Capitolo 5

Prof. Stefano Paraboschi

Funzioni scalari in SQL

- Costrutti definiti in SQL-2 che estendono la sintassi delle espressioni nella target list
  - `coalesce`
  - `nullif`
  - `case`
- Vengono valutate per ogni riga estratta dalla query
- Non aumentano il potere espressivo, ma semplificano la scrittura delle query (altrimenti bisognerebbe fare un'unione di diverse query)

`coalesce`

- Sintassi
  - `coalesce` (*list*)
- Descrizione
  - restituisce il primo valore non nullo nella lista
- Esempio:

```
select Nome,
       coalesce(DataNasc,'Non disponibile')
from Persona
```

`nullif`

- Sintassi
  - `nullif` (*input,value*)
- Descrizione
  - restituisce il valore nullo se l'espressione *input* ha come valore *value*, altrimenti restituisce *input*
- Esempio:

```
select Nome, nullif(DataNasc,'Ignota')
from Persona
```

`case`

- Sintassi
  - `case when` *Condizione* `then` *Espr*
    { `when` *Condizione* `then` *Espr* }
    `else` *Espr* `end`
- Descrizione
  - restituisce l'espressione associata alla prima condizione soddisfatta
- Esempio:

```
select Nome, case when Voto <= 10 then 'Grav. Ins.'
                  when Voto <= 17 then 'Ins.'
                  else to_char(Voto) end
from Esame
```

Altre funzioni scalari

- Funzioni matematiche:
  - `abs(), degrees(), exp(), ln(), log(), radians(), round(), sqrt(), trunc(), float(), integer(), acos(), asin(), atan(), cos(), cot(), sin(), tan()`
- Funzioni su stringhe
  - `char_length(), lower(), upper(), substring(), trim(), …`

Altre funzioni scalari

- Funzioni su dati temporali
  - `now, today` (variabili predefinite)
  - `abstime(), age(), date_part(), date_trunc(), to_char()`

  per `date_part()` e `date_trunc()` la porzione può essere
  - `year, month, day, hour, minute, second, decade, century, millennium, millisecond, microsecond, dow, week, epoch`

Altre funzioni scalari

- Normalmente i sistemi estendono ulteriormente questo insieme di funzioni
  - Funzioni per la formattazione dell'output
  - Funzioni geometriche
  - Funzioni di accesso ad alcuni servizi del sistema operativo

Uso di SQL nelle applicazioni

- L'uso diretto di SQL è riservato a relativamente pochi utenti delle basi di dati
- Normalmente si accede a un DBMS tramite un'applicazione
- L'applicazione può essere realizzata con
  - Linguaggi di quarta generazione (4GL)
    - soluzioni specializzate, spesso integrate con un particolare DBMS
  - Normali linguaggi di programmazione (COBOL, C, C++, Java, Basic, ...) che fanno uso di SQL
    - Problema: come combinare due diversi linguaggi di programmazione?

SQL Embedded

- Le applicazioni "rinchiudono" (embed) comandi SQL all'interno delle istruzioni di un linguaggio di programmazione procedurale (COBOL, C, etc.)
- I comandi SQL embedded sono preceduti da 'EXEC SQL' e chiusi con ';'
- Un precompilatore (o preprocessore) gestisce i comandi SQL, convertendoli in opportune chiamate di una libreria interna, normalmente specifica per il DBMS
- Il precompilatore può riconoscere eventuali anomalie nell'uso di SQL

Esempio di SQL Embedded

```
#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;
  }
}
```

Variabili in SQL Embedded

- Le variabili che realizzano il passaggio di informazioni tra il programma e l'ambiente SQL embedded devono essere dichiarate in una sezione opportuna
- Le variabili del programma sono precedute da ':' quando sono usate come parametri nei comandi SQL
- L'ambiente offre una variabile predefinita **`sqlca`** (SQL Communication Area) con una componente **`sqlcode`** che descrive l'esito dei comandi SQL (vale zero se il comando SQL è stato eseguito con successo)

Cursori

- Problema ulteriore: conflitto d'impedenza (impedance mismatch)
  - i linguaggi di programmazione tradizionali gestiscono i record uno alla volta (tuple-oriented)
  - SQL gestisce insiemi di tuple (set-oriented)
- I cursori fanno parte dello standard SQL-2 e risolvono questo problema
- Un cursore:
  - accede al risultato di una query in modo set-oriented
  - restituisce le tuple al programma una alla volta
- Sintassi per la definizione dei cursori:
  declare *NomeCursore* [**scroll**] **cursor for** *SelectSQL*
  [**for** < **read only** | **update** [**of** *Attributo* {, *Attributo*}]>]

Operazioni sui cursori

- Per eseguire la query associata a un cursore:
  open *NomeCursore*
- Per estrarre una tupla dal risultato della query:
  fetch [*Posizione* **from**] *NomeCursore* **into** *ListaDiFetch*
- Per disallocare il cursore, scartando il risultato della query:
  close *NomeCursore*
- Per accedere alla tupla corrente (quando un cursore accede a una tabella, nell'ambito di un comando di modifica) :
  current of *NomeCursore* (nella clausola `where`)

Esempio d'uso dei cursori

```
void MostraStipendiDipart(char NomeDip[])
{
  exec sql begin declare section;
  char Nome[20], Cognome[20];
  long int Stipendio;
  exec sql end declare section;

  exec sql declare DipImp cursor for
    select Nome, Cognome, Stipendio
    from Impiegato
    where Dipart = :NomeDip;
  exec sql open DipImp;
  exec sql fetch DipImp into :Nome,:Cognome,:Stipendio;
  printf("Dipartimento %s\n",NomeDip);
  while (sqlca.sqlcode == 0)
  {
   printf("Nome: %s %s ",Nome,Cognome);
   printf("Stipendio: %d\n",Stipendio);
   exec sql fetch DipImp into :Nome,:Cognome,:Stipendio;
  }
exec sql close DipImp;
}
```

SQL Dinamico

- SQL dinamico serve quando le applicazioni non possono sapere al momento della compilazione quale sarà il comando SQL da eseguire
- Problema principale: gestire il trasferimento di parametri tra il programma e l'ambiente SQL
- Per l'esecuzione diretta:
  execute immediate *ComandoSQL*
- Se l'esecuzione è preceduta da un'analisi del comando:
  prepare *NomeComando* **from** *ComandoSQL*
  - seguito da:
    execute *NomeComando* [**into** *TargetList*]
    [**using** *ListaParametri*]

Call Level Interface (CLI)

- Con SQL Embedded il preprocessore traduce il comando in un insieme opportuno di chiamate di una libreria interna
- Un'alternativa consite nell'offrire direttamente al programmatore una libreria di sottoprogrammi per gestire il dialogo con il DBMS
- Procedura classica
  - Si usa un sottoprogramma per creare una connessione
  - Si invia sulla connessione un comando SQL (come testo)
  - Si ottiene come risposta una struttura relazionale (la libreria offre un ricco insieme di strumenti per analizzare la struttura del risultato)
  - Al termine dell sessione si chiude la connessione

ODBC

- Open DataBase Connectivity (ODBC) è una importante Call Level Interface, creata da Microsoft
- L'architettura di ODBC prevede 4 componenti
  - L'applicazione
    - specifica il driver da utilizzare e invia ad esso i comandi SQL
  - Il driver manager
    - carica il driver chiesto effettivamente dall'applicazione
  - Il driver
    - realizza il canale di comunicazione tra l'applicazione e una particolare sorgente dati
  - La sorgente dati
    - Mantiene i dati ed esegue i comandi che arrivano dall'applicazione

Caratteristiche di ODBC

- ODBC è una soluzione di grande successo per la costruzione di applicazioni in ambiente Windows
  - La sorgente dati spesso è remota e opera su una piattaforma software completamente diversa
    - gli aspetti di distribuzione e di eterogeneità sono completamente nascosti all'applicazione
  - Le applicazioni possono facilmente cambiare la sorgente dati
  - L'interfaccia però non sfrutta i vantaggi del paradigma di programmazione a oggetti ed è abbastanza complicata
    - soluzioni alternative, basate su ODBC: OLE DB e ADO

OLE DB e ADO

- OLE DB è un'interfaccia generale per l'accesso a sorgenti dati (non solo DB relazionali)
  - a seconda delle caratteristiche della sorgente dati, sarà più o meno ricco l'insieme di comandi di accesso che verranno accettati al momento dell'esecuzione
- Si basa sul modello a oggetti COM, che caratterizza la piattaforma software Microsoft
- ActiveX Data Object (ADO) è un'interfaccia semplice per l'accesso a sorgenti OLE DB

ADO

- Il modello a oggetti di ADO si basa su 4 concetti fondamentali
  - *Connection*
    - rappresenta il canale di comunicazione
    - la sua creazione richiede la specifica della sorgente dati, dell'utente e della password
    - contiene la collezione di errori *Errors*
  - *Command*
    - contiene il comando (SQL) che si vuole far eseguire sulla sorgente dati
  - *Record*
    - rappresenta la singola tupla
  - *RecordSet*
    - rappresenta l'insieme di tuple
    - Svolge il ruolo del cursore, che grazie al modello a oggetti non deve essere introdotto esplicitamente in SQL

ADO in VisualBasic (1)

```
Public Sub InterrogaBD()
  Dim setImpiegati As ADODB.Recordset
  Dim conn As ADODB.Connection
  Dim stringaSQL As String
  Dim comSQL As ADODB.Command
  Dim messaggio As String

    ' si stabilisce la connessione
  Set conn = New ADODB.Connection
  conn.Open "mioServer", "giovanni", "passwordsegreta"

    ' si carica l'insieme degli impiegati in setImpiegati
  Set setImpiegati = New ADODB.Recordset
  stringaSQL = "select Nome, Cognome, Data " & _
               "from Impiegato order by Cognome"
  comSQL.CommandText = stringaSQL
  setImpiegati.Open comSQL, conn, , ,
  ........
```

ADO in VisualBasic (2)

```
...........
  ' si stampano i dati di tutti gli impiegati
  Do While Not setImpiegati.EOF
    messaggio = "Impiegato: " & setImpiegati!Nome & _
                setImpiegati!Cognome & "(record " & _
                setImpiegati.AbsolutePosition & _
                " di " & setImpiegati.RecordCount & ")"
    If MsgBox(messaggio, vbOkCancel) = vbCancel
      Then Exit Do
    setImpiegati.MoveNext
  Loop

  ' si fa pulizia
  setImpiegati.Close
  conn.Close
  Set setImpiegati = Nothing
  Set conn = Nothing
End Sub
```

JDBC

- Java è un moderno linguaggio di programmazione orientato agli oggetti, proposto dalla Sun Microsystems
  - È il cuore di un'architettura, basata sulla definizione rigorosa di un ambiente di esecuzione dei programmi scritti nel linguaggio (Java Virtual Machine)
    - Portabilità molto elevata
  - Dispone di una famiglia di librerie molto estesa, tra cui Java DataBase Connectivity (JDBC)
- JDBC ha una struttura simile a ODBC
  - il driver manager isola il driver responsabile di implementare il canale di comunicazione
  - esistono anche bridge JDBC/ODBC, per usare driver ODBC in un contesto JDBC
- Uso di JDBC
  - Si carica il driver
  - Si crea una connessione con la base di dati
  - Si compone il comando SQL e lo si invia alla base di dati
  - Si gestisce il risultato del comando SQL

Esempio d'uso di JDBC (1)

```
import java.sql.*;
public class PrimoJdbc {
  public static void main(String[] arg) {
    Connection conn = null;
    try {
      // Caricamento del driver
      Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
      // Apertura della connessione
      conn =
  DriverManager.getConnection("jdbc:odbc:Corsi");
    }
    catch (Exception e) { System.exit(1); }
.....................
```

Esempio d'uso di JDBC (2)

```
..............
    try {
      // Esecuzione dell'interrogazione
      Statement interrogazione = conn.createStatement();
      ResultSet risultato =
        interrogazione.executeQuery("select * from Corsi");
      while (risultato.next()) {
        String nomeCorso = risultato.getString("NomeCorso");
        System.out.println(nomeCorso);
      }
    }
    catch (Exception e) { System.exit(1); }
  }
}
```

Procedure

- Moduli di programma che svolgono una specifica attività di manipolazione dei dati
- SQL-2 permette la definizione di procedure (anche note come *stored procedures*), ma solo in forma molto limitata
- La maggior parte dei sistemi offrono delle estensioni che permettono di scrivere procedure complesse (es., Oracle PL/SQL), con strutture di controllo, variabili, eccezioni, etc.
  - si ottiene un linguaggio di programmazione completo
- Due momenti:
  - dichiarazione (DDL)
  - invocazione (DML)
- Con architettura client-server sono normalmente:
  - invocate dai client
  - memorizzate ed eseguite presso i server

Esempio : prelievo dal magazzino

Magazzino

CodProd	QtaDisp	QtaRiord
1	150	100
3	130	80
4	170	50
5	500	150

Riordino

CodProd	Data	QtaOrd

Specifica

- L'utente indica un prelievo dando il codice del prodotto e la quantità da prelevare
- Se la quantità disponibile in magazzino non è sufficiente la procedura si arresta con una eccezione
- Viene eseguito il prelievo, modificando la quantità disponibile in magazzino
- Se la quantità disponibile in magazzino è inferiore alla quantità di riordino si predispone un nuovo ordine d'acquisto

Interfaccia

```
procedure Prelievo
    ( Prod   integer,
      Quant  integer )
```

Invocazione

```
Prelievo(4,150)
```

Stato iniziale nella base di dati

CodProd	QtaDisp	QtaRiord
4	170	50

Realizzazione della procedura

1. Dichiarazione variabili
2. Lettura dello stato
3. Se la quantità disponibile è insufficiente: eccezione
4. Aggiornamento dello stato
5. Se la nuova quantità disponibile è inferiore alla quantità di riordino: emissione di un ordine

Procedura

```
procedure Prelievo (Prod integer, Quant integer) is
begin
 Q1, Q2 integer;
 X exception;
 select QtaDisp, QtaRiord into Q1, Q2
   from Magazzino
   where CodProd = Prod;
 if Q1 < Quant then raise(X);
 update Magazzino
   set QtaDisp = QtaDisp - Quant
   where CodProd = Prod;
 if Q1 - Quant < Q2 then
  insert into Riordino
    values(Prod,sysdate,Q2)
end;
```

Esempio di invocazione

```
Prelievo(4,150)
```

Prod=4, **Quant**=150

```
select QtaDisp, QtaRiord into Q1, Q2
     from Magazzino
     where CodProd = Prod;
```

CodProd	QtaDisp	QtaRiord
4	170	50

Q1 = 170, **Q2** = 50

Invocazione (continua)

`if Q1 < Quant then raise(X)` non scatta

```
update Magazzino
set QtaDisp = QtaDisp - Quant
where CodProd = Prod
```

CodProd	QtaDisp	QtaRiord
4	20	50

`Q1 - Quant < Q2` è vero:

```
  insert into Riordino
     values(Prod, sysdate, Q2)
```

CodProd	Data	QtaRiord
4	10-10-97	50

Problemi del progetto di procedure

- Decomposizione modulare delle applicazioni
- Aumento di:
  - efficienza
  - controllo
  - riuso
- Aumenta la responsabilità dell'amministratore della base di dati (rispetto al programmatore applicativo)
- Si sposta "conoscenza" dalle applicazioni allo schema della base di dati (indipendenza di conoscenza)
- Attenzione a non abusarne:
  - è una soluzione efficiente se il problema richiede di fare computazioni semplici su grandi quantità di dati
  - è inefficiente se su un dato estratto dalla base di dati si devono svolgere computazioni complesse