Atzeni, Ceri, Paraboschi, Torlone

Basi di dati

McGraw-Hill, 1996-2002

Capitolo 5:

SQL nei linguaggi di programmazione

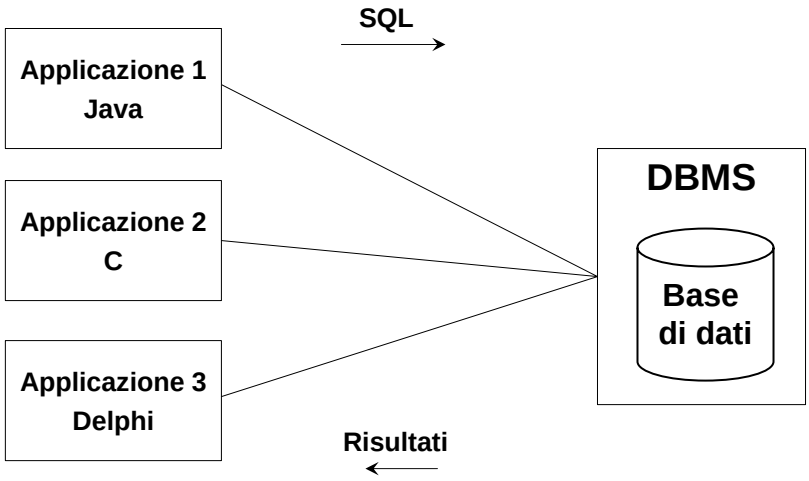
SQL e applicazioni

- **In applicazioni complesse, l'utente non vuole eseguire comandi SQL, ma programmi, con poche scelte**
- **SQL non basta, sono necessarie altre funzionalità, per gestire:**
  - **input (scelte dell'utente e parametri)**
  - **output (con dati che non sono relazioni o se si vuole una presentazione complessa)**
  - **per gestire il controllo**

SQL e linguaggi di programmazione

- Le applicazioni sono scritte in
  - linguaggi di programmazione tradizionali:
    - Cobol, C, Java, Fortran
  - linguaggi “ad hoc”, proprietari e non:
    - PL/SQL, Informix4GL, Delphi
- Vediamo solo l’approccio “tradizionale”, perché più generale

Applicazioni ed SQL: architettura



Una difficoltà importante

- **Conflitto di impedenza ("impedance mismatch") fra base di dati e linguaggio**
  - **Linguaggi di programmazione: accedono agli elementi di una tabella con un approccio *tuple-oriented* (opera su singole variabili o oggetti)**
  - **SQL: linguaggio di tipo set-oriented (opera su intere tabelle)**

Altre differenze, 1

- **Modalità di accesso ai dati e correlazione:**
  - **linguaggio: dipende dal paradigma e dai tipi disponibili; ad esempio scansione di liste o "navigazione" tra oggetti**
  - **SQL: join (ottimizzabile)**

Altre differenze, 2

- **Definizione dei tipi di base:**
  - **linguaggi: numeri, stringhe, booleani**
  - **SQL: CHAR, VARCHAR, DATE, ...**
- **Definizione dei costruttori di tipo:**
  - **linguaggio: dipende dal paradigma**
  - **SQL: relazioni e ennuple**

SQL e linguaggi di programmazione: tecniche principali

Diverse soluzioni per consentire l'uso di SQL all'interno di un normale linguaggio di programmazione:

1. **SQL immerso ("Embedded SQL")**
   - **sviluppata sin dagli anni '70**
   - **"SQL statico"**
2. **SQL dinamico**
3. **Call Level Interface (CLI)**
   - **più recente**
   - **SQL/CLI, ODBC, JDBC**

1. SQL immerso (Embedded)

- le istruzioni SQL sono “immerse” nel programma redatto nel linguaggio “ospite”
- un precompilatore (legato al DBMS) viene usato per analizzare il programma e tradurlo in un programma nel linguaggio ospite (sostituendo le istruzioni SQL con chiamate alle funzioni di una API del DBMS)

SQL immerso, un esempio

```
#include<stdlib.h>
main(){
  exec sql begin declare section;
    char *NomeDip = "Manutenzione";
    char *CittaDip = "Pisa";
    int NumeroDip = 20;
  exec sql end declare section;

  exec sql connect to utente@librobd;
  if (sqlca.sqlcode != 0) {
    printf("Connessione al DB non riuscita\n"); }
  else {
    exec sql insert into Dipartimento
           values(:NomeDip,:CittaDip,:NumeroDip);
    exec sql disconnect all;
  }
}
```

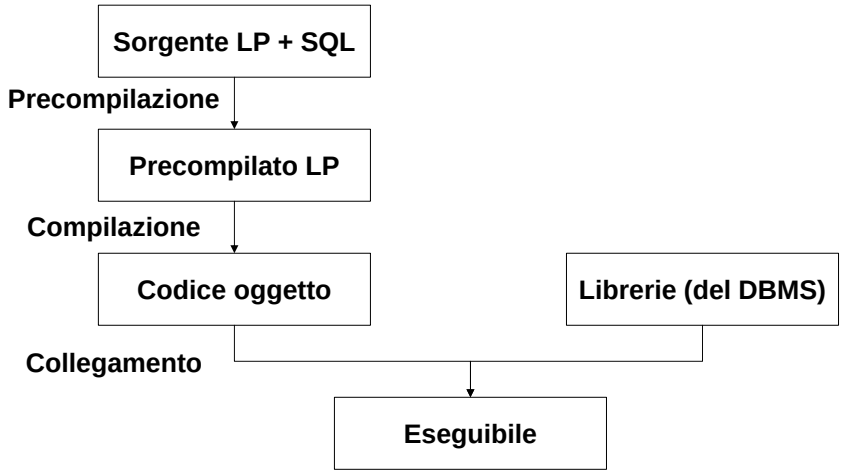
SQL immerso, commenti al codice

- EXEC SQL denota le porzioni di interesse del precompilatore:
  - definizioni dei dati
  - istruzioni SQL
- le variabili del programma possono essere usate come “parametri” nelle istruzioni SQL (precedute da “:”) dove sintatticamente sono ammesse costanti

SQL immerso, commenti al codice, 2

- `sqlca(SQL Communication Area)` è una struttura dati per la comunicazione fra programma e DBMS
- `sqlcode` è un campo di `sqlca` che mantiene il codice di errore dell’ultimo comando SQL eseguito:
  - zero: successo
  - altro valore: errore o anomalia

SQL immerso, fasi



Un altro esempio

```
int main() {
  exec sql connect to universita
    user pguser identified by pguser;
  exec sql create table studente
   (matricola integer primary key,
    nome varchar(20),
    annodicorso integer);
  exec sql disconnect;
  return 0;
}
```

L’esempio “precompilato”

```
/* These include files are added by the preprocessor */
#include <ecpgtype.h>
#include <ecpglib.h>
#include <ecpgerrno.h>
#include <sqlca.h>
int main() {
  ECPGconnect(__LINE__, "universita" , "pguser" ,
    "pguser" , NULL, 0);
  ECPGdo(__LINE__, NULL, "create table studente (
  matricola integer primary key , nome varchar ( 20 ) ,
  annodicorso integer )", ECPGt_EOIT, ECPGt_EORT);
  ECPGdisconnect(__LINE__, "CURRENT");
  return 0;
}
```

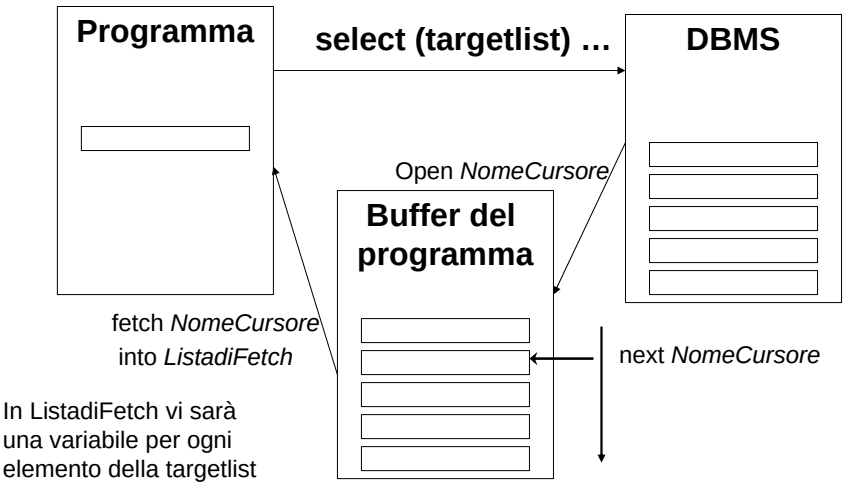

Note

- **Il precompilatore è specifico della combinazione**
  linguaggio-DBMS-sistema operativo

Interrogazioni in SQL immerso: conflitto di impedenza

- **Il risultato di una select è costituito da zero o piú ennuple:**
  - **zero o una: ok -- l'eventuale risultato puó essere gestito in un record**
  - **piú ennuple: come facciamo?**
    - **l'insieme (in effetti, la lista) non è gestibile facilmente in molti linguaggi**
- **Una soluzione: il Cursore**
  tecnica per trasmettere al programma una ennupla alla volta

Cursore



In ListadiFetch vi sarà una variabile per ogni elemento della targetlist della select (di tipo compatibile)

Nota

- **Il cursore**
  - **accede a tutte le ennuple di una interrogazione in modo globale (tutte insieme o a blocchi – è il DBMS che sceglie la strategia efficiente)**
  - **trasmette le ennuple al programma una alla volta**

Operazioni sui cursori

Definizione del cursore
declare NomeCursore [scroll] cursor for Select …

Esecuzione dell'interrogazione
open NomeCursore

Utilizzo dei risultati (una ennupla alla volta)
fetch NomeCursore into ListaVariabili

Disabilitazione del cursore
close cursor NomeCursore

Accesso alla ennupla corrente (di un cursore su singola relazione a fini di aggiornamento)
current of NomeCursore
nella clausola where

```
write('nome della citta''?');
readln(citta);
EXEC SQL DECLARE P CURSOR FOR
  SELECT NOME, REDDITO
  FROM PERSONE
  WHERE CITTA = :citta ;
EXEC SQL OPEN P ;
EXEC SQL FETCH P INTO :nome, :reddito ;
while SQLCODE = 0
do begin
      write('nome della persona:', nome, 'aumento?');
      readln(aumento);
      EXEC SQL UPDATE PERSONE
                  SET REDDITO = REDDITO + :aumento
                  WHERE CURRENT OF P
      EXEC SQL FETCH P INTO :nome, :reddito
  end;
EXEC SQL CLOSE CURSOR P
```



```
void VisualizzaStipendiDipart(char NomeDip[])
{
  char Nome[20], Cognome[20];
  long int Stipendio;
  $ declare ImpDip cursor for
      select Nome, Cognome, Stipendio
      from Impiegato
      where Dipart = :NomeDip;
  $ open ImpDip;
  $ fetch ImpDip into :Nome, :Cognome, :Stipendio;
  printf("Dipartimento %s\n",NomeDip);
  while (sqlcode == 0)
  {
      printf("Nome e cognome dell'impiegato: %s
                  %s",Nome,Cognome);
      printf("Attuale stipendio: %d\n",Stipendio);
      $ fetch ImpDip into :Nome, :Cognome,
  :Stipendio;
  }
  $ close cursor ImpDip;
}
```

Cursori, commenti

- **Per aggiornamenti e interrogazioni "scalari" (cioè che restituiscano una sola ennupla) il cursore non serve:**

```
select Nome, Cognome
   into :nomeDip, :cognomeDip
from Dipendente
where Matricola = :matrDip;
```


Cursori, commenti, 2

- **I cursori possono far scendere la programmazione ad un livello troppo basso, pregiudicando la capacità dei DBMS di ottimizzare le interrogazioni:**
  - **se "nidifichiamo" due o più cursori, rischiamo di reimplementare il join!**

Esercizio

Studenti(Matricola, Cognome, Nome)
Esami(Studente,Materia,Voto,Data)
Corsi(Codice,Titolo)
con gli ovvî vincoli di integrità referenziale

- **Stampare, per ogni studente, il certificato con gli esami e il voto medio**

Output

Matricola Cognome Nome
Materia Data Voto
…
Materia Data Voto
VotoMedio
Matricola Cognome Nome
Materia Data Voto
…
Materia Data Voto
VotoMedio
…

Esercizio

Studenti(Matricola, Cognome, Nome)
Esami(Studente,Materia,Voto,Data)
Corsi(Codice,Titolo)
Iscrizioni(Studente,AA,Anno,Tipo)
con gli ovvî vincoli di integrità referenziale

- **Stampare, per ogni studente, il certificato con gli esami e le iscrizioni ai vari anni accademici**

Output

Matricola Cognome Nome
AnnoAccademico AnnoDiCorso TipoIscrizione
…
AnnoAccademico AnnoDiCorso TipoIscrizione
Materia Data Voto
…
Materia Data Voto
Matricola Cognome Nome
AnnoAccademico AnnoDiCorso TipoIscrizione
…
AnnoAccademico AnnoDiCorso TipoIscrizione
Materia Data Voto
…
Materia Data Voto

2. SQL dinamico

- **Non sempre le istruzioni SQL sono note quando si scrive il programma (definire al momento dell'esecuzione le interrogazioni da effettuare sulla base dati)**
- **Allo scopo, è stata definita una tecnica completamente diversa, chiamata *Dynamic SQL* che permette di eseguire istruzioni SQL costruite dal programma (o addirittura ricevute dal programma attraverso parametri o da input)**
- **Non è banale gestire i parametri e la struttura dei risultati (non noti a priori)**

SQL dinamico

- **Le operazioni SQL possono essere:**
  - **eseguite immediatamente**
    `execute immediate` *SQLStatement*
  - **prima "preparate":**
    `prepare` *CommandName* `from` *SQLStatement*
    e poi eseguite (anche più volte):
    `execute` *CommandName* [`into` *TargetList*]
    [`using` *ParameterList*]

E' possibile l'uso di cursori

3. Call Level Interface

- Indica genericamente interfacce che permettono di inviare richieste a DBMS per mezzo di parametri trasmessi a funzioni
- Il programmatore ha a disposizione una libreria di funzioni che vengono invocate per realizzare il dialogo con la base di dati
- standard SQL/CLI (’95 e poi parte di SQL:1999)
- ODBC: implementazione proprietaria di SQL/CLI
- JDBC: una CLI per il mondo Java

SQL immerso e CLI

- SQL immerso permette
  - precompilazione (e quindi efficienza)
  - uso di SQL completo
- CLI
  - indipendente dal DBMS
  - permette di accedere a più basi di dati, anche eterogenee

ODBC, OLE DB e ADO

- **ODBC (Open DataBase Connectivity), interfaccia applicativa Microsoft per costruire applicazioni che facciano accesso a basi di dati relazioni di tipo eterogeneo.**
- **Le applicazioni possono accedere a dati presenti su sistemi relazionali generici, eventualmente su sistemi remoti.**
- **Il linguaggio SQL supportato da ODBC è particolarmente "ristretto", insieme minimo di istruzioni**

ODBC, OLE DB e ADO

- **Il collegamento tra una applicazione e un server richiede l'uso di un *driver*, libreria collegata alle applicazioni e da essa invocata.**
- **E' il *driver* che maschera le differenze di interazione legate al DBMS, al sistema operativo e al protocollo di rete utilizzato.**
- **Ciascun venditore deve garantire dei driver che prevedano l'uso del DBMS all'interno di una specifica rete - sistema operativo**

ODBC, OLE DB e ADO

L'accesso ai dati tramite ODBC richiede la cooperazione di 4 componenti:

1. **Applicazione: richiama le funzioni SQL per eseguire interrogazioni ed ottenere dati**
2. **Driver manager: carica i driver su richiesta dell'applicazione**
3. **Driver: eseguono le interrogazioni SQL e restituiscono i risultati alle applicazioni**
4. **Fonte di informazione (data-source): sistema che esegue le funzioni trasmesse dal client**

I codici di errore sono standardizzati per consentire il controllo durante l'esecuzione

ODBC, OLE DB e ADO

OLE DB e ADO sono due specifiche dell'architettura ODBC, nate per semplificare la realizzazione di applicazioni di accesso ai dati

- **OLE DB: permette di accedere a sorgenti dati di tipo generico, non solo sistemi relazionali ma altre tipologie di archivi, come caselle di posta elettronica o sistemi CAD/CAM.**
- **ADO (ActiveX Data Object) interfaccia di alto livello ai servizi offerti da OLE DB.**

ODBC, OLE DB e ADO

ADO si basa su 4 concetti fondamentali:

1. **Connessione: canale di comunicazione da stabilire per interagire con una sorgente di dati**
2. **Comando: stringa di caratteri che contiene l'istruzione SQL da eseguire**
3. **Tupla: (record) descrive la singola riga di una tabella e fornisce strumenti per gestire la struttura dei dati che la compongono**
4. **Insieme di tuple: (recordset) struttura che conserva i risultati dei comandi inviati alla sorgente dati, offre meccanismi per scandire l'insieme di tuple e accedere ad una tupla per volta (come per i cursori)**


```
Public SUB InterrogaDB()
   Dim setImpiegati AS ADODB.Recordset
   Dim conn AS ADODB.Connection
   …
   Set conn = New ADODB.Connection
   conn.Open "mioServer", "giovanni", "passord"
   Set setImpiegati = New ADODB.Recordset
   stringaSQL = "select Nome, Cognome, Data from
                  impiegato order by cognome"
   comSQL.commandText = stringaSQL
   setImpiegati.Open comSQL, conn, , ,
   do while not setImpiegato.EOF
      messaggio = "Impiegato: " & setImpiegati!nome & _
            setImpiegati!cognome & " (record " & _
            setImpiegati.AbsolutePosition & " di " & _
            setImpiegati.RecordCount & ")"
      if MsgBox(messaggio, vbOkCancel) = vbCancel then
         Exit Do
      setImpiegati.MoveNext
   loop
   setImpiegati.close
   conn.close
End Sub
```

JDBC

- **Una API (Application Programming Interface) di Java (intuitivamente: una libreria) per l'accesso a basi di dati, in modo indipendente dalla specifica tecnologia**
- **JDBC è una interfaccia, realizzata da classi chiamate driver**:
  - **l'interfaccia è standard, mentre i driver contengono le specificità dei singoli DBMS (o di altre fonti informative)**

I driver JDBC

- **(A titolo di curiosità; può bastare il primo tipo)**
  Esistono quattro tipi di driver (chiamati, in modo molto anonimo, tipo 1, tipo 2, tipo 3, tipo 4):
  1. **Bridge JDBC-ODBC: richiama un driver ODBC, che deve essere disponibile sul client; è comodo ma potenzialmente inefficiente**
  2. **Driver nativo sul client: richiama un componente proprietario (non necessariamente Java) sul client**
  3. **Driver puro Java con server intermedio ("middleware server"): comunica via protocollo di rete con il server intermedio, che non deve risiedere sul client**
  4. **Driver puro Java, con connessione al DBMS: interagisce direttamente con il DBMS**

Il funzionamento di JDBC, in breve

Per utilizzare i servizi di JDBC si seguono i passi seguenti:

1. **Caricamento del driver**
2. **Apertura della connessione alla base di dati**
3. **Richiesta di esecuzione di istruzioni SQL**
4. **Elaborazione dei risultati delle istruzioni SQL**

Un programma con JDBC

```
import java.sql.*;
public class PrimoJDBC {
   public static void main(String[] arg){
     Connection con = null ;
     try {
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
        String url = "jdbc:odbc:Corsi";
        con = DriverManager.getConnection(url);
        }
     catch(Exception e){
        System.out.println("Connessione fallita");
        }
     try {
        Statement query = con.createStatement();
        ResultSet result =
                query.executeQuery("select * from Corsi");
        while (result.next()){
                String nomeCorso = result.getString("NomeCorso");
                System.out.println(nomeCorso);
                }
        }
     catch (Exception e){
        System.out.println("Errore nell'interrogazione");
     }
   }
}
```

Preliminari

- L'interfaccia JDBC è contenuta nel package `java.sql`

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

- Il driver deve essere caricato (trascuriamo i dettagli)

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

- Connessione: oggetto di tipo `Connection` che costituisce un collegamento attivo fra programma Java e base di dati; viene creato da

```
String url = "jdbc:odbc:Corsi";
con = DriverManager.getConnection(url);
```


Preliminari dei preliminari: origine dati ODBC

- Per utilizzare un driver JDBC-ODBC, la base di dati (o altro) deve essere definita come "origine dati ODBC"
- In Windows (con YYY, avendo già definito la base di dati xxx.yyy da collegare):
  - Pannello di controllo
  - Strumenti di amministrazione
  - Opzione "Origini dati ODBC"
  - Bottone "Aggiungi" ("Add")
  - Nella finestra di dialogo "Crea Nuova origine dati" selezionare “YYY Driver" e nella successiva
    - selezionare il file xxx.yyy
    - attribuirgli un nome (che sarà usato da ODBC e quindi da JDBC)

Esecuzione dell'interrogazione ed elaborazione del risultato

Esecuzione dell'interrogazione

```
Statement query = con.createStatement();
ResultSet result =
  query.executeQuery("select * from Corsi");
```

Elaborazione del risultato

```
while (result.next()){
     String nomeCorso =
          result.getString("NomeCorso");
     System.out.println(nomeCorso);
}
```


JDBC: Classe Statement

- Un'interfaccia i cui oggetti consentono di inviare, tramite una connessione, istruzioni SQL e di ricevere i risultati forniti
- Un oggetto di tipo `Statement` viene creato con il metodo `createStatement` della classe `Connection`
- I metodi dell'interfaccia `Statement`:
  - `executeUpdate` per specificare aggiornamenti o istruzioni DDL
  - `executeQuery` per specificare interrogazioni e ottenere un risultato
  - `execute` per specificare istruzioni non note a priori
  - `executeBatch` per specificare sequenze di istruzioni

JDBC: Classe ResultSet

- I risultati delle interrogazioni sono forniti in oggetti di tipo `ResultSet` (interfaccia definita in `java.sql`)
- In sostanza, un result set è una sequenza di tuple su cui si può "navigare" (in avanti, indietro e anche con accesso diretto) e dalla cui tupla "corrente" si possono estrarre i valori degli attributi
- Metodi principali:
  - `next()`
  - `get`*`XXX`*`(`*`posizione`*`)`
    - es: `getString(3); getInt(2)`
  - `get`*`XXX`*`(`*`nomeAttributo`*`)`
    - es: `getString("Cognome"); getInt("Codice")`

Specializzazioni di Statement

- `PreparedStatement` premette di utilizzare codice SQL già compilato, eventualmente parametrizzato rispetto alle costanti
  - in generale più efficiente di `Statement`
  - permette di distinguere più facilmente istruzioni e costanti (e apici nelle costanti)

  i metodi `set`*`XXX`*`( , )` permettono di definire i parametri
- `CallableStatement` premette di utilizzare "stored procedure", come quelle di Oracle PL/SQL o anche le query memorizzate (e parametriche) di Access

```
import java.sql.*;
import javax.swing.JOptionPane;
public class SecondoJDBCprep {
   public static void main(String[] arg){
      try {
         Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
         String url = "jdbc:odbc:Corsi";
         Connection con = DriverManager.getConnection(url);
         PreparedStatement pquery = con.prepareStatement(
              "select * from Corsi where NomeCorso LIKE ?");
        String param = JOptionPane.showInputDialog(
                    "Nome corso (anche parziale)?");
         param = "%" + param + "%";
         pquery.setString(1,param);
         ResultSet result = pquery.executeQuery();
         while (result.next()){
           String nomeCorso = result.getString("NomeCorso");
           System.out.println(nomeCorso);
           }
       }
       catch (Exception e){System.out.println("Errore");}
   }
}
```



```
import java.sql.*;
import javax.swing.JOptionPane;

public class TerzoJDBCcall {
   public static void main(String[] arg){
      try {
         Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
         String url = "jdbc:odbc:Corsi";
         Connection con = DriverManager.getConnection(url);
         CallableStatement pquery =
              con.prepareCall("{call queryCorso(?)}");
               String param = JOptionPane.showInputDialog(
                     "Nome corso (anche parziale)?");
               param = "*" + param + "*";
               pquery.setString(1,param);
         ResultSet result = pquery.executeQuery();
         while (result.next()){
              String nomeCorso =
                    result.getString("NomeCorso");
              System.out.println(nomeCorso);
              }
      }
      catch (Exception e){System.out.println("Errore");}
   }
}
```

Altre funzionalità

- Molte, fra cui
  - username e password
  - aggiornamento dei ResultSet
  - richiesta di metadati
  - gestione di transazioni

Transazioni in JDBC

- Scelta della modalità delle transazioni: un metodo definito nell'interfaccia `Connection`:
  `setAutoCommit(boolean autoCommit)`
- `con.setAutoCommit(true)`
  - (default) "autocommit": ogni operazione è una transazione
- `con.setAutoCommit(false)`
  - gestione delle transazioni da programma
    `con.commit()`
    `con.rollback()`
  - non c'è `begin transaction`

Procedure

- SQL:1999 (come già SQL-2) permette la definizione di procedure e funzioni (chiamate genericamente “stored procedures”)
- Le stored procedures sono parte dello schema

```
procedure AssignCity(:Dep char(20), :City char(20))
   update Department
   set City = :City
   where Name = :Dep
```

- Lo standard prevede funzionalità limitate e non è molto recepito
- Molti sistemi offrono estensioni ricche (ad esempio Oracle PL/SQL e Sybase-Microsoft Transact SQL)

Oracle PL/SQL: un esempio

```
create or replace procedure aco_scheda_doc(id_doc IN integer) AS
doc_line        varchar2(4000);
n_sup           integer;
n_eve           integer;
n_similaudio    integer;
BEGIN
   aco_dati_scheda_doc(id_doc);
   aco_count_evedoc(id_doc, n_eve);
   aco_count_supdoc(id_doc, n_sup);
   aco_count_similaudiodoc(id_doc, n_similaudio);
   ……
   if (n_sup > 0) then
      doc_line := 'aco_lista_supdoc?id_doc='||id_doc||'&n_eve='||n_eve||‘
                  &n_sup='||n_sup||'&n_similaudio='||n_similaudio;
      htp.print('<tr><td width="252" height="20" bgcolor="#465d74" align="left">
                  <a href="'||doc_line||'">Supporti Collegati</a></td>
                    <td width="*" height="20”>'||to_char(n_sup)||'</td></tr>');
   else
      htp.print('<tr><td width="252" height="20" bgcolor="#465d74" align="left">
                  Supporti Collegati</td>
                    <td width="*"> 0</td></tr>');
   end if;
   ……
   htp.print('</table></body></html>');
END aco_scheda_doc;
/
show errors
```

Oracle PL/SQL: un esempio

```
create or replace procedure aco_dati_scheda_doc(id_doc IN integer) AS
doc_line        varchar2(4000);
tab_query       integer;
BEGIN
   htp.print('<html><head><title>== AESS - Scheda Documento N.: '||id_doc||'
                ==</title>……');

   query_str := 'SELECT DISTINCT formalizzato, descr_titolo_aco, descr_stato,
                                 descr_regione, descr_provincia,
                                          ……
                                 descr_contenuti2, audio, testo, pentagramma
                       FROM ut08view.documenti_t3
                       WHERE (id_documento = '||id_doc||')';

   tab_query := owa_util.bind_variables(query_str);
   colCnt := dbms_sql.execute(tab_query);
   LOOP
      status := dbms_sql.fetch_rows(tab_query);
      if (status <= 0) then
         exit;
      end if;

      dbms_sql.column_value(tab_query, 1, doc_line);
      dbms_sql.column_value(tab_query, 2, doc_line);
      dbms_sql.column_value(tab_query, 3, doc_line);
       ……
   END LOOP;
   dbms_sql.close_cursor(tab_query);
```

Oracle PL/SQL: un esempio

```
-- conta quanti eventi sono legati al documento "id_doc"
-------------------------------------------------------------
create or replace procedure aco_count_evedoc(id_doc IN integer, n_eve_found OUT
   integer) AS
BEGIN
   n_eve_found := 0;

   SELECT count(DISTINCT sigla_evento) into n_eve_found
      FROM ut08view.link_documenti_eventi
      WHERE id_documento = id_doc;
END aco_count_evedoc;
/
show errors

-- conta quanti supporti sono legati al documento "id_doc“
-------------------------------------------------------------
create or replace procedure aco_count_supdoc(id_doc IN integer, n_doc_found OUT
   integer) AS
BEGIN
   n_doc_found := 0;

   SELECT count(DISTINCT id_supporto) into n_doc_found
      FROM ut08view.link_documenti_supporti
      WHERE id_documento = id_doc;
END aco_count_supdoc;
/
show errors
```

Oracle PL/SQL: un esempio