

A decorative graphic on the left side of the slide. It consists of several vertical blue lines of varying widths and heights. Overlaid on these lines are several blue circles of different sizes. One circle is particularly large and contains the number '1'.

SVILUPPO DI APPLICAZIONI PER BASI DI DATI

1

PROBLEMA

Una larga fetta del software prodotto ricade in questa tipologia

- Scrivere un'applicazione che permetta ad un utente finale di interagire comodamente con un DB
 - visualizzare/inserire dati,
 - effettuare elaborazioni basate sui dati presenti e salvarne i risultati sul DB
 - ...
- Serve un linguaggio che possa:
 - Effettuare aggiornamenti
 - Effettuare interrogazioni
 - Effettuare I/O
 - Controllare il flusso in un modo che dipende dal risultato dell'interrogazione

richiede un buon livello di integrazione

PROBLEMA

SQL permette di

- inserire, aggiornare, cancellare i dati memorizzati in un DB
- eseguire ricerche sui dati memorizzati in un DB

ma non

- elaborazioni complesse sui dati
- interazioni con il sistema operativo

SQL non è **completo**:

- **computazionalmente**
 - non è in grado di esprimere tutte le computazioni teoricamente possibili
 - ⇐ mancano costrutti quali scelta o iterazione
- **operazionalmente**
 - non è in grado di comunicare direttamente con il sistema operativo e, attraverso di esso, con l'hardware e le periferiche
 - ⇐ mancano costrutti per scrivere in un file, stampare, sviluppare un'interfaccia utente...

SOLUZIONE

- Per estendere il potere espressivo di SQL si *combina* SQL con un linguaggio di programmazione generico
 - SQL consente accesso ottimizzato ai dati
 - il linguaggio di programmazione garantisce completezza computazionale ed operativa

APPROCCI ALL'INTEGRAZIONE ACCOPPIAMENTO INTERNO

Estensioni procedurali di SQL

- Si estende SQL con costrutti standard nei linguaggi di programmazione
- Si usa per definire ed eseguire funzioni e procedure all'**interno** del DB
 - richiamabili internamente o da qualunque applicazione
 - riuso/centralizzazione di manutenzione e verifica
 - **approccio misto all'accoppiamento**
- Permette di garantire la completezza computazionale
- Non sempre garantisce la completezza operativa
- Adottato soprattutto per nuove applicazioni che richiedono una forte interazione con il DBMS
- Lo standard SQL propone un'estensione procedurale (SQL/PSM),
ma la sintassi varia sensibilmente tra i vari DBMS
 - noi vedremo PostgreSQL: PL/pgSQL

APPROCCI ALL'INTEGRAZIONE ACCOPPIAMENTO ESTERNO

- Un linguaggio di programmazione esistente (es. C, Java) viene integrato con SQL
- Il codice viene eseguito in un ambiente **esterno** al DBMS
- La completezza (computazionale e operativa) è garantita
- Il più diffuso
 - preferito quando si estendono applicazioni esistenti per interagire con un (ulteriore) DBMS
- Due approcci principali, di lunga tradizione
 - **Librerie di funzioni**
 - **SQL ospitato**

LIBRERIE DI FUNZIONI

- L'interfaccia di comunicazione tra il DBMS ed il linguaggio di programmazione è fornita da un'API
 - Funzioni per
 - la connessione alla base di dati
 - l'esecuzione di comandi SQL
 - Librerie proprietarie dei vari DBMS
 - Librerie standard, che garantiscono interoperabilità (es. ODBC, JDBC)

SQL OSPITATO

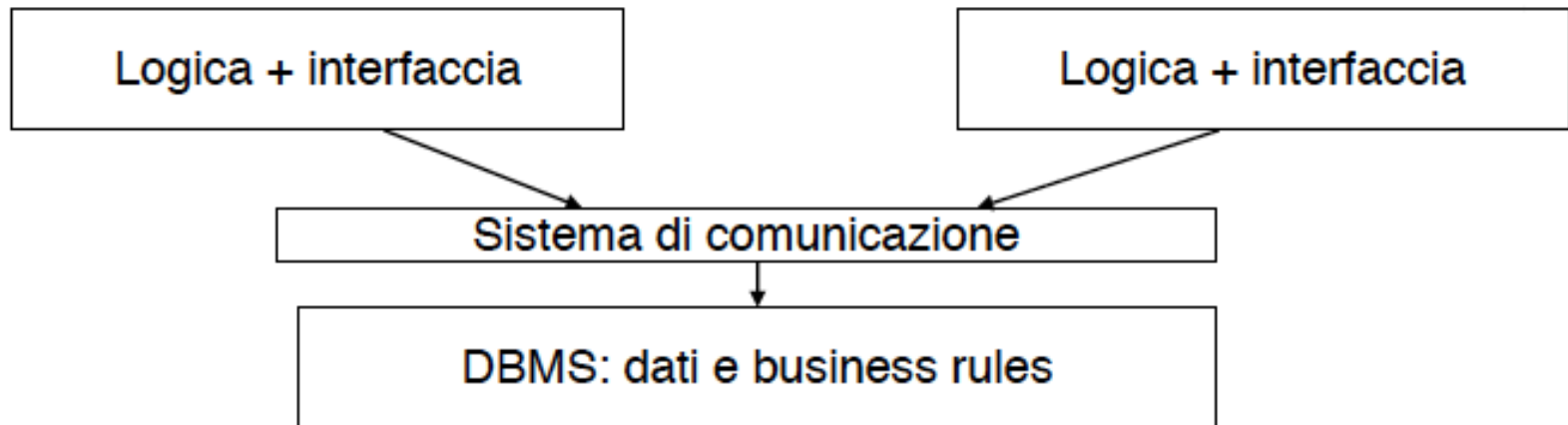
- Il linguaggio di programmazione è esteso con comandi SQL
- Uno speciale pre-compilatore traduce programmi nel linguaggio esteso in programmi nel linguaggio ospite
 - In genere, sostituisce i comandi SQL con chiamate al DBMS tramite interfacce di connessione standard o proprietarie
- Esempio
 - SQLj per ospitare comandi SQL in programmi Java

APPLICAZIONE PER BASI DI DATI

- Quattro parti:
 1. Gestione dei dati (transazioni, gestione dei dati persistenti, controllo accessi)
 2. Business rules (vincoli di integrità e dati derivati)
 3. Logica applicativa
 4. Interfaccia utente
- Diversi tipi di architettura per risolvere lo stesso problema

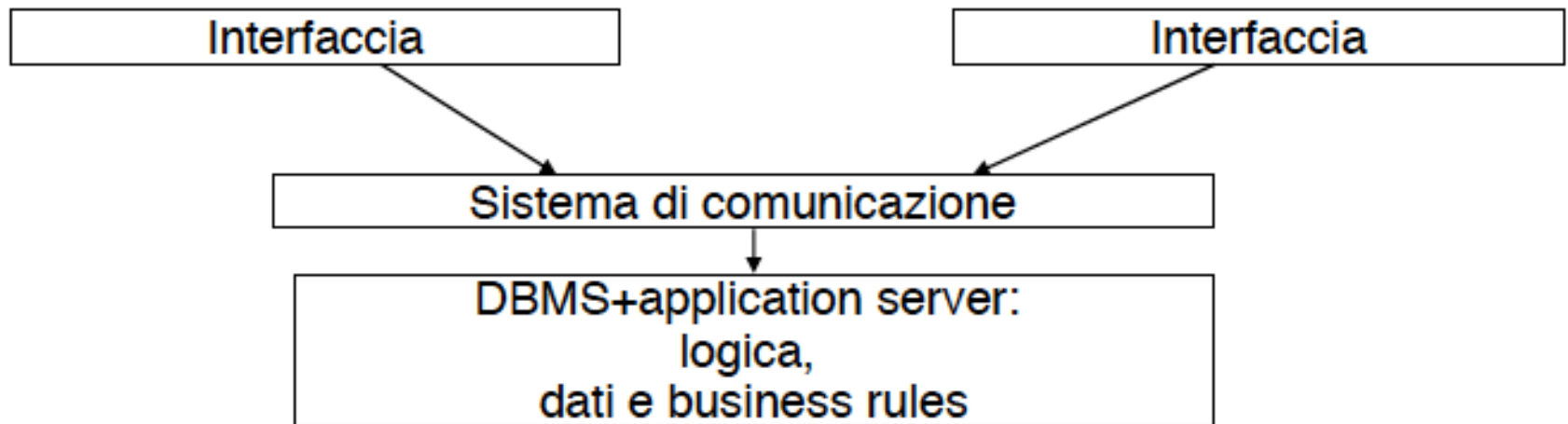
ARCHITETTURE SOFTWARE

Client-Server

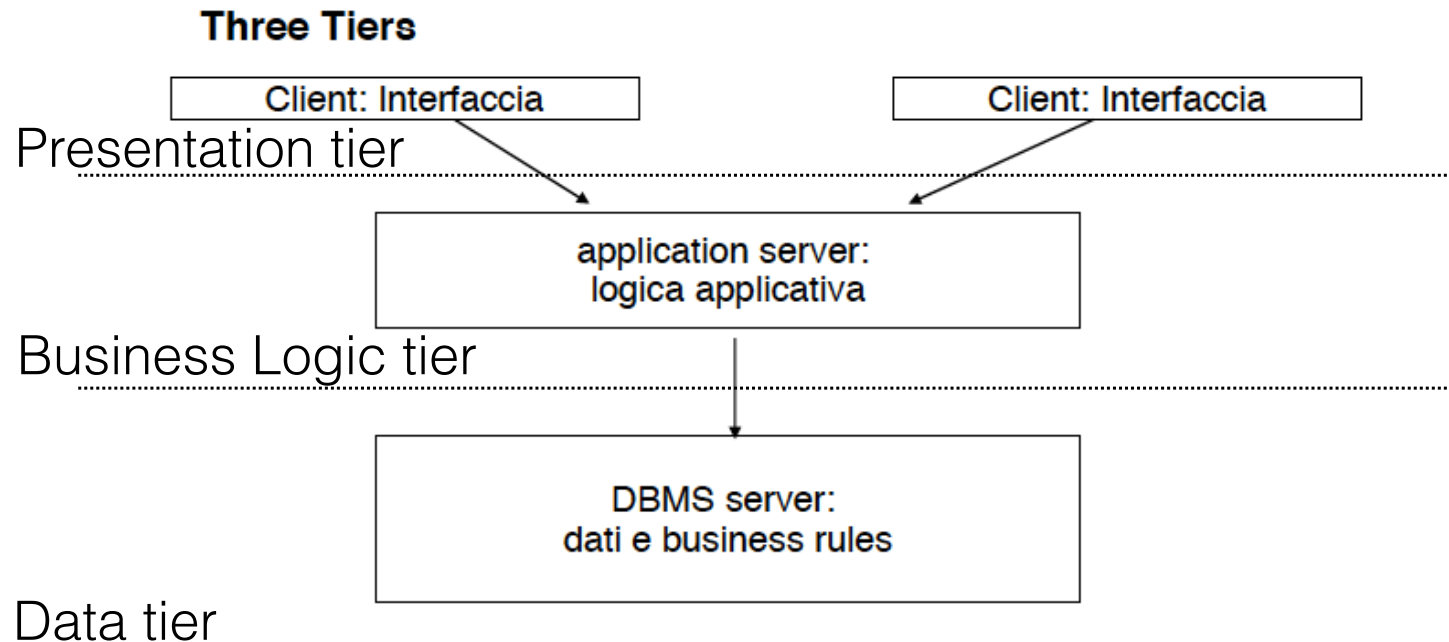


ARCHITETTURE SOFTWARE

Thin Client



ARCHITETTURA SOFTWARE



APPLICAZIONI PER BASI DI DATI - SPOILER

- Persistence Layer

- Per isolare gli aspetti legati all'interfacciamento con il DBMS in classi ad hoc dell'applicazione

- Framework di persistenza (es. LINQ/Hibernate)

- Perché l'interfacciamento tra linguaggio di programmazione e DBMS non è indolore (impedance mismatch)

- Corso di Ingegneria del Software

The left side of the slide features a series of vertical stripes in various shades of blue and teal. Overlaid on these stripes are several circles of different sizes, also in shades of blue and teal. One large circle is positioned near the top left, and several smaller circles are arranged below it, creating a decorative, abstract pattern.

INTRODUZIONE A PL/PGSQL

14

INTRODUZIONE


- La maggior parte dei DBMS supporta un'estensione procedurale di SQL
- I programmi sono in genere organizzati in routine (procedure o funzioni) che vengono
 - eseguite direttamente dal DBMS
 - chiamate da applicazioni basate su accoppiamento esterno

INTRODUZIONE

- Oracle: PL/SQL
 - SQL Server: T-SQL
 - PostgreSQL: PL/pgSQL
-
- I programmi possono solo essere eseguiti direttamente dal DBMS

CREAZIONE ROUTINE

non si differenzia la
definizione di
funzioni e procedure



```
CREATE [ OR REPLACE ] FUNCTION <nome routine>
    ( [<lista parametri>] )
[ RETURNS <tipo ritorno> ]
AS
$$ <corpo routine> $$
LANGUAGE plpgsql ;
```

<lista parametri> = <parametro>[, <lista parametri>]

<parametro>= [<modo>] [<nome parametro>]<tipo parametro>

CREAZIONE ROUTINE:

**<PARAMETRO> = [<MODO>] [<NOME PARAMETRO>]
<TIPO PARAMETRO>**

- **<modo>** rappresenta il tipo di parametro
 - Input IN (default)
 - Output OUT
 - Input/Output INOUT
- **<nome parametro>** nome parametro formale
 - se manca si usa \$n dove n è la posizione del parametro
- **<tipo parametro>**
 - un qualunque tipo previsto da PostgreSQL
 - il tipo di una colonna di una tabella esistente
 - <nome tabella>.<nome colonna>%TYPE

CREAZIONE ROUTINE

- Si vuole dichiarare un parametro *titolo* a cui verrà passato il titolo di film selezionati dalla tabella Film
- Se non ne conosciamo il tipo, oppure se assumiamo che il tipo possa cambiare nel tempo, possiamo utilizzare la seguente dichiarazione:

- ilMioTitolo **Film.titolo%TYPE**;

Nome tabella



Nome colonna tabella

Le funzioni possono restituire insiemi
RETURNS TABLE vi può essere utile per
progetto...

CREAZIONE ROUTINE:

RETURNS <TIPO RITORNO>

- <tipo ritorno>: tipo del valore restituito dalla funzione
 - Stessi tipi ammissibili rispetto ai parametri
 - Void se la funzione non restituisce nulla
 - Se almeno un parametro è OUT o INOUT, la clausola RETURNS può essere omessa
 - Se è presente, deve coincidere con il tipo dei parametri di output
 - se ci sono **più** parametri OUT o INOUT, <tipo ritorno> = record
- Quindi
 - RETURNS presente → funzione
(oppure se ci sono parametri OUT o INOUT)
 - RETURNS assente → procedura
(se **non** ci sono parametri OUT o INOUT)

CREAZIONE ROUTINE: BODY

- <Corpo routine>
 - Corpo della routine, organizzato in blocchi
 - Tutto il codice deve essere racchiuso tra \$\$ e \$\$
- Oltre a plpgsql, esistono altri linguaggi, incluso SQL semplice
 - Non li vediamo

CHIAMATA DI FUNZIONE

<nome funzione> (<lista argomenti>);

- <nome funzione> è il nome della funzione da eseguire
- <lista argomenti> è una lista di argomenti ciascuno dei quali corrisponde a
 - un'espressione per ogni parametro di input alla funzione
 - una variabile inizializzata, per ogni parametro di input output
 - **non** sono presenti i parametri di output (!)
 - servono solo a
 - dare un nome alle colonne del risultato
 - definire un tipo di ritorno complesso in modo anonimo
 - I tipi degli argomenti devono essere conformi con la dichiarazione dei parametri corrispondenti

CHIAMATA DI FUNZIONE

- La chiamata di una funzione restituisce un valore per <tipo ritorno>
- Se la clausola RETURNS non è presente (o è indicato RETURNS record), viene restituito un record
 - costituito dai valori assegnati ai parametri OUT e INOUT
 - nell'ordine in cui sono definiti
- Il risultato di una funzione è quindi sempre una tupla formata da:
 - un solo elemento, rappresentato dal valore di ritorno, se non ci sono parametri OUT e INOUT
 - i valori assegnati a parametri OUT e INOUT, nell'ordine con cui compaiono nella segnatura

ESEMPIO (1)

```
CREATE FUNCTION AggiornaVal1 (IN ilGenere CHAR(15) )  
    RETURNS void AS
```

```
$$
```

```
    <aggiorna la valutazione dei film di genere uguale ad ilGenere>
```

```
$$ LANGUAGE plpgsql;
```

Restituisce tupla vuota

ESEMPIO (2)

```
CREATE FUNCTION AggiornaVal2 (IN ilGenere CHAR(15) )  
    RETURNS NUMERIC(3,2) AS
```

```
$$
```

<aggiorna la valutazione dei film di genere uguale ad
ilGenere e restituisce valutazione media dei film con quel
genere>

```
$$ LANGUAGE plpgsql;
```

Restituisce tupla formata da un unico elemento, che
rappresenta la valutazione media

ESEMPIO(3)

CREATE FUNCTION

AggiornaVal3 (IN ilGenere CHAR(15), OUT val NUMERIC(3,2))
AS

\$\$

<aggiorna la valutazione dei film di genere uguale ad ilGenere e restituisce valutazione media dei film con quel genere **nel parametro di output val**>

\$\$ LANGUAGE plpgsql;

Restituisce tupla formata da un unico elemento, che rappresenta la valutazione media

ESEMPIO (4)

CREATE FUNCTION

```
AggiornaVal4 (IN ilGenere CHAR(15),  
              OUT val NUMERIC(3,2), OUT minAnno INTEGER )  
AS
```

```
$$
```

<aggiorna la valutazione dei film di genere uguale ad ilGenere e restituisce:

(1) valutazione media dei film con quel genere nel parametro di output val

(2) l'anno di produzione del film piu` vecchio, con genere ilGenere, nel parametro di output minAnno>

```
$$ LANGUAGE plpgsql;
```

Restituisce tupla formata da due valori

CHIAMATA DI FUNZIONE

Una chiamata di funzione può comparire

- in un comando SQL
- in una istruzione PL/pgSQL

in ogni posizione in cui può comparire [un insieme formato da] una singola tupla

ESEMPIO

- **SELECT AggiornaVal1('drammatico');**
aggiorna la valutazione dei film di genere drammatico e restituisce tabella vuota;
- **SELECT AggiornaVal2('drammatico');**
aggiorna la valutazione dei film di genere drammatico e restituisce una tabella di grado 1, costituita da un'unica tupla contenente la valutazione media dei film di genere 'drammatico'
- **SELECT AggiornaVal3('drammatico');**
aggiorna la valutazione dei film di genere drammatico e restituisce una tabella di grado 1, costituita da un'unica tupla contenente la valutazione media dei film di genere 'drammatico'

manca il parametro out nella chiamata
il risultato ha una colonna denominata
val

ESEMPIO

- **SELECT AggiornaVal4('drammatico');**

aggiorna la valutazione dei film di genere drammatico e restituisce **una tabella di grado 1**, costituita da un'unica tupla contenente come unico valore un **record**, composto dalla valutazione media dei film di genere 'drammatico' e dall'anno del film drammatico più vecchio

- record come tipo di dato non visto a lezione

- **SELECT * FROM AggiornaVal4('drammatico');**

aggiorna la valutazione dei film di genere drammatico e **restituisce una tabella di grado 2**, costituita da un'unica tupla contenente la valutazione media dei film di genere 'drammatico' e dall'anno del film drammatico più vecchio

CORPO ROUTINE

Il corpo di ogni programma/routine è composto da due sezioni

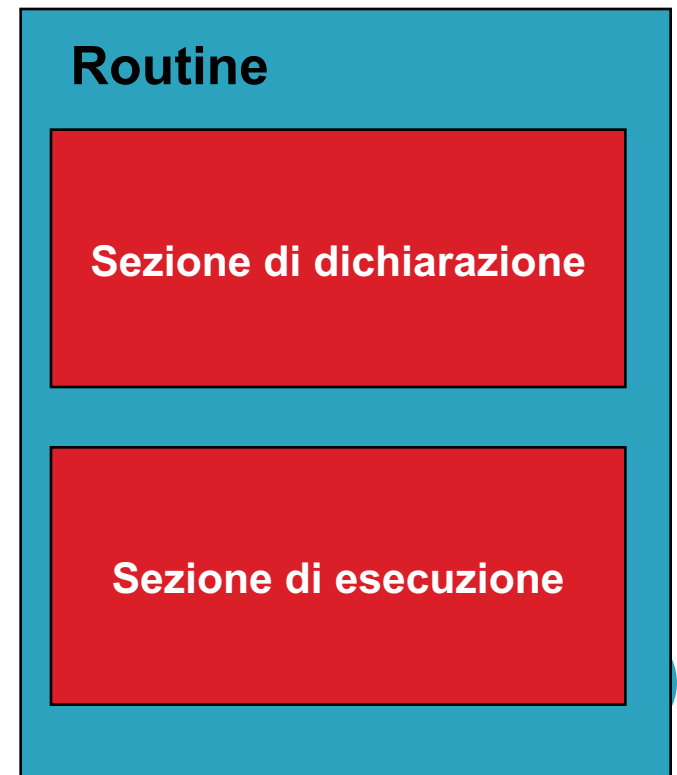
o sezione di dichiarazione

- dichiarazioni di **tutte** le variabili del programma
- opzionale

o sezione di esecuzione

- codice da eseguire
 - costrutti procedurali (istruzioni)
 - +
 - statement SQL
- obbligatorio

```
[ DECLARE <dichiarazioni> ]  
  BEGIN  
    <istruzioni>  
  END;
```



SEZIONE DI DICHIARAZIONE

- Singola dichiarazione di variabile/costante

<nome> [CONSTANT] *<tipo>*

[NOT NULL]

[{ DEFAULT | := } *<espressione>*];

- CONSTANT \Rightarrow *<nome>* non può essere riassegnata
- NOT NULL \Rightarrow non si può assegnare NULL a *<nome>*
- All'entrata nel blocco a *<nome>* viene assegnato
 - se è presente DEFAULT o :=, il valore ottenuto dalla valutazione di *<espressione>*
 - altrimenti NULL

SEZIONE DI DICHIARAZIONE - ESEMPI

DECLARE

```
valutaz NUMERIC(3,2) DEFAULT 2.50;  
regista VARCHAR(20) := 'quentin tarantino';  
genere CONSTANT CHAR(15) := 'drammatico';  
codCli DECIMAL(4) NOT NULL :=0000;
```

- Possibili usi corretti
 - assegnare 'pinco palla' a regista
 - assegnare 1234 a codCli
- Possibili errori
 - assegnare 'cinque' a valutaz
 - assegnare qualsiasi cosa a genere
 - assegnare NULL a codCli

SEZIONE DI ESECUZIONE

- Contiene costrutti procedurali e statement SQL
 - che comunicano fra loro usando le variabili precedentemente dichiarate
- In SQL le variabili possono essere usate in
 - clausola WHERE di statement SELECT, DELETE, UPDATE
 - clausola SET di uno statement di UPDATE
 - clausola VALUES di uno statement di INSERT
 - clausola INTO di uno statement SELECT (si veda oltre)
- Nella parte procedurale ad una variabile può essere assegnato un valore ottenuto da una query

ASSEGNAZIONE

<nome var> := <espressione>

- <nome var> ed <espressione> devono avere lo stesso tipo

- **Esempio**

```
DECLARE ilGenere CHAR(15);
```

```
BEGIN
```

```
    ilGenere := 'comico';
```

```
    ...
```

```
    ilGenere := (SELECT genere FROM Film  
WHERE titolo = 'pulp fiction' AND regista =  
'quentin tarantino');
```

```
    ...
```

```
END;
```

Assegnando alla variabile un valore calcolato da una query si collegano i costrutti imperativi e quelli di SQL

OUTPUT

Utile soprattutto in fase di debugging

In produzione

- codice eseguito dal server
- l'utente dell'applicazione finale non vede nulla

`RAISE NOTICE <stringa> [, <espressione>]`

- <stringa> da stampare a video
 - se contiene %, a video % viene sostituito con il valore di <espressione>
- La stringa viene visualizzata nella zona 'Messaggi' dell'output
- **Esempio**

```
DECLARE ilGenere CHAR(15);
```

```
BEGIN
```

```
    ilGenere := 'comico';
```

```
    RAISE NOTICE 'Il genere considerato e ``%', ilGenere;
```

```
    ...
```

```
END;
```

ESECUZIONE COMANDI SQL

- I comandi che non restituiscono tuple (INSERT, DELETE, UPDATE, CREATE TABLE ...) sono istruzioni
 - È sufficiente scrivere il comando nel corpo della funzione

- **Esempio**

```
DECLARE
```

```
    ilTitolo VARCHAR(30);
```

```
    ilRegista VARCHAR(20);
```

```
BEGIN
```

```
    ilTitolo := 'Pulp Fiction';
```

```
    ilRegista := 'quentin tarantino';
```

```
    UPDATE Film
```

```
    SET valutaz = valutaz * 1.1
```

```
    WHERE titolo = ilTitolo AND regista = ilRegista;
```

```
END;
```

ESECUZIONE COMANDI SQL

- Per comandi SELECT che restituiscono **esattamente** una tupla, i valori degli attributi della tupla possono essere inseriti direttamente in variabili utilizzando lo statement SELECT ... INTO

- Esempio**

```
DECLARE
```

```
    laValutaz NUMERIC(3,2);
```

```
BEGIN
```

```
    [...]
```

```
    SELECT valutaz INTO laValutaz
```

```
    FROM Film
```

```
    WHERE titolo = 'Mediterraneo' AND regista =  
        'gabriele salvatores';
```

```
END;
```

equivalente a

```
laValutaz :=  
    (SELECT valutaz  
     FROM Film  
     WHERE titolo =  
         'Mediterraneo' AND regista  
         = 'gabriele salvatores');
```

ESECUZIONE COMANDI SQL

- Se la query restituisce più di una tupla, nelle variabili vengono inseriti i valori degli attributi della **prima tupla** restituita
- Se la query restituisce un insieme vuoto, viene assegnato NULL alle variabili
- Se si vuole che in **entrambi** i casi precedenti venga generato un errore, è necessario utilizzare la parola chiave **STRICT**

ESECUZIONE COMANDI SQL

ESEMPIO

```
DECLARE
```

```
    laValutaz NUMERIC(3,2);
```

```
BEGIN
```

```
    [...]
```

```
    SELECT valutaz INTO STRICT laValutaz
```

```
    FROM Film
```

```
    WHERE titolo = 'Mediterraneo';
```

```
END;
```

Se esiste più di un film dal titolo 'Mediterraneo'
oppure non ne esiste nessuno si ha un errore

STRUTTURE DI CONTROLLO – COSTRUTTI DI SCELTA

```
IF <expr booleana>  
    THEN <istruzioni>  
    [ELSE <istruzioni>]  
END IF;
```

```
IF <expr booleana>  
    THEN <istruzioni>  
    [ELSEIF <expr booleana>  
        THEN <istruzioni> ]  
END IF;
```

ESEMPIO

DECLARE

infoNoleggi INTEGER;

laValutaz NUMERIC(3,2);

BEGIN

[...]

IF infoNoleggi > 5000 THEN laValutaz := 3.00;

ELSEIF infoNoleggi > 3000 THEN laValutaz := 2.00;

ELSE laValutaz := 1.00;

END IF;

INSERT INTO Film

VALUES ('kill bill I','quentin tarantino',2003,'thriller',laValutaz);

END;

STRUTTURE DI CONTROLLO – CICLI

ITERAZIONE INCERTA

[WHILE <expr booleana>] LOOP <istruzioni> END LOOP;

- clausola WHILE **opzionale**

- se assente equivale a WHILE true
- se presente il ciclo termina se la condizione diventa falsa

- nel blocco LOOP si possono utilizzare

- EXIT [WHEN <expr booleana>] per uscire dal ciclo
- CONTINUE [WHEN <expr booleana>] per passare all'esecuzione dell'iterazione successiva
- in entrambi i casi se è presente la clausola WHEN <expr booleana> l'azione viene eseguita solo se l'espressione si valuta a true
- in caso di cicli annidati EXIT/CONTINUE fanno riferimento al ciclo più interno
 - si può fare in modo di far riferimento anche ad un altro ciclo più esterno usando le **label**, se vi servisse guardate il manuale

STRUTTURE DI CONTROLLO – CICLI

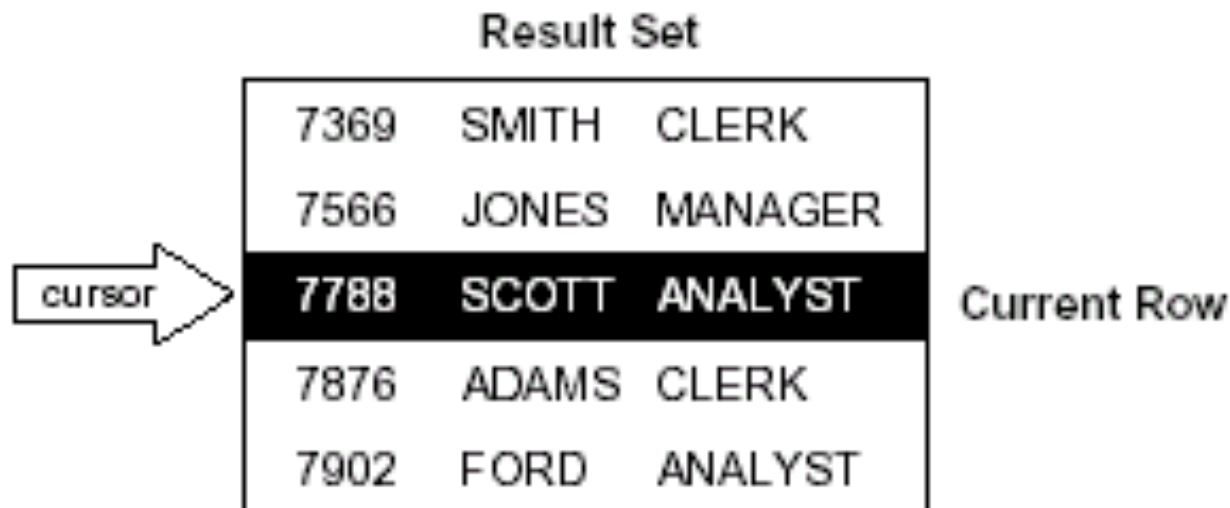
ITERAZIONE CERTA

```
FOR <var indice> IN [ REVERSE ] <expr> .. <expr>  
[ BY <expr> ] LOOP <istruzioni> END LOOP;
```

- Iterazione **certa** perché le due espressioni sono valutate **una sola volta prima** di iniziare l'esecuzione
- **var indice** = variabile di tipo **intero**
 - automaticamente dichiarata dal ciclo e inizializzata con la prima espressione
 - ha il ciclo come scope
 - automaticamente incrementata del passo ad ogni ciclo
 - se REVERSE è specificata **decrementata**
 - quando raggiunge il valore della seconda espressione si esegue il ciclo per l'ultima volta
- La clausola BY permette di definire il passo
 - se manca, il passo di default è 1

ESECUZIONE COMANDI SQL

- Se il comando `SELECT` restituisce un insieme di tuple spesso è necessario iterare sul risultato
- Per farlo si usano i **cursori**
 - Un cursore è un puntatore ad una tupla contenuta nel risultato di un'interrogazione SQL
 - Un cursore è quindi associato alla valutazione di un'interrogazione
 - In ogni momento, la tupla puntata dal cursore, e quindi i valori degli attributi di tale tupla, possono essere letti



CURSORI

○ Operazioni sui cursori:

- **Dichiarazione**

<nome cursore> CURSOR FOR <select statement>

associa il cursore all'interrogazione

- Da inserire nella sezione di dichiarazione

- **Apertura**

OPEN <nome cursore>

esegue l'interrogazione associata al cursore e lo inizializza prima della prima tupla del risultato

- **Posizionamento**

sposta il cursore avanti o indietro ed opzionalmente legge i valori della tupla di arrivo e li assegna (vedi prossima slide)

- **Chiusura**

CLOSE <nome cursore>

disabilita il cursore

- se si vuole usarlo dopo averlo chiuso è necessario riaprirlo
- se lo si riapre si riesegue la query

CURSORE POSIZIONAMENTO

Ci sono molte possibili opzioni (vedi manuale)

La più comune è l'**avanzamento**

FETCH <nome cursore> [**INTO** <lista variabili>]

che posiziona il cursore sulla tupla del risultato successiva a quella corrente

Con l'opzione **INTO** è possibile inserire in variabili i valori degli attributi della tupla puntata dal cursore **dopo** l'avanzamento

- se tale tupla non esiste vengono assegnati dei null
- il numero di variabili deve corrispondere al numero di colonne della query associata al cursore
- ad ogni variabile dell'elenco, da sinistra a destra, è assegnato il valore della colonna corrispondente come posizione
- il tipo di dato di ogni variabile deve essere lo stesso o compatibile con quello della colonna corrispondente

STATO DI UNA OPERAZIONE

- Per verificare lo stato di un'operazione, è disponibile la variabile **FOUND**
- Inizialmente è posta a FALSE
- Viene posta a TRUE dalle seguenti operazioni
 - SELECT INTO, se viene restituita una tupla
 - UPDATE, INSERT, DELETE, se almeno una tupla viene aggiornata
 - FETCH, se dopo lo spostamento il cursore punta ad una tupla (non ha quindi raggiunto la fine del result set)
 - Ciclo FOR, se viene eseguita almeno una iterazione

ESEMPIO

DECLARE

ilTitolo VARCHAR(30);

ilRegista VARCHAR(20);

valElevata CURSOR FOR

SELECT titolo, regista FROM Film WHERE valutaz > 3.00;

BEGIN

[...]

OPEN valElevata;

FETCH valElevata INTO ilTitolo, ilRegista;

WHILE **FOUND** LOOP

BEGIN

... elaborazione di ilTitolo e ilRegista ...

FETCH valElevata INTO ilTitolo, ilRegista;

END;

END LOOP;

CLOSE valElevata;

END;

Attenzione!!!

Non «esageriamo» con i cursori: vanno usati **solo se** l'elaborazione che vogliamo fare su titolo e regista non si riesce a fare in modo set-oriented con comando SQL

SQL DINAMICO

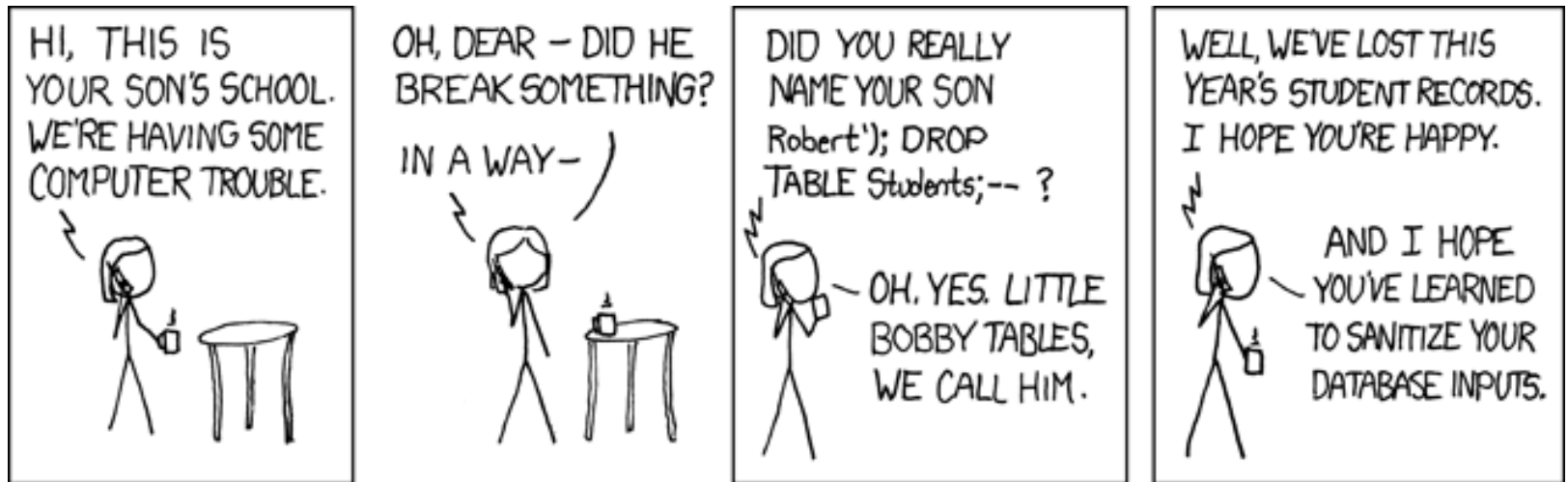
- PL/pgSQL permette di eseguire comandi creati durante l'esecuzione dell'applicazione

EXECUTE <espressione di tipo stringa>
[**INTO** [**STRICT**] <lista variabili>]

- < espressione di tipo stringa > sarà il codice SQL da eseguire (non può contenere SELECT INTO)
- La stringa contenuta può anche essere costruita a tempo di esecuzione
- La clausola INTO si comporta come nel comando SELECT INTO, se la stringa è un comando di SELECT

LITTLE BOBBY TABLES

- La stringa contenuta può anche essere costruita a tempo di esecuzione...



ESEMPIO

DECLARE

ilTitolo VARCHAR(30);

ilRegista VARCHAR(20);

ilComando VARCHAR(100) := 'SELECT titolo, regista, valutaz FROM Film';

BEGIN

[...]

IF (SELECT valutaz FROM Film

WHERE titolo = ilTitolo AND regista = ilRegista) <= 1.00 THEN

ilComando := 'DELETE FROM Video

WHERE titolo = ilTitolo AND regista = ilRegista';

ELSEIF (SELECT valutaz FROM Film

WHERE titolo = ilTitolo AND regista = ilRegista) >= 4.00 THEN

ilComando := 'UPDATE Video SET tipo = "d"

WHERE titolo = ilTitolo AND regista = ilRegista AND tipo = "v" ';

END IF;

EXECUTE ilComando;

END;

ESEMPIO

DECLARE

 ilTitolo VARCHAR(30);

 ilRegista VARCHAR(20);

 ilComando VARCHAR(100) := 'SELECT AVG(valutaz) FROM Film';

BEGIN

 [...]

 IF (ilRegista IS NOT NULL) THEN

 ilComando := ilComando || ' WHERE regista=ilRegista';

 IF (ilTitolo IS NOT NULL) THEN

 ilComando := ilComando || ' AND titolo=ilTitolo';

 END IF;

 ELSEIF (ilTitolo IS NOT NULL) THEN

 ilComando := ilComando || ' WHERE titolo=ilTitolo';

 END IF;

EXECUTE ilComando;

END;

GESTIONE DEGLI ERRORI

- Gli errori si gestiscono tramite il meccanismo delle eccezioni
- Le eccezioni possono essere catturate all'interno di un qualunque blocco
- Esistono due tipi di eccezioni:
 - **di sistema**, definite e sollevate direttamente dal sistema
 - **utente**, definite e sollevate dal programma applicativo
 - Non le vediamo

GESTIONE DEGLI ERRORI

BEGIN

<istruzioni>

EXCEPTION

WHEN <condizione> [**OR** <condizione> ...] **THEN**

<handler_statements>

[**WHEN** <condizione> [**OR** <condizione> ...] **THEN**

<handler_statements> ...]

END;

Analogo a

try

{<istruzioni>}

catch (<exception>)

{<handler_statements>}

[**catch** (<exception>)

{<handler_statements>} ...]

GESTIONE DEGLI ERRORI

ANALOGO A JAVA, C#, C++...

- Se si verifica un errore e viene quindi sollevata un'eccezione
 - il controllo passa alla lista di eccezioni
 - si determina la prima condizione soddisfatta dall'errore rilevato e l'handler-statement corrispondente viene eseguito
 - si passa all'esecuzione della prima istruzione dopo il blocco
- Se nessuna condizione viene soddisfatta dall'errore, l'errore viene propagato al blocco più esterno
- Se un'eccezione non viene gestita a nessun livello, la funzione viene abortita
- La gestione delle eccezioni è costosa, va fatta solo quando è ragionevole aspettarsi che un errore si possa verificare

GESTIONE DEGLI ERRORI

PECULIARITÀ

- Le eccezioni non sono tipate
 - non si può diversificare l'handler sul tipo delle eccezioni (come nei catch)
- Si sceglie quale handler usare sulla base del **valore** dell'eccezione
- Eccezione in SQL = la variabile globale e predefinita **SQLSTATE** ha assunto un valore nella lista dei codici di errore
 - Il valore 00000 indica che non si è verificato alcun errore
 - Ci sono valori che indicano warning
 - il programma non abortisce, ma l'esecuzione ha avuto problemi
 - esempio: il risultato era troppo grosso ed è stato troncato
- Ogni condizione controlla se **SQLSTATE** ha un particolare valore
 - bisogna conoscere i codici di errore per poter programmare
 - oltre ai codici numerici si possono usare delle **costanti** standard (molto meglio)

GESTIONE DEGLI ERRORI – ALCUNI CODICI

Error Code	Meaning	Constant
Class 00 — Successful Completion		
00000	SUCCESSFUL COMPLETION	successful_completion
Class 01 — Warning		
01000	WARNING	warning
0100c	DYNAMIC RESULT SETS RETURNED	dynamic_result_sets_returned
01008	IMPLICIT ZERO BIT PADDING	implicit_zero_bit_padding
01003	NULL VALUE ELIMINATED IN SET FUNCTION	null_value_eliminated_in_set_function
01007	PRIVILEGE NOT GRANTED	privilege_not_granted
01006	PRIVILEGE NOT REVOKED	privilege_not_revoked
01004	STRING DATA RIGHT TRUNCATION	string_data_right_truncation
01P01	DEPRECATED FEATURE	deprecated_feature
Class 02 — No Data (this is also a warning class per the SQL standard)		
02000	NO DATA	no_data
02001	NO ADDITIONAL DYNAMIC RESULT SETS RETURNED	no_additional_dynamic_result_sets_returned

ESEMPIO

```
CREATE FUNCTION AggiornaVal(ilGenere CHAR(15)) AS
DECLARE
    laVal NUMERIC(3,2);
    ilTitolo VARCHAR(30);
    ilRegista VARCHAR(20);
    valCr CURSOR FOR SELECT titolo, regista, valutaz FROM Film WHERE genere = ilGenere;
BEGIN
    SELECT AVG(valutaz) INTO laVal
    FROM Film WHERE genere = ilGenere;
    IF (laVal < 4) THEN UPDATE Film SET valutaz = valutaz * 1.05;
    ELSE UPDATE Film SET valutaz = valutaz * 0.95;
    END IF;
    OPEN valCr;
    FETCH valCr INTO ilTitolo, ilRegista, laVal;
    WHILE FOUND LOOP
        BEGIN
            RAISE NOTICE 'Titolo: %', ilTitolo;
            RAISE NOTICE 'Regista: %', ilRegista;
            RAISE NOTICE 'Valutazione: %', laVal;
            FETCH valCr INTO ilTitolo, ilRegista, laVal;
        END;
    END LOOP;
    CLOSE valCr;
    EXCEPTION
    WHEN no_data THEN RAISE NOTICE 'Errore, dati non trovati';
END;
```