

Basi di dati

Corso di laurea in Ingegneria Informatica
Scuola di Ingegneria — Università di Pisa

Oracle MySQL
A.A. 2017-2018

5
Ing. Francesco Pistolesi
Postdoctoral Researcher
Data Science and Engineering Lab
Dipartimento di Ingegneria dell'Informazione
francesco.pistolesi@iet.unipi.it

Stored procedure



Stored procedure

➤ sono invocate mediante chiamata

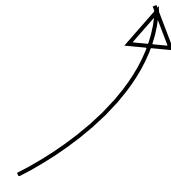
Sono **procedure dichiarativo-procedurali** memorizzate nel DBMS



supportate dalla versione 5.0 di MySQL

Stored procedure: a cosa servono?

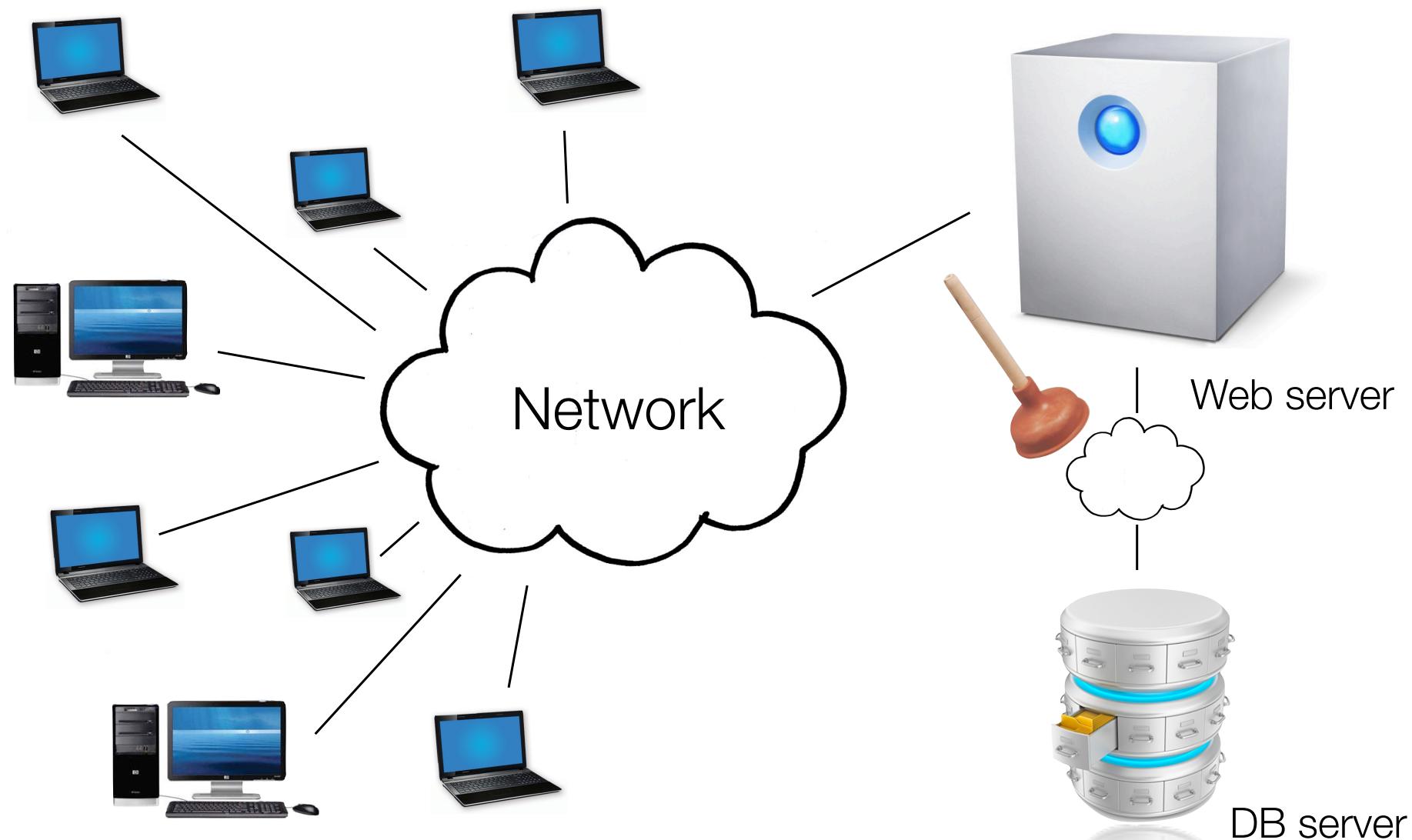
sono compilate e inserite in una cache



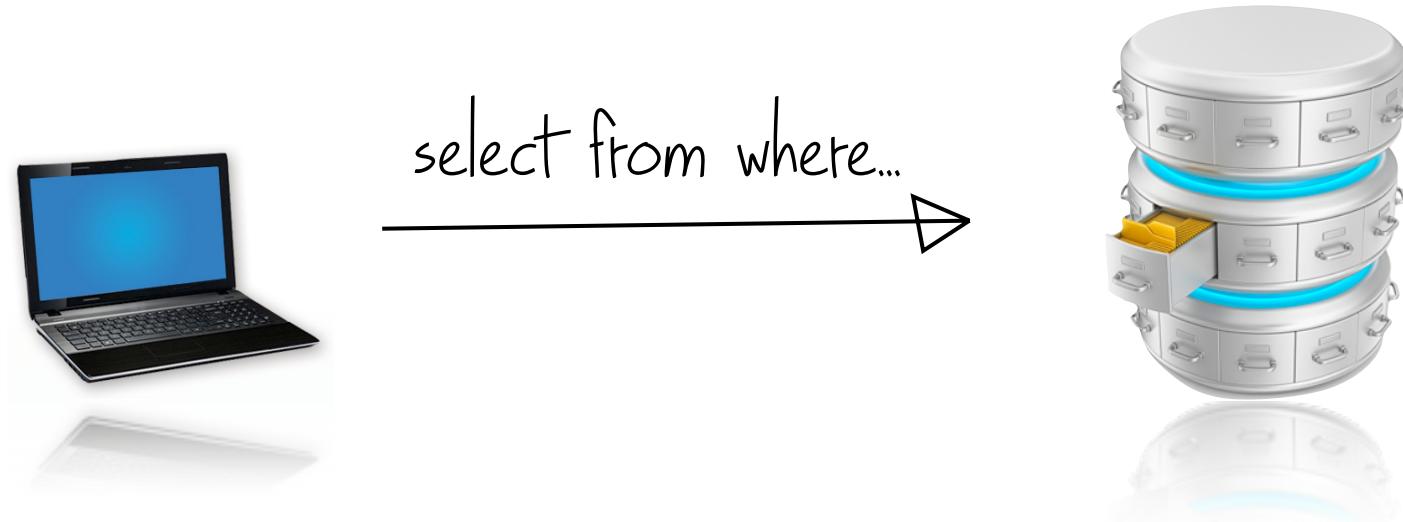
Lo scopo principale è il miglioramento delle prestazioni

se un'applicazione usa più volte una stored procedure,
MySQL utilizza la versione nella cache

Architettura multi-tier

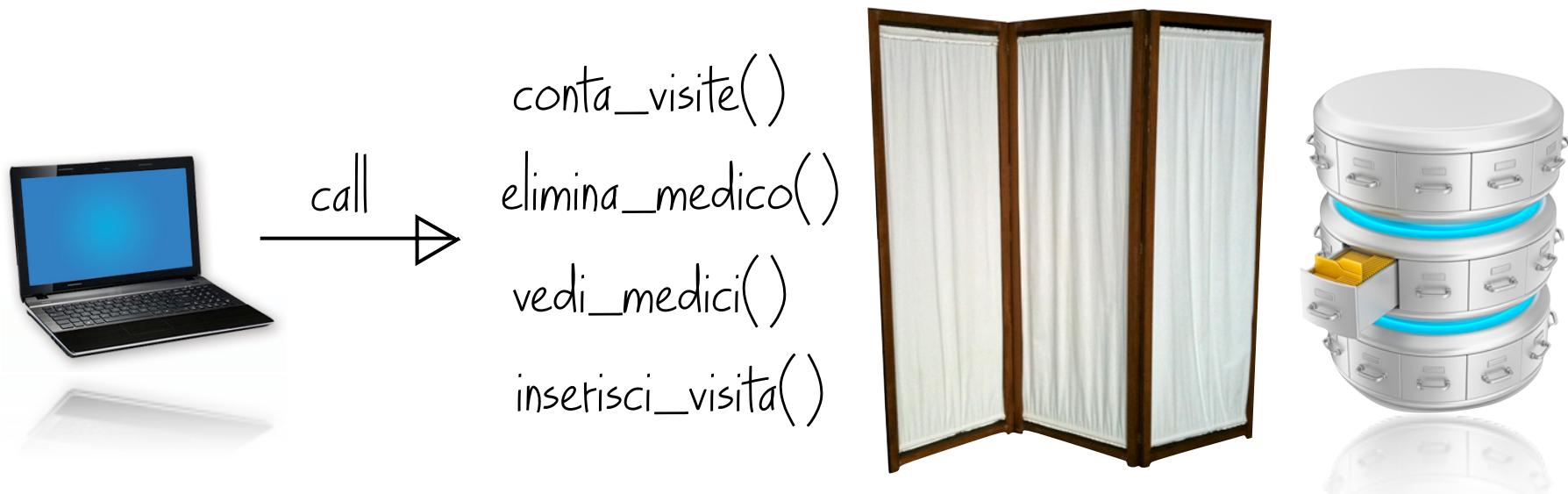


Accesso diretto ai dati



i dati devono essere accessibili (grant, e accesso alla macchina)
e si deve padroneggiare l'uso di SQL

Accesso mediante stored procedure

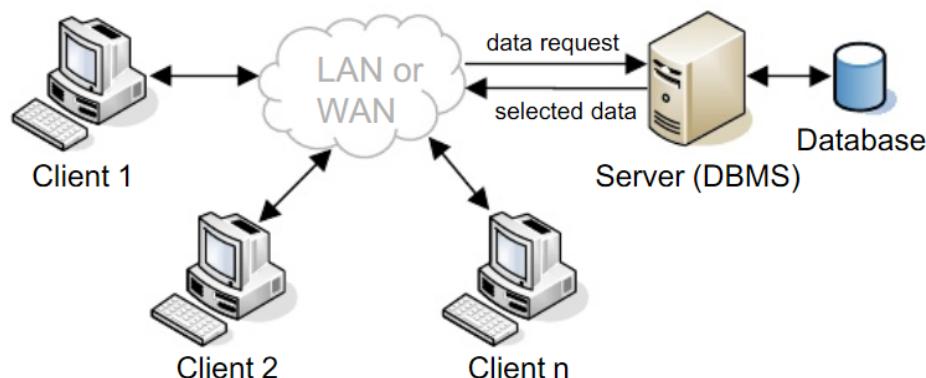


dati mascherati ed SQL mascherato

Prestazioni

il carico è spostato sul server (DBMS)

Mediante le stored procedure, le applicazioni **inviano solo una chiamata**, non il codice della query.



il traffico sulla rete è drasticamente ridotto, e l'utente non deve scrivere codice SQL complesso

Sicurezza

il codice può essere mascherato

i dati "grotzzi" non sono visibili

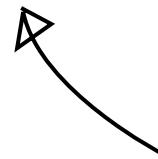
Le applicazioni possono essere autorizzate a eseguire stored procedure,
ma **avere accesso vietato alle tabelle**

restrizioni impostate con opportuni grant



Riuso del codice

devono avere i permessi per invocare le procedure



Una stored procedure è come un servizio che gli utenti delle applicazioni che usano il database possono **utilizzare senza scrivere codice**

Ma però...

carico delle CPU sul server

uso di memoria sul server



non sono banali da scrivere

debug difficoltoso

Un semplice esempio

Scrivere una stored procedure che restituisca le specializzazioni mediche offerte dalla clinica

Un semplice esempio

Scrivere una stored procedure che stampi le specializzazioni mediche offerte dalla clinica

Esecuzione

The screenshot shows the MySQL Workbench interface with the following details:

- Toolbar:** Includes icons for Run, Stop, Export Wizard, New, Load, Save, Save As, Grid View, Form View, Image, Text, and Hex.
- Tab Bar:** Shows multiple tabs, all titled "Untitled @Clinica (localhost)".
- Query Editor:** Contains the following SQL code:

```
1 DROP PROCEDURE IF EXISTS mostra_specializzazioni;
2
3 DELIMITER $$
4
5 CREATE PROCEDURE mostra_specializzazioni()
6 BEGIN
7     SELECT DISTINCT Specializzazione
8     FROM Medico;
9 END $$
```
- Status Bar:** Shows "9:7".
- Message Area:** A table titled "Message" with two rows:

Query	Message
DROP PROCEDURE IF EXISTS mostra_specializzazioni	OK
CREATE PROCEDURE mostra_specializzazioni()	OK

Chiamata

Scrivere una stored procedure che restituisca le specializzazioni mediche offerte dalla clinica

CALL mostra_specializzazioni();

chiamando la stored procedure si ottiene il risultato restituito dall'esecuzione del body

Esecuzione della chiamata

The screenshot shows the MySQL Workbench interface. The title bar reads "Untitled @db_root (localhost)". The toolbar includes icons for Run, Stop, Export Wizard, New, Load, Save, Save As, Grid View, Form View, Image, Text, and Hex. The query editor tab is active, showing the following SQL code:

```
1 CALL mostra_specializzazioni();  
2  
3
```

The result pane below displays the output of the stored procedure:

Specializzazione
Cardiologia
Gastroenterologia
Medicina generale
Nefrologia
Neurologia
Oculistica
Ortopedia
Otorinolaringoiatria
Psichiatria

At the bottom of the result pane, the command "CALL mostra_specializzazioni();" is shown again, along with the time "0.01 sec elapsed". The status bar at the bottom indicates "9 records".

Variabili locali

Sono utilizzate per **memorizzare informazioni intermedie** di ausilio



devono essere dichiarate tutte insieme prima di essere usate!

Variabili locali: sintassi

int, double, char, varchar,
date, datetime...

(tipizzazione forte, non si possono
dichiarare senza specificare il tipo)

senza default value
il valore iniziale è NULL

DECLARE nome_variabile tipo(size) **DEFAULT** valore_default;

→ capacità della variabile
(se non settata, è impostata al valore default)

Assegnamento

È possibile **assegnare un valore a una variabile** in due modalità:
istruzione **SET** oppure **SELECT+INTO**

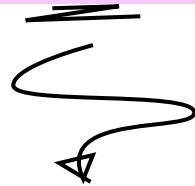
i due modi sono **alternativi**, dipende dai gusti



Assegnamento statico con SET

Creare una variabile contenente il minimo numero di visite da effettuare, impostato a 20.

```
DECLARE min_visite_mensili INT DEFAULT 0;  
SET min_visite_mensili = 20;
```



la sintassi più precisa imporrebbe l'uso di `:=`, ma i due punti
possono essere omessi se si usa l'istruzione `SET`

Assegnamento calcolato con SELECT+INTO

Creare una variabile contenente il numero di visite effettuate nel mese in corso

```
DECLARE visite_mese_attuale INT DEFAULT 0;  
  
SELECT COUNT(*) INTO visite_mese_attuale  
FROM Visita V  
WHERE MONTH(V.Data) = MONTH(CURRENT_DATE)  
      AND YEAR(V.Data) = YEAR(CURRENT_DATE);
```

Assegnamento calcolato con SET

Creare una variabile contenente il numero di visite effettuate nel mese in corso

```
DECLARE visite_mese_attuale INT DEFAULT 0;
```

```
SET visite_mese_attuale =
(
    SELECT COUNT(*)
    FROM Visita V
    WHERE MONTH(V.Data) = MONTH(CURRENT_DATE)
        AND YEAR(V.Data) = YEAR(CURRENT_DATE);
);
```

Utilizzo errato delle variabili



```
DECLARE visite_mese_attuale VARCHAR(255);  
SELECT * INTO visite_mese_attuale  
FROM Visita  
WHERE MONTH(Data) = MONTH(CURRENT_DATE)  
AND YEAR(V.Data) = YEAR(CURRENT_DATE);
```

una variabile non può contenere un result set

Variabili user-defined

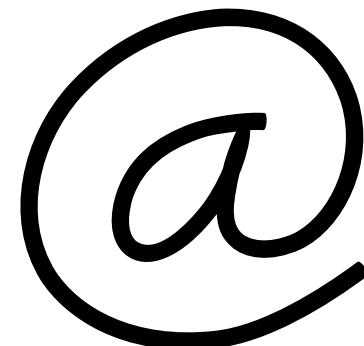
tipizzazione debole

(non si specifica il tipo della variabile, può contenere qualsiasi tipo di dato, e tipi di dato diversi in istanti diversi)

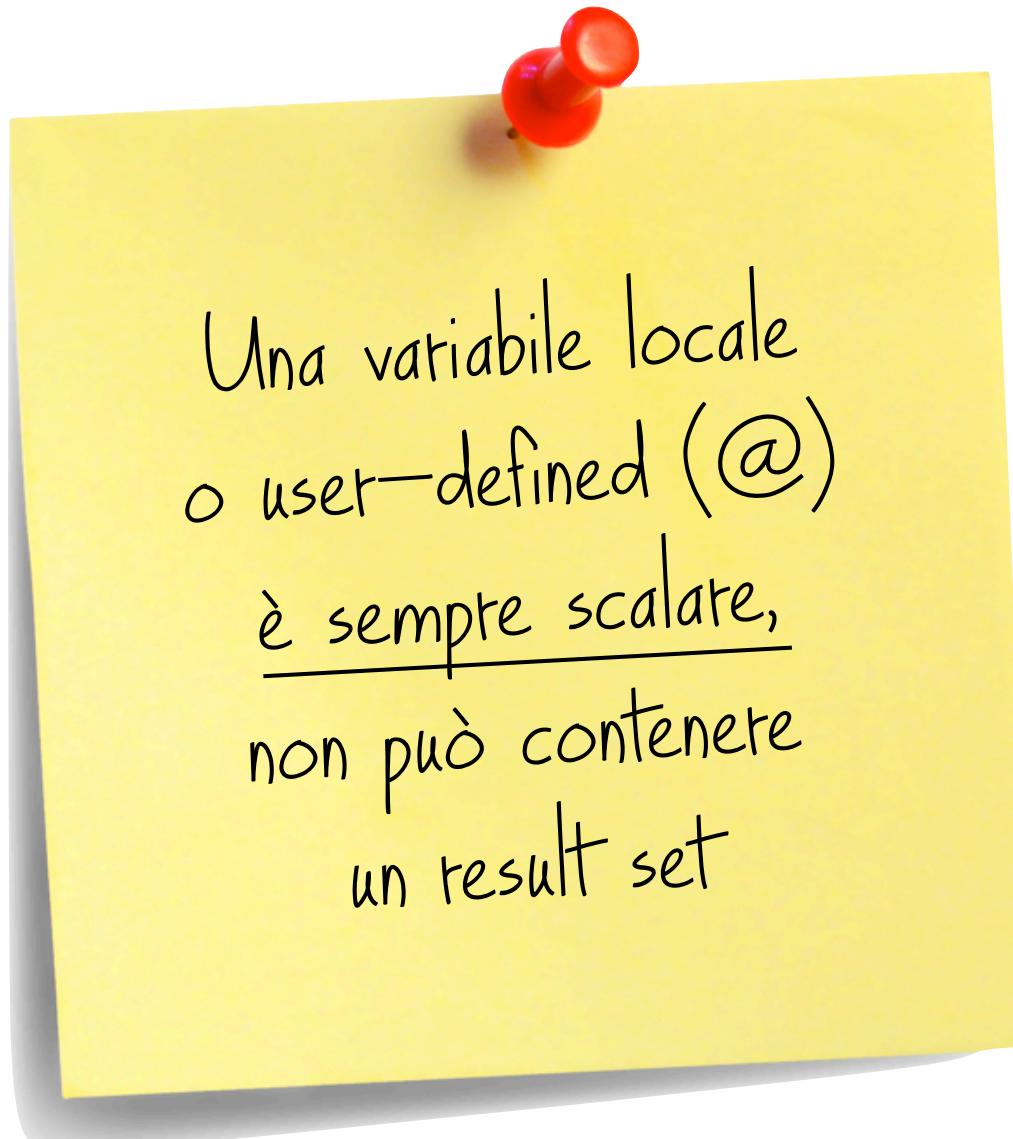


Sono inizializzate dall'utente **senza necessità di dichiarazione**, e il loro ciclo di vita equivale alla durata della connection a MySQL server

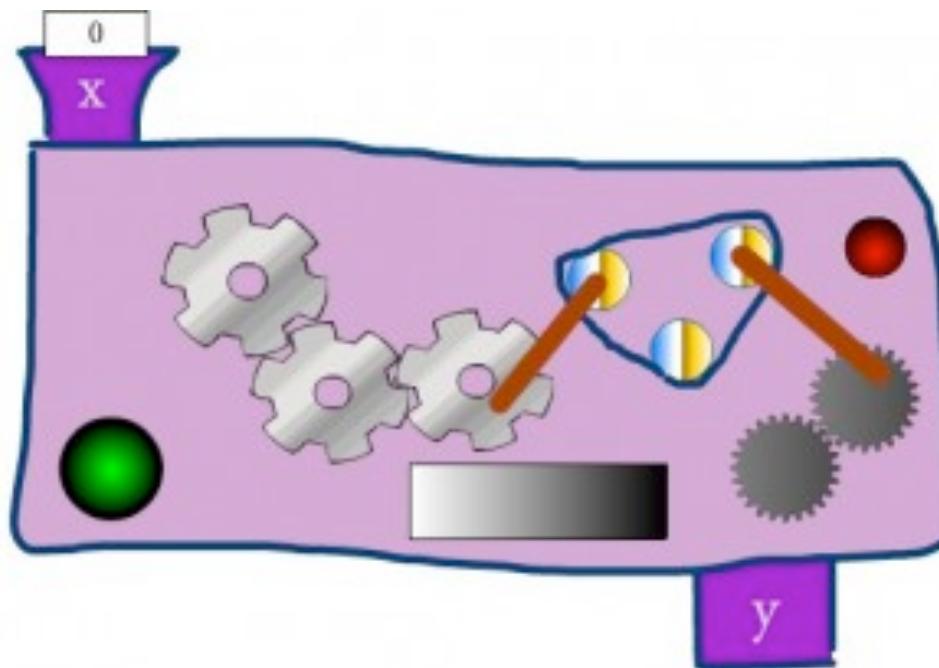
il contenuto è visibile ovunque, ma solo all'utente che le ha inizializzate, sono case insensitive e il loro identificatore deve iniziare con '@'



Attenzione

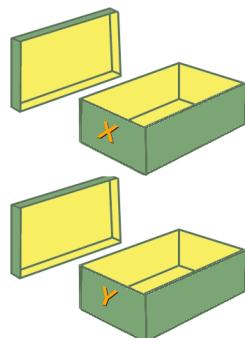


Parametri



Parametri

Una stored procedure MySQL accetta parametri di tipo
ingresso, uscita e ingresso-uscita



permettono di comunicare col chiamante

Ingresso

IN

equivalente al passaggio per valore



Un parametro in ingresso **può essere letto**, ma non modificato

i parametri sono in ingresso per default
(se non specificato diversamente)

Ingresso: esempio

Scrivere una stored procedure che stampi la parcella media di una specializzazione specificata come parametro

```
DROP PROCEDURE IF EXISTS parcella_media_spec;  
  
DELIMITER $$  
CREATE PROCEDURE parcella_media_spec(IN _specializzazione VARCHAR(100))  
BEGIN  
    SELECT AVG(M.Parcella)  
    FROM Medico M  
    WHERE M.Specializzazione = _specializzazione;  
END $$  
DELIMITER ;  
  
CALL parcella_media_spec('Ortopedia'); ← chiamata
```

Esecuzione

The screenshot shows the MySQL Workbench interface with the following details:

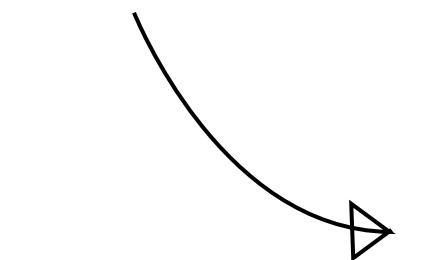
- Title Bar:** Untitled @db_root (localhost)
- Toolbar:** Run, Stop, Export Wizard, New, Load, Save, Save As, Grid View, Form View, Image, Text, Hex.
- Tab Bar:** ClinicaNe... (closed), Untitled @Clinica (closed), Untitled @Clinica (closed), Untitled @Clinica (closed), medico @db_root (closed), Untitled @db_root (closed).
- Query Editor:** Contains the following SQL code:

```
1 DROP PROCEDURE IF EXISTS parcella_media_spec;
2
3 DELIMITER $$
4
5 CREATE PROCEDURE parcella_media_spec(IN _specializzazione VARCHAR(100))
6 BEGIN
7     SELECT AVG(M.Parcella)
8     FROM Medico M
9     WHERE M.Specializzazione = _specializzazione;
10 END $$
11
12 DELIMITER ;
13
14 CALL parcella_media_spec('Ortopedia');
```
- Message Panel:** Shows the result of the query: AVG(M.Parcella) | 170.0000
- Result Panel:** Shows the call to the procedure: CALL parcella_media_spec('Ortopedia') | 0.01 sec elapsed

Uscita

OUT

Un parametro di uscita **può essere modificato** per assumere il
valore del risultato della stored procedure



nella chiamata si possono usare
variabili user-defined (quelle che iniziano con '@')

Uscita: esempio

Scrivere una stored procedure che restituisca il numero di pazienti visitati da medici di una data specializzazione, ricevuta come parametro

```
DROP PROCEDURE IF EXISTS tot_pazienti_visitati_spec;

DELIMITER $$

CREATE PROCEDURE tot_pazienti_visitati_spec(
    IN _specializzazione VARCHAR(100),
    OUT totale_pazienti_ INT)

BEGIN
    SELECT COUNT (DISTINCT V.Paziente) INTO totale_pazienti_
    FROM Visita V
        INNER JOIN
            Medico M ON V.Medico = M.Matricola
    WHERE M.Specializzazione = _specializzazione;
END $$

DELIMITER ;
```

```
CALL tot_pazienti_visitati_spec('Neurologia', @quantiPazienti);
```

```
SELECT @quantiPazienti;
```

Esecuzione

The screenshot shows the MySQL Workbench interface with the following details:

- Toolbar:** Includes Run, Stop, Export Wizard, New, Load, Save, Save As, Grid View, Form View, Image, Text, and Hex.
- Query Editor:** Titled "Untitled @db_root (localhost)". It contains the following SQL code:

```
1 DROP PROCEDURE IF EXISTS tot_pazienti_visitati_spec;
2
3 DELIMITER $$ 
4 CREATE PROCEDURE tot_pazienti_visitati_spec(
5     IN _specializzazione VARCHAR(100),
6     OUT totale_pazienti_ INT)
7 BEGIN
8     SELECT COUNT(DISTINCT V.Paziente) INTO totale_pazienti_
9     FROM Visita V
10    INNER JOIN
11        Medico M ON V.Medico = M.Matricola
12    WHERE M.Specializzazione = _specializzazione;
13 END $$ 
14 DELIMITER ;
15
16 CALL tot_pazienti_visitati_spec('Neurologia', @quantiPazienti);
17
18 SELECT @quantiPazienti;
```

- Result Window:** Shows the output of the last query: "@quantiPazienti | 18".
- Status Bar:** Displays "SELECT @quantiPazienti" and "0.00 sec elapsed".
- Bottom Buttons:** Includes standard window control buttons (+, -, checkmark, X).

Ingresso-uscita

INOUT

La stored procedure riceve il parametro e **può leggerlo e modificarlo**,
la modifica è visibile al chiamante

equivalente al passaggio per reference

Ingresso-uscita: esempio

Considerare una variabile user-defined contenente la data in cui è stata effettuata la visita più recente. Scrivere una stored procedure che aggiorna la variabile.

Ingresso-uscita: esempio

Considerare una variabile user-defined contenente la data in cui è stata effettuata la visita più recente. Scrivere una stored procedure che aggiorna la variabile.

```
SET @dataUltimaVisita = (
    SELECT V.Data
    FROM Visita V
    WHERE V.Data = (
        SELECT MAX(Data)
        FROM Visita
    )
);
```

Ingresso-uscita: esempio

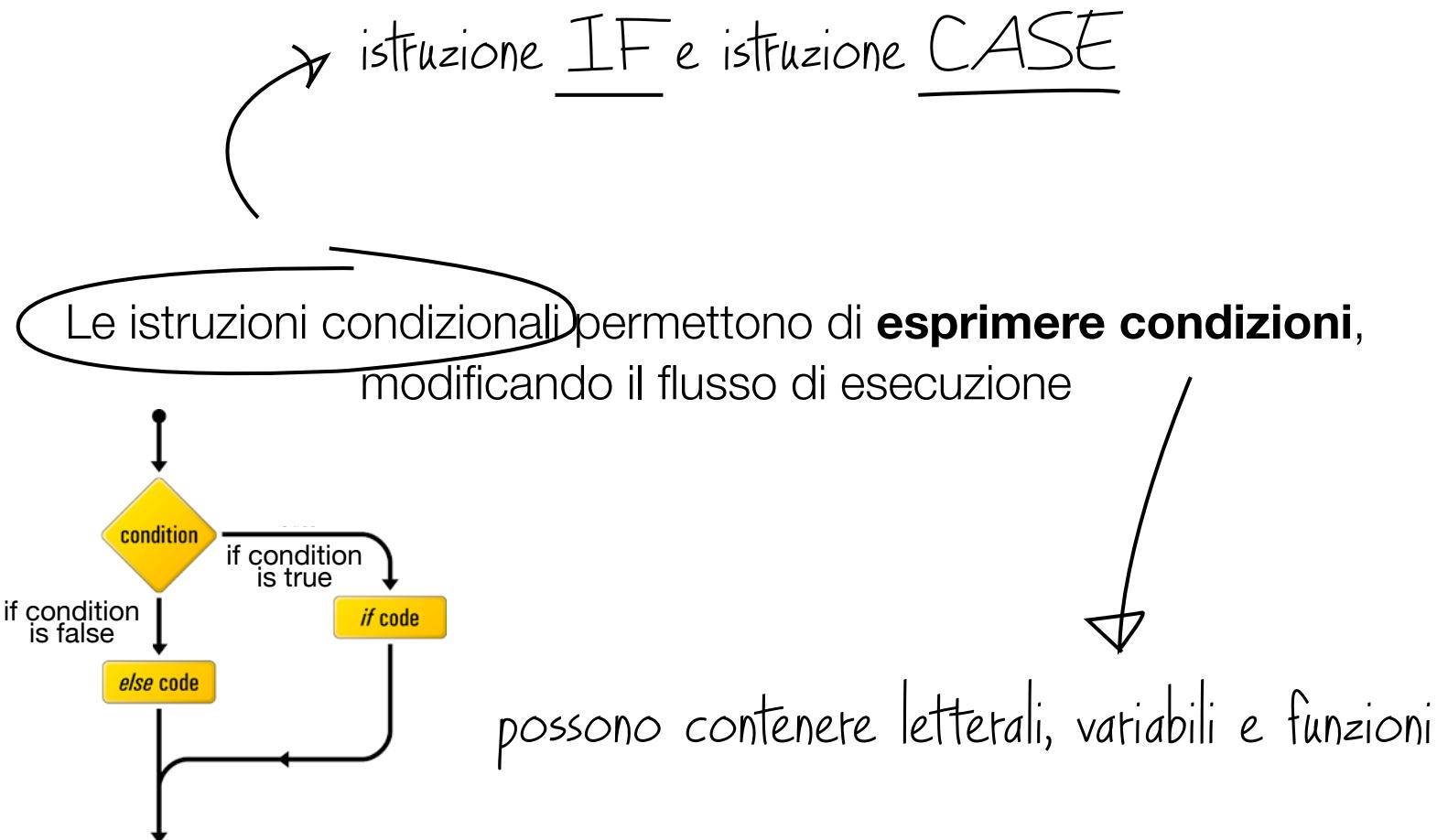
Considerare una variabile user-defined contenente la data in cui è stata effettuata la visita più recente. Scrivere una stored procedure che aggiorna la variabile.

```
DELIMITER $$  
CREATE PROCEDURE aggiorna_data_ultima_visita(  
    INOUT _data_UV_ VARCHAR(100))  
BEGIN  
  
    SELECT MAX(Data) INTO _data_UV_  
    FROM Visita;  
  
    END $$  
DELIMITER ;  
  
CALL aggiorna_data_ultima_visita(@dataUltimaVisita);  
SELECT @dataUltimaVisita;
```

Istruzioni condizionali



Istruzioni condizionali



Istruzione IF

parte facoltativa

IF if_condition **THEN**

-- blocco istruzioni if true

ELSEIF elseif_1_condition **THEN**

-- blocco istruzioni elseif_1

:

ELSEIF elseif_N_condition **THEN**

-- blocco istruzioni elseif_N

ELSE

-- blocco istruzioni else

END IF ;

obbligatorio il punto e virgola!

Esempio

Scrivere una stored procedure che riceva come parametro un intero t e una specializzazione s e restituisca in uscita `true` se il totale delle visite della specializzazione s nel mese in corso è superiore all'intero t , `false` se è inferiore e `NULL` se è uguale.

Soluzione

```
1  DROP PROCEDURE IF EXISTS visite_sopra_soglia;
2
3  DELIMITER $$ 
4  CREATE PROCEDURE visite_sopra_soglia(IN _t INT, IN _s VARCHAR(100), OUT passed BOOLEAN)
5  BEGIN
6      DECLARE visite_mese_attuale INT DEFAULT 0;
7      SET visite_mese_attuale = (
8          SELECT COUNT(*)
9              FROM Visita V
10             INNER JOIN
11                 Medico M ON V.Medico = M.Matricola
12            WHERE M.Specializzazione = _s
13            AND MONTH(V.`Data`) = MONTH(CURRENT_DATE)
14            AND YEAR(V.`Data`) = YEAR(CURRENT_DATE)
15      );
16
17      IF visite_mese_attuale > _t THEN
18          SET passed = TRUE;
19      ELSEIF visite_mese_attuale < _t THEN
20          SET passed = FALSE;
21      ELSE
22          SET passed = NULL;
23  END IF;
24  END $$ 
25  DELIMITER ;
26
27  CALL visite_sopra_soglia(10, 'Otorinolaringoiatria', @controllo);
```

Istruzione CASE

CASE

WHEN condition_1 **THEN**

-- blocco istruzioni_1

:

WHEN condition_N **THEN**

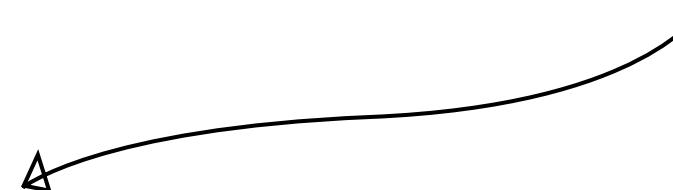
-- blocco istruzioni_N

END CASE ;

equivale all'istruzione 'switch' del C++

Più variabili di uscita

Scrivere una stored procedure che restituisca la data in cui un paziente, il cui codice fiscale è passato come parametro, è stato visitato per la prima volta, e il nome e cognome del medico che lo ha visitato in tale circostanza. In caso di più medici, per semplicità, selezionarne uno.



Considerando che Data in Visita fa parte della chiave, in uno stesso giorno più medici possono visitare lo stesso paziente

Soluzione

Scrivere una stored procedure che restituisca la data in cui un paziente, il cui codice fiscale è passato come parametro, è stato visitato per la prima volta, e il nome e cognome del medico che lo ha visitato in tale circostanza. In caso di più medici, per semplicità, selezionarne uno.

```
1  DELIMITER $$  
2  ▼CREATE PROCEDURE primaVisitaPaziente(IN paziente VARCHAR(16),  
3                                         OUT dataPrimaVisita DATE,  
4                                         OUT cognomeMedico VARCHAR(100),  
5                                         OUT nomeMedico VARCHAR(100))  
6  ▲  
7  ▼BEGIN  
8      SELECT MIN(V.`Data`) INTO dataPrimaVisita  
9      FROM Visita V  
10     WHERE V.Paziente = paziente;  
11  
12     SELECT M.Cognome, M.Nome INTO cognomeMedico, nomeMedico  
13     FROM Visita V INNER JOIN Medico M ON V.Medico = M.Matricola  
14     WHERE V.Paziente = paziente  
15         AND V.`Data` = dataPrimaVisita  
16         LIMIT 1;  
17  ▲END $$  
18  DELIMITER ;
```

Un dubbio ci attanaglia...

...e se volessi restituire un insieme
di record come risultato?



attendete prego...

Istruzioni iterative



Istruzioni iterative

→ istruzioni WHILE, REPEAT e LOOP

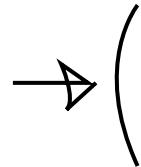
Le istruzioni iterative permettono di **ripetere blocchi di codice**,
dipendentemente dalla veridicità di una condizione



)
la condizione può valutare anche funzioni

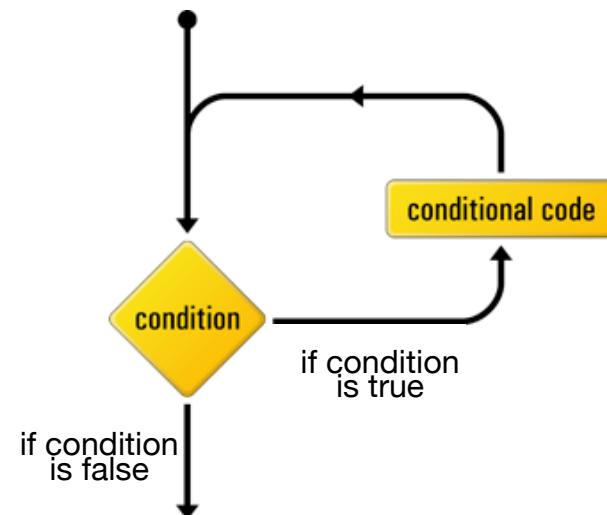
While

eseguito finché
la condizione è
TRUE



WHILE condition **DO**
 -- blocco istruzioni
END WHILE;

la condizione è controllata prima di ogni iterazione



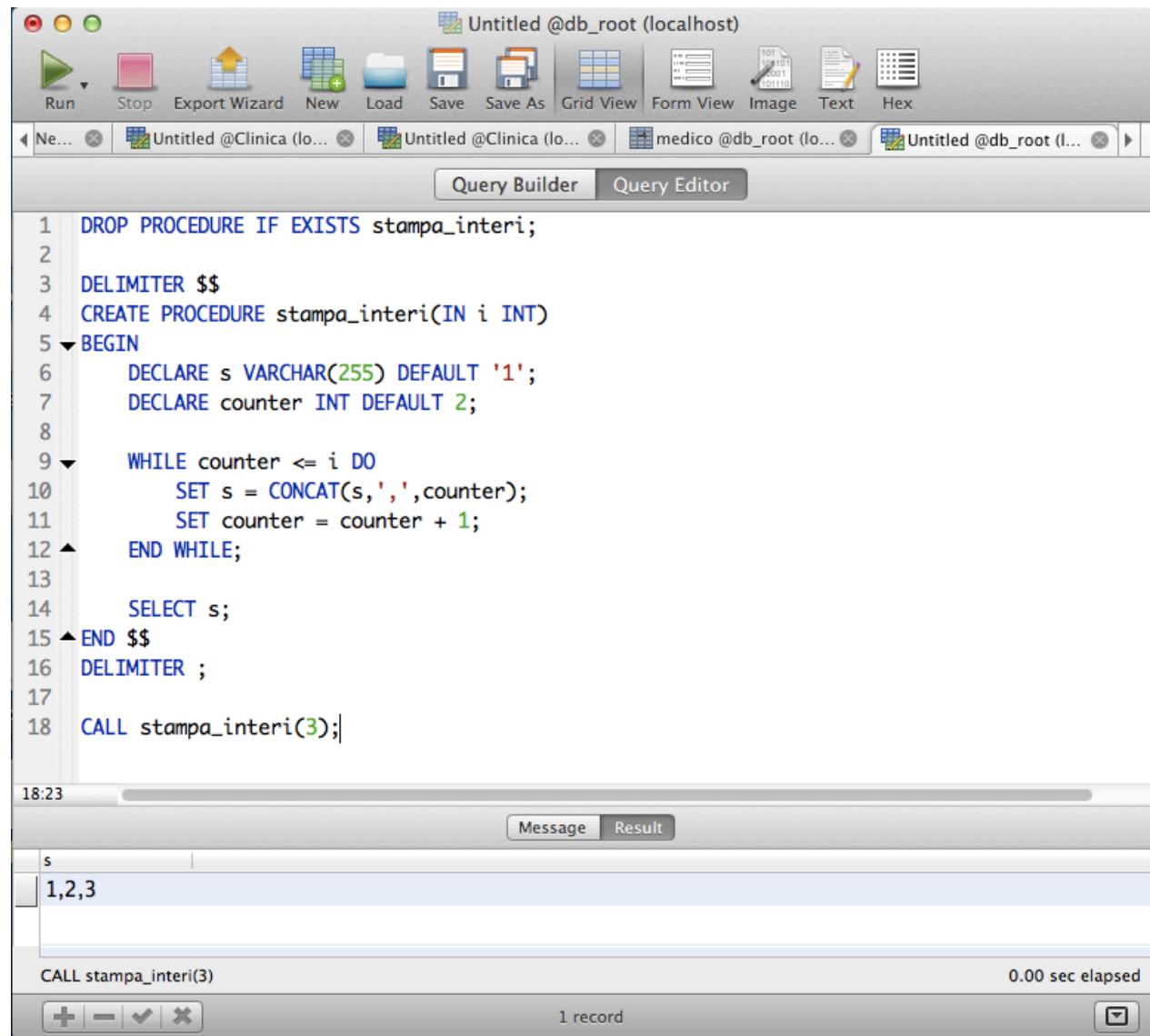
Esempio...programmazione spinta

Scrivere una stored procedure che riceve in ingresso un intero i e stampa a video i primi i interi separati da virgola, in ordine crescente



Esempio: se $i=3$, l'uscita deve essere 1, 2, 3

Soluzione



```
1 DROP PROCEDURE IF EXISTS stampa_interi;
2
3 DELIMITER $$;
4 CREATE PROCEDURE stampa_interi(IN i INT)
5 BEGIN
6     DECLARE s VARCHAR(255) DEFAULT '';
7     DECLARE counter INT DEFAULT 1;
8
9     WHILE counter <= i DO
10         SET s = CONCAT(s, ',', counter);
11         SET counter = counter + 1;
12     END WHILE;
13
14     SELECT s;
15 END $$;
16 DELIMITER ;
17
18 CALL stampa_interi(3);
```

18:23

Message Result

s
1,2,3

CALL stampa_interi(3) 0.00 sec elapsed

+ - ✓ ✎

1 record

Repeat



eseguito fintantoché non
si verifica la condizione

(UNTIL contiene
una condizione d'uscita)

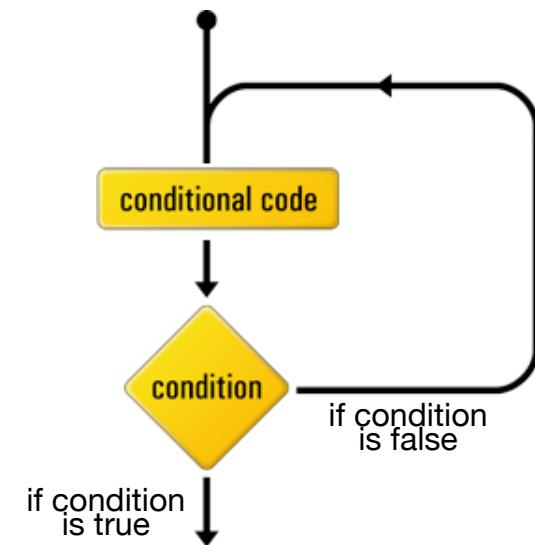
la condizione è controllata dopo ogni iterazione

REPEAT

-- blocco istruzioni

UNTIL condition

END REPEAT:



Esempio precedente con REPEAT

The screenshot shows the MySQL Workbench interface with the following details:

- Toolbar:** Run, Stop, Export Wizard, New, Load, Save, Save As, Grid View, Form View, Image, Text, Hex.
- Tab Bar:** Untitled @db_root (localhost), Untitled @Clinica (localhost), Untitled @Clinica (localhost), medico @db_root (localhost), Untitled @db_root (localhost).
- Query Editor:** Contains the following MySQL code:

```
1 DROP PROCEDURE IF EXISTS stampa_interi2;
2
3 DELIMITER $$ 
4 CREATE PROCEDURE stampa_interi2(IN i INT)
5 BEGIN
6     DECLARE s VARCHAR(255) DEFAULT '1';
7     DECLARE counter INT DEFAULT 2;
8
9     REPEAT
10        SET s = CONCAT(s,',',counter);
11        SET counter = counter + 1;
12    UNTIL counter > i
13    END REPEAT;
14
15    SELECT s;
16 END $$ 
17 DELIMITER ;
18
19 CALL stampa_interi2(3);
```
- Message/Result Tab:** Shows the output of the query.

s
1,2,3
- Statistics:** CALL stampa_interi2(3) | 0.01 sec elapsed | 1 record.

Loop

loop_label: **LOOP**

-- blocco istruzioni e check di condizioni

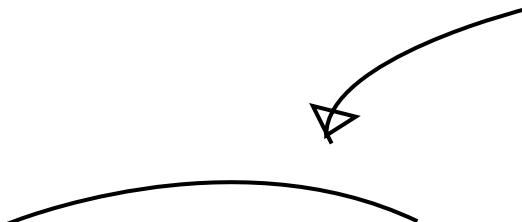
END LOOP;



le condizioni di uscita sono gestite dal programmatore

Istruzioni di salto

sono sempre usate in concomitanza con blocchi if o case



Le istruzioni **LEAVE** e **ITERATE** permettono di **interrompere** un ciclo o **passare all'iterazione successiva**, rispettivamente



leave è l'equivalente di break, mentre iterate di continue...

Esempio

Scrivere una stored procedure che riceve in ingresso un intero i e stampa a video i primi i interi dispari, separati da virgola

Esempio: se $i=5$, l'uscita deve essere 1, 3, 5

Soluzione

```
1  DROP PROCEDURE IF EXISTS stampa_dispari;
2  DELIMITER $$ 
3  CREATE PROCEDURE stampa_dispari(IN i INT)
4  BEGIN
5      DECLARE s VARCHAR(255) DEFAULT '';
6      DECLARE counter INT DEFAULT 1;
7
8      IF i>0 THEN
9          SET s = '1';
10     END IF;
11
12     scan: LOOP
13         SET counter = counter + 1;
14
15         IF counter = i+1 THEN
16             LEAVE scan;
17         END IF;
18
19         IF (counter % 2) = 0 THEN
20             ITERATE scan;
21         ELSE
22             SET s = CONCAT(s,',',counter);
23         END IF;
24     END LOOP;
25
26     SELECT s;
27 END $$ 
28 DELIMITER ;
```

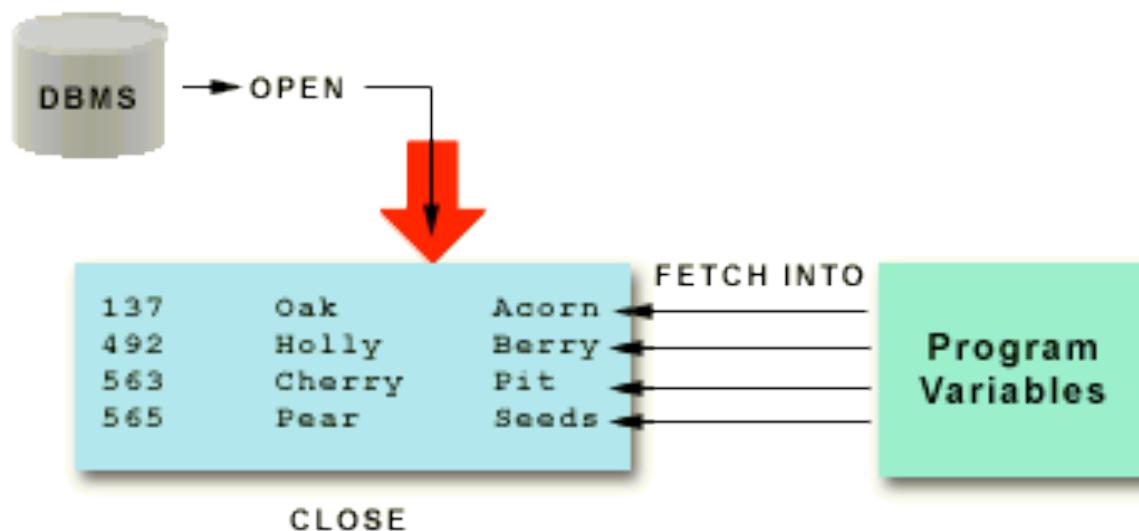
```
graph TD; L12[12 scan: LOOP] --> L13[13 SET counter = counter + 1]; L13 --> L15[15 IF counter = i+1 THEN]; L15 --> L16[16 LEAVE scan]; L16 --> L17[17 END IF]; L17 --> L19[19 IF (counter % 2) = 0 THEN]; L19 --> L20[20 ITERATE scan]; L20 --> L21[21 ELSE]; L21 --> L22[22 SET s = CONCAT(s,',',counter)]; L22 --> L23[23 END IF]; L23 --> L24[24 END LOOP]; L24 --> L26[26 SELECT s]; L26 --> L12;
```

Qual è l'errore...se c'è?

```
1  DROP PROCEDURE IF EXISTS stampa_dispari_maligna;
2  DELIMITER $$ 
3  CREATE PROCEDURE stampa_dispari_maligna(IN i INT)
4  BEGIN
5      DECLARE s VARCHAR(255) DEFAULT '';
6      DECLARE counter INT DEFAULT 1;
7
8      IF i>0 THEN
9          SET s = '1';
10     END IF;
11
12     scan: LOOP
13     IF counter = i THEN
14         LEAVE scan;
15     END IF;
16
17     IF (counter % 2) = 0 THEN
18         ITERATE scan;
19     ELSE
20         SET s = CONCAT(s, ',' ,counter);
21     END IF;
22
23     SET counter = counter + 1; ←
24 END LOOP;
25
26     SELECT s;
27 END $$ 
28 DELIMITER ;
```

non è mai eseguita e genera
un loop infinito

Cursori



Cursori

come farebbe un puntatore

Permettono di **scorrere un result set**, solo in avanti, per effettuare delle azioni all'interno di istruzioni iterative



Cursori: dichiarazione

```
DECLARE NomeCursore CURSOR FOR  
SQL query ;
```

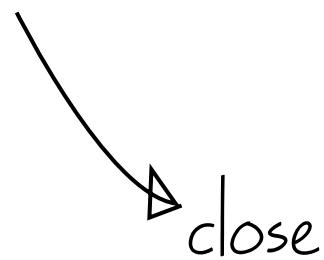


i cursori si possono dichiarare solo immediatamente dopo
la dichiarazione di tutte le variabili

Cursori: apertura, fetch e chiusura



Per usare un cursore lo si deve **aprire**, poi è possibile **effettuare il prelievo** riga per riga, infine lo si deve **chiudere**



Sintassi



apertura

OPEN NomeCursore;



prelievo

FETCH NomeCursore **INTO** ListaVariabili



chiusura

CLOSE NomeCursore;

Handler

È un **gestore di situazioni**, utile *tra l'altro* nei cursori per riconoscere quando esso è giunto alla fine del result set



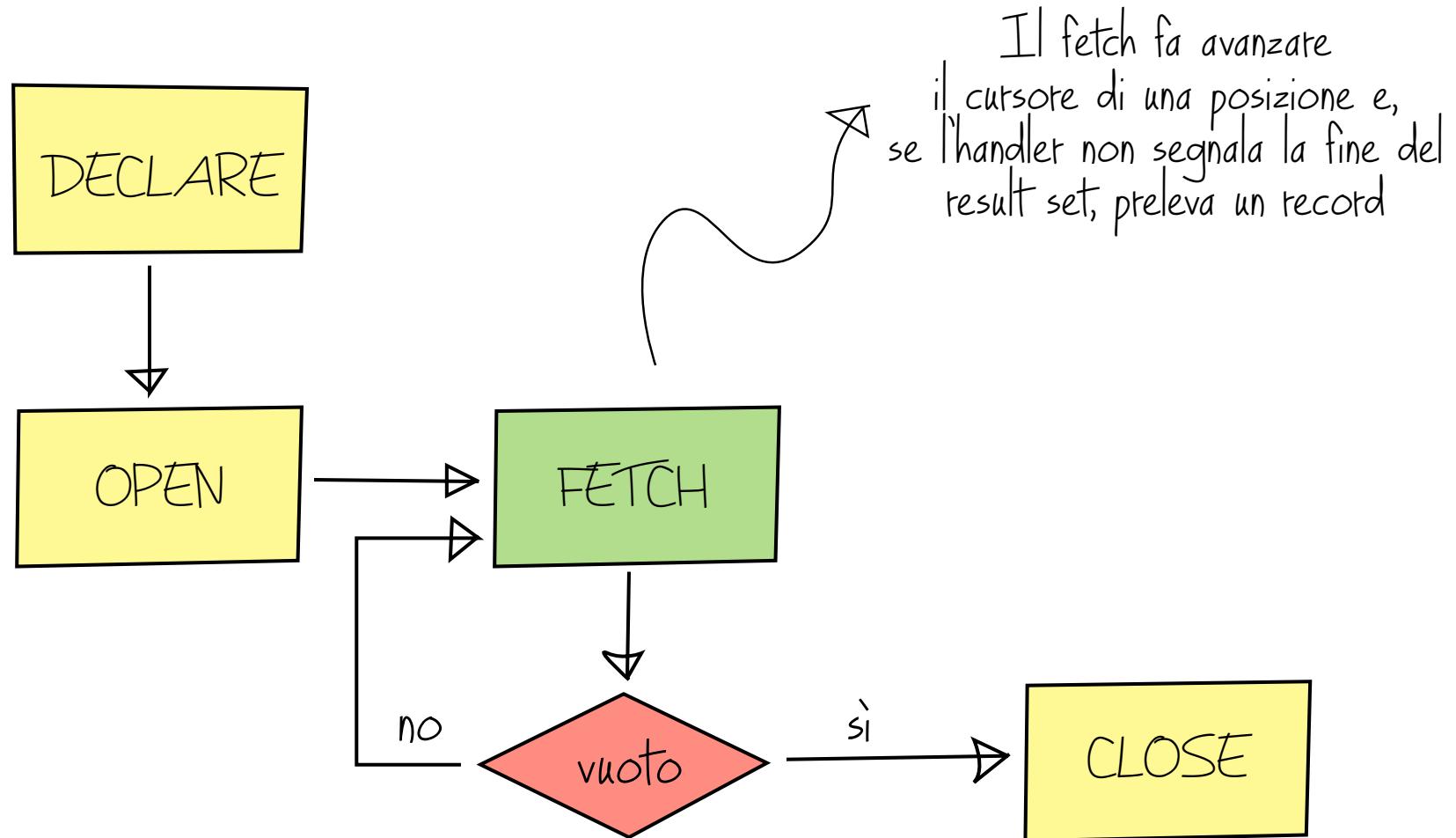
possono essere definiti dall'utente *dopo* le definizioni delle variabili e dei cursori

Esempio: il not found handler

```
DECLARE CONTINUE HANDLER  
FOR NOT FOUND SET finito = 1;
```

“continue” significa che il flusso continua, uscendo
per sempre dal cursore

Come funziona un cursore?



Stored procedure con cursore

Scrivere una stored procedure che restituisca il codice fiscale dei pazienti visitati da un solo medico per una data specializzazione, organizzati in una stringa formattata del tipo “codFiscale1, codFiscale2, … , codFiscaleN”

```

1 DELIMITER $$ 
2 CREATE PROCEDURE PazientiSingoloMedico(IN specializzazione CHAR,
3                                     INOUT codiciFiscali VARCHAR (255))
4 BEGIN
5     DECLARE finito INTEGER DEFAULT 0;
6     DECLARE codiceFiscale VARCHAR(255) DEFAULT "";
7
8     -- dichiarazione del cursore
9     DECLARE cursoreCodici CURSOR FOR
10        SELECT V.Paziente
11        FROM Visita V INNER JOIN Medico M ON V.Medico = M.Matricola
12        WHERE M.Specializzazione = specializzazione
13        GROUP BY V.Paziente
14        HAVING COUNT(DISTINCT V.Medico) = 1;
15
16     -- dichiarazione handler
17     DECLARE CONTINUE HANDLER
18         FOR NOT FOUND SET finito = 1;
19
20     OPEN cursoreCodici;
21
22     -- ciclo di fetch per il prelievo
23     preleva: LOOP
24         FETCH cursoreCodici INTO codiceFiscale;
25         IF finito = 1 THEN
26             LEAVE preleva;
27         END IF;
28         SET codiciFiscali = CONCAT(codiceFiscale, ";", codiciFiscali);
29     END LOOP preleva;
30     CLOSE cursoreCodici;
31 END $$ 
32 DELIMITER ;

```

Nota: la capienza massima di un varchar è 20000 caratteri, per esigenze maggiori usare text.

Chiamata

```
SET @codiciPazienti = '';
CALL PazientiSingoloMedico('Ortopedia', @codiciPazienti);
SELECT @codiciPazienti;
```

il risultato viene inserito in `@codiciPazienti` come
unica stringa formattata

Gestione degli errori



Gestione degli errori

MySQL permette di gestire agevolmente **errori ed eccezioni**

si basa sugli handler



il blocco di codice responsabile può essere
terminato o ripreso effettuando l'exception handling

Handler: sintassi e semantica



DECLARE action **HANDLER FOR** condition_value
statement(s);



se si verifica una condizione il cui valore è condition_value,
MySQL esegue statement(s) e continua/termina il blocco
attuale dipendentemente dalla tipologia di action scelta

Action: possibili valori

l'esecuzione del blocco begin-end continua



Può assumere un valore a scelta fra **CONTINUE** oppure **EXIT**



termina l'esecuzione del blocco in cui
l'handler è dichiarato

Condition value

valori SQLSTATE

codici errore MySQL

SQLEXCEPTION

Particolare condizione (o classe di condizioni) che **attivano l'handler**

SQLWARNING

NOTFOUND

Statement

Può essere un'istruzione **semplice oppure un blocco**

racchiuso fra begin-end



Continue handler: esempio

Scrivere un handler che imposta a 1 una variabile d'errore qualora si verifichi un'eccezione, ma non interrompa comunque la processazione.

l'esecuzione non termina

```
DECLARE CONTINUE HANDLER FOR SQLEXCEPTION  
SET errore = 1;
```



quando l'handler si attiva,
imposta la variabile "errore" a 1

Exit handler: esempio

Scrivere un handler che, al verificarsi di un'eccezione, annulli tutte le precedenti operazioni, stampi un messaggio d'errore, e interrompa la processazione.

l'esecuzione termina

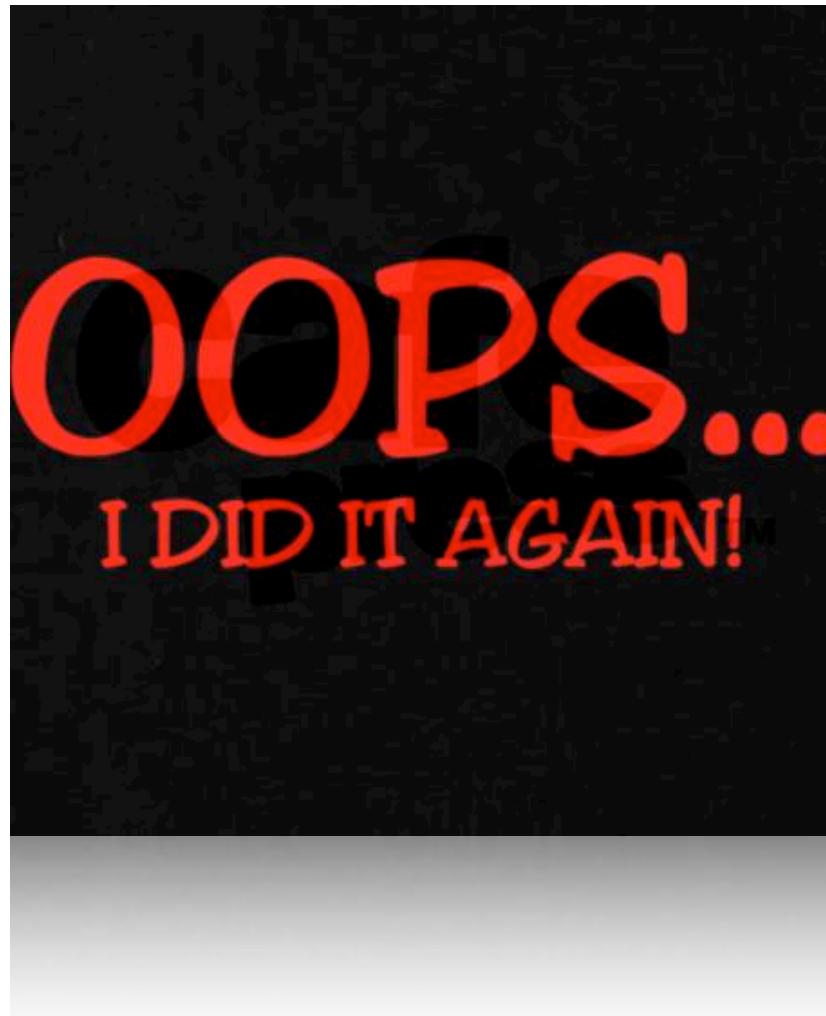
```
DECLARE EXIT HANDLER FOR SQLEXCEPTION
BEGIN
    ROLLBACK;
    SELECT 'Si è verificato un errore!';
END;
```

Not found handler

Scrivere un handler che, al verificarsi di un'eccezione di tipo NOT FOUND, imposti a 1 una variabile d'errore e permetta di continuare la processazione

```
DECLARE CONTINUE HANDLER FOR NOT FOUND  
    SET errore = 1;
```

Sollevare errori



Sollevare errori

fatto grave ↪
avvertimento minore ↪

È possibile restituire un **errore** o un **warning** al chiamante di una stored procedure

Conclusion qualcosa non è andato come avrebbe dovuto

Signal

dichiarata con declare condition



```
SIGNAL SQLSTATE | condizione_etichettata  
SET condition_information_item_name1 = value1,  
    condition_information_item_name2 = value2,  
    ...  
    condition_information_item_nameN = valueN;  
  
    ➔ MESSAGE_TEXT, MYSQLERRNO, ...
```

Esempio

Scrivere una stored procedure che aumenta la parcella di un dato medico di 10 euro solo se questa non è superiore alla media delle parcelle.

Nel caso la parcella non sia da aggiornare, deve essere sollevato un errore generico (SQLSTATE ‘45000’) che ne specifichi la causa

```

1  DELIMITER $$

2

3  DROP PROCEDURE IF EXISTS aumentaParcella;
4  CREATE PROCEDURE aumentaParcella(matricolaMedico VARCHAR(255))
5
6  BEGIN
7
8      DECLARE mediaParcelle DOUBLE DEFAULT 0;
9      DECLARE parcellaMedico DOUBLE DEFAULT 0;
10
11     -- calcolo della parcella media dei medici
12     SELECT AVG(Parcella) INTO mediaParcelle
13     FROM Medico;
14
15     -- ricerca della parcella del medico
16     SELECT Parcella INTO parcellaMedico
17     FROM Medico
18     WHERE Matricola = matricolaMedico;
19
20     -- controllo di flusso
21     IF parcellaMedico >= mediaParcelle THEN
22         SIGNAL SQLSTATE '45000'
23         SET MESSAGE_TEXT = 'Medico con parcella superiore alla media';
24     ELSE
25         UPDATE Medico
26             SET Parcella = Parcella + 10
27             WHERE Matricola = matricolaMedico;
28     END IF
29
30 END $$

31
32 DELIMITER ;

```

Stored function



Stored function

Restituiscono un solo valore e sono **richiamabili anche da statement SQL**

utili per encapsulare formule o aggregazioni personalizzate

Stored function: sintassi



CREATE FUNCTION function_name(parametro1, ..., parametroN)
RETURNS datatype **DETERMINISTIC** | **NOT DETERMINISTIC**



crea una funzione `function_name`, avente i parametri
`{parametro1, ..., parametroN}`, che restituisce un risultato di tipo
datatype, deterministico o no.

Cosa vuol dire risultato deterministico?

Una function è deterministica se restituisce un **risultato invariante** a fronte delle chiamate effettuate con gli stessi valori per i parametri d'ingresso

...una function che calcola la media è deterministica



Esempio

Scrivere una function che, preso in ingresso un numero di visite effettuate da un medico, restituiscia: ‘low’ se il numero di visite è inferiore a 20; ‘medium’ se il numero di visite è compreso fra 20 e 50; ‘high’ se il numero di visite supera 50.

deterministica o non deterministica?



Soluzione

```
1  DELIMITER $$  
2  DROP FUNCTION IF EXISTS rank;  
3  
4  CREATE FUNCTION rank(totaleVisite INT)  
5    RETURNS VARCHAR(6) DETERMINISTIC  
6  
7  BEGIN  
8      -- variabile risultato  
9      DECLARE ranking VARCHAR(6) DEFAULT "";  
10  
11     -- assegnamento del ranking su condizione  
12     CASE  
13         WHEN totaleVisite < 20 THEN  
14             SET ranking = 'low';  
15         WHEN totaleVisite BETWEEN 20 AND 50 THEN  
16             SET ranking = 'medium';  
17         WHEN totaleVisite > 50 THEN  
18             SET ranking = 'high';  
19     END CASE;  
20  
21     -- restituzione del risultato  
22     RETURN (ranking);  
23  END $$  
24  DELIMITER ;
```

Chiamata

Una function può essere chiamata dalle **query SQL**, dalle **stored procedure**, dai **trigger** e dagli **event**



Utilizzo di una function

Scrivere una stored procedure che restituisca la posizione in classifica nel mese in corso di un medico, passato come parametro, sfruttando la function `rank()`

```

1  DELIMITER $$ 
2  DROP PROCEDURE IF EXISTS classificaMedico;
3  CREATE PROCEDURE classificaMedico(IN matricola VARCHAR(5), OUT classe VARCHAR(6))
4
5  BEGIN
6
7      DECLARE visiteMeseCorrente INT DEFAULT 0;
8      DECLARE matricolaValida INT DEFAULT 0;
9
10     SELECT COUNT(*) INTO matricolaValida
11     FROM Medico M
12     WHERE M.Matricola = matricola;
13
14    IF matricolaValida = 0 THEN
15        BEGIN
16            SET classe = 'NULL';
17            SIGNAL SQLSTATE '45000'
18            SET MESSAGE_TEXT = 'Medico inesistente!';
19        END;
20    ELSE
21        BEGIN
22            SELECT COUNT(*) INTO visiteMeseCorrente
23            FROM Visita V
24            WHERE V.Medico = matricola
25            AND MONTH(V.Data) = MONTH(CURRENT_DATE);
26
27            SELECT rank(visiteMeseCorrente) INTO classe;
28        END;
29    END IF;
30
31 END $$ 
32 DELIMITER ;

```

Il colpo finale

Scrivere una stored procedure che produca **una classifica di tutti i medici** della clinica sfruttando la function `rank()`

Temporary table

sono cancellate alla fine della sessione

Sono tabelle **temporanee** che mantiene risultati utili all'interno di una sessione



Ma...



Temporary table vs. view

Una temporary table **non è ricalcolata** ogni volta che viene utilizzata



è a tutti gli effetti una tabella, ma al momento dell'utilizzo potrebbe contenere dati non più aggiornati

Ranking con temporary table

Scrivere una stored procedure che produca **una classifica di tutti i medici** della clinica sfruttando la function rank()

```
1  DELIMITER $$  
2  DROP PROCEDURE IF EXISTS classificaMedici;  
3  CREATE PROCEDURE classificaMedici()  
4  
5    BEGIN  
6      CREATE TEMPORARY TABLE IF NOT EXISTS _Classifica(  
7          Medico varchar(100) NOT NULL,  
8          Rank varchar(10) NOT NULL,  
9          PRIMARY KEY (Medico)  
10     ) ENGINE=InnoDB DEFAULT CHARSET=latin1;  
11  
12     TRUNCATE TABLE _Classifica;  
13  
14     INSERT INTO _Classifica  
15         SELECT V.Medico, rank(COUNT(*))  
16         FROM Visita V  
17         WHERE MONTH(V.Data) = MONTH(CURRENT_DATE)  
18         GROUP BY V.Medico;  
19     END $$  
20  DELIMITER ;
```

Il comando "truncate table" permette di svuotare la tabella

Chiamata

```
1 CALL classificaMedici();
2 SELECT * FROM _Classifica;
```

ranking dei medici
rispetto ai criteri
della funzione rank()



Medico	Rank
001	high
003	low
004	low
005	low
006	medium
007	low
008	high
009	low
010	high
011	low
014	low
015	low
017	medium
018	low

Per casa

Per casa vi lascio alcuni esercizi d'esame, li potete trovare sulla mia pagina. Ci sono anche le soluzioni, ma non guardatele, altrimenti secondo me non serve a niente.

Es. 4 del 10-07-2014

Es. 3 del 26-07-2014

Es. 4 del 08-01-2015

Es. 4 del 25-02-2015

Es. 3 del 10-06-2015

Es. 3 del 26-09-2015

Giovedì **10 Maggio** faremo **1 sola ora** (dalle 11:30 alle 12:30) a causa della sospensione della didattica. Durante tale ora, farò le 10 slide finali di questo pacchetto (cioè le **stored function**) e vedremo **almeno uno degli esercizi per casa**. Recupererò la prossima settimana il tempo sottratto all'esercitazione dalla sospensione della didattica.