Interazione[1] linguaggio PHP con DB *MySQL*

Un problema sempre più frequente dei webmaster è quello di dover gestire intere sezioni dei loro siti in modo semplice e veloce. La gestione è quanto più efficiente, tanto più è possibile effettuare **modifiche frequenti** di contenuto, ma, a volte, anche di veste grafica. Un sito che aspiri ad aver un certo successo deve anche offrire una **consistente quantità di informazioni**, ma è impensabile dover modificare centinaia di pagine ogni volta si apporti anche il minimo aggiornamento.

In tutto questo, per fortuna, ci sono venuti in aiuto i linguaggi **orientati al web publishing** come PHP, ASP o Perl. Purtroppo tutto questo si è rivelato subito insufficiente: i linguaggi sono ottimi per la creazione di ***pagine dinamiche***, ma non offrono nessuna possibilità di memorizzazione dei dati. La soluzione attuale è quindi quella di utilizzare ***parallelamente*** a tali ***linguaggi*** un ***database***.

Ad esempio si fa interagire **PHP** con il database relazionale **MySQL**. Questa accoppiata è, al giorno d'oggi, una delle più diffuse in rete in quanto abbiamo a disposizione gratuitamente un linguaggio solido, capace di sopportare grandi carichi di lavoro, e un database (in realtà un DBMS) dalle notevoli qualità tecniche.

È importante prima di tutto chiarire un concetto: si ha a disposizione un database che non viene memorizzato in un file specifico ma in un ***insieme di file***. Anche se possedete un dominio e vi siete rivolti a un servizio di hosting a pagamento, questo non vi permetterà di accedere ai file: potrete ***modificarli indirettamente*** (quindi tramite query), ma non potrete copiarli o salvarli. Questo crea una limitazione nel senso che quando volete distribuire uno script, non potete fornire con esso anche il database. Dovete quindi fare in modo che l'utente crei le tabelle necessarie all'interno del suo database: potremmo quindi fornire all'utente accesso ad interfacce (come **phpMyAdmin**[2]) per crearsi tabelle con determinate caratteristiche ma questo metodo richiede una serie di conoscenze da parte dell'utente che probabilmente non ha. Quindi gli forniremo uno script che creerà per lui tutte le tabelle necessarie.

Connessione al database

Innanzi tutto vedremo le operazioni necessarie per la **connessione al database**.
Prima di poter comunicare con un DB abbiamo infatti bisogno di creare un "collegamento" fra lo script e MySQL.

Avremo bisogno di alcune informazioni relative all'accesso al database: l'**host** da cui si può raggiungere MySQL (generalmente è *localhost*); **username** e **password** per l'accesso al database; il **nome** del database. Questi quattro parametri vengono forniti dall'amministratore del nostro spazio web e prevediamo di memorizzare in altrettante variabili tali dati relativi alla configurazione dello script che salveremo con estensione **.php** per motivi di sicurezza.

Se qualcuno cercasse, infatti, di visualizzare questa pagina con il browser, vedrebbe solo una pagina vuota.

Il webserver, infatti, grazie a quest'estensione, prima di passare la pagina al browser, la farà elaborare dal modulo Php. Visto che non è previsto nessun output, sul browser verrà visualizzata solo una pagina bianca.

---

1 *PHP e MySQL, la guida*: http://www.html.it/guide/guida-php-e-mysql-pratica/

2 **PhpMyAdmin** non è altro che un'interfaccia grafica che permette di amministrare **MySql**, un tipo di database che immagazzina qualsiasi tipo di dati in strutture chiamate tabelle; con PhpMyAdmin, in pratica, si può visualizzare il contenuto di un database; creare, modificare, cancellare intere tabelle o singoli record; fare un backup dei dati contenuti; visualizzare informazioni interessanti sul db.

```
// parametri di connessione
$db_host = "localhost";                    // il server che si trova sulla macchina locale
$db_user = "";
$db_password = "";

// parametri del database
$db_name = "";
```

Nelle prime versioni, fin da PHP 2.0, che non prevedevano l'uso di oggetti, l'***estensione nativa per MySQL*** rendeva disponibili ***funzioni*** (da **PHP 7** non più supportate) per realizzare tutte le fasi necesarie nell'interagire con tale DBMS; a partire da PHP 5.0 si usa l'***estensione MySQLi*** *(MySQL improved)* che può essere programmata tramite un **approccio Object Oriented** sebbene mantenga la possibilità di poter essere utilizzata anche con un approccio **procedurale**.

Una terza possibilità, dalla versione 5.1, è data dal ricorso all'***estensione*** PHP Data Objects (***PDO***), completamente orientata agli oggetti, e che non può quindi essere utilizzata con un approccio procedurale. La caratteristica principale[3] che differenzia questa estensione rispetto a MySQLi è che essa dispone di driver per **diversi DBMS**, non limitandosi esclusivamente a MySQL. In sostanza **PDO** (nel confronto) fornisce un *livello di astrazione* per l'interazione con le basi di dati.

Solo alcune delle funzioni dell'estenzione nativa sono oggi deprecate.

Per **connettersi**[4], allora, ad un database ***MySQL*** remoto

estensione nativa	***estensione MySQLi***
Uso della funzione **mysql_connect()** che prevede come argomenti l'**host** da cui si può raggiungere MySQL, **username** e **password** per l'accesso al database: **mysql_connect (*server, login, password*);** Tale funzione consente il collegamento con il database. Questa funzione se riceve un valore lo passa alla variabile *resource*[5] (nel nostro esempio $db). <? **$db** = mysql_connect ($db_host, $db_user, $db_password); ?>	Estensione **procedurale**: uso della funzione **mysqli_connect()** Estensione con **approccio ad oggetti**: creeremo un'**istanza** **new mysqli** ($db_host, $db_user, $db_password) o **new mysqli** ($db_host, $db_user, $db_password, **$db_name**) <? **$db** = **new mysqli** ($db_host, $db_user, $db_password) ?>

I valori da inserire solitamente sono dati dal server dove risiede il database (in fase di registrazione utente dovete inserire la password ed il login).

Se la connessione ha buon fine restituisce un *identificatore* alla connessione che si memorizza nella variabile **$db**. Useremo questa variabile ogni volta che vorremo fare un'operazione sul database.

Se la connessione non dovesse andare a buon fine (per esempio se uno dei parametri fosse sbagliato) verrebbe restituito FALSE.

---

3 Svantaggi: leggermente meno performante di MySQLi e non ha accesso diretto ad alcune caratteristiche delle ultime versioni di MySQL come i *multiple statements*.

4 Interessante e sintetico confronto nel creare una connessione (MySQLi OO, MySQLi procedurale, PDO)

5 Nuovo tipo di variabile che ci serve come *puntatore* al database

Quindi dovremo **verificare il buon esito della connessione** utilizzando la funzione **die()** che interrompe l'esecuzione dopo aver scritto a video tutto quello che contengono le parentesi:

estensione nativa	***estensione MySQLi con approccio ad ogggetti***
<? if ($db == FALSE) **die** ("Errore nella connessione[6]. Verificare i parametri"); ?> Una volta stabilita la connessione, il passo successivo è **selezionare il database** col quale vogliamo lavorare. Per questo si usa la funzione **mysql_select_db(*nomedb, connessione*)**: col primo parametro passiamo il nome del db al quale vogliamo connetterci, col secondo l'identificativo di connessione (cioè quello che abbiamo ottenuto da mysql_connect). Questa funzione **restituisce** un valore booleano che indica se la selezione del database è riuscita o no. Quindi si prevede di verificare con quale database si vuole lavorare e di controllare nuovamente la riuscita dell'operazione: <? mysql_select_db ($db_name, $db) **or** die ("Errore nella selezione del database. Verificare i parametri"); ?>	*Caso* $db = **new mysqli** ($db_host, $db_user, $db_password, **$db_name**) ***// contemporaneo test connessione e db:*** <? if ($db ->**connect_error**) **die** ("Errore nella connessione[7] e selezione del database. Verificare i parametri"); ?> *Caso* $db= **new mysqli** ($host, $user, $password); ***// test connessione:*** if ($db->**connect_error**) { die("Errore connessione: " . $db->**connect_error**); } ***// test esistenza del database:*** if($db->**select_db**($db_name) == 0) { echo("Il database non esiste"); exit; }

Segmento che può prevedere il salvataggio dei parametri di connessione e del db in altro file:

config.inc.php <? **// parametri del database** $db_host = "localhost"; $db_user = "*account*"; $db_password = ""; **// opzionale** $db_name = "my_*account*"; ?>	<? **require ("config.inc.php"); // *include* *fase test*** **// connessione con estensione MySQLi** $db = **new mysqli**($db_host, $db_user, $db_password, $db_name); if ($db **->connect_error**)die ("Errore nella connessione e selezione del database. Verificare i parametri nel file config.inc.php"); **// …… esecuzione di query** ?>

6 Volendo conoscere il motivo della mancata connessione, si usa la funzione senza parametri **mysql_error()** che effettua la stampa dell'errore segnalato dal server MySql impostando la seguente sintassi:

die("Errore nella connessione a MySql: " . mysql_error());

7 Volendo conoscere il motivo della mancata connessione: $db ->connect_error supportato dopo PHP 5.2.9 e 5.3.0
if ($db->connect_error)
die("Connessione fallita, motivi: " . $d->connect_error); // pur se esiste **mysqli_error()**

Esecuzione di una query

Si è arrivati alla parte fondamentale del colloquio con un database, cioè l'esecuzione di una query.

estensione nativa	*estensione MySQLi*
Per eseguire la query si usa la funzione **mysql_query (*query, connessione*)** alla quale viene passata la query da eseguire insieme all'identificativo di connessione. Anche questa funzione **restituisce** un valore, per il quale però dobbiamo distinguere due possibilità rispetto al tipo di query che abbiamo lanciato:	**mysqli_query ($db, $query)** o **$db->query($query);** *nb:* si noti la diversa posizione dei parametri nell'uso di MySQLi procedurale

- Se si tratta di una **query di interrogazione** (SELECT, SHOW, EXPLAIN, DESCRIBE), la funzione restituisce un **identificativo del risultato** (cioè un'altra variabile di tipo resource), che ci servirà successivamente, se la query è andata a buon fine; se invece MySql ha rilevato degli errori, la funzione restituisce FALSE;
- Se invece si tratta di una **query di aggiornamento** (INSERT, UPDATE, DELETE e tutte le altre diverse da quelle viste prima), la funzione restituirà in ogni caso un valore booleano, ad indicare se l'esecuzione è andata a buon fine oppure no.

A questo punto dobbiamo richiamare l'attenzione ancora una volta sulla necessità di **verificare il risultato** della nostra query, importante più che mai in questa situazione in quanto è molto facile commettere errori in una query. Vediamo quindi un esempio:

```
$query = 'SELECT * FROM tabella';
```

estensione nativa	*estensione MySQLi*
$ris = mysql_query(**$query, $db**) or die("Errore nella query: " . mysql_error());	$ris = **mysqli_query($db, $query**); o **$ris = $db->query($query);**

In questo modo, se la query ha avuto successo la variabile $ris conterrà l'identificativo del risultato, che ci servirà successivamente per **leggere** le righe restituite al volo. Se invece la query non va a buon fine, lo script si blocca segnalando l'errore. Nel caso in cui avessimo voluto eseguire una query di aggiornamento, avremmo potuto evitare di assegnare il risultato ad una variabile.

Per retrocompatibilità, ogni funzione dell'estenzione mysqli procedurale ha la stessa sintassi delle funzioni dell'estensione nativa: si noti, invece, la diversa sintassi in questo caso:

mysqli_query (connection, query, *resultmode*);

resultmode è una costante opzionale:

- MYSQLI_USE_RESULT (usata se si devonorecuperare grandi quantità di dati)
- MYSQLI_STORE_RESULT (di default)

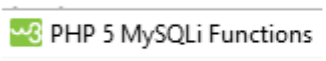
Verifica dei risultati della query - *estensione nativa / MySQLi* procedurale

Il fatto che una query sia stata eseguita correttamente non significa necessariamente che abbia prodotto dei risultati. Può infatti verificarsi il caso in cui una query, pur essendo perfettamente corretta, non produce alcun risultato, ad esempio perché le condizioni che abbiamo specificato nella clausola WHERE non sono mai verificate sulle tabelle interessate.

Se vogliamo sapere **quante** righe sono state **restituite** da una SELECT, possiamo usare la funzione **mysql_num_rows(*risultato*) / mysqli_num_rows(*risultato*)**, che ci **restituisce** il numero di righe contenute dall'identificativo del risultato che le passiamo. Se invece abbiamo eseguito una query di aggiornamento (INSERT, UPDATE, DELETE) e vogliamo sapere **quante** righe sono state **modificate**, possiamo usare **mysql_affected_rows (*connessione*) / mysqli_affected_rows (*connessione*)** , che ci **restituisce** il numero di righe modificate dall'ultima query di aggiornamento.

Estensione nativa per MySQL

```
$query = 'SELECT * FROM tabella';
$ris = mysql_query($query,$db) or die("Errore nella query: " . mysql_error());

/* $righe riceve il numero di righe restituite dalla SELECT */
$righe = mysql_num_rows($ris);

$query = "UPDATE tabella SET campo1='valore' WHERE campo2='val'";
mysql_query($query,$db) or die("Errore nella query: " . mysql_error());

 /* $righe riceve il numero di righe modificate da UPDATE */
$righe = mysql_affected_rows($db);
```



Estensione per RDBMS MySQL version 4.1.13 e successive

```
$query = 'SELECT * FROM tabella';
$ris = mysqli_query($query,$db) or die("Errore nella query: " . mysqli_error());

 /* $righe riceve il numero di righe restituite dalla SELECT */
$righe = mysqli_num_rows($ris);

$query = "UPDATE tabella SET campo1='valore' WHERE campo2='val'";
mysqli_query($query,$db) or die("Errore nella query: " . mysqli_error());

/* $righe riceve il numero di righe modificate da UPDATE */
$righe = mysqli_affected_rows($db);
```

È importante notare la differenza nel parametro da passare alle due funzioni:
mentre **mysql_num_rows() / mysqli_num_rows()** richiede un *identificativo di risultato*, **mysql_affected_rows() / mysqli_affected_rows()** richiede un *identificativo di connessione*; infatti, come abbiamo visto prima, una query di aggiornamento **non** restituisce un identificativo di risultato.

Lettura dei risultati di una SELECT

Come abbiamo visto prima, una volta effettuata una query di interrogazione abbiamo a disposizione un identificativo del suo risultato. Per poter leggere questo risultato possiamo utilizzare funzioni dedicate che **ogni volta** che vengono chiamate, **restituiscono una riga** del risultato:

estensione nativa	*estensione MySQLi*
La funzione **mysql_fetch_array (*risultato*)** restituisce una riga del risultato sotto forma di array potendo ***accedere con indice numerico*** ai campi; quando non ci sono più righe da leggere, la funzione restituisce FALSE.	**mysqli**_fetch_array (***risultato***) o **$ris->fetch_array();**

Quindi, per scorrere tutto il risultato, dovremo usare questa funzione come condizione di un ciclo, che si concluderà quando restituisce FALSE. In questo modo **non abbiamo bisogno** di sapere a priori quante sono le righe contenute nel risultato stesso.

```
$query = 'SELECT * FROM tabella';
```

estensione nativa	*estensione MySQLi*
$ris = mysql_query($query, $db) or die("Errore nella query: " . mysql_error()); while($riga = **mysql_fetch_array**($ris)) { ***//codice che elabora i dati*** }	$ris = **mysqli_query($db, $query);** o **$ris = $db->query($query);** while($riga = **mysqli**_fetch_array($ris)) { ***//codice che elabora i dati*** }

Ogni volta che questo ciclo viene eseguito, quindi, avremo a disposizione, nella variabile $riga, una riga del nostro risultato. Questa variabile è in effetti un **array** che contiene i valori delle colonne (valori dei campi) restituiti dalla query. Gli indici dell'array sono i nomi delle colonne, ed i loro valori sono i valori estratti dal database. Si potranno recuperare i valori estratti sia con indici numerici sia con indici associativi.

Segmento di codice – estensione MySQLi con approccio ad oggetti

controllo: si realizza il ciclo solo se la query ha ritornato almeno una riga

```
$ris = $db->query($query);
if ($ris->num_rows > 0)
{  echo "<table>";                                          // si visualizzano i dati in forma tabellare
   echo "<tr><th>Nome</th><th>Telefono</th></tr>";
   while ($row = $ri ->fetch_assoc())                      // approccio ai risultati come array associativo
    {
       echo "<tr><td>". $row["nome"]. "</td><td>". $row["telefono"]. "</td></tr>";
    }
    echo "</table>";
}
else
{  echo "nessun risultato";   // si evita il ciclo
}
```

Rivediamo dunque l'esempio di prima, specificando per maggior chiarezza quali colonne vogliamo estrarre dalla tabella:

```
$query = 'SELECT nome, indirizzo, telefono FROM tabella';
```

estensione nativa	*estensione MySQLi*
$ris = mysql_query($query,$db) or die("Errore nella query: " . mysql_error()); while($riga = **mysql_fetch_array**($ris)) { print"Nome: $riga[nome] "; print"Indirizzo: $riga[indirizzo] "; print"Telefono: $riga[telefono] "; print" "; }	$ris = **mysqli_query($db, $query);** while($riga = **mysqli_fetch_array**($ris)) { print"Nome: $riga[nome] "; print"Indirizzo: $riga[indirizzo] "; print"Telefono: $riga[telefono] "; print" "; }

Con questo ciclo quindi stamperemo tutti i valori estratti dalla query, separando con una riga vuota i blocchi relativi ad ogni record. Nella select abbiamo estratto le colonne 'nome', 'indirizzo' e 'telefono', e quindi l'array $riga conterrà tre elementi con questi indici. In realtà, l'array $riga contiene anche altri tre elementi, con **indici numerici** 0, 1 e 2, che contengono sempre gli stessi dati (nome, indirizzo e telefono) nell'ordine in cui li abbiamo indicati nella select.

Questi dati sono alquanto inutili, in quanto è molto più comodo, ovviamente, usare gli indici alfanumerici con i nomi delle colonne.

Se vogliamo evitare di ricevere questi dati aggiuntivi, togliendo così un po' di lavoro a PHP, possiamo chiamare la funzione **mysql_fetch_array() / mysqli_fetch_array()** specificando il parametro aggiuntivo **MYSQL_ASSOC** (è una costante, va scritta in maiuscolo senza $ davanti); in alternativa, possiamo usare la funzione

mysql_fetch_assoc (risultato) / mysqli_fetch_assoc (risultato)

che equivale a mysql_fetch_array() /mysqli_fetch_array() ma restituisce solo gli indici associativi.

```
/* solo indici associativi */
$riga = mysql_fetch_array($ris,MYSQL_ASSOC);

 // solo indici associativi
$riga = mysql_fetch_assoc($ris);
```

```
/* solo indici associativi */
$riga =
mysqli_fetch_array($ris,MYSQL_ASSOC);

 // solo indici associativi
$riga = mysqli_fetch_assoc($ris);
 o
$riga = $ris->fetch_assoc();
```

In *temi* di *Sistemi e Reti* proposti agli Esami di Stato:

a.s.2015/2016 ***PHP estensione nativa*** (pg.10) e tipiche soluzioni a temi di *Informatica* in a.s. precedenti, *Suppletiva 2016 proposta come area di progetto 2018* ***PHP estensione MySQLi procedurale*** (pg. 11-12), *a.s.2017/18* ***PHP estensione MySQLi procedurale*** (pg.9) o altra proposta ***PHP estensione MySQLi con approccio ad oggetti*** (pg. 8-9) nel confronto con **ASPX** e linguaggio **C#** (pg.10-11)

Con l'introduzione del **modello ad oggetti**[8], possiamo recuperare come oggetti le t-ple che costituiscono il *recordset* cioè la tabella dinamica generata da una query di interrogazione; si prelevano infatti le singole righe della tabella, usando la funzione:

mysql_fetch_object (*risultato*) / mysqli_fetch_object (*risultato*)

che restituisce la riga corrente e potendo recuperare i valori di ogni campo come **attributi di oggetti**:

```
$riga = mysql_fetch_object($ris);          /* preleva la riga corrente della tabella dinamica */

while($riga = mysql_fetch_object($ris))
{
        echo "$riga->nome   $riga->telefono   $riga->indirizzo <br>\n ";
      // visualizza scorrendo tutte le righe ed inserisce uno spazio tra valori in ogni riga
}
```

oppure, sfruttando la possibilità di gestire con echo più stringhe divise da virgole:

```
$riga = mysql_fetch_object($ris); /* preleva la riga corrente della tabella dinamica */
while($riga = mysql_fetch_object($ris))
{
        echo "$riga->nome" , "   $riga->telefono" , "   $riga->indirizzo <br>\n" ;
        // visualizza scorrendo tutte le righe ed inserisce uno spazio tra valori in ogni riga
}
```

Si è quindi completata questa veloce carrellata sulle principali **funzioni** da utilizzare per interagire da PHP con un database MySql.

Rimane da citare la possibilità di chiudere esplicitamente la connessione

estensione nativa	***estensione MySQLi***
Funzione **mysql_close (*connessione*)** che serve per chiudere la connessione aperta con mysql_connect(), ma in pratica questa funzione è usata pochissimo, in quanto PHP si preoccupa da solo, al termine dello script, di chiudere le connessioni che abbiamo aperto. **mysql_close**($db); **mysql_close**(); */* non specificando, saranno chiuse tutte le connessioni */*	**mysqli_close (*connessione*)** o **$db->close();** **mysqli_close**($db); **mysqli_close**();

nb: in **PHP 7 non sono più incluse le *mysql functions*** che non vanno più considerate un'alternativa valida. Se si usa l'estensione **MySQLi** (*MySQL improved*), si sfrutta la caratteristica peculiare di questa estensione: quella di disporre di una doppia interfaccia: procedurale e **ad oggetti**.

Esempio di connessione a MySQL con ***estensione*** mysqli nell'***approccio ad oggetti***:

```
$mysqli = new mysqli('localhost', 'username', 'password', 'nome_database');
```

8 Supporto di base agli oggetti da **PHP 4.0**; supporto completo agli oggetti e migliori performance da **PHP 5.0**

Creazione di una tabella

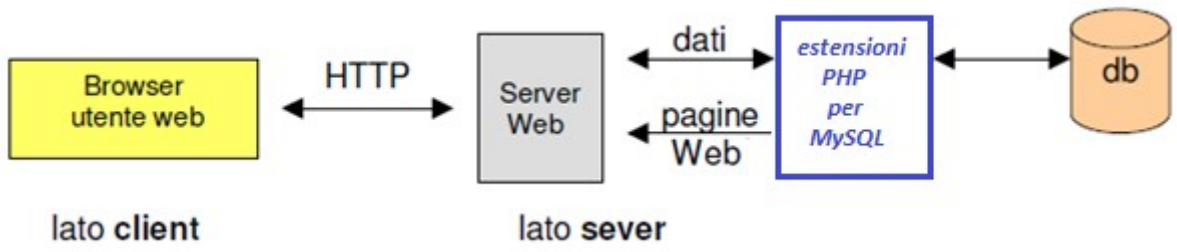
Per creare una tabella e controllare la riuscita dell'operazione:

```
<?
$query = "CREATE TABLE news (id INT (5) UNSIGNED not null AUTO_INCREMENT, titolo
VARCHAR (255) not null , testo TEXT not null , data INT (11) , autore VARCHAR (50) , mail
VARCHAR (50) , PRIMARY KEY (id))";

// controllo usando l'estensione mysqli procedurale

 $result = mysqli_query($db, $query);
 if ($result)

    echo "L'installazione e' stata eseguita correttamente";

else

     echo "Errore durante l'installazione";

 mysqli_close($db);
?>
```

Oppure:

```
<?
$query = "CREATE TABLE news (id INT (5) UNSIGNED not null AUTO_INCREMENT, titolo
VARCHAR (255) not null , testo TEXT not null , data INT (11) , autore VARCHAR (50) , mail
VARCHAR (50) , PRIMARY KEY (id))";

// controllo usando l'estensione mysqli ad oggetti

 $result = $db->query($query);
 if ($result)

    echo "L'installazione e' stata eseguita correttamente";

else

     echo "Errore durante l'installazione";

 $d-> close();  // chiusura usando l'estensione mysqli ad oggetti
?>
```

nb:
i precedenti segmenti di codice prevedono già creato il database e realizzata la corretta connessione al database usando gli opportuni paametri



Esempio di **creazione della tabella** ***contatti*** **(se non esiste)** con **estensione** **PDO**

http://new345.altervista.org/PHP/db/OO/install_contattiPDO.php ***(da guida in italiano)***

Inserimento di record in una tabella

Esempio di inserimento con campi obbligatori titolo e testo poichè id si autoincrementa

Nb: se non forniamo – al momento della creazione – il contenuto dei campi *not null*, l'inserzione non può avvenire.

```
<?
$query = "INSERT INTO news (titolo, testo, data, autore, mail) VALUES ('vecchio articolo', 'Ecco un articolo', '1002664800', 'prof. Biasotti', 'pbiasotti@libero.it')";

// esecuzione e controllo usando l'estensione mysqli procedurale
// con data 1002664800  in formato timestamp
// cioè numero di secondi trascorsi a partire dall'ora 00:00 del 1 gennaio 1970
```

```
 $result = mysqli_query($db, $query);
 if ($result)

   echo "L'articolo e' stato inserito correttamente";

else

   echo "Erorre durante l'inserimento";

echo "<br/>";
?>
```

NB: analogamente, si potranno realizzare aggiornamenti

o cancellazioni

Estrazione da una tabella

Esempi di estrazione e visualizzazione con uso di identificatore delle singole righe cioè la variabile $result

```
<?
$query = "SELECT * FROM news ";

// estensione mysqli procedurale
$result = mysqli_query($db, $query);
while ($row = mysqli_fetch_array($result))
{ echo "$row[testo] " . date("j/n/y", $row[data]) . " - $row[titolo] <br/>"; }

/*  date("j/n/y", $row[data]) è una funzione PHP
    che in base a una data in timestamp (nel nostro caso memorizzato in $row[data]
    crea la data nel formato g/m/aa
*/

// funzione opzionale infatti la chiusura della connessione al database è automatica
// tutte le connessioni vengono chiuse automaticamente alla fine della pagina.

mysqli_close($db);
?>
```

Oppure: stessa query di estrazione

```
<?
$query = "SELECT * FROM news ";
// estensione mysqli OO
$result = $db->query($query);
while ($row = $result->fetch_assoc())
{ echo "$row[testo] " . date("j/n/y", $row[data]) . " - $row[titolo] <br/>"; }

/*  date("j/n/y", $row[data]) è una funzione PHP che in base a una data in timestamp (nel nostro
    caso memorizzato in $row[data] crea la data nel formato g/m/aa
*/

 $db->close(); // opzionale infatti la chiusura della connessione al database è automatica
               // tutte le connessioni vengono chiuse automaticamente alla fine della pagina.
?>
```

Altra query di estrazione

Esempio di estrazione in **ordine cronologico** di **sole 3 righe**:

```
<?
$query = "SELECT id,data,titolo FROM news ORDER BY data DESC LIMIT 0,3";
?>
```

NB: analogamente, si potranno realizzare estrazioni da più tabelle (*join*)

con vari *criteri di selezione*

e ancora si potranno ***deallocazione*** tabelle, indici, database

Prepared statements

Uno degli ambiti di utilizzo pratico più importanti per i *Prepared Statements* è quello che prevede l'invio di parametri per le query tramite **campi form** compilabili dagli utenti o **querystring** presenti nella URL di una pagina web (e pertanto facilmente alterabili dall'esterno e quindi potenziamente pericolosi in quanto utilizzabili per veicolare istruzioni malevole): con l'uso dei *prepared statements*, invece, gli input non vengono determinati a priori dallo sviluppatore e il loro **contenuto rimane sconosciuto all'applicazione fino al momento dell'elaborazione**.

Un vantaggio di tipo sintattico dato da PDO è la possibilità di **assegnare dei nomi** ai parametri indicati nei *prepared statements* (come vedremo più avanti), senza dover utilizzare il formalismo meno intuitivo di MySQLi basato sui segnaposto (*placeholders*).

Prepared statements[9] **– *estensione MSQLi con approccio OO***

- ***esempio di estrazione*** ***http://new345.altervista.org/PHP/db/OO/index_MySQLi_OO.php***

Prepared statements – ***estensione PHP Data Objects (PDO[10])***

- ***esempio di inserimento in tabella MyGuest*** *personalizzando (hosting free altervista)*

9 Chiara e completa Guida pratica PHP e MySQL con sezione dedicata *Un'applicazione completa con MySQLi*
10 Dispensa dedicata all'estensione PDO