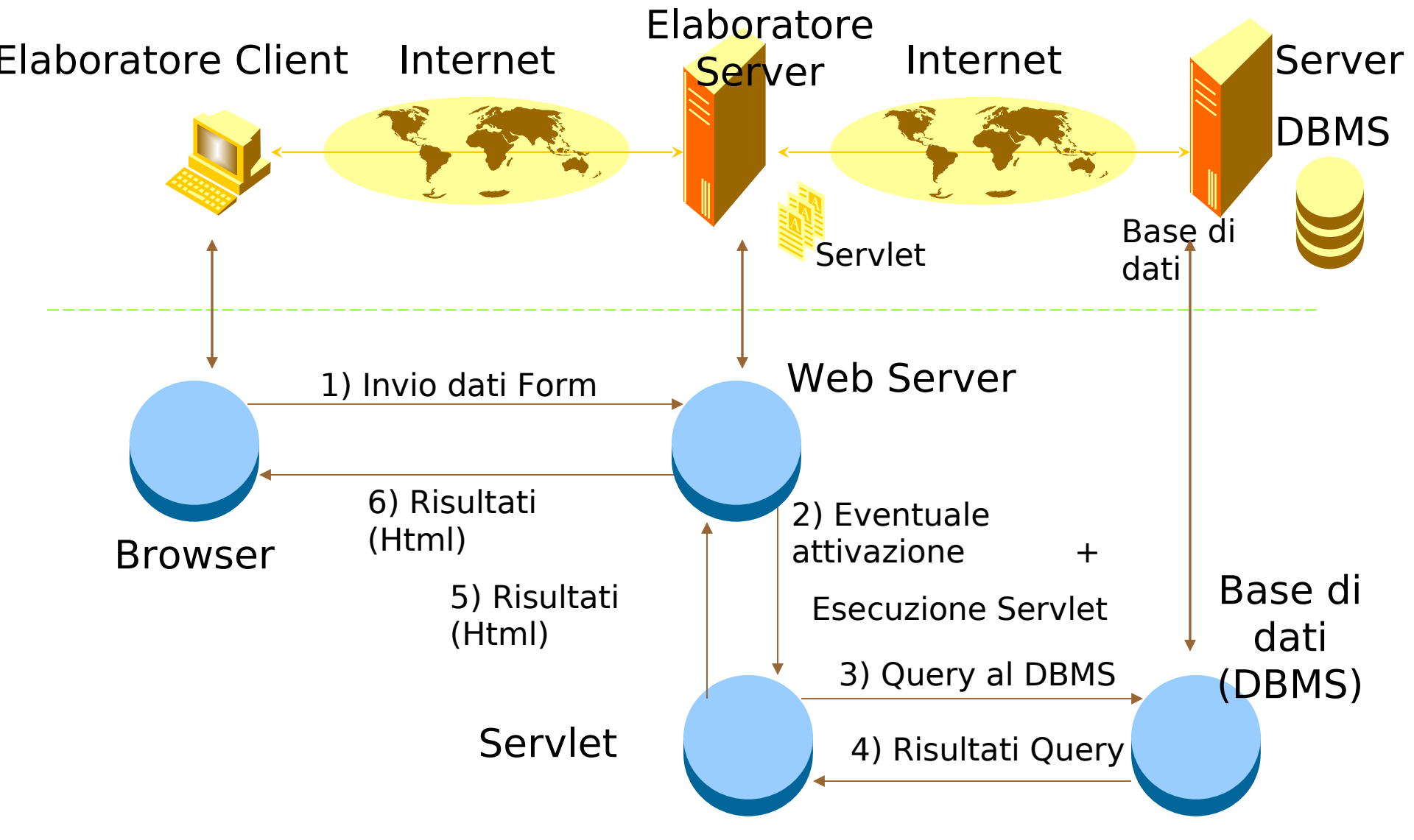
Java Servlet

- Altri vantaggi:
  - indipendenza dalla piattaforma, grazie a Java
  - sicurezza gestita mediante security manager della JVM
  - gestione degli errori con il meccanismo delle eccezioni Java
  - disponibilità: distribuzione gratuita di Java Servlet Development Kit, contenente la libreria Java Servlet
- Web Server deve essere esteso con un servlet engine

Servlet: funzionamento

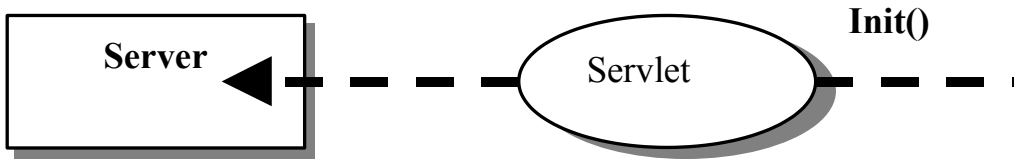


Servlet: interazione con base di dati

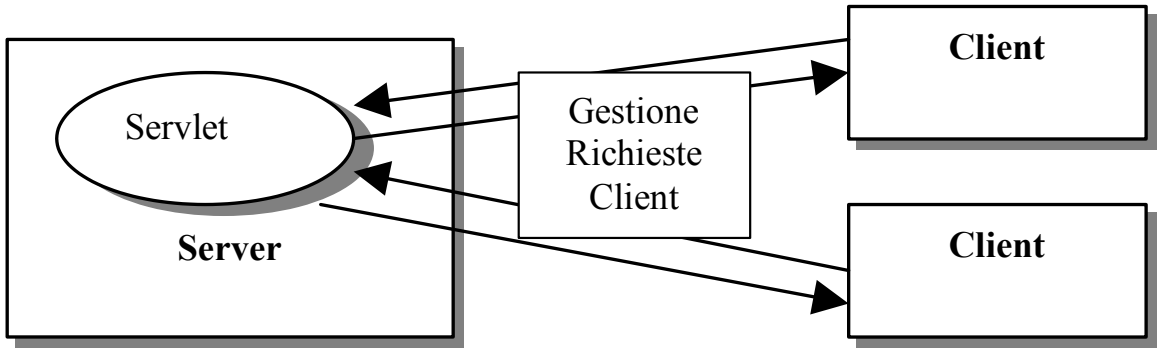


Ciclo di vita

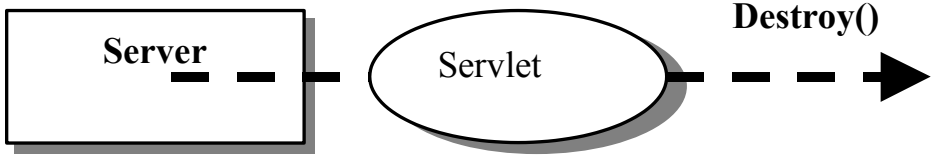
Caricamento e inizializzazione



Gestione richieste



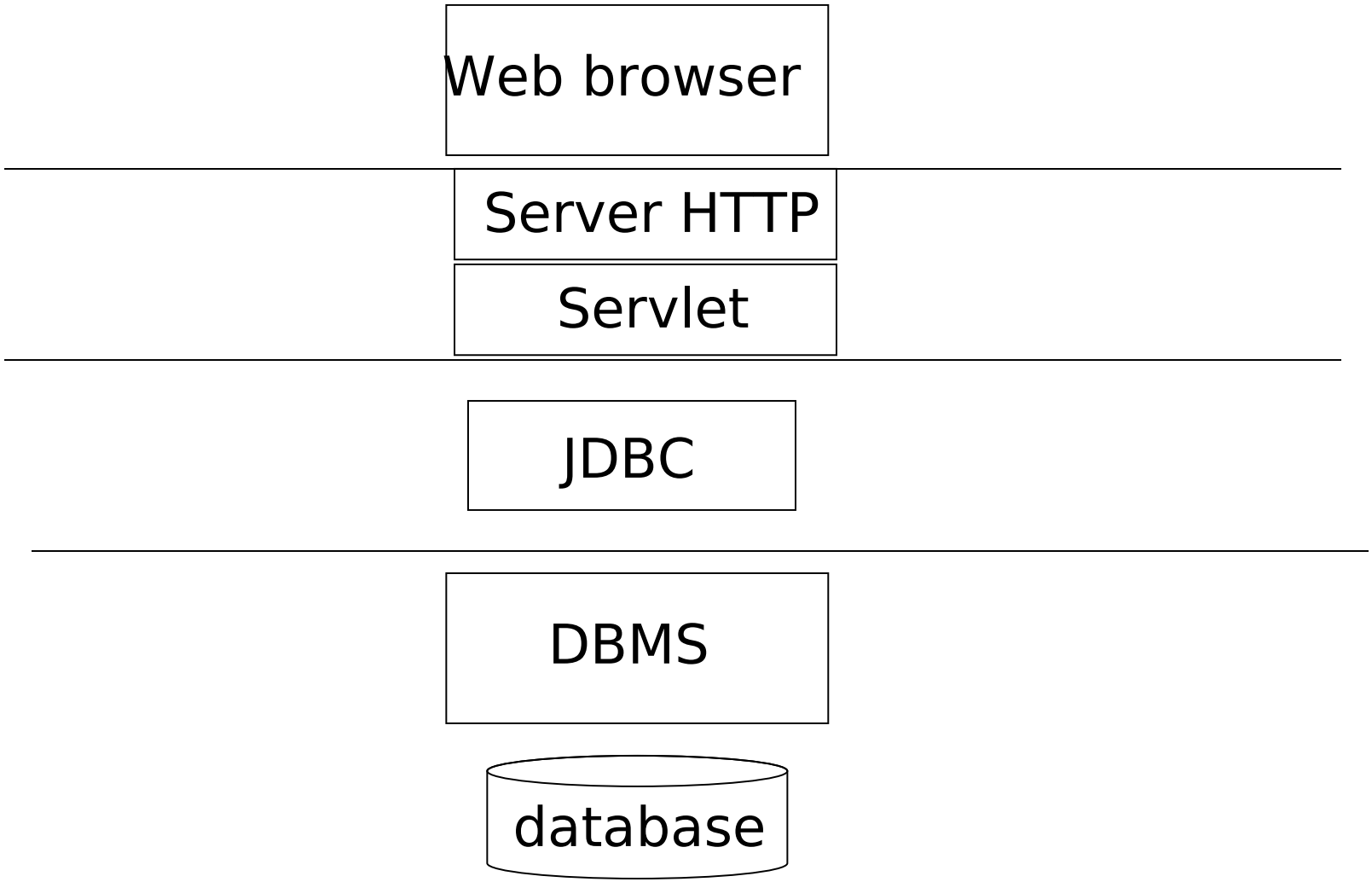
Rimozione servlet



Interazione con il DBMS

- Init:
  - connessione
- DoGET, DoPost:
  - interazione
- Destroy:
  - sconnessione

Servlet: livelli

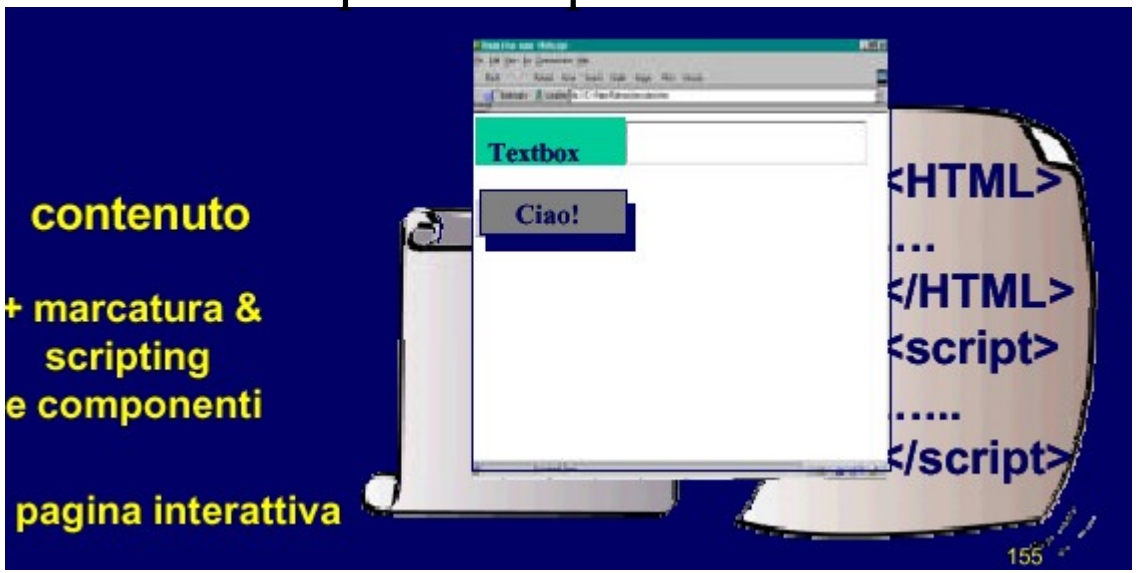


I linguaggi di scripting server-side

- CGI, Java Servlet sono architetture general-purpose per eseguire applicazioni via HTTP e sono complesse da programmare
- la presentazione è prodotta dal codice stesso:
  - i programmatori devono scrivere del codice per produrre HTML
    - non sono tenuti a conoscerlo
  - gli esperti di HTML devono conoscere il linguaggio utilizzato (es. Java)
    - non sono tenuti a conoscerlo
- Attualmente in alternativa a questi approcci sono disponibili i cosiddetti linguaggi di scripting lato server (Server-Side) che permettono di generare pagine dinamiche
- Una pagina dinamica è costituita da HTML e da codice compreso tra tag speciali (<% e %>)
  - chiara separazione tra contenuto e presentazione
- Esistono molti linguaggi di scripting come : JSP,ASP, PHP
- Il Web Server deve essere esteso in modo da riconoscere le pagine dinamiche e possa inviarle al motore di scripting (Script Engine) che le interpreta e le esegue restituendo il risultato dell'elaborazione

Differenze tra scripting client/server side

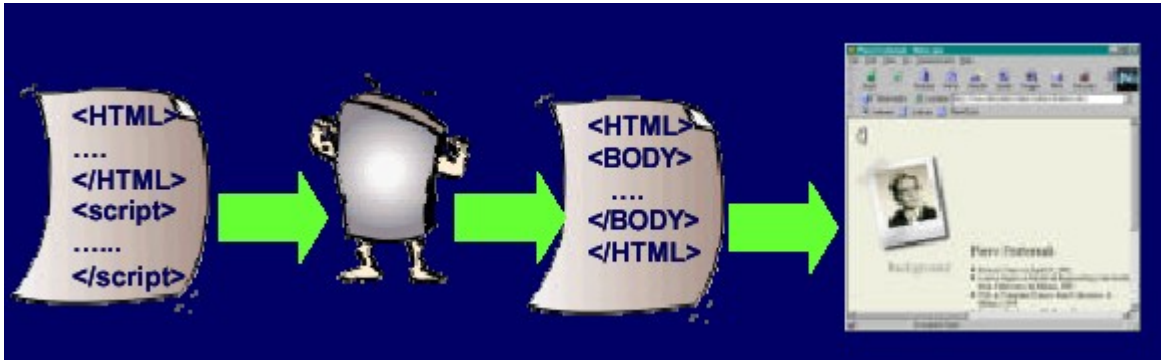
Il codice script è interpretato dal browser



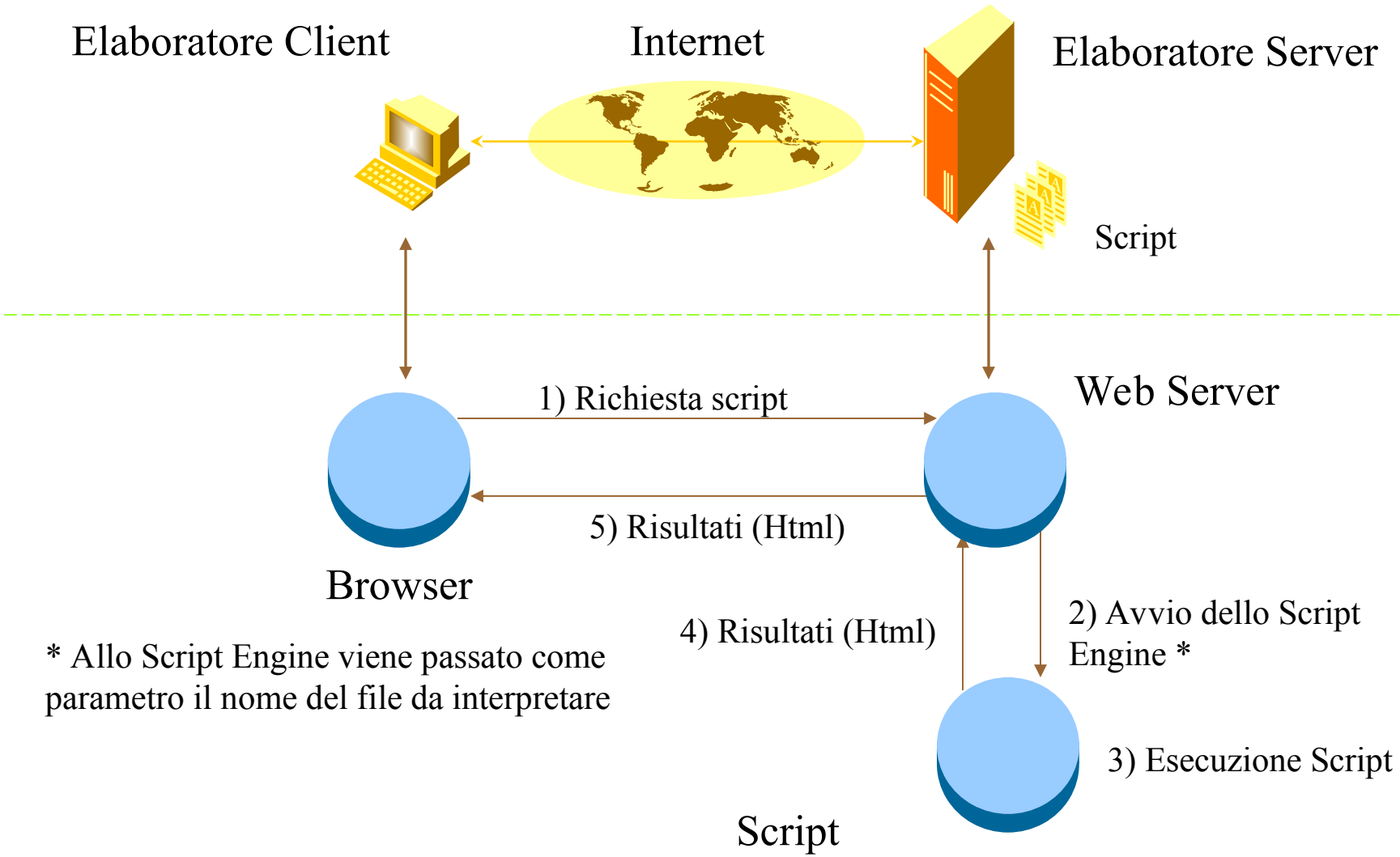
CLIENT

Il codice script è interpretato dal server
il browser riceve HTML puro

SERVER

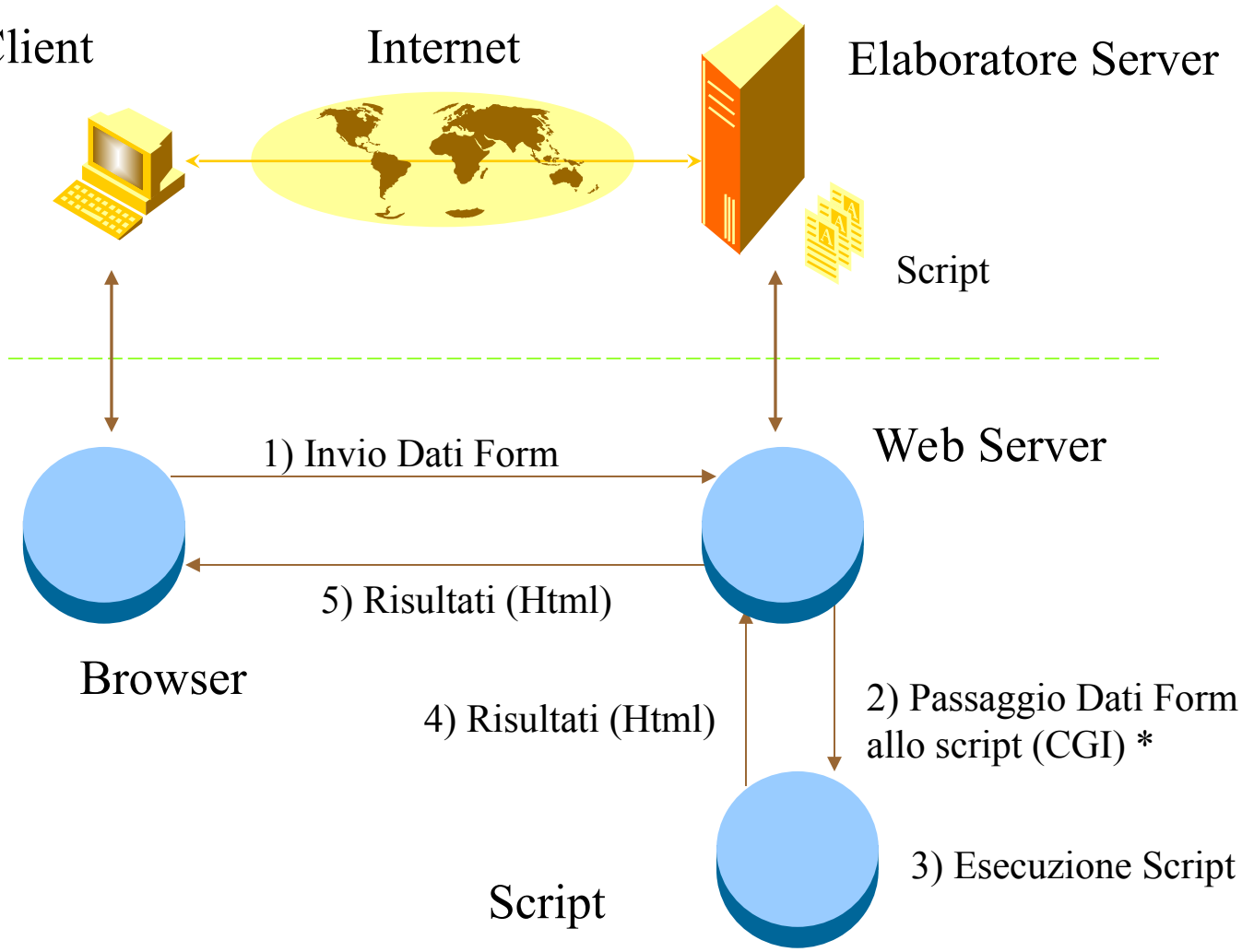


Il funzionamento di base



* Allo Script Engine viene passato come parametro il nome del file da interpretare

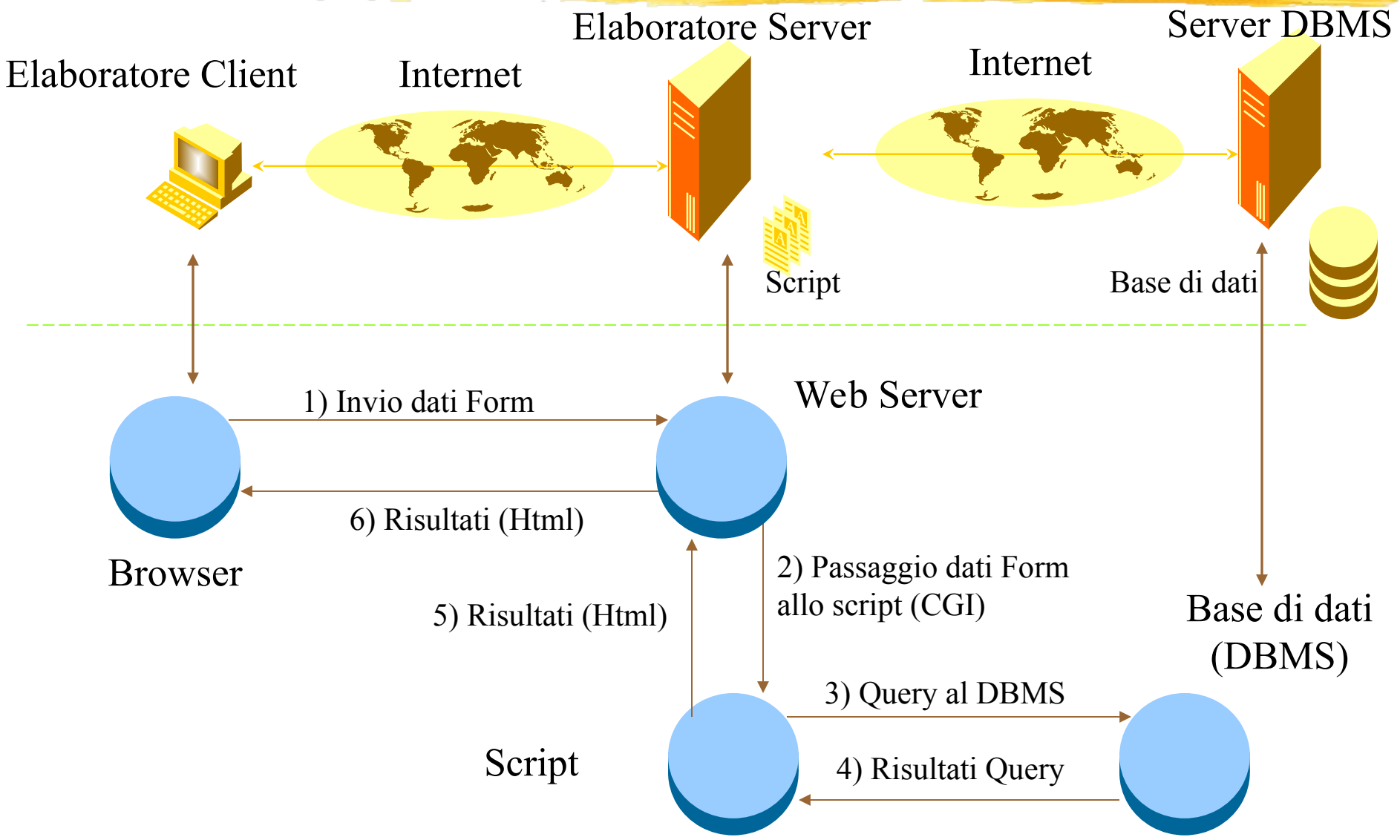
Il funzionamento con le form



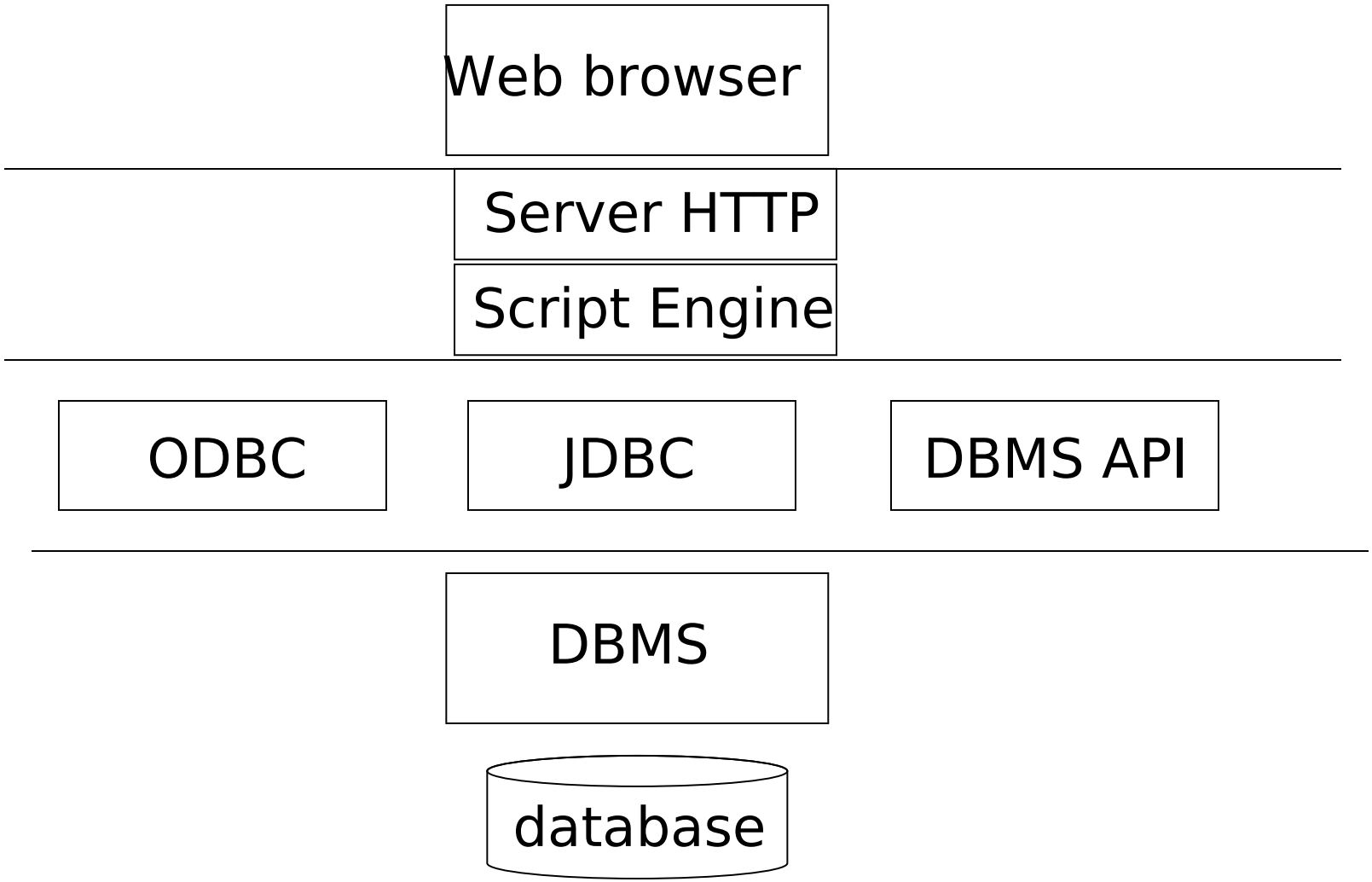
* 1) Allo Script Engine viene passato come parametro il nome del file da interpretare

2) Lo standard CGI fornisce i dati inviati tramite la form allo script tipicamente mediante variabili d'ambiente

Interazione con il DBMS



Livelli


Web browser
Server HTTP
Script Engine
ODBC
JDBC
DBMS API
DBMS
database

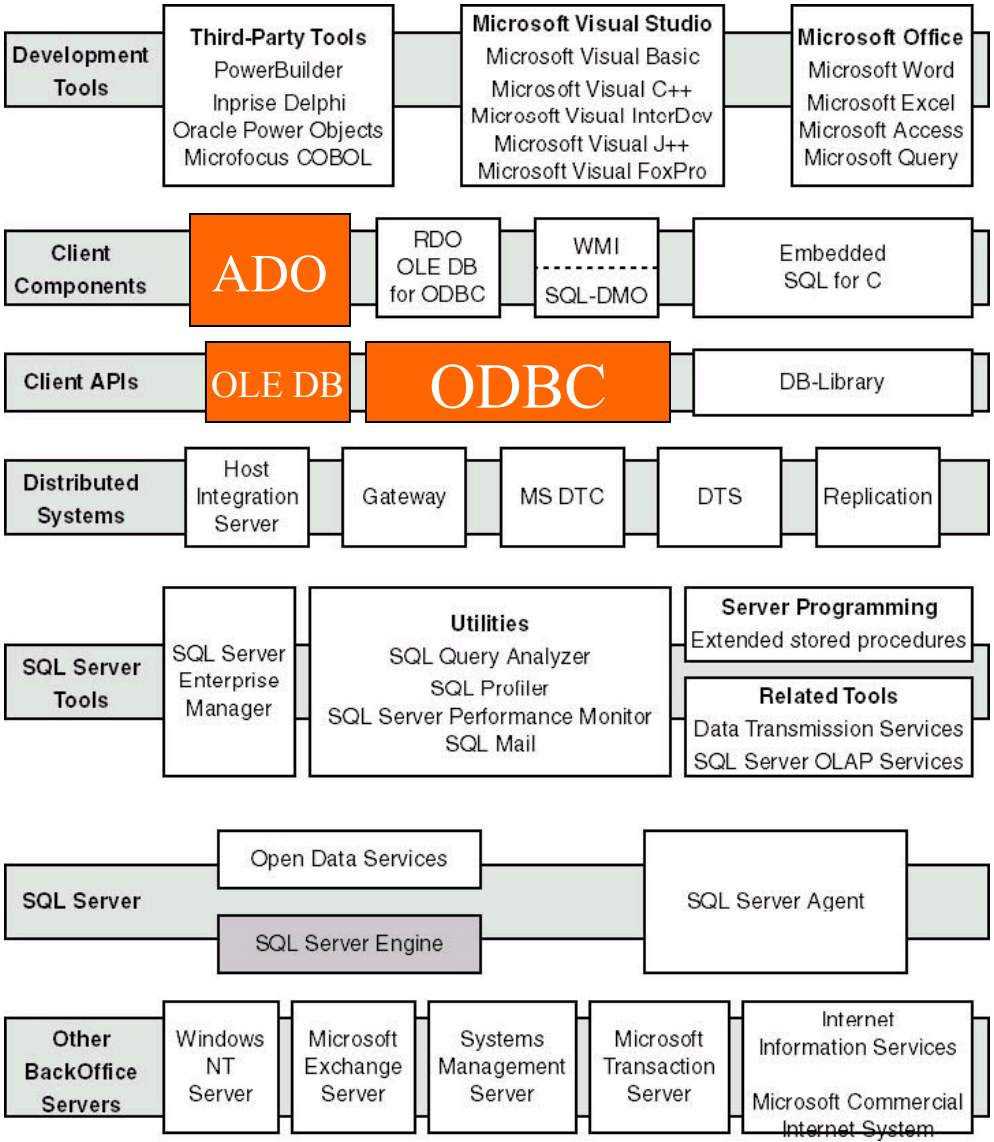
ASP

- ASP (Active Server Pages) è un linguaggio di scripting Server-Side
- Soluzione di Microsoft per creare pagine Web dinamiche
- Sito ufficiale: WWW.MICROSOFT.COM
- Caratteristiche:
  - non è portabile
  - indipendenza dal tipo di browser utilizzato
    - il browser vede solo pagine HTML
      - l'utente non ha bisogno di programmi proprietari o estensioni del browser
  - sintassi semplice (insieme limitato di statement Visual Basic)
  - nasconde la presenza di script agli utenti
  - supporto ODBC per i maggiori DBMS
  - interpretato

Le pagine in ASP

- Le Active Server Pages (ASP) sono inserite nei file con estensione .asp.
- Un file .asp è un file testuale che contiene i seguenti elementi:
  - Tag HTML
  - Codice di script racchiuso tra Tag speciali <% ... %>
  - il linguaggio di scripting non è fissato
    - noi useremo Jscript, versione Microsoft di JavaScript
- approccio object-oriented

ASP in SQL Server



ASP e DBMS

- La comunicazione con un DBMS avviene tramite ADO, un'interfaccia object-oriented, molto simile come strutturazione a JDBC
- Oggetti principali supportati da ADO:
  - *Connection*: rappresenta una singola connessione al DBMS
  - *Command*: comando da inviare al DBMS
    - può anche contenere parametri
  - *Recordset*: insieme ottenuto come risultato di un'interrogazione
  - *Record*: elemento di un recordset
  - *Error*: dettaglio errori DBMS, specifici ad una connessione

Flusso tipico

- Connessione alla base di dati
- specifica comando
- esecuzione comando (eventualmente settando parametri)
- analisi risultato
- generazione output da inviare al client

Connessione

- Due possibilità
  - connessione a OLE DB non la vediamo
  - connessione a ODBC

```
<% var cnn;
  //Create a connection object.
  cnn = Server.CreateObject("ADODB.Connection");
  //Open a connection using the ODBC connection string.
  cnn.Open (”DSN=giuntiDSN;UID=giunti;PWD=corsoDBMS”);
%>
```

```
<% var cnn;
  //Create a connection object.
  cnn = Server.CreateObject("ADODB.Connection");
  //Open a connection using the ODBC  connection string.
  cnn.Open(”Driver={SQL Server};Server=PAPERINO;UID=
     giunti;PWD=corsoDBMS;database=giuntiDB”);
%>
```

Esecuzione statement

- Il modo più semplice di eseguire uno statement utilizza la connessione

```
<% var cnn,strSQL;
  //Create a connection object.
  cnn = Server.CreateObject("ADODB.Connection");
  //Open a connection using the ODBC  connection string.
  cnn.Open ("Driver={SQL Server};Server=PAPERINO;UID=
     giunti;PWD=corsoDBMS;database=giuntiDB”);
//Define SQL statement.
  strSQL = "INSERT INTO Impiegati(Nome,Dip#,Mansione,Data_a,Stipendio)"+
     " VALUES ('catania',1,impiegato,12/3/89,1000)"   ;
  //Use the Execute method to issue a SQL query to database.
  cnn.Execute(strSQL);
%>
```

Esecuzione query

- Se viene eseguita una query, il risultato può essere analizzato utilizzando un oggetto di tipo RecordSet
- la specifica dello statement può avvenire anche a livello recordset
- un RecordSet rappresenta un cursore sul risultato di una query
- è possibile definire vari tipi di cursori
- come in JDBC, proprietà a metodi per:
  - stabilire se abbiamo analizzato tutti I record:
    - proprietà EOF
  - muoversi sulla tupla successiva
    - metodo MoveNext

Esempio

```
<%
    ...
  //Instantiate a Recordset object.
  rstImpiegati = Server.CreateObject("ADODB.Recordset");

  //Open a recordset using the Open method
  //and use the connection established by the Connection object.
  strSQL = "SELECT Nome from Impiegati ”;
  rstImpiegati.Open (strSQL, cnn);

  //Cycle through record set and display the results
  //and increment record position with MoveNext method.
    while (! rstImpiegati.EOF)    {
    Response.Write(rstImpiegati("Nome") + "<BR>”);
                rstImpiegati.MoveNext;
    }
%>
```

Esecuzione statement

- Un modo più flessibile di eseguire uno statement prevede la creazione di un un oggetto *Command*
- Vantaggio:
  - si possono settare le proprietà dello statement, ad esempio il tipo
    - maggiore efficienza in esecuzione
  - si possono costruire statement preparati con o senza parametri
- per settare le proprietà è utile utilizzare delle costanti di sistema
  - il file che le contiene deve essere incluso
  - <!-- #include File = ..\adojavas.inc -->

Esempio

```
<%
    ...
  //create a command object
   com = Server.CreateObject("ADODB.Command");
  //Define SQL statement.
  strSQL = "INSERT INTO Impiegati(Nome,Dip#,Mansione,Data_a,Stipendio)"+
    " VALUES ('catania',1,impiegato,12/3/89,1000)" ;
//set command properties
   com.ActiveConnection = cnn;
   com.CommandType = adCmdText;   //costante che indica che il comando è
    uno statement SQL
   com.CommandText = strSQL;
  //Use the Execute method to issue a SQL query to database.
   com.Execute() ;
%>
```

Possibili opzioni

Costante	Tipo di Query
....	
adCmdStoreProc	Stored procedure
adCmdText	SQL statement
adCmdUnknown	Type of the command is not known (default)
adCmdUnspecified	Unspecified command type argument
....	

Statement con parametri (preparate)

- Per preparare uno statement, è necessario porre a True la proprietà *Prepared* associata all'oggetto comando
- per specificare I parametri:
  - un oggetto di tipo comando è associato ad una collezione Parameters che rappresenta I parametri
  - è necessario aggiungere un elemento alla collezione per ogni parametro
  - è necessario specificare il nome, il tipo, la direzione (IN/OUT) e la dimensione del parametro
  - se non si specifica la direzione, il parametro è considerato un parametro di input
    - quindi negli esempi non specifichiamo la direzione

Esempio

```
<%

...

//Instantiate Command object; use ActiveConnection property to
    attach //connection to Command object.

cmn= Server.CreateObject("ADODB.Command") ;

cmn.ActiveConnection = cnn ;

//Define SQL query.

cmn.CommandText = ”DELETE  FROM Impiegati WHERE Nome = ? ”;

//Save a prepared (or pre-compiled) version of the query specified in
    CommandText 'property before a Command object's first execution.

cmn.Prepared = True;
```

Esempio (continua)

```
//Define query parameter configuration information.
cmn.Parameters.Append (
    cmn.CreateParameter(”nomeP",adVarChar,,30 ));
//Define and execute statement
cmn(”nomeP") = ”Rossi”;
cmn.Execute();
//Define and execute statement again
cmn(”nomeP") = ”Verdi”;
cmn.Execute();


. . .
%>
```

Osservazione

- Sono possibili tre metodi di esecuzione
  - sull'oggetto connection
    - specifichiamo almeno la stringa corrispondente allo statement da eseguire
  - sull'oggetto RecordSet
    - specifichiamo la stringa corrispondente allo statement da eseguire e la connessione
    - possiamo analizzare I risultati di una query e muoverci con il cursore
  - sull'oggetto Command
    - specifichiamo la stringa corrispondente allo statement da eseguire,la connessione, ed eventualmente il tipo di statement
    - possiamo specificare se lo statement è preparato o meno, ed eventualmente associare parametri

Form HTML, ASP e ADO

- É possibile utilizzare I valori passati attraverso una form per settare I parametri di uno statement SQL
- Tutto come prima, cambia solo l'assegnazione dei valori ai parametri

```
//Define query parameter configuration information.
cmn.Parameters.Append (
    cmn.CreateParameter(”nomeP",adVarChar, ,30 ));
//Define statement if the type of the request is POST
cmn(”nomeP") = Request.Form(”Nome") ();
//Define statement if the type of the request is GET
cmn(”nomeP") = Request.QueryString(”Nome")() ;
cmn.Execute();
```

Errori

- Un oggetto Error contiene I dettagli relativi ad un errore che si è verificato nel fornitore dati
- possono essere di varia natura:
  - lo statement SQL inviato non è corretto
  - alcune funzionalità non sono supportate
- gli errori vengono associati tramite una collezione ad una connessione

Esempio

```
//Use the Execute method to issue a SQL statement to database.
  cnn.Execute(strSQL);

//check if some errors have been generated
if (cnn.Errors.Count >0)
    {
    Response.Clear();
    for(intcount = 0;intcount<cnn.Errors.Count; intcount++)
      {
        Response.Write("Error Number:" + cnn.Errors(intcount).Number();
        Response.Write("Error Description:" +
        cnn.Errors(intcount).Description();

    }
```

Gestione connessioni

- Una connessione può essere chiusa con il metodo close dell'oggetto Connection

  con.close

- se la connessione deve essere usata diverse volte nel contesto della stessa applicazione, potrebbe essere utile definire la connessione a livello applicazione
- se la connessione deve essere usata diverse volte dallo stesso utente, potrebbe essere utile definire la connessione a livello sessione
- Problema: in questo modo le connessioni potrebbero rimanere inattive per lungo tempo

Gestione connessioni

- Soluzione:
  - a livello applicazione o sessione si definiscono solo le stringhe di connessione
  - si lasciano gestire le connessioni dal Web Server mediante connection pooling
- Connection pooling:
  - le connessioni, una volta aperte, vengono inserite in un pool
  - quando è necessaria una connessione, il Web Server seleziona una connessione nel pool, evitando di riattivare il processo di connessione

Transazioni

- Le transazioni possono essere gestite direttamente dal DBMS, inviando gli statement adeguati per creare la transazione
- é possibile anche utilizzare metodi dell'oggetto Connection:
  - BeginTrans: inizia una transazione nel DBMS a cui si è connessi
  - CommitTrans: termina con successo la transazione corrente
  - RollbackTrans: effettua il rollback della transazione corrente

Un esempio interessante

- Supponiamo che la tabella dipartimenti contenga un campo “homepage” che contiene la URL dell’home page dei dipartimenti
- si vuole generare una pagina HTML che tramite una form, attivi una pagina asp che permetta di recuperare le URL di tutti I dipartimenti, attivandole come collegamento ipertestuale nella pagina restituita al client

Esempio di codice

```
<%
...
rstURL = Server.CreateObject("ADODB.Recordset");

strSQL = "SELECT HomePage from Dipartimenti ”;
rstURL.Open  (strSQL, cnn);


 while (! rstURL.EOF)
    {         %>
    <A HREF = “<%= rstURL(“HomePage”) ; %>”>
    <%= rstURL(“HomePage”);%> </A>
    <%      rstImpiegati.MoveNext;
    }
...
%>
```

Esercitazione proposta

- Estendere la pagina HTML e la pagina ASP generate durante il corso di ASP in modo da:
  - prendere in input tramite form I dati di un corso
  - effettuarne l'inserimento nella base di dati
  - restituire un adeguato messaggio al client
- Estendere la pagina HTML e la pagina ASP generate durante il corso di ASP in modo da:
  - prendere in input qualche valore
  - eseguire una query utilizzando I valori inseriti da form e determinare I corsi corrispondenti
  - inserire gli indirizzi Web relativi ai corsi trovati in una tabella
  - HTML, da restituire al client

Esercitazione proposta

- Estendere la pagina HTML e la pagina ASP generate durante il corso di ASP in modo da:
  - prendere in input tramite form un valore
  - cancellare I corsi utilizzando nella condizione di selezione il valore preso in input
- generare una pagina HTML, contenente un link per ogni pagina precedentemente costruita (inserimento, query cancellazione)

Osservazione conclusiva

- Abbiamo visto che esistono diverse tecnologie per interagire con una base di dati via Web
- ciascuna tecnologia presenta vantaggi e svantaggi
- Applet:
  - semplici da usare
  - gestiscono l'interfaccia grafica
  - limitazioni sull'accesso al DBMS
  - mix di interfaccia e logica applicativa
- Servet:
  - flessibili
  - gestione interfaccia tramite HTML
  - nessuna limitazione sull'accesso al DBMS
  - mix di HTML e logica applicativa
    - il programmatore deve conoscere HTML
- linguaggi di scripting
  - flessibili
  - gestione logica applicativa tramite linguaggio di scripting
  - mix di HTML e script
    - lo sviluppatore HTML deve conoscere il linguaggio di script e la logica applicativa

Osservazione conclusiva

- Quindi, per creare un'applicazione Web che interagisce con le basi di dati è necessario sviluppare tre componenti:
  - interfaccia
  - logica applicativa (come interagisco con la base di dati)
  - flusso applicativo (come gestisco l'intera applicazione)
- in un contesto aziendale, difficilmente esistono persone esperte su tutti questi aspetti
  - l'esperto di basi di dati non necessariamente conosce bene HTML
- come risolvere il problema?

Osservazione conclusiva

- La scelta di Java può essere di aiuto
- come abbiamo visto, Java supporta tre tecnologie:
  - Applet
  - Servlet
  - JSP
- Possibile soluzione:
  - uso Applet per gestire l'interfaccia client
    - pagine statiche, non dipendenti dal contenuto del DB
    - no logica, no flusso applicazione
  - uso Servlet per gestire la logica e il flusso
    - no interfaccia
  - uso JSPper gestire l'interfaccia di risposta
    - pagine dinamiche, dipendenti dal contenuto del DB
    - no logica, no flusso
- necessità di fare comunicare Servlet e JSP (possibile)

Architettura

- L’interazione basi di dati e Web richiede:
  - A livello fisico:
    - l’introduzione di un ulteriore livello nella tipica architettura client/server del Web
    - necessità di gateway per comunicare con il DBMS
      - JDBC, ODBC, DBMS API
  - A livello logico:
    - chiara distinzione tra flusso, logica e presentazione

XML & DBMS

XML e Database: il problema

- Problema:
  - è possibile/necessario memorizzare documenti XML in un DBMS?
  - Quale tecnologia è necessaria a questo scopo?

- Risposta:
  - è certamente possibile memorizzare e gestire documenti XML in un DBMS
  - la tecnologia necessaria a questo scopo dipende dal perché vogliamo gestire documenti XML in un DBMS

Tipologie di documenti XML

- Due possibili usi per documenti XML:
  - *Data Centric*: i documenti possono rappresentare lo strumento con il quale dati tradizionali (es. relazionali) vengono trasferiti su Web
    - XML come veicolo per trasporto di dati
    - Esempio: ordini di vendita, scheduling di voli, menù
  - *Document Centric*: l'informazione è rappresentata dal documento in sé
    - XML come modello per la rappresentazione dei dati
    - Esempio: libri, documenti in genere

Documenti Data Centric

- Struttura regolare
- livello di dettaglio piuttosto fine
- contenuto omogeneo
- l’ordine con cui gli elementi allo stesso livello appaiono è ininfluente
- Utilizzati per “machine consumption”
- Esempi: ordini di vendita, scheduling di voli, menù,…

Esempio: ordini di vendita

```
<Orders>
   <SalesOrder SONumber="12345">
         <Customer CustNumber="543">
           <CustName>ABC Industries</CustName>
           ...
         </Customer>
         <OrderDate>981215</OrderDate>
         <Line LineNumber="1">
           <Part PartNumber="123">
             <Description>
                    Turkey wrench: Stainless steel, one piece...
             </Description>
             <Price>9.95</Price>
           </Part>
           <Quantity>10</Quantity>
         </Line>
         <Line LineNumber="2">
           ...
         </Line>
   </SaleOrder>
</Orders>
```

Documenti Document Centric

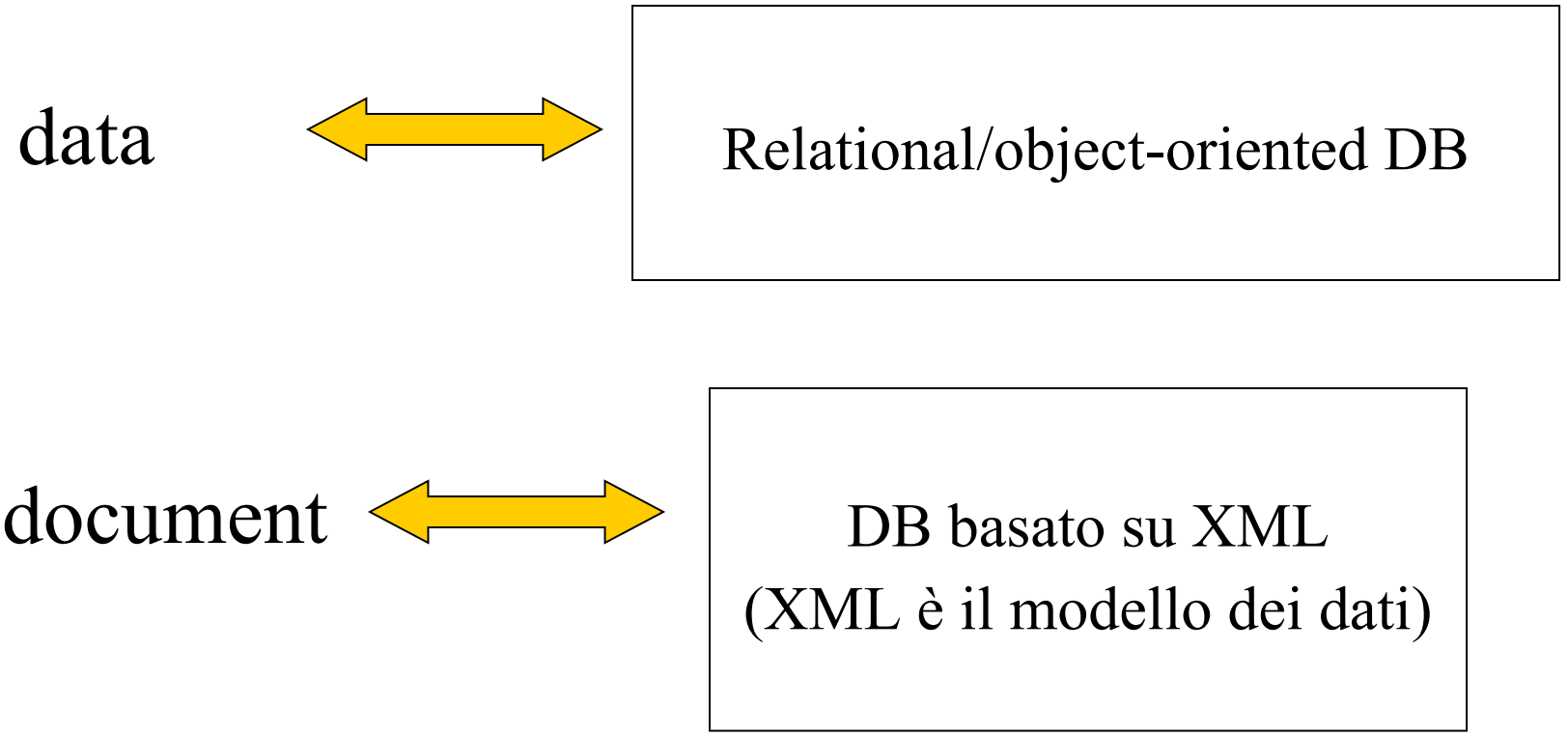
- Struttura irregolare
- Livello di dettaglio meno fine
- contenuto eterogeneo
- l’ordine degli elementi allo stesso livello è significativo
- in genere progettati per “human consumption”
- Esempi: libri, email, …

Product Description

```
<Product>
<Name>Turkey Wrench</Name>
<Developer>Full Fabrication Labs, Inc.</Developer>
<Summary>Like a monkey wrench, but not as big.</Summary>
<Description>
<Para>The Turkey wrench, which comes in both right- and left-handed
  versions ....</Para>
<Para>You can:</Para>
<List>
  <Item><Link URL="Order.htm">Order your turkey wrench</Link></Item>
  <Item><Link URL="Wrench.html">Read about wrenches</Link></Item>
  <Item><Link URL="catalog.zip">Download the catalog</Link></Item>
</List>
  ....
</Description>
</Product>
```

XML e DBMS

- Ciascuna tipologia di documenti richiede una particolare tecnologia per la sua gestione



XML e DBMS

- Due categorie di DBMS:
  - XML-Native DBMS:
    - comprendono un insieme di nuovi sistemi la cui architettura è stata progettata per supportare totalmente le funzionalità necessarie alla gestione di documenti XML
    - tecnologia non ancora matura
    - utili per Document Centric
    - Esempio:eXcelon
  - XML-Enabled DBMS:
    - comprendono tutti i DBMS che mantengono integra la propria architettura estendendola con funzionalità necessarie alla gestione di documenti XML
    - sono tipicamente Object-Relational (DB2, Oracle8i,...)
    - utili per Data Centric e parzialmente per Document Centric

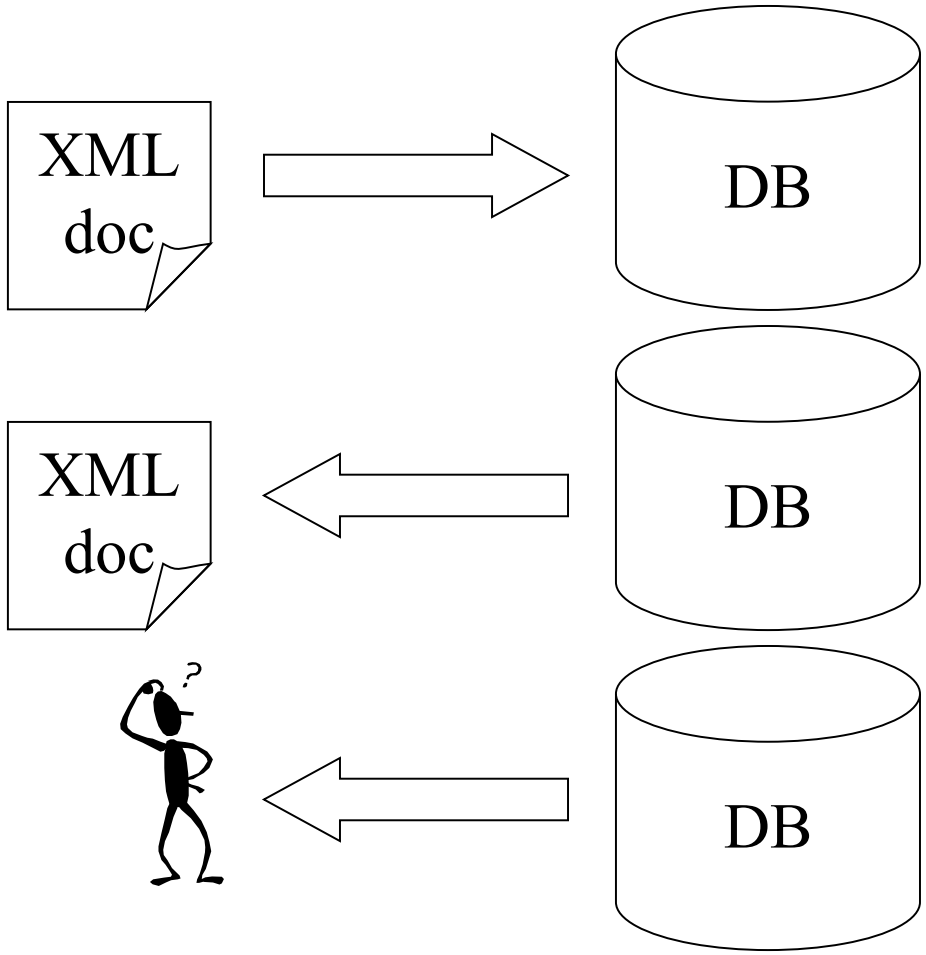
XML e DBMS

- Nel seguito.
  - Problematiche relative alla gestione di documenti Data Centric e Document Centric in XML-Enabled DBMS

XML-Enabled DBMS e documenti Data Centric

Problematiche per Data Centric

- Tre problematiche di base:
  - come rappresentare i dati contenuti nei documenti XML nel DBMS
  - come generare documenti XML partendo dai dati contenuti nel DBMS
  - come interrogare i dati estratti da documenti XML



Rappresentazione dati

- È necessario definire un mapping tra la struttura dei documenti XML e lo schema del DB
  - Per memorizzare i dati contenuti in un documenti XML in un DB, deve esistere una o più tabelle con lo schema richiesto dal mapping
- rappresentazione strutturata
- Vantaggi:
  - approccio piuttosto semplice
  - i dati sono facilmente interrogabili
- Svantaggi:
  - Scarsa flessibilità: la tabella deve essere conforme al documento
  - il documento di partenza non è più recuperabile

DBMS relazionale

- Un documento XMl viene rappresentato come una singola tabella o un insieme di tabelle
- la struttura del documento XML è simile alla seguente:

```
<database>
<table>
<row>
<column1>...</column1>
<column1>...</column1>
...
</row>
...
</table>
...
</database>
```

- approccio tipico per DBMS relazionali, object-relational

Esempio

Documento XML

```
<clienti>
   <row>
     <numero> 7369 </numero>
     <nome> PAUL </nome>
     <cognome> SMITH </cognome>
   </row>
   <row>
     <numero> 7000 </numero>
     <nome> STEVE </nome>
     <cognome> ADAM </cognome>
   </row>
</clienti>
```

Tabella Clienti

Numero	Nome	Cognome
2000	MIKE	SCOTT
7369	PAUL	SMITH
7000	STEVE	ADAM

Esempio

Documento XML

```
<clienti>
  <row>
      <numero> 7369 </numero>
       <lista_clienti>
      <cliente>
            <nome> PAUL </nome>
              <cognome> SMITH
</cognome>
      </cliente>
            <cliente>
            <nome> STEVE </nome>
              <cognome> ADAM
</cognome>
          </cliente>
        </lista_clienti>
      </row>
</clienti>
```

Tabella Lista_Clienti

Numero
2000
7369

Tabella Clienti

Numero	Num_cliente	Nome	Cognome
2000	1	MIKE	SCOTT
7369	2	PAUL	SMITH
7369	3	STEVE	ADAM

Interrogazione dati

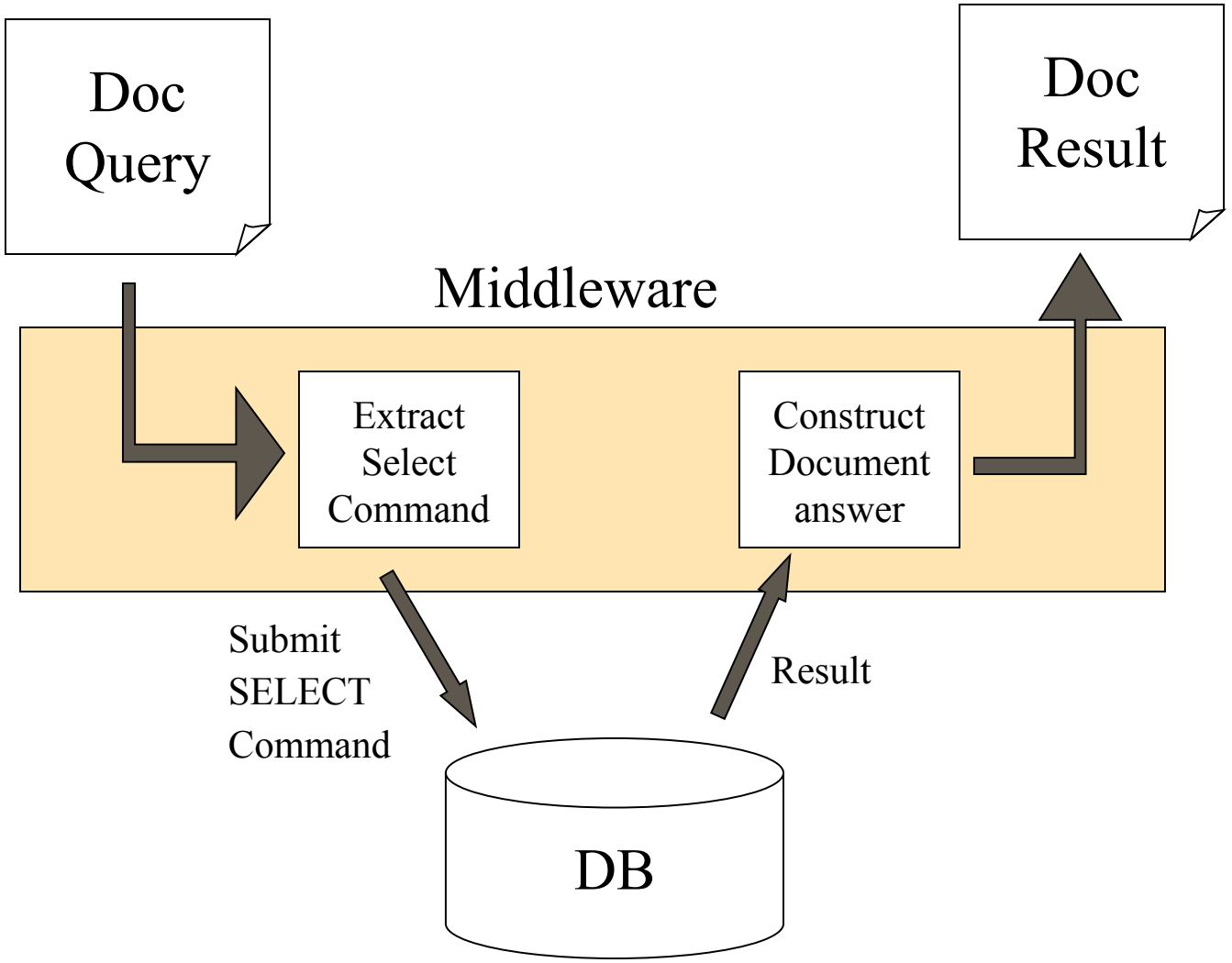
- Poiché i dati vengono rappresentati secondo il modello supportato dal DBMS (es. relazionale), è possibile utilizzare i linguaggi supportati dal DBMS per l'interrogazione dei dati memorizzati
- approccio template-based:
  - la query viene rappresentata nel documento XML
  - necessità di middleware

Flight Information

```
<?xml version="1.0">
<FlightInfo>
   <Intro>The following flights have available seats:</Intro>
   <SelectStmt>
        SELECT Airline, FltNumber, Depart, Arrive FROM Flights
   </SelectStmt>
   <Conclude>We hope one of these meets your needs</Conclude>
</FlightInfo>
```

```
<?xml version="1.0">
<FlightInfo>
   <Intro>The following flights have available seats:</Intro>
   <Flight>
        <Row>
           <Airline>ACME</Airline><FltNumber>123</FltNumber>
           <Depart>Dec 12, 1998
   13:43</Depart><Arrive>...<Arrive>
        <Row>
   </Flight>
   <Conclude>We hope one of these meets your needs</Conclude>
</FlightInfo>
```

Interrogazione dati



Generazione documenti XML

- Problema: fornire una rappresentazione XML ai dati recuperati tramite query dal DBMS
- si utilizza il mapping inverso rispetto a quello utilizzato per la memorizzazione
- operazione importante per attribuire un formato standard ai dati ritrovati, prima di inviarli sulla rete

Esempio

```
SELECT nome, cognome
FROM Clienti
WHERE Numero = “7369
```

Tabella Clienti

Numero	Nome	Cognome
2000	MIKE	SCOTT
7369	PAUL	SMITH
7000	STEVE	ADAM

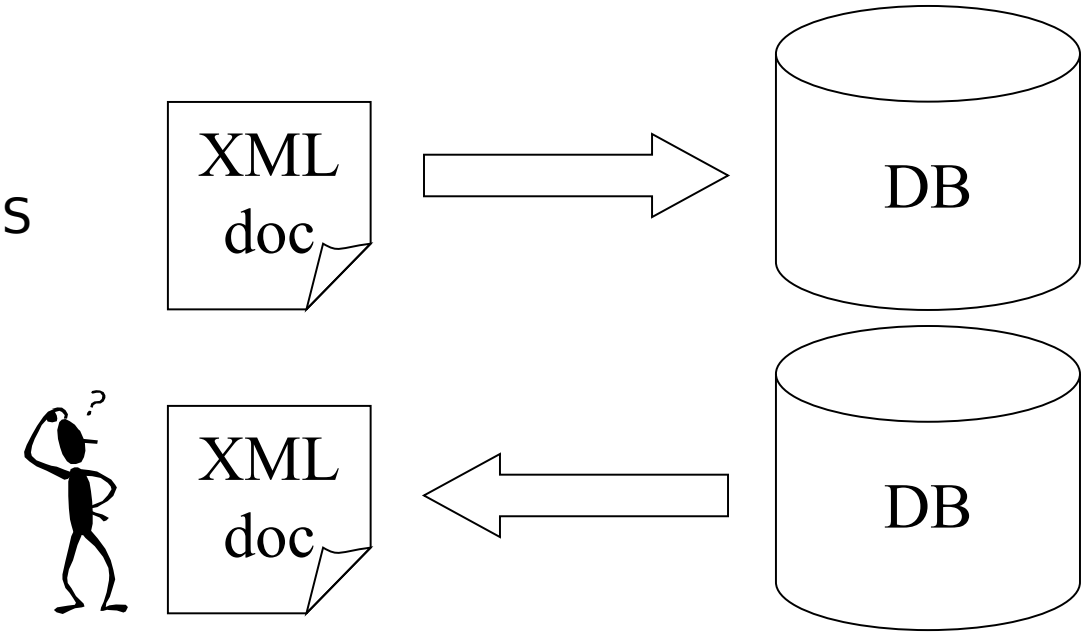
Documento XML

```
<clienti>
   <row>
      <nome> PAUL </nome>
      <cognome> SMITH </cognome>
   </row>
</clienti>
```

XML-Enabled DBMS e documenti Document Centric

Problematiche per Document Centric

- Due problematiche di base:
  - come rappresentare i documenti XML nel DBMS
  - come interrogare i documenti XML



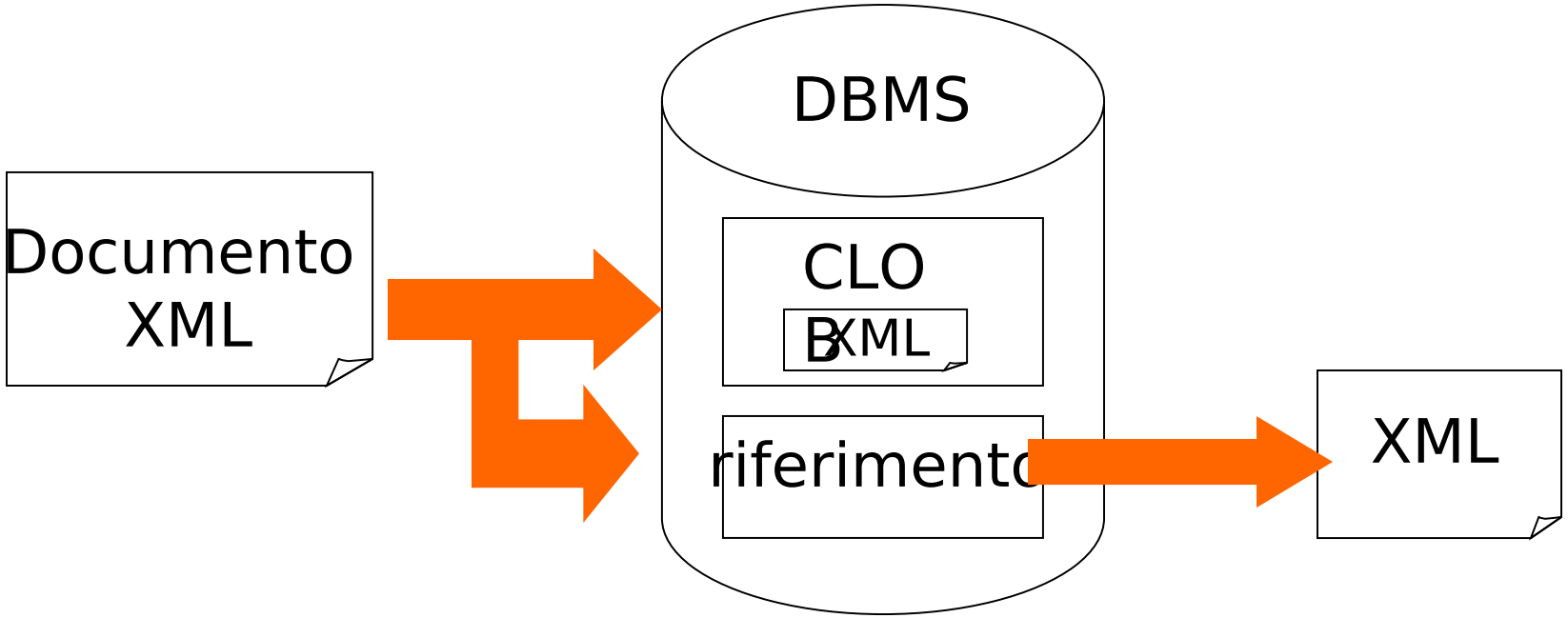
Rappresentazione

- Permette di mantenere integro il documento XML
- Due approcci:
  - rappresentazione non strutturata
    - documento come unico oggetto
  - rappresentazione ibrida
    - documento parzialmente rappresentato secondo la rappresentazione strutturata e parzialmente secondo la rappresentazione non strutturata

Rappresentazione non strutturata

- Il documento viene tipicamente mappato in un singolo campo di una tabella di tipo:
  - CLOB (Character Large Object): il documento è fisicamente contenuto nel campo della tabella
    - alcuni DBMS (IBM DB2) supportato tipi ad hoc: XMLVARCHAR
  - riferimento: il campo contiene il riferimento al documento, memorizzato altrove, sul file system
- Vantaggi:
  - flessibile
- Svantaggi:
  - i dati sono non strutturati
  - interrogazione più complessa
  - la tabella può contenere documenti eterogenei (diversi DTD)

Rappresentazione non strutturata



Esempio

Documento XML

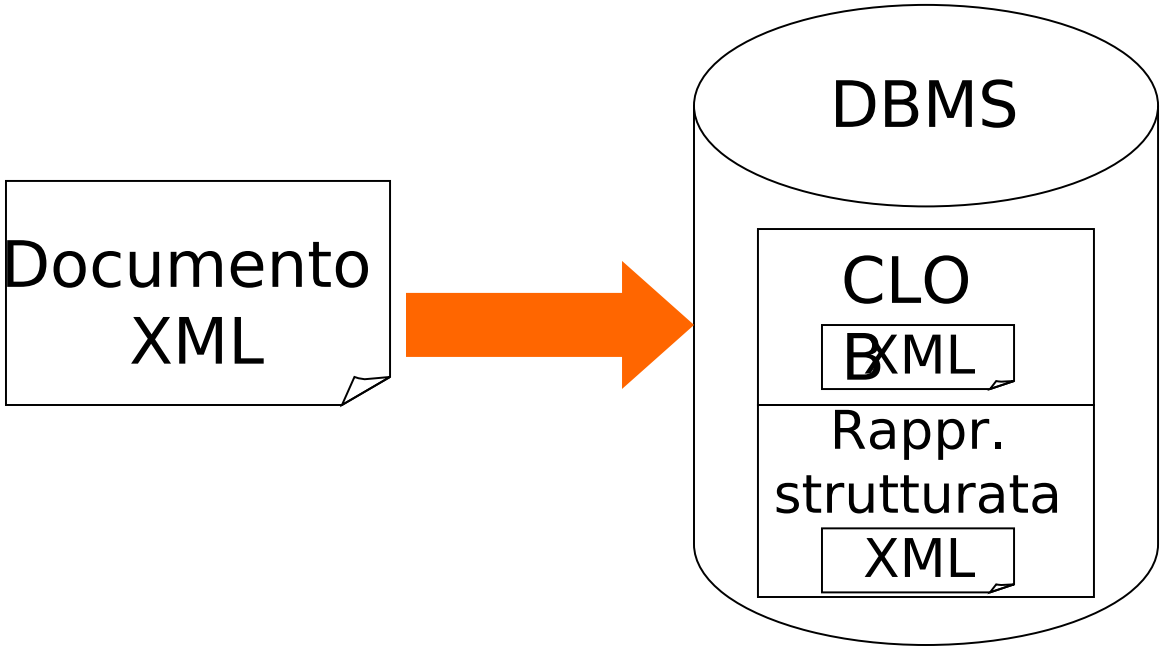
```
<clienti>
   <row>
     <numero> 7369 </numero>
     <nome> PAUL </nome>
     <cognome> SMITH </cognome>
   </row>
   <row>
     <numero> 7000 </numero>
     <nome> STEVE </nome>
     <cognome> ADAM </cognome>
   </row>
</clienti>
```

Tabella Clienti

Id	Documento_XML
10	<clienti> <row> <numero> 7369 </numero> <nome> PAUL </nome> <cognome> SMITH </cognome> </row> <row> <numero> 7000 </numero> <nome> STEVE </nome> <cognome> ADAM </cognome> </row> </clienti>

Rappresentazione ibrida

- Rappresentazione che combina rappresentazione strutturata e non strutturata

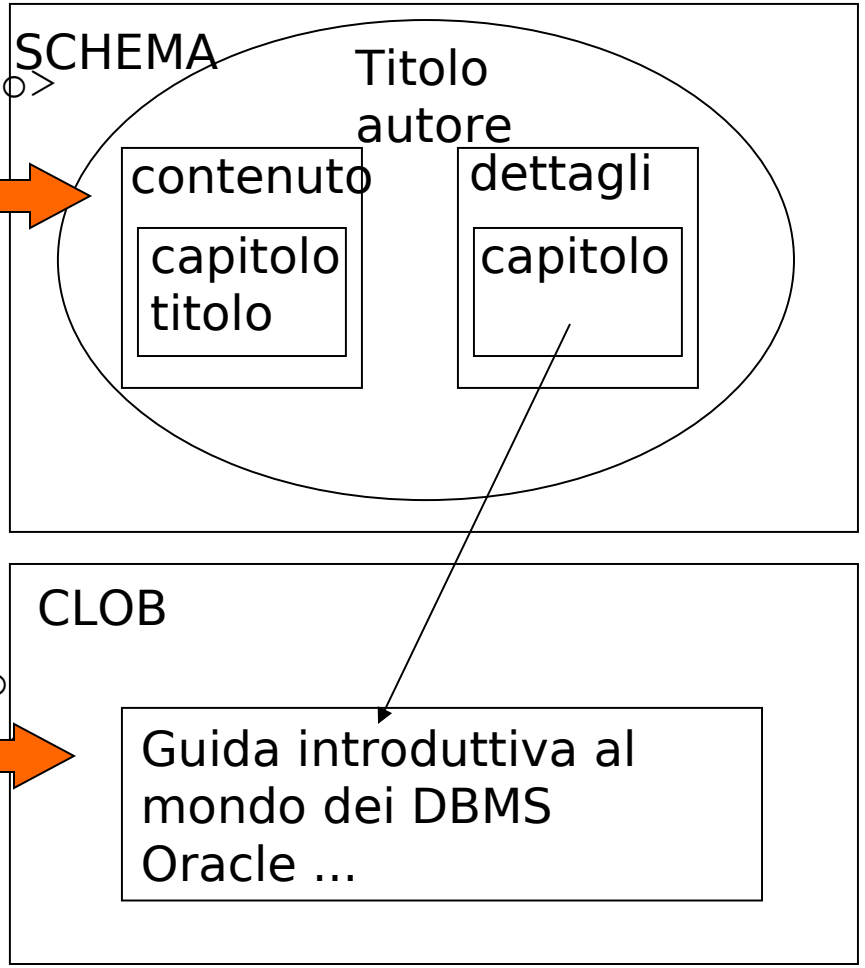


Esempio

Documento XML

```
<libro>
       <titolo> Oracle Guide </titolo>
       <autore> M. Abbey  </autore>
  <contenuto>
         <capitolo n='1'>
              <titolo> Introduzione
</titolo>
                …
         </capitolo>
    …
  </contenuto>
  <dettagli>
     <capitolo n='1'>
       <sezione n='1'>
         Guida introduttiva al mondo
                    dei DBMS Oracle
       </sezione>
     </capitolo>
  </dettagli>
 …
</libro>
```

Tabella LIBRO



Interrogazione documenti

- Dal punto di vista del DBMS, un documento memorizzato in modo non strutturato non è che un documento di testo
- in genere i DBMS supportano strumenti per ritrovare i documenti in base al contenuto
- nel caso di documenti XML, mettono a disposizione operatori avanzati da utilizzare in statement SQL per recuperare documenti XML in base al contenuto

XML e Oracle 8i

- XML-enabled
- supporta rappresentazione strutturata, non strutturata in campi CLOB e BFILE, e ibrida
- interrogazione rappresentazione non strutturata tramite Intermedia Context
- generazione documenti XML a partire dal contenuto DB

XML e IBM DB2

- XML enabled
- supporta rappresentazione strutturata, non strutturata in campi ad hoc, e ibrida
- Nuovi tipi di dato:
  - XMLVARCHAR: documenti XML memorizzati come VARCHAR
  - XMLCLOB: documenti XML memorizzati come CLOB
  - XMLFILE: riferimento ad un documento XML, memorizzato su file system
- interrogazione rappresentazione non strutturata tramite:
  - operatori specifici, che permettono di navigare la struttura del documento
  - Text Extender, che supporta funzionalità aggiuntive di analisi del contenuto (ne parleremo nel contesto Multimedia)
- generazione documenti XML a partire dal contenuto DB

Supporto Tecnologie Internet in SQL Server

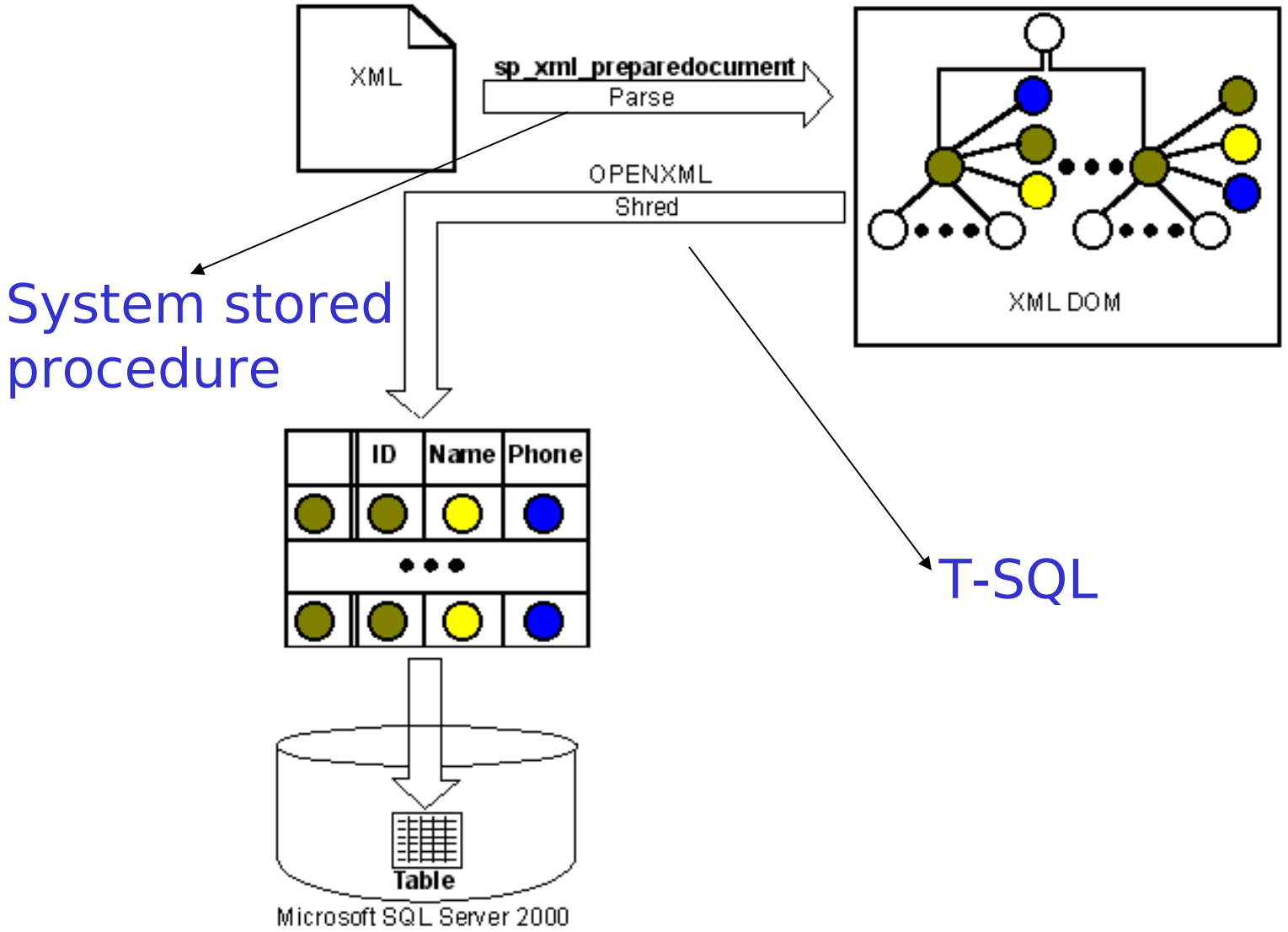
Funzionalità

- SQL Server 2000 è un XML-enabled DBMS
- supporta le seguenti funzionalità:
  - gestione documenti document centric:
    - tramite campi di tipo text (nessun supporto particolare)
  - gestione documenti data centric:
    - generazione documenti XML a partire dal contenuto della base di dati
    - inserimento di documenti data-centric
  - supporto per XDR (XML-Data Reduced) schema
    - viste in formato XML sullo schema di una base di dati
    - interrogazione di tali viste con XPath
  - accesso a SQL Server da HTTP

Gestione documenti document- centric

- T-SQL supporta comandi per:
  - generare un insieme di tuple da un documento XML
  - generare un documento XML come risultato di una query

Generazione tuple da XML



Fasi

- Generazione DOM tree tramite la stored procedure sp_xml_preparedocument
  - Parametri (nell'ordine):
    - OUT: hdoc OUTPUT (handler alla nuova rappresentazione)
    - IN: variabile di tipo testuale, contenente il documento XML
- estrazione tuple da document:
  - funzione OPENXML
- rimozione handle con stored procedure sp_xml_removedocument

Esempio: generazione handle

```
DECLARE @hdoc int
DECLARE @doc varchar(1000)
SET @doc ='
<ROOT>
<Customer CustomerID="VINET" ContactName="Paul Henriot">
   <Order CustomerID="VINET" EmployeeID="5" OrderDate="1996-07-04T00:00:00">
      <OrderDetail OrderID="10248" ProductID="11" Quantity="12"/>
      <OrderDetail OrderID="10248" ProductID="42" Quantity="10"/>
   </Order>
</Customer>
<Customer CustomerID="LILAS" ContactName="Carlos Gonzlez">
   <Order CustomerID="LILAS" EmployeeID="3" OrderDate="1996-08-16T00:00:00">
      <OrderDetail OrderID="10283" ProductID="72" Quantity="3"/>
   </Order>
</Customer>
</ROOT>'
--Create an internal representation of the XML document.
EXEC sp_xml_preparedocument @hdoc OUTPUT, @doc
-- Remove the internal representation.
exec sp_xml_removedocument @hdoc
```

Generazione tuple

- OPENXML richiede quattro parametri:
  - un XML handle, generato da sp_xml_preparedocument (idoc)
  - un'espressione Xpath che identifica I nodi del documento dai quali creare le tuple (xpathexpr)
  - una descrizione dello schema delle tuple da generare (SchemaDeclaration|TableName)
  - mapping tra lo schema delle tuple e I nodi restituiti dall'espressione Xpath (flag + indicazioni in clausola WITH)

OPENXML**(***idoc* **int** [**in**],xpathexpr **nvarchar**[**in**],[*flags* **byte**[**in**]]**)**
[WITH (*SchemaDeclaration* | *TableName*)]

Due parole su XPath

- Xpath permette di navigare la struttura ad albero di un documento XML, utilizzando una sintassi simile a quella utilizzata per navigare nel file system, estesa con l'utilizzo di condizioni di selezione (predicati)
- Restituisce I nodi finali del cammino specificato
- per ogni nodo lungo il percorso, è possibile specificare dei predicati
- per specificare il passaggio da un livello all'altro: /
  - /Customers/ContactName
- per identificare un attributo: @nome attributo
- per specificare una condizione: []
  - /Customer/Order[@OrderID="1"]
- per specificare il nodo padre: ..
- Per specificare il nodo corrente: .

Esempi

- Seleziona dalla radice (elemento più esterno), I clienti con attributo ClientID = ="ALFKI"
  - Customer[@CustomerID="ALFKI"]
- supponendo che Customer abbia come sottoelemento Order, seleziona tutti gli ordini in cui OrderID = 1
  - /Customer/Order[@OrderID="1"]
- Supponendo che Customer ammetta come sottoelemento ContactName, selezionare tutti I Customer che:
  - hanno almeno un ContactName
    - /Customer[ContacName]
  - non hanno ContactName
    - /Customer[not(ContacName)]

Esempi

- Selezionare gli ordini che si riferiscono al cliente identificato da "ALFKI"
  - /Customer/Order[../@CustomerID="ALFKI"]
  - /Customer[@CustomerID="ALFKI"]/Order
- selezionare l’ordine identificato da "Ord-10643”, relativo al cliente identificato da "ALFKI"
  - /Customer[@CustomerID="ALFKI"]/Order[@OrderID="Ord-10643"]
- selezionare I clienti che hanno effettuato almeno un ordine di un prodotto inquantità superiore a 5
  - /Customer[Order/OrderDetail[@Quantity>5]]

Esempi

- Selezionare I vari prodotti ordinati, che complessivamente costano più di 2000
  - /child::OrderDetail[@UnitPrice * @Quantity = 2000]
- seleziona il cliente "ALFKI" e il cliente “ANATR”
  - /Customer[@CustomerID="ALFKI" or @CustomerID="ANATR"]

Esempio 1

```
DECLARE @idoc int
DECLARE @doc varchar(1000)
SET @doc ='
<ROOT>
<Customer CustomerID="VINET" ContactName="Paul Henriot">
  <Order OrderID="10248" CustomerID="VINET" EmployeeID="5"
       OrderDate="1996-07-04T00:00:00">
    <OrderDetail ProductID="11" Quantity="12"/>
    <OrderDetail ProductID="42" Quantity="10"/>
  </Order>
</Customer>
<Customer CustomerID="LILAS" ContactName="Carlos Gonzlez">
  <Order OrderID="10283" CustomerID="LILAS" EmployeeID="3"
       OrderDate="1996-08-16T00:00:00">
    <OrderDetail ProductID="72" Quantity="3"/>
  </Order>
</Customer>
</ROOT>'
```

Esempio 1a

```
-- Create an internal representation of the XML document.
EXEC sp_xml_preparedocument @idoc OUTPUT, @doc
-- Execute a SELECT statement using OPENXML rowset provider.
SELECT *
FROM OPENXML (@idoc, '/ROOT/Customer',1)
        WITH (CustomerID  varchar(10),
                ContactName varchar(20))
-- Remove the internal representation of the XML document
EXEC sp_xml_removedocument @idoc
```

RISULTATO

```
CustomerID ContactName
---------- --------------------
VINET      Paul Henriot
LILAS      Carlos Gonzlez
```

Attribute-centric mapping:
gli attributi del rowset vengono mappati sugli attributi con nome uguale specificati nella clausola WIT

Esempio 1b

```
-- Create an internal representation of the XML document.
EXEC sp_xml_preparedocument @idoc OUTPUT, @doc
-- Execute a SELECT statement using OPENXML rowset provider.
SELECT *
FROM OPENXML (@idoc, '/ROOT/Customer',2)
        WITH (CustomerID  varchar(10),
              ContactName varchar(20))
-- Remove the internal representation of the XML document
EXEC sp_xml_removedocument @idoc
```

RISULTATO

```
CustomerID ContactName
---------- --------------------
NULL       NULL
```

Element-centric mapping:
si considera il contenuto
dei sottolementi semplici
(in questo caso non ci sono)

Esempio 2

```
DECLARE @idoc int
DECLARE @doc varchar(1000)
SET @doc ='
<ROOT>
<Customer>
    <CustomerID>VINET</CustomerID>
    <ContactName>Paul Henriot</ContactName>
     <Order OrderID="10248" CustomerID="VINET" EmployeeID="5"
        OrderDate="1996-07-04T00:00:00">
     <OrderDetail ProductID="11" Quantity="12"/>
     <OrderDetail ProductID="42" Quantity="10"/>
  </Order>
</Customer>
<Customer>
    <CustomerID>LILAS</CustomerID>
    <ContactName>Carlos Gonzlez</ContactName>
    <Order OrderID="10283" CustomerID="LILAS" EmployeeID="3"
        OrderDate="1996-08-16T00:00:00">
     <OrderDetail ProductID="72" Quantity="3"/>
  </Order>
</Customer>
</ROOT>'
```

Esempio 2a

```
-- Create an internal representation of the XML document.
EXEC sp_xml_preparedocument @idoc OUTPUT, @doc
-- Execute a SELECT statement using OPENXML rowset provider.
SELECT *
FROM OPENXML (@idoc, '/ROOT/Customer',2)
        WITH (CustomerID  varchar(10),
              ContactName varchar(20))
-- Remove the internal representation of the XML document
EXEC sp_xml_removedocument @idoc
```

RISULTATO

```
CustomerID ContactName
---------- --------------------
VINET      Paul Henriot
LILAS      Carlos Gonzlez
```

Element-centric mapping:
si considera il contenuto
dei sottolementi semplici
(in questo caso non ci sono)

Esempio 3

```
DECLARE @idoc int
DECLARE @doc varchar(1000)
SET @doc ='
<ROOT>
<Customer CustomerID="VINET" ContactName="Paul Henriot">
  <Order OrderID="10248" CustomerID="VINET" EmployeeID="5"
       OrderDate="1996-07-04T00:00:00">
    <OrderDetail ProductID="11" Quantity="12"/>
    <OrderDetail ProductID="42" Quantity="10"/>
  </Order>
</Customer>
<Customer CustomerID="LILAS" ContactName="Carlos Gonzlez">
  <Order OrderID="10283" CustomerID="LILAS" EmployeeID="3"
       OrderDate="1996-08-16T00:00:00">
    <OrderDetail ProductID="72" Quantity="3"/>
  </Order>
</Customer>
</ROOT>'
```

Esempio 3a

```
-- Create an internal representation of the XML document.
    EXEC sp_xml_preparedocument @idoc OUTPUT, @doc
-- Execute a SELECT stmt using OPENXML rowset provider.
     SELECT *
     FROM OPENXML (@idoc, '/ROOT/Customer/Order/OrderDetail')
                WITH (OrderID int                    '../@OrderID',
                      CustomerID varchar(10)     '../@CustomerID',
                      OrderDate datetime                  '../@OrderDate',
                      ProdID int                   '@ProductID',
                      Qty int                      '@Quantity')
```

RISULTATO

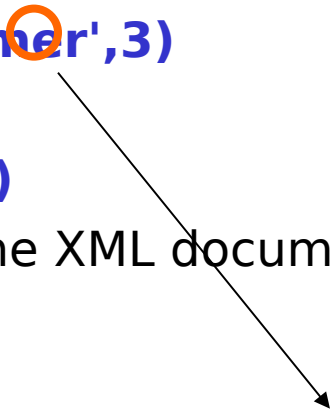
```
OrderID CustomerID OrderDate ProdID Qty
-----------------------------------------------------------
 10248 VINET 1996-07-04 00:00:00.000 11 12
10248 VINET 1996-07-04 00:00:00.000 42 10
10283 LILAS 1996-08-16 00:00:00.000 72 3
```

Esempio 4

```
DECLARE @idoc int
DECLARE @doc varchar(1000)
SET @doc ='
<ROOT>
<Customer CustomerID="VINET" >
 <ContactName>Paul Henriot</ContactName>
  <Order OrderID="10248" CustomerID="VINET" EmployeeID="5"  OrderDate="1996-07 04">
    <OrderDetail ProductID="11" Quantity="12"/>
    <OrderDetail ProductID="42" Quantity="10"/>
  </Order>
</Customer>
<Customer CustomerID="LILAS" >
  <ContactName>Carlos Gonzlez</ContactName>
  <Order OrderID="10283" CustomerID="LILAS" EmployeeID="3" OrderDate="1996-08-16">
    <OrderDetail ProductID="72" Quantity="3"/>
  </Order>
</Customer>
</ROOT>'
```

Esempio 4a

```
-- Create an internal representation of the XML document.
EXEC sp_xml_preparedocument @idoc OUTPUT, @doc
-- Execute a SELECT statement using OPENXML rowset provider.
SELECT *
FROM OPENXML (@idoc, '/ROOT/Customer',3)
        WITH (CustomerID  varchar(10),
              ContactName varchar(20))
-- Remove the internal representation of the XML document
EXEC sp_xml_removedocument @idoc
```



RISULTATO

```
CustomerID ContactName
---------- --------------------
VINET      Paul Henriot
LILAS      Carlos Gonzlez
```

Attribute +Element-centric mapping:
si considerano prima gli attributi e po
gli elementi

Esempio 4b

- Supponiamo di avere una tabella T(CustomerId, ContactName)

```
-- Create an internal representation of the XML document.
EXEC sp_xml_preparedocument @idoc OUTPUT, @doc
-- Execute a SELECT statement using OPENXML rowset provider.
SELECT *
FROM OPENXML (@idoc, '/ROOT/Customer',3)
        WITH T
-- Remove the internal representation of the XML document
EXEC sp_xml_removedocument @idoc
```

- applicabile quando:
  - esiste una tabella con la struttura voluta
  - non si devono specificare pattern XPath

Esempio 5

```
DECLARE @idoc int
DECLARE @doc varchar(1000)
SET @doc ='
<ROOT>
<Customer CustomerID="VINET" ContactName="Paul Henriot">
  <Order OrderID="10248" CustomerID="VINET" EmployeeID="5"
        OrderDate="1996-07-04T00:00:00">
    <OrderDetail ProductID="11" Quantity="12"/>
    <OrderDetail ProductID="42" Quantity="10"/>
  </Order>
</Customer>
<Customer CustomerID="LILAS" ContactName="Carlos Gonzlez">
  <Order OrderID="10283" CustomerID="LILAS" EmployeeID="3"
        OrderDate="1996-08-16T00:00:00">
    <OrderDetail ProductID="72" Quantity="3"/>
  </Order>
</Customer>
</ROOT>'
```

Esempio 5a

```
-- Create an internal representation of the XML document.
EXEC sp_xml_preparedocument @idoc OUTPUT, @doc
-- Execute a SELECT statement using OPENXML rowset provider.
SELECT *
FROM OPENXML (@idoc,
    '/ROOT/Customer/Order/OrderDetail/@ProductID') WITH ( ProdID
                int       '.',
                  Qty               int       '../@Quantity',
                  OID int                     '../../@OrderID')
-- Remove the internal representation of the XML document
EXEC sp_xml_removedocument @idoc
```

RISULTATO

ProdID	Qty	OID
11	12	10248
42	10	10248
72	3	10283

Generazione XML da tuple

- È possibile generare un documento XML a partire dalle tuple ottenute come risultato di una query
- dato uno statement SELECT
  - la clausola FOR XML formatta il risultato come documento XML
  - è possibile dare indicazioni su come il documento debba essere formattato
    - RAW
    - AUTO
    - EXPLICIT

Limitazioni

- Non si può usare in:
  - sottoquery
  - aggregazioni
  - definizione di viste
  - analisi con cursore
  - ...

Modalità RAW

- Trasforma ogni tupla del risultato in un elemento XML con identificatore “row”
- ogni colonna non null viene mappata in un attributo dell’elemento

Esempio 1

```
SELECT TOP 2 Customers.CustomerID, Orders.OrderID,
    Orders.OrderDate
FROM Customers, Orders
WHERE Customers.CustomerID = Orders.CustomerID
ORDER BY Customers.CustomerID
FOR XML RAW
```

RISULTATO

```
<row CustomerID="ALFKI" OrderID="10643" OrderDate="1997-08-
    25T00:00:00"/>
<row CustomerID="ALFKI" OrderID="10692" OrderDate="1997-10-
    03T00:00:00"/>
```

Modalità AUTO

- Restituisce elementi XML annidati
- ogni tabella che compare nella clausola FROM per la quale almeno un attributo compare nella SELECT viene rappresentata con un elemento XML con lo stesso nome
- le colonne della SELECT diventano attributi dell'elemento corrispondente alla tabella a cui appartengono, con lo stesso nome
- é possibile dire che le colonne della SELECT devono diventare sottoelementi
  - clausola ELEMENTS
- il nesting degli elementi dipende dall'ordine con cui gli attributi appaiano nella clausola SELECT
  - leftmost = più esterno

Esempio 1

```
SELECT TOP 2 Customers.CustomerID, Orders.OrderID,
    Orders.OrderDate
FROM Customers, Orders
WHERE Customers.CustomerID = Orders.CustomerID
ORDER BY Customers.CustomerID
FOR XML AUTO
```

RISULTATO

```
<Customers CustomerID="ALFKI">
    <Orders OrderID="10643" OrderDate="1997-08-25T00:00:00"/>
    <Orders OrderID="10692" OrderDate="1997-10-03T00:00:00"/>
</Customers>
```

Esempio 2

```
SELECT TOP 2  Orders.OrderID, Customers.CustomerID,
    Orders.OrderDate
FROM Customers, Orders
WHERE Customers.CustomerID = Orders.CustomerID
ORDER BY Customers.CustomerID
FOR XML AUTO
```

RISULTATO

```
<Orders OrderID="10643" OrderDate="1997-08-25T00:00:00">
    <Customers CustomerID="ALFKI"/>
</Orders>
<Orders OrderID="10692" OrderDate="1997-10-03T00:00:00">
    <Customers CustomerID="ALFKI"/>
</Orders>
```

Esempio 3

```
SELECT TOP 2 Customers.CustomerID, Orders.OrderID, Orders.OrderDate
FROM Customers, Orders
WHERE Customers.CustomerID = Orders.CustomerID
ORDER BY Customers.CustomerID
FOR XML AUTO, ELEMENTS
```

RISULTATO

```
<Customers>
    <CustomerID>ALFKI</CustomerID>
    <Orders>
    <OrderID>10643</OrderID>
    <OrderDate>1997-08-25T00:00:00</OrderDate>
    </Orders>
    <Orders>
    <OrderID>10692</OrderID>
    <OrderDate>1997-10-03T00:00:00</OrderDate>
    </Orders>
</Customers>
```

Esempio 4

```
SELECT TOP 2 C.CustomerID, O.OrderID, O.OrderDate
FROM Customers C, Orders O
WHERE C.CustomerID = O.CustomerID
ORDER BY C.CustomerID
FOR XML AUTO
```

RISULTATO

```
<C CustomerID="ALFKI">
    <O OrderID="10643" OrderDate="1997-08-25T00:00:00"/>
    <O OrderID="10692" OrderDate="1997-10-03T00:00:00"/>
</C>
```

Modalità EXPLICIT

- Permette di controllare il formato del documento XML
- affinché possa essere attivata, la query SQL deve avere una certa forma
- non la vediamo

Tipi di accesso da HTTP

- Esecuzione query SQL
- Esecuzione query template
  - documenti XML che al loro interno contengono la specifica di una o più query SQL
  - il template può essere contenuto in un file o specificato nell'indirizzo
- Esecuzione query Xpath su schemi XDR
  - uno schema XDR (XML-Data Reduced) è una vista XML sul contenuto del database
- Accesso ad oggetti contenuti nel database

Come attivare le funzionalità Internet

- É necessario creare una directory virtuale in IIS
- questo permette di stabilire una connessione con un'istanza di SQL Server
- la directory virtuale è associata ad informazioni di connessione che permettono di accedere un particolare database
- il nome del server IIS e della directory virtuale compaiono nell'URL da utilizzare per comunicare con SQL Server
- Se la directory virtuale si chiama "myDB" e il server Web si chiama "PAPERINO", e' possibile collegarsi al database usando il seguente indirizzo:
  - http://paperino/myDB
- Tutto cio' che vogliamo visualizzare deve essere un document XML

Esecuzione query SQL da HTTP

- La stringa SQL viene passata come valore associata al parametro “sql”, fornito direttamente con l’indirizzo
- Se la query restituisce tuple con piu’ di un campo, e’ necessario che il risultato venga prodotto in formato XML
- Se viene restituito piu’ di un elemento XML (quindi piu’ di un record) e’ necessario inserire un tag fittizio
  - Si passa il valore ROOT associato alla variabile “root “
- Esempio:
  - http://paperino/myDB?sql=select * from impiegati FOR XML AUTO&root=ROOT
  - http://paperino/myDB?sql=select nome from impiegati

Esecuzione Stored Proc da HTTP

- In modo simile si possono eseguire stored procedure
  - http://paperino/myDB?sql+EXECUTE myProc+val1+val2&root=ROOT
- In questo caso, il documento XML deve essere generato dalla stored procedure

Esecuzione template

- Scrivere lunghi statement SQL nell'URL può essere problematico
- Soluzione:
  - Inserire le query SQL in template
- Un template e' un documento XML ben formato contenente una o piu' query SQL o Xpath
- I templati devono essere salvati in directory di tipo *template*, all'interno della directory reale corrispondente alla directory virtuale
  - se la directory si chiama "myTemplate" e il template Template1.xml:
    - http://paperino/myDB/myTemplate/Template1.xml
- Anche in questo caso, la query deve restituire un risultato in formato XML
  - l'elemento radice è inserito direttamente dal template

Forma generale di un template per eseguire query SQL

```
<ROOT xmlns:sql="urn:schemas-microsoft-com:xml-sql" sql:xsl='XS
   FileName' >
<sql:header>
   <sql:param>..</sql:param>
   <sql:param>..</sql:param>
</sql:header>
<sql:query> sql statement(s) </sql:query>
</ROOT>
```

Esempio 1: SELECT

```
<ROOT xmlns:sql="urn:schemas-microsoft-com:xml-sql">
<sql:query>
    SELECT top 2 CustomerID, CompanyName FROM Customers
    FOR XML AUTO
</sql:query>
</ROOT>
```

http://paperino/myDB/template/template1.xml

```
- <ROOT xmlns:sql="urn:schemas-microsoft-com:xml-sql">
-   <Customers CustomerID="ALFKI" CompanyName="Alfreds
                - Futterkiste" />
-  <Customers CustomerID="ANATR" CompanyName="Ana Trujillo
                                Emparedados y helados" />
```

Esempio 2: stored proc

```
<ROOT xmlns:sql="urn:schemas-microsoft-com:xml-
   sql">
          <sql:query> exec CategoryInfo </sql:query>
</ROOT>
```

```
<ROOT xmlns:sql="urn:schemas-microsoft-com:xml-sql">
      <Categories CategoryName="Condiments" />
 </ROOT>
```

Esempio 3: stored proc con parametri

```
CREATE PROCEDURE CategoryInfoWithInputParam
   @CategoryName varchar(35)
AS
SELECT CategoryName, Description
FROM Categories
WHERE Categories.CategoryName =
   @CategoryName
FOR XML AUTO
```

Esempio 3 (continua)

```
<ROOT xmlns:sql='urn:schemas-microsoft-com:xml-sql'>
 <sql:header>
    <sql:param name='CategoryName'>
            Condiments
    </sql:param>
</sql:header>
<sql:query >
    exec CategoryInfoWithInputParam @CategoryName
</sql:query>
</ROOT>
```

Template e HTML

- I template possono anche essere attivati da form HTML
- idea:
  - si associa il template ad un campo
  - al submit, questa informazione viene inviata al server
  - il server esegue il template specificato
- conviene usare il metodo POST, per non avere problemi con la lunghezza del template
  inviatoAction: indirizzo directory virtuale

Esempio

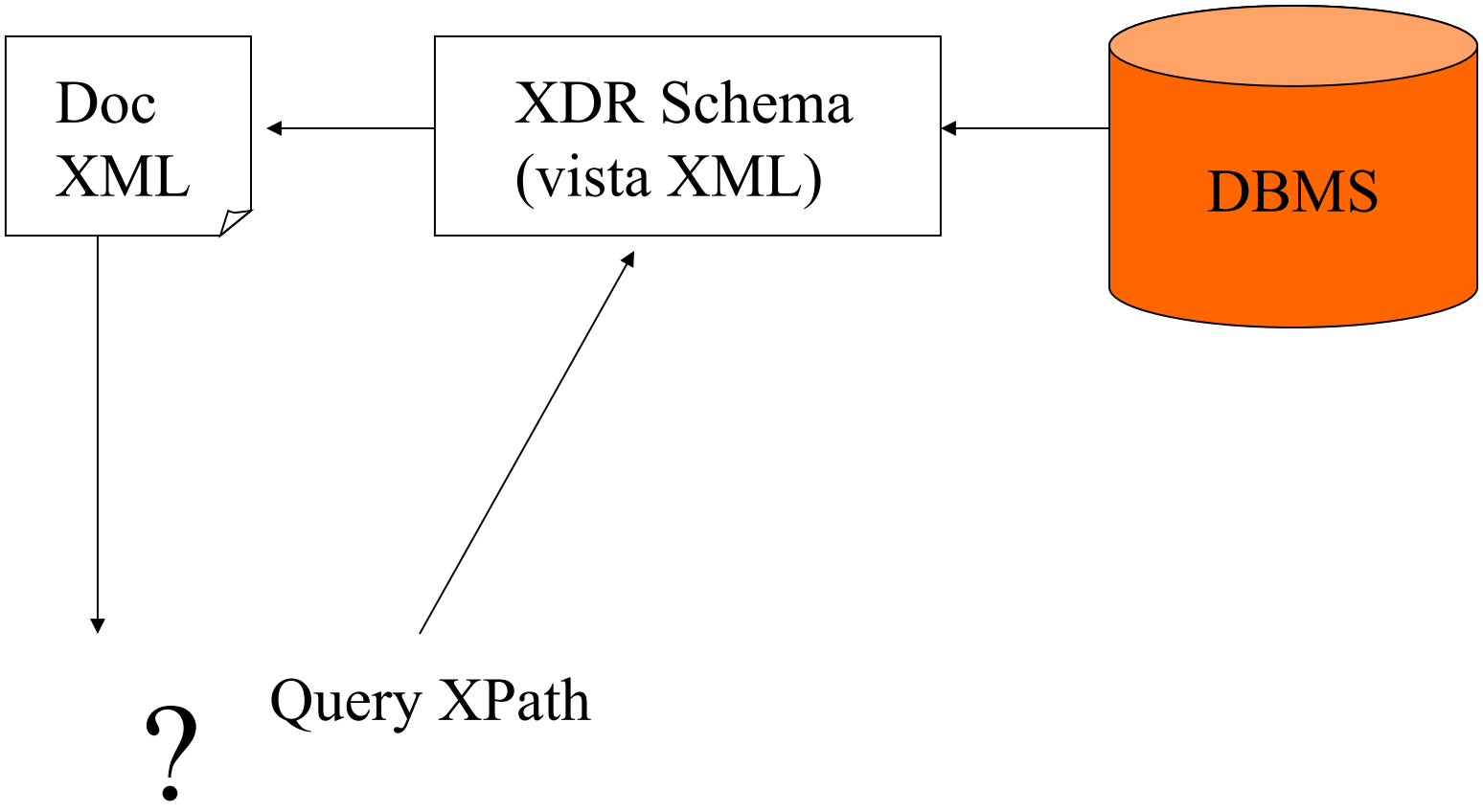
```
<head>
<TITLE>Sample Form </TITLE>
</head>
<body>
For a given employee ID, employee first and last name is retrieved.
<form action="http://IISServer/myDB" method="POST">
<B>Employee ID Number</B>
<input type=text name=EmployeeID value='1'>
<input type=hidden name=contenttype value=text/xml>
<input type=hidden name=template value='
<ROOT xmlns:sql="urn:schemas-microsoft-com:xml-sql" >
<sql:query>
  SELECT FirstName, LastName
  FROM    Employees
  WHERE    EmployeeID= 1
  FOR XML AUTO
</sql:query>
</ROOT>
'>
<input type="submit">
</form>
</body>
```

Esecuzione query XPath

- Xpath può essere utilizzato in SQL Server per interrogare documenti XML
- tali documenti devono essere creati definendo opportune viste XML sulla base di dati
  - tali viste possono essere specificate utilizzando gli schemi XML-Data Reduced (XDR)

- Quindi, in SQL Server per utilizzare Xpath, è necessario:
  - Creare viste XML del database tramite schemi XML-Data Reduced (XDR)
  - Interrogare tali viste con Xpath

Idea di base



Viste XML

- E’ possibile creare viste XML del contenuto del database utilizzando XML-Data Reduced Schema (XDR)
- Uno schema XDR descrive la struttura del documento XML che si vuole generare, quindi descrive la vista XML che si vuole eseguire sulla base di dati
- A differenza del DTD, mi permette di descrivere la struttura di un documento XML utilizzando la sintassi XML
- La struttura del documento deve quindi essere mappata sulla struttura del database

Viste XML

- Se si vogliono recuperare dati da una sola tabella e se I nomi delle tabelle e dei campi dai quali vengono estratti I dati coincide con il nome degli attributi e degli elementi con I quali si vuole costruire il documento XML
  - sintassi immediata
- se la condizione precedente nonè soddisfatta, si utilizzano le “annotazioni”
  - informazioni che specificano come effettuare Il mapping tra elementi e attributi XML e dati nel database

Mapping di default

```
<?xml version="1.0" ?>
 <Schema        xmlns="urn:schemas-microsoft-com:xml-data"
xmlns:dt="urn:schemas-microsoft-com:datatypes"
xmlns:sql="urn:schemas-microsoft-com:xml-sql">
<ElementType name="Employees" >
<AttributeType name="EmployeeID" />
        <AttributeType name="FirstName" />
<AttributeType name="LastName" />

<attribute type="EmployeeID" />
<attribute type="FirstName" />
<attribute type="LastName" />
</ElementType>
</Schema>
```

Risultato

```
<ROOT>
<Employees EmployeeID ="1" FirstName="N
   LastName="Davolio"></Employee>
<Employees EmployeeID ="2" FirstName="A
   LastName="Fuller"></Employee>
</ROOT>
```

Tabella

nployees(EmployeeID,FirstName,LastName)

Mapping di default: sottoelementi

```
<?xml version="1.0" ?>
<Schema xmlns="urn:schemas-microsoft-com:xml-data"
        xmlns:dt="urn:schemas-microsoft-com:datatypes"
        xmlns:sql="urn:schemas-microsoft-com:xml-sql">

<ElementType name="FirstName" content="textOnly" />
<ElementType name="LastName"  content="textOnly" />

<ElementType name="Employees" >
    <AttributeType name="EmployeeID"”/>

    <attribute type="EmployeeID"/>
    <element type="FirstName"/>
    <element type="LastName"/>
</ElementType>
</Schema>
```

Risultato

```
<ROOT xmlns:sql="urn:schemas-
      microsoft-com:xml-sql">
  <Employees EmployeeID="1">
    <FirstName>Nancy</FirstName>
    <LastName>Davolio</LastName>
  </Employees>
</ROOT>
```

Tabella

nployees(EmployeeID,FirstName,LastName)

Mapping espliciti

- É possibile specificare come gli elementi e gli attributi del file XML vengono associati alle tabelle e ai campi delle tabelle
- si utilizzano le “annotazioni”
- per indicare a quale relazione si riferisce un certo elemento:
  - sql:relation
- per indicare a quale campo è associato un elemento o un attributo:
  - sql:field
- esiste un ampio numero di annotazioni, che permette di creare schemi XDR molto flessibili

Esempio 1

```
<?xml version="1.0" ?>
<Schema xmlns="urn:schemas-microsoft-com:xml-data"
        xmlns:dt="urn:schemas-microsoft-com:datatypes"
        xmlns:sql="urn:schemas-microsoft-com:xml-sql">
  <ElementType name="Customer" sql:relation="Customers" >
    <AttributeType name="CustomerID" />
    <AttributeType name="ContactName" />
    <AttributeType name="Phone" />

    <attribute type="CustomerID" />
    <attribute type="ContactName" />
    <attribute type="Phone" />
  </ElementType>
</Schema>
```

Risultato

```
<ROOT xmlns:sql="urn:schemas-
        microsoft-com:xml-sql">
  <Customer CustomerID="ALFKI"
        ContactName="Maria Ande
        Phone="030-0074321" />
</ROOT>
```

Tabella

ustomers(CustomerID,ContactName,Phone)

Esempio 2

```
<?xml version="1.0" ?>
<Schema xmlns="urn:schemas-microsoft-com:xml-data"
        xmlns:dt="urn:schemas-microsoft-com:datatypes"
        xmlns:sql="urn:schemas-microsoft-com:xml-sql">
  <ElementType name="Customer" sql:relation="Customers" >
    <AttributeType name="CustomerID" />
    <AttributeType name="ContactName” sql:field= “Contact” />
    <AttributeType name="Phone" />

    <attribute type="CustomerID" />
    <attribute type="ContactName" />
    <attribute type="Phone" />
  </ElementType>
</Schema>
```

Risultato

```
<ROOT xmlns:sql="urn:schemas-
        microsoft-com:xml-sql">
  <Customer CustomerID="ALFKI"
        ContactName="Maria Ande
        Phone="030-0074321" />
</ROOT>
```

Tabella

ustomers(CustomerID,Contact,Phone)

Generazione automatica di schemi XDR

- É possibile generare schemi XDR in modo automatico, eseguendo query SQL
- Idea: eseguo una query SQL specificando che
  - il risultato deve essere formattato in XML
  - lo schema del risultato deve essere formattato con XDR schema
- Clausola FOR XML <modalità>, XMLDATA

Esempio

```
SELECT TOP 2 Customers.CustomerID, Orders.OrderID
FROM Customers, Orders
WHERE Customers.CustomerID = Orders.CustomerID
ORDER BY Customers.CustomerID
FOR XML RAW , XMLDATA
```

RISULTATO

```
<Schema name="Schema3" xmlns="urn:schemas-microsoft-com:xml-data"
    xmlns:dt="urn:schemas-microsoft-com:datatypes">
  <ElementType name="row" content="empty">
    <AttributeType name="CustomerID" dt:type="string"/>
    <AttributeType name="OrderID" dt:type="i4"/>

    <attribute type="CustomerID"/>
    <attribute type="OrderID"/>
  </ElementType>
</Schema>

<row xmlns="x-schema:#Schema3" CustomerID="ALFKI" OrderID="10643" />
<row xmlns="x-schema:#Schema3" CustomerID="ALFKI" OrderID="10692" />
```

Come interrogare le viste XDR?

- É possibile utilizzare Xpath per interrogare le viste XDR
- la query Xpath deve essere rappresentata:
  - nell'URL
  - all'interno di un template
- per potere essere interrogati, gli schemi XDR devono essere memorizzati in una directory di tipo *schema*, all'interno della directory reale corrispondente alla directory virtuale
- il template deve indicare su quale schema deve essere eseguito

Utilizzo di Xpath query

- Direttamente nell'URL
  - http://paperino/myDB/schema/

    XDRSchema1.xml/Customer[@CustomerID="ALFKI"]?root=ROOT

- all'interno di un template

```
<ROOT xmlns:sql="urn:schemas-microsoft-com:xml-sql">
  <sql:xpath-query mapping-schema=”nomefileXDRschema.xml">
    /Customer[@CustomerID="ALFKI"]
  </sql:xpath-query>
</ROOT>
```

Esempio

- XDR Schema:

```
<?xml version="1.0" ?>
    <Schema        xmlns="urn:schemas-microsoft-com:xml-data"  xmlns:dt="urn:schemas-
   microsoft-com:datatypes"         xmlns:sql="urn:schemas-microsoft-com:xml-sql">
   <ElementType name="Employees" >
   <AttributeType name="EmployeeID" />
          <AttributeType name="FirstName" />
   <AttributeType name="LastName" />

   <attribute type="EmployeeID" />
   <attribute type="FirstName" />
   <attribute type="LastName" />
   </ElementType>
</Schema>
```

- Xpath: /Employees[@EmployeeID=”1"]

Esempio (continua)

- Template:

```
<ROOT xmlns:sql="urn:schemas-microsoft-com:xml-sql">
  <sql:xpath-query mapping-schema="SampleSchema1.xml">
    /Employees[@EmployeeID=”1"]
  </sql:xpath-query>
</ROOT>
```

- Possibile risultato

```
<ROOT>

<Employees EmployeeID ="1" FirstName="Nancy ”
   LastName="Davolio"></Employee>

</ROOT>
```

Esercitazione proposta

- Generare una directory virtuale associata al proprio database
- provare ad eseguire query SQL da HTTP
- definire un template per eseguire uno statement SQL e provare ad eseguirlo
- definire semplici documenti XML ed utilizzare OPENXML per formattarne il contenuto in forma relazionale ed inserirlo in una tabella esistente
- definire query sul database costruito e generare il risultato in formato XML
- costruire un semplice schema XDR per recuperare alcuni dati da una vostra tabella; interrogare quindi lo schema utilizzando un template Xpath e un’espressione di comando direttamente da URL