Una classe Php per l'accesso a Firebird

Parte 3: Implementazione metodi specifici per la gestione delle Transazioni

Con una serie di 3 articoli vediamo come sviluppare una soluzione RDBMS accedendo al database Firebird con PHP

di Raoul Scarazzini

Negli articoli precedenti abbiamo costruito parte di una classe PHP per l'accesso ad un database Firebird. Allo stato attuale delle cose, la nostra classe ci consente di effettuare query, navigare nei resultset ed operare sui campi BLOB.

In questo articolo cercheremo di completare il lavoro provando a capire cosa sono le Transazioni ed implementando un metodo che ci consenta di gestirle al meglio.

Panoramica sulle Transazioni

Nell'operare su basi dati complesse può capitare che per effettuare una determinata operazione sia necessario più di un singolo passo, ad esempio: un cliente si collega ad un servizio B2B ed inserisce il suo ordine.

Quella che appare come una banale operazione di inserimento, in realtà a livello dati potrebbe comportare ad esempio la scrittura della testata d'ordine nella tabella ordini, dei dettagli ordine in un'altra tabella contenente le varie righe, l'aggiornamento della tabella dello storico ordini ed infine l'aggiornamento del campo "data ultima operazione" nella tabella dei clienti.

Come si può facilmente capire, la singola operazione "inserisci ordine" è in realtà composta da 4 sotto-operazioni che dipendono l'una dall'altra in maniera inscindibile: se venissero inserite le righe di dettaglio dell'ordine senza che l'operazione di registrazione della testata sia andata a buon fine, il database si troverebbe in uno stato inconsistente e non permetterebbe di effettuare l'inserimento restituendo un messaggio di errore.

Per mantenere la coerenza della nostra base dati, nel momento in cui una qualsiasi delle nostre sotto-operazioni dovesse fallire, anche tutte le altre andrebbero annullate.

In questo caso ed in mille altri simili, fare a meno delle Transazioni è praticamente impossibile.

Per definizione quindi, una Transazione è un'insieme di istruzioni SQL che deve essere trattato come una unità "atomica", cioè non scomponibile. Una Transazione viene completata solo se tutte le operazioni che la compongono vengono eseguite con successo. Nel caso in cui una delle operazioni fallisce oppure la Transazione viene esplicitamente annullata, tutte le operazioni precedenti devono essere ritenute nulle, come se non fossero mai state effettuate.

Usare le Transazioni risulta inoltre particolarmente utile nell'implementazione di meccanismi di locking (blocco) per permettere un accesso concorrente ai dati di un database, in quanto le modifiche operate sul database dalle istruzioni di una Transazione non hanno effetto su questo (e quindi non sono visibili ad altre Transazioni) fino a quando la Transazione non viene completata con successo.

Ogni volta che una Transazione viene avviata può concludersi in due modi: Successo o Insuccesso.

In caso di successo tutti gli inserimenti, cancellazioni e aggiornamenti vengono registrati nel database, in caso di insuccesso, il database non sarà in alcun modo modificato e si troverà nello stato immediatamente precedente all'avvio della Transazione.

Avviare una Transazione in Firebird

In Firebird ogni operazione effettuata su un database in realtà è parte di una Transazione. Nel momento stesso in cui viene effettuata una connessione ad un database, viene avviata, in maniera automatica e trasparente ai nostri occhi, una Transazione "di default", che parte cioè automaticamente, senza bisogno di avviarla tramite istruzioni SQL.

Questo concetto viene spiegato con un esempio nel Listato 1, e va chiarito da subito che non limita in alcun modo la possibilità di definire proprie Transazioni.

Per avviare manualmente una Transazione, è necessario utilizzare questa istruzione SQL:

```
SET TRANSACTION <ACCESS MODE> <LOCK RESOLUTION>
<ISOLATION LEVEL> <TABLE RESERVATION> <DATABASE
SPECIFICATION>
```

Cerchiamo di capire in maniera sintetica il significato di ciascuno dei parametri passabili all'istruzione:

Listato 1 : Le transazioni di "default"

Proviamo a fare una veloce prova sul db di test /home/janet/dbtest.gdb (creato negli scorsi articoli) utilizzando l'utilissimo tool da linea di comando "isql" presente nella directory di installazione di Firebird (generalmente /opt/interbase/bin), effettuando un inserimento nella tabella PEOPLE seguito da un'istruzione di ROLLBACK :

```
[root@conflitto janet]# /opt/interbase/bin/isql
Use CONNECT or CREATE DATABASE to specify a database
SQL> CONNECT /home/janet/dbtest.gdb USER sysdba PASSWORD
masterkey;
Database:  /home/janet/dbtest.gdb, User: sysdba
SQL> SELECT PCODE, PNAME, PSURNAME FROM PEOPLE;

       PCODE PNAME                           PSURNAME
============ ====================== ==============================

           0 Vito                            Corleone
           1 Michael                         Corleone
           2 Sonny                           Corleone
           3 Fredo                           Corleone
           4 Tom                             Hagen
           5 Peter                           Clemenza
           6 Kay                             Adams Corleone

SQL> INSERT INTO PEOPLE (PCODE, PNAME, PSURNAME) VALUES (6,
'Luca', 'Brasi');
SQL> SELECT PCODE, PNAME, PSURNAME FROM PEOPLE;

       PCODE PNAME                           PSURNAME
============ ====================== ==============================

           0 Vito                            Corleone
           1 Michael                         Corleone
           2 Sonny                           Corleone
           3 Fredo                           Corleone
           4 Tom                             Hagen
           5 Peter                           Clemenza
           6 Kay                             Adams Corleone
           6 Luca                            Brasi

SQL> ROLLBACK;
SQL> SELECT PCODE, PNAME, PSURNAME FROM PEOPLE;

       PCODE PNAME                           PSURNAME
============ ====================== ==============================

           0 Vito                            Corleone
           1 Michael                         Corleone
           2 Sonny                           Corleone
           3 Fredo                           Corleone
           4 Tom                             Hagen
           5 Peter                           Clemenza
           6 Kay                             Adams Corleone

SQL> QUIT;
```

All'inizio in tabella, effettuando la prima select, osserviamo come siano presenti solamente 6 record. Una volta effettuato l'inserimento, e rilanciata la select, ci accorgiamo di come il nuovo record inserito esista.
Forzando il ROLLBACK della Transazione, il record inserito sparisce, riportando il database allo stato inziale, come se noi non avessimo effettuato nessun tipo di operazione.
Questo dimostra come pur non avendo dichiarato esplicitamente l'avvio della Transazione, siamo in grado di controllarne il flusso tramite le istruzioni ROLLBACK e COMMIT.
Un'altra piccola nota: nel momento in cui viene effettuata una ROLLBACK o una COMMIT sulla Transazione di default, ne viene subito avviata una nuova, a sottolineare come ogni operazione sul database viene obbligatoriamente associata ad una Transazione. È questo lo stesso motivo per cui rimanendo nell'ambiente isql possiamo continuare a lanciare ROLLBACK o COMMIT a rotazione pur non effettuando nessun tipo di operazione: queste istruzioni chiuderanno la Transazione di default e Firebird sistematicamente ne creerà una nuova.

ACCESS MODE: può essere READ ONLY o READ WRITE, questo parametro descrive il tipo di accesso effettuabile dalla Transazione sulle tabelle del database, se di sola lettura o di lettura e scrittura.

LOCK RESOLUTION: può essere WAIT o NO WAIT e descrive il tipo di comportamento che la Transazione deve assumere di fronte a record posti in stato di blocco (LOCK). Nel primo caso la Transazione attenderà sino a che il record non sarà sbloccato, nel secondo caso genererà un errore di tipo "LOCK CONFLICT".

ISOLATION LEVEL: può essere SNAPSHOT, SNAPSHOT TABLE STABILITY o READ COMMITTED. Questo parametro descrive l'iterazione della Transazione con le altre (eventuali) in corso, il tipo di accesso alle medesime tabelle. Se dichiariamo SNAPSHOT, verrà restituita ad un'altra Transazione la vista del database al momento in cui la nostra è iniziata; indicando SNAPSHOT TABLE STABILITY non verranno consentite modifiche da parte di altre Transazioni alle tabelle che la nostra sta leggendo o updatando; per finire, con READ COMMITTED verrà consentito l'update sull'ultima versione registrata (quindi conclusa con una COMMIT RETAIN, la cui spiegazione viene affrontata nel paragrafo "Terminare una Transazione") della riga in questione.

TABLE RESERVATION: consente indicando RESERVING di bloccare immediatamente l'accesso alle tabelle che verranno indicate.

DATABASE SPECIFICATION: indicando USING si può specificare una serie di database disponibili ai quali questa Transazione può accedere.

Utilizzare la Transazione di default equivale a dichiarare quanto segue:

```
SET TRANSACTION READ WRITE WAIT ISOLATION LEVEL
SNAPSHOT;
```

Quindi per necessità diverse, saranno diversi i parametri che verranno passati all'istruzione SQL.

È possibile anche nominare le Transazioni, per distinguerle l'una dall'altra. L'argomento in sè richiederebbe un

articolo a parte e non riguarda direttamente lo scopo del nostro articolo, quindi ricordo che per approfondire l'argomento un ottimo strumento gratuito rimane la documentazione di Interbase 6 scaricabile dal sito ufficiale di Firebird:

```
http://firebird.sourceforge.net/index.php?op=doc&id=userdoc
```

in particolare il capitolo 4 del libro "EMBEDDED SQL GUIDE": "WORKING WITH TRANSACTIONS".

Terminare una Transazione

Ora che è chiaro come viene avviata una Transazione, prima di proseguire cerchiamo di capire come è possibile terminarla. Come già accennato, i due stadi possibili sono successo ed insuccesso.

Le istruzioni associate a questi stadi sono COMMIT e ROLLBACK.

In caso di COMMIT tutti i cambiamenti effettuati all'interno del database diventano permanenti.

In caso di ROLLBACK, tutti i cambiamenti effettuati vengono annullati. Tipicamente, una istruzione di ROLLBACK viene utilizzata in caso di errore durante lo svolgimento delle operazioni della Transazione per riportare il database allo stato iniziale.

Concludere una Transazione con uno di questi due metodi significa liberare le risorse allocate, rendendo più efficiente e veloce il database.

C'è un ulteriore parametro passabile ad entrambe le istruzioni, questo è RETAIN.

Se viene aggiunta questa voce ad una delle due istruzioni, la Transazione non viene terminata, ma vengono invece registrate (o annullate) tutte le modifiche effettuate dall'avvio a quel momento. La Transazione rimane dunque aperta, passibile di altre istruzioni SQL comprese nuove COMMIT RETAIN o ROLLBACK RETAIN. È facile capire anche come il lanciare immediatamente dopo un comando comprensivo della voce RETAIN un'istruzione di tipo COMMIT o ROLLBACK non produca effetto alcuno se non il terminare la Transazione in corso.

Utilizzare le Transazioni in PHP

Il nostro obiettivo è quello di creare un metodo che consenta alla nostra classe di operare con le Transazioni.

Il supporto che PHP fornisce per gestire le Transazioni di Firebird si basa sul concetto di handle, ed anche se già dovrebbe essere chiaro in quanto parte fondamentale dei precedenti articoli, facciamo un veloce riepilogo: un handle è un indirizzo di memoria al quale è associata la nostra connessione. In PHP questo indirizzo di memoria viene generalmente registrato all'interno di una variabile che viene usata ogni qualvolta si vogliono effettuare query sul database.

Il metodo exec_query, presente nella nostra classe infatti necessita (essendo basato sul metodo PHP ibase_query) di due parametri: l'handle del database e la stringa sql da eseguire.

Per le transazioni, il concetto è lo stesso. Ad ogni Transazione avviata corrisponde un handle, tutte le operazioni che si vorranno effettuare all'interno di questa dovranno far riferimento al suo handle specifico.

Per comodità definiremo un solo metodo per la gestione di tutte e tre le operazioni che in base ad un parametro definito, avvierà una Transazione, la completerà con una COMMIT o la annullerrà con una ROLLBACK.

Le funzioni che PHP mette a disposizione in merito all'argomento sono tre:

ibase_trans: richiede in input gli argomenti della Transazione (vedi Listato 2) e l'handle del database selezionato, mentre restituisce in output l'handle della transazione.

ibase_commit: dato in input l'handle della Transazione, effettua il commit e restituisce un valore booleano tra TRUE o FALSE a seconda di successo o insuccesso.

ibase_rollback: dato in input l'handle della Transazione, effettua il rollback ed in output si comporta alla stessa maniera di ibase_commit.

Organizzando il nostro metodo, potremmo pensare di usare due parametri : il primo denominato $action che conterrà la descrizione dell'operazione da svolgere: "start" per l'avvio

Listato 2 : Gli argomenti di ibase_trans

La scelta del parametro trans_args passabile alla funzione ibase_trans va effettuata tra questi valori predefiniti :

```
IBASE_DEFAULT,
IBASE_READ
IBASE_WRITE
IBASE_COMMITTED
IBASE_CONSISTENCY
IBASE_CONCURRENCY
IBASE_REC_VERSION
IBASE_REC_NO_VERSION
IBASE_WAIT
IBASE_NOWAIT
```

Ciascuno di questi guida uno specifico comportamento della Transazione.
Purtroppo non esiste nemmeno sul sito ufficiale di PHP una spiegazione specifica su ciascuna singola voce, solo alcune (le più usate) sono abbastanza autoesplicative.
Indicando IBASE_DEFAULT ad esempio, si avvia una Transazione le cui proprietà sono quelle di default ossia access mode READ ONLY, lock resolution NO WAIT ed isolation level SNAPHOT.
Nell'attesa che il supporto di PHP da questo punto di vista migliori, è necessario sperimentare in ambiente di prova le opzioni che si vogliono utilizzare in modo da non incorrere in spiacevoli inconvenienti.

Listato 3: Test con ROLLBACK e COMMIT

```
<html>
<body>

<?php
 // Inclusione file ib_class
 include("ib_class.inc");

 // Creazione classe Ibase_DB
 $db = new ib_class("localhost", "/home/janet/dbtest.gdb", "", 0,
3, "SYSDBA", "masterkey");

 switch ($_POST["selection"]) {
  case "rollback" :
   $tr_id = $db->transaction("start", IBASE_DEFAULT);
   echo ibase_errmsg();
   // Test Rollback
   $ssql = "INSERT INTO PEOPLE (pcode, pname, psurname) VALUES
(?, ?, ?)";
   // Definisco i parametri
   $db->add_param(7);
   $db->add_param("Luca");
   $db->add_param("Brasi");
   // Esecuzione query
   if (!$db->exec_query($tr_id, $ssql))
    // C'e' un errore, lo visualizzo
    echo "Si è verificato un errore : " . $db->Error;
   else
    $db->transaction("rollback", $tr_id);
   break;

  case "commit" :
   $tr_id = $db->transaction("start", IBASE_DEFAULT);
   // Test Commit
   $ssql = "INSERT INTO PEOPLE (pcode, pname, psurname) VALUES
(?, ?, ?)";
   // Definisco i parametri
   $db->add_param(7);
   $db->add_param("Luca");
   $db->add_param("Brasi");
   // Esecuzione query
   if (!$db->exec_query($tr_id, $ssql))
    // C'e' un errore, lo visualizzo
    echo "Si è verificato un errore : " . $db->Error;
   else
    $db->transaction("commit", $tr_id);
   break;

  case "delete":
   // Cancello gli eventuali inserimenti
   $ssql = "DELETE FROM PEOPLE WHERE pcode = ?";
   // Definisco i parametri
   $db->add_param(7);
   // Esecuzione query
   if (!$db->exec_query(0, $ssql))
    // C'e' un errore, lo visualizzo
    echo "Si è verificato un errore : " . $db->Error;
   break;

  default : break;
 }
?>

<form action="<?=$_SERVER["PHP_SELF"];?>" method="post">
<input type="radio" name="selection" value="rollback">Inserisci
ed effettua rollback (il record non viene inserito)
<br>
<input type="radio" name="selection" value="commit">Inserisci ed
effettua commit (il record viene inserito)
<br>
<input type="radio" name="selection" value="delete">Cancella il
record inserito
<br>
<input type="submit">
</form>
<p>Stato tabella :

<?php
 $ssql = "SELECT a.PCODE, a.PNAME, a.PSURNAME FROM PEOPLE a ORDER
BY a.PCODE";

 // Esecuzione query
 if (!$db->exec_query(0, $ssql))
  // C'e' un errore, lo visualizzo
  $db->display_error(1, $ssql);
 else
 {
  echo "<table border=\"1\">";
  // Creo le colonne della tabella
  while (list($colonna)=each($db->Fields_Info))
   echo "<th>" . $db->Fields_Info[$colonna]["alias"] . "</th>";
  // Creo le righe della tabella
  while ($db->next_record())
  {
   echo "<tr>";
   for ($fld = 0; $fld < $db->Num_Fields; $fld++)
    echo "<td>" . $db->Record[$db->Fields_Info[$fld]["alias"]] .
"</td>";
   echo "</tr>";
  }
  echo "</table>";
 }
?>

</body>
</html>
```

di una transazione e "commit" o "rollback" per il termine di questa in uno dei due modi. Il secondo parametro, $tr_id, lo considereremo un "jolly" nel senso che in caso di avvio della Transazione conterrà gli argomenti di questa (si veda sempre il Listato 2), mentre per commit o rollback conterrà l'handle della Transazione da terminare.

A questo punto mettendo in pratica quanto accennato, il codice del nostro metodo, potrebbe presentarsi come segue:

```
function transaction ($action, $tr_id = NULL)
{
 switch ($action)
 {
  case "start" :
   $this->connect();
   return ibase_trans($tr_id, $this->Link_ID);
   break;
  case "commit" :
   return ibase_commit($tr_id);
   break;
  case "rollback" :
   return ibase_rollback($tr_id);
   break;
  default :
   return NULL;
   break;
 }
}
```

Tramite il costrutto switch, decido che tipo di operazione

effettuare. Il valore restituito dalla funzione dipenderà dall'operazione selezionata: in caso di "start" sarà l'handle della Transazione, mentre per gli altri due casi sarà TRUE in caso di successo o FALSE se l'operazione fallisce.

Se viene indicato in $action un valore diverso dai tre illustrati, la funzione restituisce NULL.

Unica nota è il fatto che per azione uguale a "start" prima di eseguire la funzione ibase_trans, richiamo la funzione $this->connect() vista nel primo articolo, per fare in modo che la proprietà $this->Link_ID della classe sia un database handle effettivo.

Dalla teoria alla pratica

Una volta aggiunto questo nuovo metodo alla nostra classe, possiamo procedere con la creazione di un esempio effettivo.

Struttureremo lo script in modo che presenti una checkbox che consenta di scegliere una fra tre opzioni: in un caso effettuerà un inserimento nella tabella PEOPLE e terminerà con una ROLLBACK, nell'altro con una COMMIT ed in più aggiungeremo la possibilità di cancellare il record eventualmente inserito (in modo da poter ripetere l'esempio più volte). Alla fine effettueremo una SELECT sulla tabella e ne visualizzeremo l'esito in modo da osservare i cambiamenti prodotti dalle nostre operazioni.

La checkbox porterà allo script la variabile $_POST["selection"] che tramite il costrutto switch presente nello script, selezionerà l'operazione da svolgere.

Il risultato finale sarà quanto appare nel Listato 3: Lo script presenta argomenti trattati nei precenti articoli quali il passaggio di parametri alla query e la stampa in una tabella del result set di una query e che qui non verranno ulteriormente approfonditi, le parti da esaminare a fondo sono quelle in cui viene avviata la Transazione

```
$tr_id = $db->transaction("start", IBASE_DEFAULT);
```

In questo modo assegnamo alla variabile $tr_id l'handle della transazione e proprio in funzione di questo, la query di inserimento verrà lanciata con questa variabile come parametro di handle:

```
if (!$db->exec_query($tr_id, $ssql))
```

Se l'operazione ha successo, viene effettuata (a seconda della scelta) l'operazione di rollback o commit che terrà come riferimento sempre $tr_id come handle:

```
$db->transaction("rollback", $tr_id);
```

o

```
$db->transaction("commit", $tr_id);
```

Il risultato consentirà di visualizzare il comportamento del database nei due diversi casi: solo per COMMIT il record viene inserito, la modifica non avrà effetto per ROLLBACK.

Un ulteriore ed interessante test potrebbe riguardare l'argomento passato alla Transazione in fase di avvio: se al posto di IBASE_DEFAULT indichiamo IBASE_READ e proviamo ad inserire il record, sia che effettuiamo una commit, sia che effettuiamo una rollback, otteniamo questo errore:

"attempted update during read-only transaction", ciò conferma ancora di più l'importanza che ricoprono gli argomenti passati alla Transazione in fase di avvio.

Il risultato finale dopo una COMMIT appare in figura 1.

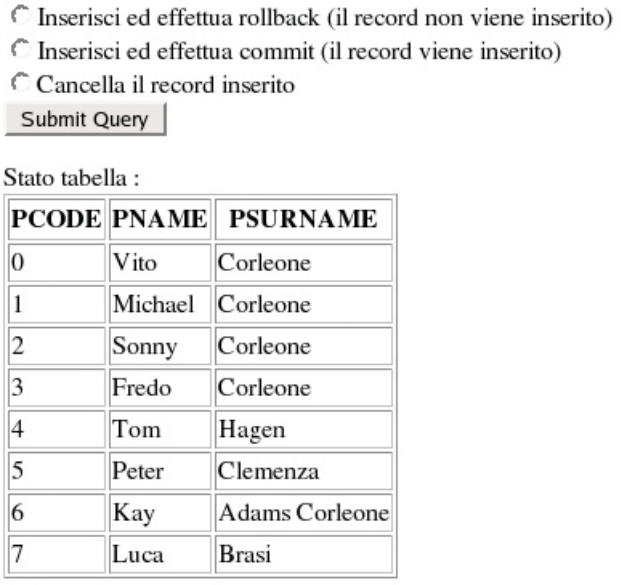

Figura 1

Conclusioni

Bene. Abbiamo finito. La nostra classe ora consente di effettuare tutte le operazioni che generalmente si fanno su un database Firebird.

Inutile sottolineare un'ultima volta come l'implementazione di tutti i metodi presentati è puramente indicativa. Le potenzialità di un progetto simile sono pressochè infinite.

Raoul Scarazzini - http://web.tiscali.it/rascasoft
<rascasoft@tiscali.it>

Nota:

Tutti i Listati citati nel testo sono riportati integralmente nella versione elettronica del documento ottenibile all'indirizzo **www.dossier.duke.it** indicando il Codice Documento **L0412RS3**