

## Funzioni SQL e integrazione con i linguaggi di programmazione

### Funzioni SQL

- Costrutti definiti in SQL-2 che estendono la sintassi delle espressioni nella target list
  - `coalesce`
  - `nullif`
  - `case`
- Vengono valutati 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 in SQL

- 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 in SQL

- 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 in SQL

- 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@librodb;
    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 consiste 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 della 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 richiesto 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); }
}
}
```