Una classe PHP per l'accesso a Firebird

Parte 1: costruzione della classe e implementazione dei metodi base

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

Raoul Scarazzini

Nell'esteso panorama degli RDBMS (Relational Database Management System) si sta pian piano ritagliando una fetta di mercato un progetto giovane, ma già molto maturo: Firebird. Nato da una costola del più famoso Interbase della Borland, FireBird rappresenta un ottimo punto di partenza per quanti vogliono implementare soluzioni database stabili ed efficienti.

Cercheremo in questa serie di articoli di imparare ad accedere a questo motore per database da PHP, costruendo, sulla base della programmazione ad oggetti, una classe di accesso personalizzata che chiameremo ib_class.

L'ideale, per essere subito operativi, è che nel sistema su cui si lavorerà ci sia installato il motore per database Firebird (la versione attuale e consigliata è la 1.0.3), un web-server (per esempio Apache, 1.3.x o 2.x è indifferente) che supporti Php (4.3.x) e che lo stesso PHP sia compilato con supporto Interbase (e quindi Firebird).

Prima di iniziare

Quella che segue è una breve panoramica su ciò che è la programmazione ad oggetti, su ciò che sono le classi, e su come vengono implementate ed utilizzate in PHP. Essendo l'obiettivo primario di questo articolo creare una classe PHP specifica, se si desidera approfondire gli argomenti in questione si può far riferimento al testo "Object Oriented Programming – Guida completa" edito da Apogeo per quanto riguarda lo studio della programmazione ad oggetti, mentre all'indirizzo http://it.php.net/oop per trovare spiegazioni ed esempi inerenti la programmazione ad oggetti in PHP.

Inoltre non dimentichiamoci mai del buon vecchio google (www.google.com) per effettuare qualsiasi tipo di ricerca tecnica.

```
OOP : Object Oriented Programming.
```

I concetti di classe ed oggetto (o istanza) sono alla base della programmazione ad oggetti (OOP, Object Oriented Programming). Una classe rappresenta l'astrazione di un oggetto mentre un oggetto è istanza di una classe: Creare un oggetto significa istanziare una classe.

Una struttura dati è definita classe quando racchiude in se delle variabili (denominate proprietà o attributi) e le funzioni per la manipolazione di queste (denominate metodi).

È importante che la struttura dati di una classe sia trasparente per chi la utilizza: si opererà su questa solo tramite i metodi in essa definiti, quindi sarà necessario conoscere come questi operano e quale è la loro funzione, non come sono stati implementati. In questo modo si garantisce che la classe diventi fruibile non solo da chi l'ha costruita, bensì da tutti coloro che conoscono come questa funziona e cosa fa.

Panoramica sulle classi in PHP

PHP è un linguaggio procedurale che però supporta il paradigma della programmazione orientata agli oggetti, proprio per questo possiede una serie di parole chiave necessarie ad implementare classi.

Una classe in PHP viene creata tramite la parola chiave class e nel corpo della sua dichiarazione si trovano tutte le variabili (proprietà) e le funzioni (metodi) che ne descrivono l'implementazione.

Le proprietà di una classe vengono dichiarate tramite la parola chiave var mentre per i metodi viene usata la parola chiave function.

Per poter accedere alle funzioni e alle variabili interne di una classe, viene usata la pseudo-variabile $this. Questo perchè, in fase di progettazione, non si conosce il nome con cui la classe verrà istanziata all'interno dei vari programmi. Questa pseudo-variabile può essere quindi letta come "questo oggetto", this appunto.

Per scorrere proprietà e metodi di un oggetto viene usato l'operatore "->" che consente, un pò come succede nello scorrimento delle directory tramite l'operatore "/" di reperire un valore associato ad un percorso: se volessi accedere al file file.txt sotto la cartella /files dovrei far riferimento a

questo così: /files/file.txt, negli oggetti la sintassi per accedere ai dati richiama questo concetto: $this->nomeproprietà per le proprietà e $this->nomemetodo() per i metodi.

È da notare come la variabile si chiami $this->nomeproprietà, e non $this->$nomeproprietà, pur essendo questa dichiarata come var $nomeproprietà: ciò accade perchè le variabili in PHP vanno scritte con un unico simbolo dollaro.

Come ultima cosa è necessario far notare che, al di fuori della classe stessa, non è mai buona norma assegnare dei valori alle prioprietà di un oggetto, è molto meglio creare dei metodi che effettuino questi assegnamenti.

In PHP 5 è stato introdotto un sistema per evitare che le proprietà di un oggetto vengano modificate all'esterno di questo: dichiarando una variabile con la parola chiave private anzichè var.

Visto che implementeremo la nostra classe con la versione 4 di PHP, per il momento consideriamo quanto detto solo a livello teorico, cercando di non assegnare direttamente i valori alle proprietà.

Il costruttore, questo sconosciuto

Se tra i metodi dichiarati all'interno di una classe ne esiste uno con il nome uguale a questa, esso viene chiamato costruttore. Il costruttore viene invocato ogni volta che un'istanza della classe viene creata ed ha il compito di inizializzare le proprietà del nucoo oggetto. In PHP ciò avviene tramite la parola chiave new:

```
$nomeoggetto = new nomeclasse();
```

Se non esiste un costruttore, new istanzierà solamente la classe e sarà necessario che ogni parametro abbia dei valori di default.

Una volta creato l'oggetto, le proprietà di questo saranno accessibili in questo modo $nomeoggetto->nomeproprietà ed i metodi con $nomeoggetto->nomemetodo()

Non è obbligatorio avere un costruttore all'interno di una classe, ma questo metodo è talmente utile da non poterne fare a meno.

Per creare una classe che si rispetti quindi, si fa presto a capire come il primo metodo da implementare sia il costruttore.

Prepariamoci al lavoro

Ora che abbiamo introdotto tutti gli argomenti che tratteremo nel corso dell'articolo, siamo pronti per iniziare a lavorare.

Si può partire analizzando quali dovranno essere le funzionalità della nostra classe in base alle operazioni che dovrà svolgere: essendo una classe per l'accesso ad un database, sicuramente dovrà consentirci di stabilire una connessione, di effettuare query sulle tabelle e di navigare all'interno dei risultati delle nostre interrogazioni. E questo solo per cominciare.

È facile intuire come ad ogni funzionalità indicata, dovrà corrispondere un metodo.

Ciascuno di questi metodi dovrà basarsi sulle funzioni che PHP mette a disposizione per l'accesso ad Interbase/Firebird. Un elenco completo di queste funzioni lo si può trovare sul sito ufficiale di PHP, all'indirizzo http://it.php.net/manual/it/ref.ibase.php.

Quello che dovremo fare, per iniziare, sarà realizzare quattro metodi base:

- Un metodo che consenta a chi opera con la classe di istanziarla: dovremo creare cioè il costruttore, partendo dal presupposto che questo metodo dovrà inizializzare le proprietà dell'oggetto ed avere lo stesso nome della classe;
- Un metodo che permetta di connettersi ad un database Firebird basato sulla funzione ibase_connect;
- Un metodo che consenta di effettuare delle query. La funzione che ci verrà in aiuto a questo punto sarà ibase_query;
- Un ultimo metodo che permetta di operare sui resultset delle query, di scorrere cioè all'interno dei risultati in modo da poterli presentare nei nostri programmi.

È chiaro che questi sono i metodi base, nella stesura di questi necessiteremo di crearne degli altri "minori", sino ad ottenere una classe pronta per essere usata negli accessi a database Interbase/Firebird.

Per concludere, cercheremo di sviluppare un esempio che acceda ad un database di test (la cui creazione è illustrata nel riquadro1) e che sfrutti tutti i metodi sinora illustrati.

Bene, pronti? Via!

Creiamo il file

Partendo dal presupposto che la classe si chiamerà ib_class creiamo un file denominato ib_class.inc che contenga l'intestazione della nostra classe insieme alle prime proprietà essenziali: il nome host, che identifica appunto l'Host presso il quale "alloggia" il nostro db, il percorso (path) del database insieme allo username ed alla password di accesso, il Set di caratteri, il numero di buffers ed il dialetto sql usato dal server:

```
<?php

class ib_class {

var $Host          = "";      // Host
var $Database      = "";      // Database
var $User          = "";      // Utente
var $Password      = "";      // Password
var $Charset       = "";      // Set di caratteri
var $Buffers       = 0;       // Numero di buffers
var $Dialect       = 3;       // Dialetto SQL
```

```
};

?>
```

Quanto sopra può essere considerato come la vera base della nostra classe. Queste proprietà non a caso rappresentano i dati da passare alla funzione ibase_connect per effettuare la connessione al database che vedremo più avanti.

Ora si può procedere con la creazione di tutti gli altri metodi (e proprietà) specifici, il risultato finale sarà quanto appare nel file ib_class.inc contenuto nel CD, ma chiaramente procedere passo per passo aiuterà a capire meglio i concetti.

Istanziare la classe tramite il costruttore

Il nostro costruttore non dovrà fare altro che assegnare i valori che gli vengono passati dal chiamante (tramite la funzione new) alle proprietà del nostro oggetto.

Il metodo dovrà dichiarare i parametri che gli devono essere passati in input (che rispecchiano le proprietà sino ad ora dichiarate della nostra classe) e tramite l'ausilio della pseudo-variabile $this effettuare dei semplici assegnamenti.

Ecco come potrebbe apparire:

```
function ib_class ($Host, $Database, $Charset,
$Buffers, $Dialect, $User, $Password)
{
 $this->Host = $Host;
 $this->Database = $Database;
 $this->Charset = $Charset;
 $this->Buffers = $Buffers;
 $this->Dialect = $Dialect;
 $this->User = $User;
 $this->Password = $Password;
}
```

Un esempio di come si potrà creare un istanza del nostro oggetto in uno script è quello che segue:

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

In questo modo, viene istanziata la classe ib_class nella variabile $db. Essa verrà inizializzata con i valori relativi al database di test che abbiamo creato seguendo quanto spiegato nel riquadro 1. L'istanza della classe (il nostro oggetto) sarà da qui in avanti rappresentata dalla variabile $db.

Piccola considerazione: è facile capire come, dichiarando $db come variabile di sessione in questo modo:

```
$_SESSION["db"] = $db;
```

non avremo bisogno ogni qualvolta ci servirà il nostro oggetto, di reistanziare la classe script per script. Potremo garantire così, tramite la propagazione delle sessioni, di avere in ogni script il nostro oggetto istanziato correttamente. Interessante no?

Connettersi ad un database Firebird

Ogni volta che si stabilisce una connessione con Firebird (ed in genere con tutti gli RDBMS) a questa viene assegnato un'indirizzo di memoria, il suo handle. Tutte le iterazioni con il database (query, transazioni etc.) andranno effettuate utilizzando un handle specifico.

Quello che a noi interessa ottenere è il valore relativo al link di una connessione e registrarlo nel nostro oggetto in una nuova proprietà, denominata $this->Link_ID.

La funzione PHP che consente di stabilire una connessione con un database Firebird è ibase_connect: Questa funzione necessita in input dei dati relativi al database (altro non sono che le proprietà che abbiamo già dichiarato del nostro oggetto) ed in caso di successo restituisce l'handle della connessione.

Il metodo che andremo a dichiarare semplicemente eseguirà la funzione ibase_connect con i parametri del nostro oggetto e assegnerà l'handle restituito da questa funzione alla proprietà Link_ID.

Il valore restituito dal metodo sarà vero o falso, a seconda dell'esito della funzione ibase_errmsg che a sua volta restitusce (se esiste) il testo del messaggio di errore.

A livello di codice, il nostro metodo sarà così strutturato:

```
function connect()
{
  $this->Link_ID = @ibase_connect(
  $this->Host . ":" .
  $this->Database,
  $this->User,
  $this->Password,
  $this->Charset,
  $this->Buffers,
  $this->Dialect,
  $this->Role);
 // Se c'è errore allora metto il messaggio in
➥Error e ritorno FALSE

 $this->Error = ibase_errmsg();
 return ibase_errmsg() ? FALSE : TRUE;

}
```

ibase_connect automaticamente controlla se esiste già un link oppure ne va creato uno nuovo e lo restituisce in output. In questi termini il nostro metodo potrà essere invocato più volte di fila senza che necessariamente stabilisca per ogni

esecuzione una nuova connessione. Come si può intuire, questa funzionalità favorisce il basso traffico di rete (e quindi la maggiore velocità).

Da notare il carattere "@" che precede la chiamata alla funzione ibase_connect. Questo indica al motore PHP di non visualizzare errori o warnings in uscita dalla funzione. Ciò è molto utile se si personalizza la gestione degli errori, come abbiamo fatto noi, assegnando alla nuova proprietà $this->Error del nostro oggetto il valore di uscita della funzione ibase_errmsg. In questo modo potremo decidere quando e come far apparire gli eventuali messaggi di errore, senza generare immotivati timori negli utilizzatori dei nostri programmi ;-)

Effettuare delle query

Il metodo per l'esecuzione delle query diverrà essenziale per qualsiasi operazione ci troveremo a fare. Proprio per questo è necessario porre particolare cura nella sua impementazione.

La funzione PHP da tenere come riferimento è ibase_query. Per essere eseguita con successo, questa necessita dell'handle del database e di una valida stringa sql. Se ha successo e vi sono righe nel risultato (come si ha ad esempio con le query SELECT), restituisce un identificatore di risorsa, mentre se ha successo ma non ci sono righe risultato, restituisce semplicemente TRUE. In caso di fallimento il valore restituito sarà FALSE.

Il concetto di identificatore di risorsa, non si scosta molto da quello di handle: esso rappresenta infatti la parte di memoria riservata al resultset (l'insieme dei valori risultanti da una query di tipo SELECT) generato da una query.

Il nostro metodo deve eseguire queste operazioni:

- Controllo esistenza di una connessione
- Esecuzione della query
- Controllo esistenza di un resultset e settaggio della nuova proprietà $this->Query_ID con il valore dell'identificatore di risorsa.
- Restituzione del valore "vero" o "falso" a seconda dell'esito della query.

I parametri che il nostro metodo riceverà in input saranno due: il database handle, $dbh e la stringa sql da eseguire, $query_string. È da notare come nella dichiarazione della variabile $dbh venga ad essa assegnato un valore di default, ciò risulterà molto utile nella verifica della validità che viene effettuata all'inizio della funzione.

Se $dbh è valorizzato (non deve essere nullo e deve essere una risorsa), allora verrà usato questo handle per svolgere le operazioni, in caso contrario verrà richiamata la funzione connect. Passare un handle diverso da quello di default, tornerà utile quando in futuro parleremo di transazioni.

Esaminiamo insieme il codice di questo metodo:

```
function exec_query($dbh = NULL, $query_string)
{
 if (!is_null($dbh) && is_resource($dbh))
  $this->Link_ID = $dbh;
 else
  $this->connect();

 array_unshift($this->Query_Params, $this->Link_ID,
➥$query_string);
 $this->Query_ID =
➥@call_user_func_array("ibase_query", $this-
>Query_Params);

 if ($this->Query_ID)
 {
  $this->Row = -1;
  // La query è andata a buon fine, setto la
➥variabile Query_Sql della classe ...
  $this->Query_Sql = $query_string;
  // ...Azzero i parametri...
  $this->Query_Params = array();
  // ...E se non è una operazione di modifica creo
➥i metadata
  if ((substr(strtoupper($query_string), 0, 6) !=
➥"INSERT")&&
      (substr(strtoupper($query_string), 0, 6) !=
➥"UPDATE")&&
      (substr(strtoupper($query_string), 0, 6) !=
➥"CREATE")&&
      (substr(strtoupper($query_string), 0, 6) !=
➥"DELETE"))
   $this->metadata();
 }

 // Se c'è errore allora metto il messaggio in
➥Error e ritorno FALSE
 $this->Error = ibase_errmsg();
 return ibase_errmsg() ? FALSE : TRUE;
}
```

Dopo aver effettuato il controllo sul database handle già descritto, particolare attenzione va posta verso le due righe

```
 array_unshift($this->Query_Params, $this->Link_ID,
➥$query_string);
 $this->Query_ID =
➥@call_user_func_array("ibase_query",
➥$this->Query_Params);
```

Queste fanno riferimento alla nuova proprietà $this->Query_Params del nostro oggetto, un array che contiene i parametri che la funzione ibase_query richiede in input.

array_unshift inserisce uno o più elementi all'inizio di un array, mentre call_user_func_array passa come parametri ad una funzione indicata i valori presenti in un array. Questo significa che se $this->Query_Params è vuota, ad ibase_query() saranno passati solamente due parametri (handle e stringa sql), mentre nel caso contrario tutti i valori presenti in $this->Query_Params verranno passati a ibase_query nell'ordine in cui sono presenti nell'array.

Per spiegare il funzionamento di queste due righe, consideriamo questa query che vuole estrarre tutti i record della tabella PEOPLE che abbiano PSURNAME uguale a "Corleone":

```
$ssql = "SELECT a.* FROM PEOPLE WHERE a.PSURNAME = ?";
ibase_query($dbh, $ssql, "Corleone");
```

anzichè scrivere direttamente

```
$ssql = "SELECT a.* FROM PEOPLE WHERE a.PSURNAME =
➥'Corleone'";
ibase_query($dbh, $ssql);
```

abbiamo indicato il carattere "?" al posto della stringa effettiva fra apici e nel richiamare la procedura ibase_query non abbiamo passato come parametri solamente l'hanlde e la stringa sql, ma anche "Corleone".

Applicando questi concetti al nostro oggetto, consideriamo un nuovo metodo denominato add_param(); il cui scopo è quello di accodare i parametri all'interno dell'array $this->Query_Params:

```
function add_param($param)
{
 $this->Query_Params[] = $param;
}
```

e realizziamo l'esempio sopracitato:

```
$ssql = "SELECT a.* FROM PEOPLE WHERE a.PSURNAME <> ?";
$db->add_param("Corleone");
$db->exec_query(0, $ssql);
```

Una volta eseguita la query con la funzione ibase_query, se $this->Query_ID viene valorizzato, e quindi la query ha avuto esito positivo, vengono settate le nuove proprietà $this->Row (che rappresenta il numero di riga corrente) e $this->Query_Sql (contenente il codice sql eseguito) ed infine viene azzerato l'array dei parametri.

Per concludere, se l'istruzione sql è una SELECT, viene richiamata la funzione $this->metadata():

```
function metadata()
{
 $c=0;
 while ($campo = ibase_field_info
➥($this->Query_ID,$c))
 {
  $info[$c]["name"] = $campo["name"];
  $info[$c]["alias"] = $campo["alias"];
  $info[$c]["relation"] = $campo["relation"];
  $info[$c]["length"] = $campo["length"];
  $info[$c]["type"] = $campo["type"];
  $info[$c]["index"] = $c;
  $c++;
 }
 $this->Fields_Info=$info;
 $this->Num_Fields=ibase_num_fields
➥($this->Query_ID);
 // L'array viene anche restituito
 return $info;
}
```

Il cui scopo è quello di creare l'array associativo $this->Fields_Info con le proprietà dei campi della tabella risulato attraverso la funzione ibase_field_info(), la quale, richiamata per ogni campo, restituisce un array con informazioni relative a questo nella forma name, alias, relation, length e type.

$this->metadata() assegna anche alla nuova proprietà Num_Fields dell'oggetto il risultato della funzione ibase_num_fields() che restituisce il numero di campi della tabella risultato.

Infine exec_query controlla che non vi siano errori nello stesso identico sistema di connect, restituendo TRUE o FALSE a seconda del valore restituito da ibase_errmsg().

Operare sui resultset delle query

Prima di addentrarci nel primo esempio effettivo di utilizzo della nostra classe è necessario capire come fare a muoversi all'interno di un resultset restituito da una query.

La funzione PHP che ci viene in aiuto in questo caso è ibase_fetch_row. Questa funzione necessita in input dell'identificatore di risorsa relativo ad una query eseguita e restituisce un array con struttura indicecampo->valore. Il primo passo da compiere quindi è creare una proprietà di tipo array nel nostro oggetto denominata Record che servirà a contenere l'associazione restituita da ibase_fetch_row.

Ma noi vogliamo di più.

Oltre a questa associazione ne vogliamo creare un'ulteriore di tipo nomecampo->valore, in questo modo non saremo costretti a ricordarci quale sia l'indice di un campo per conoscerne il valore, basterà sapere il nome di questo ed accederci tramite Record["nomecampo"].

Ad esempio se il primo campo di un resultset sarà denominato CHIAVE avremo due modi per accedere al suo valore : tramite l'indice campo $db->Record[0] oppure tramite il

nome campo $db->Record[“CHIAVE”].

Analizziamo nel dettaglio il nostro metodo:

```
function next_record()
{
 if ($this->Record = ibase_fetch_row
➥($this->Query_ID))
 {
  $this->Row += 1;
  $stat = 1;
 }
 else
 {
  if ($this->Auto_Free)
   $this->free_result();
  $stat = 0;
 }
 //Assegnazione valori Array associativo
➥Record[“nomecampo”] = valorecampo
 $flds = $this->Num_Fields;
 for ($fld = 0; $fld < $flds; $fld++)
  $this->Record[$this->Fields_Info[$fld][“alias”]]
➥= $this->Record[$fld];
 //ritorno stat: 1=0K, 0=nessun record & effettuato
➥il free se c’è auto_free
 return $stat;
}
```

La proprietà dell’oggetto Row (che rappresenta il numero di riga corrente) viene incrementata fino a che l’assegnazione $this->Record = ibase_fetch_row($this->Query_ID) produce esito positivo.

L’associazione Record[“nomecampo”]->valore, viene creata partendo dalla proprietà $this->Fields_Info[$fld][“alias”] che è stata valorizzata dal metodo $this->metadata() in exec_query();

Ciclando sul numero di campi del resultset ($this->Num_Fields), aggiunge un elemento all’array record che ha per chiave il nome del campo e per valore quello relativo all’indice di questo.

next_record restituirà 1 fino a che non si troverà sull’ultimo record, momento in cui la variabile $stat (e quindi il valore restituito dal metodo) sarà 0. A questo punto verrà effettuato un check anche sulla nuova proprietà $this->Auto_Free, se questa è valorizzata, allora verrà lanciato il metodo $this->free_result():

```
function free_result()
{
 ibase_free_result($this->Query_ID);
 $this->Query_ID = 0;
}
```

Questo metodo funziona sulla base della funzione ibase_free_result, che libera la memoria allocata da un resultset.

Mettiamo in pratica

L’esempio contenuto nel riquadro 2 contiene tutti gli argomenti che abbiamo trattato sin’ora: Una volta incluso il file “ib_class.inc” contenente il codice della nostra classe si procede con la creazione di una sua istanza nella variabile $db.

La fase successiva prevede la dichiarazione della stringa contenente il codice SQL che vogliamo far eseguire. In questo caso si tratta di voler estrarre dalla tabella PEOPLE tutte le persone che hanno un cognome (PSURNAME) contenente la parola “Corleone” indipendentemente dalle maiuscole o minuscole.

A questo punto viene aggiunto il parametro stringa di confronto e viene eseguita la query.

Da notare come l’if sia impostato direttamente sull’esito della query. Se la query non va a buon fine, viene visualizzato il testo dell’errore mentre in caso di successo viene creata una tabella con i risultati visualizzati.

Prima verranno stampate con un ciclo while le colonne <th> della tabella utilizzando l’array $db->Fields_Info che è stato creato da $this->metadata in $db->exec_query, infine lo script ciclerà sulla funzione $db->next_record fino a che questa restituirà un valore effettivo (cioè fino a che non arriverà alla fine dell’elenco) visualizzando il valore di ciascun campo in una cella.

L’output effettivo di questo script non dovrebbe essere troppo diverso da quello presentato nella figura 1.

Conclusioni

Chiaramente l’implementazione di tutti i metodi dell’oggetto è puramente indicativa. Ciascuno di essi infatti può essere modificato ed ottimizzato in funzione delle proprie specifiche esigenze. Una personalizzazione mirata è alla base dello sviluppo di un buon progetto.

In questo primo articolo abbiamo realizzato la base per la nostra classe di accesso, nei successivi implementeremo nuovi metodi per renderla più completa possibile: introdurremo metodi specifici per la gestione dei campi Blob, delle transazioni e per tutte le varie tipologie di campi.

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

Approfondimento

Tutti i sorgenti citati nel testo sono riportati nella versione elettronica ottenibile all’indirizzo **www.dossier.duke.it** indicando il Codice Documento **L0410RS1**

Riquadro 1: Creazione ambiente di test

Prima di iniziare a fare i test con gli script PHP e la nostra classe è necessario creare un database di prova, sul quale poter combinare tutti i possibili danni di cui siamo capaci senza nuocere a nessuno...
Assumiamo come directory di lavoro /home/janet e creiamo un file denominato dbtest.sql con il nostro editor di testo preferito:

```
#vi /home/janet/dbtest_create.sql
```

scrivendoci all'interno i dati per la creazione del nostro db:

```
SET SQL DIALECT 3;

CREATE DATABASE 'localhost:/home/janet/dbtest.gdb'  USER 'SYSDBA' PASSWORD 'masterkey';

CREATE TABLE PEOPLE (
    pcode    INTEGER NOT NULL,
    psurname VARCHAR(30) NOT NULL,
    pname    VARCHAR(30) NOT NULL,
    pquote   BLOB SUB_TYPE TEXT SEGMENT SIZE 240,
    PRIMARY KEY (pcode)
);

INSERT INTO PEOPLE (pcode, psurname, pname, pquote)
 VALUES (0, 'Corleone', 'Vito', 'I''ll make him an offer he can''t refuse.');

INSERT INTO PEOPLE (pcode, psurname, pname, pquote)
 VALUES (1, 'Corleone', 'Michael', 'Don''t ask me about my business, Kay.');

INSERT INTO PEOPLE (pcode, psurname, pname, pquote)
 VALUES (2, 'Corleone', 'Sonny', 'Goddamn FBI don''t respect nothin''.');

INSERT INTO PEOPLE (pcode, psurname, pname, pquote)
 VALUES (3, 'Corleone', 'Fredo', 'I''m smart and I want respect !');

INSERT INTO PEOPLE (pcode, psurname, pname, pquote)
 VALUES (4, 'Hagen', 'Tom', 'Mr. Corleone never asks a second favor once he''s refused the
first, understood ?');

INSERT INTO PEOPLE (pcode, psurname, pname, pquote)
 VALUES (5, 'Clemenza', 'Peter', 'Mikey, why don''t you tell that nice girl you love her ?');

COMMIT;
```

Una volta scritto quanto sopra, salviamo il file e lanciamo questo comando come utente root:

```
#/opt/interbase/bin/isql -i /home/janet/dbtest_create.sql
```

Se non riceviamo messaggi di errore, dovremmo trovarci nella directory /home/janet il nostro file dbtest.gdb pronto da usare.

Riquadro 2 : il primo script con la nostra classe

```
<html>
<body>
<?php
 // Inclusione file ib_class
 include(“ib_class.inc”);

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

 // Stringa sql
 $ssql = “SELECT a.PSURNAME, a.PNAME FROM PEOPLE a WHERE upper(a.PSURNAME) LIKE upper(?)”;

 // Aggiungo il parametro
 $db->add_param(“%Corleone%”);

 // Esecuzione query
 if (!$db->exec_query(0, $ssql))
  // C’e’ un errore, lo visualizzo
  echo “Si è verificato un errore : “ . $db->Error;
 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>” . $db->Record[$fld] . “</td>”;
   echo “</tr>”;
  }
  echo “</table>”;
 }
?>
</body>
</html>
```

Installazione

Quelle che seguono sono le istruzioni per l'installazione del sistema base su una distribuzione Linux con gestione dei pacchetti basata su RPM, quindi RedHat, Fedora, Mandrake, SuSe etc.
Per Debian è tutto molto più semplice in quanto tramite il comando apt-get si possono installare in una sola volta i pacchetti relativi a FireBird, Apache e PHP (già configurato con il modulo Interbase/Firebird, senza quindi la necessità di configurazione).

Installazione di FireBird SuperServer

Scarichiamo da **http://firebird.sourceforge.net** il pacchetto rpm della versione Super Server (che si differenzia dalla versione "Classic" in quanto non genera un processo diverso per ogni richiesta, ma gestisce le varie richieste con un particolare sistema denominato "threading").
Al momento in cui scrivo, la versione di Firebird stabile è la 1.0.3, il file è **FirebirdSS-1.0.3.972-0.i386.rpm** Installiamo con il comando:

```
#rpm -ivh FirebirdSS-1.0.3.972-0.i386.rpm
```

il nostro rpm, al termine dell'installazione il servizio dovrebbe automaticamente venire avviato.
Per controllare che ciò sia avvenuto, lanciamo il comando

```
#netstat -natp
```

e verfichiamo che tra le righe di output ce ne sia una simile a questa:

```
Proto Recv-Q Send-Q Local Address Foreign Address State PID/Program name
tcp    0     0 0.0.0.0:3050    0.0.0.0:*    LISTEN    615/ibserver
```

la quale ci conferma che il nostro programma e' in ascolto sulla porta 3050.
Se non dovessimo trovare la riga sopraelencata, avviamo il servizio manualmente con il comando

```
#/etc/init.d/firebird start
```

Per terminare, facciamo in modo che il servizio venga avviato durante il caricamento del sistema:

```
#chkconfig –level 345 firebird on
```

Installazione di Apache e PHP con supporto Interbase (FireBird)

Scarichiamo da **http://httpd.apache.org/dist/httpd/** i file sorgenti di Apache in /usr/src/ (o in una qualsiasi altra directory, questa sarà assunta per convenzione). Al momento in cui scrivo la versione corrente è la 1.3.31 quindi il file da scaricare sarà **apache_1.3.31.tar.gz**
Per quanto riguarda PHP invece, il link di riferimento è **http://www.php.net/downloads.php** e la versione corrente è la 4.3.7 quindi il file da scaricare è **php-4.3.7.tar.bz2**
Iniziamo a scompattare i sorgenti, spostandoci in /usr/src/ così:

```
#cd /usr/src/
```

e lanciando i comandi :

```
#tar -xzvf apache_1.3.27.tar.gz
#tar -xjvf php-4.3.7.tar.bz2
```

a questo punto in /usr/src abbiamo due directory denominate rispettivamente apache-1.3.31 e php-4.3.7.

Proseguiamo spostiandoci in /usr/src/apache_1.3.31

```
#cd /usr/src/apache_1.3.31
```

per lanciare il comando:

```
#./configure
```

A questo punto Apache è configurato con le impostazioni base. Questo step è necessario per poter iniziare a lavorare sull'installazione di PHP. Dalla directory /usr/src/php-4.3.7

```
#cd /usr/src/php-4.3.7
```

lanciamo il comando

```
#./configure –with-interbase=/opt/interbase
➥-with-apache=../apache_1.3.31
```

Che indica al compilatore di iniziare ad elaborare i sorgenti configurando PHP con il supporto ad InterBase (e quindi FireBird) e apache.
Per compilare ed installare digitiamo

```
#make && make install
```

Ora PHP è configurato, non rimane che ritornare nella directory /usr/src/apache_1.3.31 ed effettuare l'installazione di Apache, indicando di configurare il nostro programma affinchè abiliti il modulo "libphp4.a" creato dall'installazione di PHP:

```
#cd /usr/src/apache_1.3.31
#./configure --activate-module=src/modules/php4/libphp4.a
#make && make install
```

A questo punto i nostri sorgenti sono stati compilati ed installati. Per teminare tutte le operazioni, torniamo in /usr/src/php-4.3.7 e copiamo il file di configurazione di PHP php.ini-dist in /usr/local/lib

```
#cp /usr/src/php-4.3.7/php.ini-dist /usr/local/lib/php.ini
```

Questo file contiene un lungo elenco di opzioni impostabili per personalizzare il funzionamento di PHP.
Esistono anche delle opzioni personalizzate per firebird/interbase. Ad esempio per fare in modo che le date resituite dalle nostre selezioni siano in formato Italiano, basta aggiungere al file le seguenti righe:

```
[Interbase]
;Italian date format
ibase.dateformat = "%d/%m/%Y";
ibase.timestampformat = "%d/%m/%Y %H:%M:%S";
ibase.timeformat = "%H:%M:%S";
```

Anche Apache ha unfile di configurazione, denominato httpd.conf che l'installazione ha messo nella directory /usr/local/apache/conf

```
#cd /usr/local/apache/conf
```

Questo file ci permetterà di personalizzare il funzionamento del nostro web server, in particolare, per fare in modo

che tutte le estensioni .php e .phtml vengano passate dal web server al parser PHP è necessario aggiungere questa riga nell'area "Document TYPE":

```
AddType application/x-httpd-php .php .phtml
```

Per completare, copiamo in /usr/sbin gli eseguibili di apache che troviamo nella directory /usr/local/apache/bin :

```
#cp /usr/local/apache/bin/ht* /usr/sbin/
```

E Ora che il nostro web server è installato bisogna configurarlo affinche il suo daemon venga avviato durante il caricamento del sistema.
Scaricando da questo indirizzo
http://web.tiscali.it/rascasoft/janet/httpd lo script httpd e copiandolo nella cartella /etc/init.d.
L'ultima cosa da fare è impostare apache affinchè sia nell'elenco dei servizi da caricare all'avvio (deve essere ad on per i runlevel 3,4,5):

```
#chkconfig –level 345 httpd on
```

Proviamo a lanciare:

```
#/etc/init.d/httpd start
```

e verifichiamo che apache si avvii correttamente. Per fare cio' lanciamo il comando

```
#netstat -natp
```

e verfichiamo che tra le righe di output ce ne sia una simile a questa:

```
Proto Recv-Q Send-Q Local Address Foreign Address State PID/Program name
tcp    0    0    0.0.0.0:80    0.0.0.0:*    LISTEN    6372/httpd
```

la quale ci conferma che il nostro programma è in ascolto sulla porta 80.
A questo punto accedendo tramite browser all'indirizzo "localhost", ci troveremo di fronte alla pagina TEST di Apache. Questo significa che il webserver è configurato correttamente e sta funzionando.
Come già detto, tramite il file /usr/local/apache/conf/httpd.conf in poco tempo si può capire come personalizzare a dovere l'esecuzione del nostro demone.
È bene ricordare che per rendere effettiva ogni modifica fatta alla configurazione il demone va sempre riavviato con il comando

```
/etc/init.d/httpd restart
```