23. MySQL: Stored procedures e stored functions

Le **stored procedures** (procedure memorizzate) sono un'altra delle caratteristiche la cui assenza è stata a lungo sottolineata dai detrattori di MySQL: con la versione 5.0 si è finalmente posto rimedio a questa assenza.

Una stored procedure è un insieme di istruzioni SQL che vengono memorizzate nel server con un nome che le identifica; tale nome consente in seguito di rieseguire l'insieme di istruzioni facendo semplicemente riferimento ad esso. Vediamo come creare una stored procedure:

CREATE PROCEDURE *nome* ([*parametro*[,...]])
[SQL SECURITY { DEFINER | INVOKER }] *corpo*
//*parametri:*
[IN | OUT | INOUT] *nomeParametro tipo*

Come abbiamo detto, ogni procedura è identificata da un nome. Inoltre la procedura è attribuita ad uno specifico database (a partire da MySQL 5.0.1), esattamente come una tabella. Di conseguenza la procedura viene assegnata ad un database al momento della creazione, ed i nomi referenziati al suo interno si riferiranno allo stesso database, a meno che non siano qualificati con un nome di database specifico. In fase di creazione, quindi, se indichiamo il nome della procedura senza specificare il database questa sarà assegnata al db attualmente in uso.

Ogni procedura può avere uno o più parametri, ciascuno dei quali è formato da un nome, un tipo di dato e l'indicazione se trattasi di parametro di input, di output o entrambi. Se manca l'indicazione, il parametro è considerato di input.

La clausola SQL SECURITY stabilisce se, al momento dell'esecuzione, la procedura utilizzerà i permessi dell'utente che la sta eseguendo o quelli dell'utente che l'ha creata (il default è DEFINER).

Vediamo adesso un esempio concreto di stored procedure:

CREATE PROCEDURE *nomeProc* (IN *param1* INT, OUT *param2* INT)
SELECT COUNT(*) INTO *param2* FROM *tabella*
WHERE *campo1 = param1*;

Questa istruzione crea una procedura chiamata 'nomeProc' nel database in uso; la procedura usa un parametro in input e uno in output, entrambi interi, ed effettua il conteggio delle righe in tabella in cui il valore di campo1 corrisponde al primo parametro; il risultato della query viene memorizzato nel secondo parametro attraverso la clausola INTO.

Ecco come chiamare la procedura e visualizzare il risultato:

CALL *nomeProc* (5, @a);
SELECT @a;

Con l'istruzione CALL effettuiamo la chiamata della procedura (immaginando che il database attualmente in uso sia lo stesso a cui la procedura è associata), passando il valore 5 come parametro

di input e la variabile @a come parametro di output, nel quale verrà memorizzato il risultato. La SELECT successiva visualizza il valore di tale variabile dopo l'esecuzione.

Nell'esempio appena visto la stored procedure conteneva una semplice SELECT; è possibile invece creare procedure che contengono sintassi complesse comprendenti più istruzioni: in pratica, dei veri e propri script, con la possibilità di controllare il flusso attraverso vari costrutti (IF, CASE, LOOP, WHILE, REPEAT, LEAVE, ITERATE). Inoltre è possibile utilizzare i **cursori** per gestire i resultset.

Ecco un esempio di procedura di una certa complessità:

```
DELIMITER //
CREATE PROCEDURE procedura1 (param1 INT, param2 CHAR(3), OUT param3 INT)
BEGIN
        DECLARE finito INT default 0;
        DECLARE a INT;
        DECLARE b CHAR(50);
        DECLARE cur1 CURSOR FOR SELECT id,nome
                FROM clienti WHERE cat = param2;
        DECLARE CONTINUE HANDLER FOR SQLSTATE '02000'
                SET finito = 1;
        OPEN cur1;
        SET param3 = 0;
        FETCH cur1 INTO a,b;
        ciclo: WHILE NOT finito DO
                IF param3 < param1 THEN
                        SET param3 = param3 + 1;
                        FETCH cur1 INTO a,b;
                ELSE
                        LEAVE ciclo;
                END IF;
        END WHILE ciclo;
END; //
DELIMITER ;
```

Analizziamo questo codice.

Per prima cosa notiamo il comando **DELIMITER**, che serve al client mysql per modificare il normale delimitatore delle istruzioni, che sarebbe il punto e virgola. Infatti, siccome la stored procedure contiene più istruzioni, al suo interno il punto e virgola viene utilizzato più volte. Di conseguenza, se vogliamo riuscire a memorizzare la procedura, dobbiamo comunicare al server che il delimitatore è un altro; in caso contrario, al primo punto e virgola penserebbe che la nostra CREATE sia terminata. Decidiamo quindi che il nuovo delimitatore sia un doppio slash.

Dichiariamo la procedura col nome *procedura1* e tre parametri, di cui i primi due di input e l'ultimo di output. Il codice da eseguire deve essere racchiuso tra le clausole **BEGIN** ed **END**; all'interno troviamo tre istruzioni **DECLARE** che dichiarano altrettante **variabili**, quindi la dichiarazione di un **cursore**, infine la dichiarazione di un **HANDLER** per l'SQLSTATE 02000, di cui vedremo tra poco l'utilità.

La prima vera operazione è l'apertura del cursore (**OPEN**), che esegue la query associata utilizzando uno dei parametri di input. Di seguito, dopo avere inizializzato il parametro di output (che servirà da contatore delle righe lette), eseguiamo la prima **FETCH**, cioè la lettura della prima riga di risultato della query, i cui valori sono assegnati alle variabili a e b. Di seguito c'è un ciclo

WHILE (al quale abbiamo assegnato proprio il nome 'ciclo') che viene eseguito fintanto che il valore di 'finito' è falso, cioè è uguale a zero come da inizializzazione. Tale valore verrà modificato nel momento in cui non ci sono più righe da leggere, come specificato con l'handler (02000 è infatti lo stato che indica tale situazione).

All'interno del ciclo si verifica se la variabile 'param3′ ha già raggiunto il valore di 'param1′, che era l'altro parametro in input: in questo caso l'istruzione **LEAVE** consente di abbandonare il ciclo stesso; in caso contrario si incrementa la variabile e si esegue una nuova lettura del cursore, sempre riportando i risultati nelle variabili a e b (l'esempio è solo dimostrativo, in quanto non utilizziamo mai questi valori).

Il ciclo quindi termina quando sono finite le righe del cursore oppure quando ne abbiamo lette tante quante indicate dal primo parametro, a seconda di quale dei due eventi si verifica prima. Al termine dell'esecuzione il parametro di output conterrà il numero di righe lette. L'istruzione DELIMITER viene utilizzata di nuovo per ripristinare il delimitatore punto e virgola, una volta che la procedura è stata memorizzata.

Vediamo quindi quali elementi possiamo trovare in una stored procedure:

- **Variabili**: sono quelle dichiarate con l'istruzione DECLARE, più i parametri ricevuti dalla procedura
- **Condizioni**: corrispondono a codici errore di MySQL oppure valori di SQLSTATE ai quali possiamo dare un nome per poi gestirli con un HANDLER
- **Cursori**: si usano per eseguire query che restituiscono un resultset quando vogliamo scorrere tale resultset all'interno della procedura
- **Handler**: istruzioni da eseguire al verificarsi di particolari condizioni; tali condizioni possono essere, oltre a quelle definite in precedenza, direttamente un SQLSTATE, oppure un codice errore o altre condizioni: SQLWARNING, NOT FOUND, SQLEXCEPTION. La dichiarazione deve anche specificare se, al verificarsi della condizione, la procedura deve continuare o terminare (CONTINUE o EXIT)

Tutti questi elementi sono creati con l'istruzione DECLARE, e devono trovarsi all'inizio del codice (dopo BEGIN) nell'ordine in cui li abbiamo elencati.

Vediamo ora cosa possiamo trovare nel codice vero e proprio:

- **Variabili**: possono essere manipolate con l'istruzione **SET** o ricevere i valori delle query con **INTO**
- **Cursori**: si aprono con **OPEN**, si chiudono con **CLOSE** (MySQL li chiude in automatico al termine della procedura se non lo facciamo esplicitamente), e si scorrono con **FETCH**, che provvede a leggere una riga e mettere i risultati in una o più variabili
- **Controllo di flusso**: sono costrutti che ci permettono di inserire una logica nella procedura. Abbiamo **IF** per le condizioni, **LOOP**, **REPEAT** e **WHILE** per i cicli, **ITERATE** e **LEAVE** per la gestione dall'interno dei cicli stessi

Vediamo un po' di codice di esempio:

```
DECLARE a INT default 0;
DECLARE cond1 CONDITION FOR 1045;
DECLARE cond2 CONDITION FOR SQLSTATE '02000';
DECLARE cur1 CURSOR FOR query;
DECLARE EXIT HANDLER FOR cond1 SET variabile = valore;
DECLARE CONTINUE HANDLER FOR cond2 SET variabile = valore;
DECLARE CONTINUE HANDLER FOR SQLWARNING;
OPEN cur1;
FETCH cur1 INTO variabile1 variabile2;
CLOSE cur1;
IF condizione THEN istruzioni
    [ELSEIF condizione THEN istruzioni] ...
    [ELSE istruzioni]
END IF;
[nome:] LOOP
    istruzioni
END LOOP [nome];
[nome:] REPEAT
    istruzioni
    UNTIL condizione
END REPEAT [nome];
[nome] WHILE condizione DO
    istruzioni
END WHILE [nome];
ITERATE nomeCiclo;
LEAVE nomeCiclo;
```

Nella prima riga definiamo una variabile; nella seconda diamo un nome al codice errore 1045; nella terza all'SQLSTATE '02000'; nella quarta definiamo un cursore. Di seguito definiamo gli handler per le due condizioni definite in precedenza, specificando che nel caso della prima interromperemo la procedura, mentre nel secondo la continueremo; in entrambi i casi impostiamo il valore di una variabile. Definiamo poi un handler anche per la condizione di SQLWARNING. Le tre istruzioni successive effettuano apertura, lettura e chiusura del cursore.

Abbiamo poi la sintassi della IF, seguita da quelle relative ai cicli; possiamo notare che il ciclo LOOP non ha condizione di uscita (dovremo quindi uscirne con un LEAVE o attraverso un handler), mentre il ciclo REPEAT ha una condizione di uscita (cioè termina quando la condizione è vera), ma viene comunque eseguito almeno una volta; il ciclo WHILE ha una condizione di permanenza (termina quando diventa falsa), e può anche non essere eseguito mai.

Con LEAVE abbandoniamo un ciclo, mentre con ITERATE passiamo all'esecuzione successiva (saltiamo cioè, per una esecuzione, la parte di codice interna al ciclo successiva alla ITERATE). I nomi dei cicli sono facoltativi, ma diventano indispensabili qualora dobbiamo referenziarli con LEAVE o ITERATE.

Una volta creata, la stored procedure può essere eliminata con **DROP PROCEDURE**, o modificata con **ALTER PROCEDURE**. Non è tuttavia possibile modificare il codice della procedura: per fare questo dovremo eliminarla e ricrearla.

ALTER PROCEDURE *nomeProcedura* SQL SECURITY { DEFINER | INVOKER };
DROP PROCEDURE *nomeProcedura*;

I permessi necessari

Per creare una stored procedure dobbiamo avere il privilegio CREATE ROUTINE, nonchè il privilegio SUPER in caso siano abilitati i log binari di MySQL. Per modificarla o eliminarla ci serve il privilegio ALTER ROUTINE (che viene assegnato automaticamente al creatore della procedura). Per eseguirla infine è necessario il privilegio EXECUTE (anche questo assegnato in automatico all'autore) nonchè il permesso di accesso al database che la contiene.

I controlli sulle istruzioni contenute dipendono dal valore impostato per il parametro SQL SECURITY: nel caso sia DEFINER (il default), sarà l'utente che ha definito la procedura che deve avere i permessi necessari; se invece il valore del parametro è INVOKER i permessi sono richiesti a chi esegue la routine.

Le stored functions

Le stored functions sono simili alle stored procedures, ma hanno uno scopo più semplice, cioè quello di definire vere e proprie funzioni, come quelle già fornite da MySQL. Esse restituiscono un valore, e non possono quindi restituire resultset, al contrario delle stored procedures. Nelle versioni di MySQL precedenti alla 5.0 esistevano le "user-defined functions", che venivano memorizzate esternamente al server. Ora queste funzioni sono ancora supportate, ma è sicuramente consigliabile utilizzare le nuove stored functions.

Vediamo come crearle:

CREATE FUNCTION *nome* ([*parametro*[,...]])
RETURNS *tipo*
[SQL SECURITY { DEFINER | INVOKER }] *corpo*
//*parametri:*
nomeParametro tipo

Rispetto alle stored procedures, vediamo che si aggiunge la clausola RETURNS che specifica che tipo di dato la funzione restituisce. Inoltre nella lista dei parametri non esiste distinzione fra input e output, in quanto i parametri sono solo in input.

Esistono poi le istruzioni **ALTER FUNCTION** e **CREATE FUNCTION** che sono analoghe alle corrispondenti relative alle procedure. Anche le considerazioni sulla sicurezza sono le stesse relative alle stored procedures. Il codice contenuto può essere lo stesso, con la differenza che, come abbiamo già detto, non sono previsti parametri in output e non è possibile restituire un resultset. é invece **obbligatorio** restituire un dato del tipo indicato nella clausola RETURNS, attraverso l'istruzione **RETURN *valore***.

Per informazioni più approfondite su stored procedures e stored functions vi rimandiamo al manuale ufficiale di MySQL.