Linguaggi di Programmazione per il Web – Parte 4

PHP – Hypertext Preprocessor

L'interazione con MYSQL

Estensione MySQLi

Autore

Prof. **Rio Chierego**
riochierego@libero.it

Siti Utili

http://www.riochierego.it/mobile

http://www.html.it/

http://www.mrwebmaster.it

http://www.php.net/

http://php.html.it/

http://mysql.it

PHP & MySql

Collegarsi ad un **database** e interagire con i dati in esso archiviati è indubbiamente uno dei compiti principali di un sito Web dinamico costruito con PHP.

PHP ci dà la possibilità di connetterci con un numero elevatissimo di database server (**MySql**, PostgreSql, Oracle, Microsoft Sql Server, Access, Sybase, Informix, mSql ecc.).

Prenderemo in considerazione **MySql**, che è il database server che si è affermato prepotentemente negli ultimi anni per la sua velocità e la sua stabilità, oltre che per il fatto di essere open source.

PHP & MySql

Lo sviluppatore interessato a costruire delle applicazioni che possano colloquiare con **MySQL**, il DBMS (*Database Management System*) Open Source per la gestione di database relazionali più utilizzato in Rete con PHP, ha a sua disposizione principalmente tre scelte corrispondenti ad altrettante estensioni.

1) estensione nativa per MySQL

N.B. presente dalla prima versione 2.0 di PHP ma deprecata a partire dalla versione 5.5

2) estensione MySQLi (*MySQL improved*)

N.B. utilizzabile a partire dalla versione PHP 5.0 sia con l'approccio procedurale sia con l'approccio Object Oriented

3) estensione *PHP Data Objects* (PDO)

N.B. completamente orientata alla programmazione Object Oriented

PHP & MySql – estensione MySQLi

MySQLi è l'abbreviazione di MySQL *improveded* ("perfezionato") ed è il nome di un'estensione per il linguaggio PHP che è stata realizzata per mettere a disposizione degli sviluppatori delle funzionalità più avanzate rispetto a quelle messe a disposizione **dall'estensione MySQL nativa** per l'interazione tra le applicazioni Web based e il noto DBMS Open Source MySQL.

Ma perché utilizzare questa estensione invece della classica estensione MySQL nativa utilizzata in milioni di applicazioni?

In realtà non è possibile sostenere in maniera assoluta che l'estensione MySQLi sia preferibile rispetto alla sua alternativa più classica, in molti sostengono addirittura che si tratti di un progetto ancora incompleto e non ben supportato; grazie a MySQLi vi sono però degli indubbi vantaggi che meritano di essere sottolineati.

PHP & MySql – estensione MySQLi

- può essere utilizzata sia all'interno di applicazioni realizzate seguendo il **paradigma procedurale** sia all'interno di applicazioni realizzate seguendo il **paradigma Object Oriented**;
- fornisce il supporto nativo al protocollo binario di MySQL introdotto con la versione 4.1;
- supporta ***stored procedures*, query multiple** e ***transactions***;
- permette impostazioni avanzate per la connessione tramite **mysqli_init()**;
- supporta le ***prepared statements*** per il *caching* delle interrogazioni;
- garantisce prestazioni più elevate dell'estensione nativa MySQL, è ritenuta più sicura e fornisce in genere migliori risposte in fase di *debugging*.

PHP & MySql – estensione MySQLi

- Come anticipato, gli strumenti forniti da MySQLi possono essere impiegati sia all'interno di applicazioni il cui codice è stato realizzato secondo il paradigma procedurale che in uno script creato grazie all'*Object-Oriented Programming*; infatti, come si vedrà a breve nel dettaglio, la gran parte delle funzioni messe a disposizione da questa estensioni possono essere sostituite da un metodo concepito per uno scopo analogo.

- Le funzioni dell'estensione MySQLi sono state concepite per rendere questo strumento il più possibile compatibile con l'estensione MySQL; nella maggior parte dei casi è infatti possibile ritrovare le funzioni classiche con una "i" finale aggiunta in coda al nome: per esempio, **mysql_connect()** è stata così "tradotta" in **mysqli_connect()** mentre **mysql_query()** è diventata **mysqli_query()**

N.B. Ciò nonostante è opportuno sottolineare che, al di là delle somiglianze relative al nome, non tutte le funzioni di MySQLi possono essere utilizzare nello stesso modo di quelle fornite da MySQL

Connessione a MySQL con MySQLi

Per la connessione tra un'applicazione Web based realizzata in PHP e un Database Manager MySQL tramite l'estensione MySQLi, sono disponibili **due procedure**:

(-) una prima basata sul paradigma procedurale

(-) una seconda basata invece sull'approccio per oggetti.

N.B. Nel primo caso la connessione avviene per "funzione", nel caso della OOP essa avviene invece per "istanza"

La connessione effettuata utilizzando la programmazione procedurale si basa su una funzione specifica denominata **mysqli_connect()** che accetta come parametri i seguenti dati:

- **hostname**: il nome di Rete della macchina ospitante il Database server, ad esempio un indirizzo IP o "localhost" nel caso di installazioni locali;
- **username**: il nome dell'utente da utilizzare per la connessione, esso dovrà avere i privilegi necessari per l'accesso e la manipolazione dei dati;
- **password**: la parola chiave necessaria per l'autenticazione dell'utente che si desidera utilizzare in connessione;
- **database**: il nome del database che si desidera interfacciare alla propria applicazione

Connessione a MySQL con MySQLi

I più accorti si saranno immediatamente resi conto di un'importante differenza tra le funzioni **mysqli_connect() e mysql_connect()** dell'estensione nativa**,**

la seconda infatti non accetta come parametro il nome della base di dati da selezionare, a questo scopo è infatti destinata la funzione **mysql_select_db()**.

N.B. MySQLi mette a disposizione una funzione omologa, denominata non a caso mysqli_select_db(), che però deve essere utilizzata esclusivamente nel caso in cui si presenti la necessità di modificare il database predefinito (cioè quello selezionato tramite mysqli_connect()).

Connessione a MySQL con MySQLi

Esempio di apertura/chiusura di una connessione con approccio procedurale

```
<?php
// connessione a MySQL con mysqli_connect()
// definizione delle variabili
$host = "localhost";
$user = "username";
$pass = "password";
$db = "database";
 // connessione al DBMS
$connessione = mysqli_connect ($host, $user, $pass, $db) or
                    die("Connessione non riuscita " . mysqli_connect_error() . " "  . mysqli_connect_errno() );
.....
 mysqli_close ($connessione) or
                    die("Chiusura connessione fallita " . mysqli_error($connessione) . " "  .
mysqli_errno($connessione) );
?>
```

Connessione a MySQL con MySQLi

Esempio di apertura/chiusura di una connessione con approccio Object Oriented

Per quanto riguarda invece le connessioni tramite approccio per oggetti, è stato già detto che in questo caso il fulcro della procedura è basato essenzialmente sul concetto di istanza

```
<?php
// connessione a MySQL per istanza
 // definizione delle variabili
$host = "localhost";
$user = "username";
$pass = "password";
$db = "database";
 // connessione al DBMS
$mysqli= new mysqli ($host, $user, $pass, $db);
if($mysqli->connect_error)
    {
   die ("<br>Connessione non riuscita " . $mysqli->connect_error . " " . $mysqli->connect_errno);
    }
.....
$mysqli->close();
?>
```

Connessione a MySQL con MySQLi

Esempio di connessione basato sull'approccio Object Oriented

Nel esempio proposto, viene effettuata un'istanza dell'oggetto "$connessione" appartenente alla classe "mysqli"; come è possibile osservare, questa operazione prevede il passaggio degli stessi parametri previsti per la funzione **mysqli_connect().**

Nel codice precedente viene introdotta anche un'altra funzione, **mysqli_connect_errno()** che ha il compito di produrre, in caso di malfunzionamento, il numero identificativo dell'errore notificato in connessione dal Database manager; se questo numero dovesse essere generato, allora la funzione **mysqli_connect_error()** comunicherà all'utente la tipologia di eccezione verificatasi.

Connessione a MySQL con MySQLi

- La funzione **mysqli_connect() e l'istanza alla classe mysqli di connessione** non sono le uniche soluzioni disponibili per l'accesso al Database manager MySQL messe a disposizione da MySQLi.

- In alternativa, è possibile utilizzare la funzione **mysqli_real_connect()** (con l'approccio procedurale) o il relativo metodo **mysqli::real_connect** (con l'approccio Object oriented);

- la funzione mysqli_real_connect() funziona in modo molto simile a mysqli_connect() per aprire una connessione al DBMS, ma presenta come differenza sostanziale la possibilità di specificare dei parametri addizionali (**Flags**).

Connessione a MySQL con MySQLi

Funzione mysqli_real_connect() da usare con approccio procedurale

La funzione mysqli_real_connect() funziona in modo molto simile a mysqli_connect() per aprire una connessione al DBMS, ma presenta come differenza sostanziale la possibilità di specificare dei parametri addizionali (**Flags**).

I parametri addizionali previsti per questa funzione sono cinque e rappresentano altrettante opzioni di connessione:

- **MYSQLI_CLIENT_COMPRESS**: permette l'utilizzo del protocollo di compressione;
- **MYSQLI_CLIENT_FOUND_ROWS**: restituisce il numero di record rilevati, non quello dei record coinvolti dall'interrogazione;
- **MYSQLI_CLIENT_IGNORE_SPACE**: consente l'utilizzo di spazi vuoti dopo i nomi delle funzioni e permette di rendere riservati tutti i nomi delle funzioni;
- **MYSQLI_CLIENT_INTERACTIVE**: attiva l'impostazione di MySQL **interactive_timeout** espressa in secondi invece di **wait_timeout** (sempre espressa in secondi), in questo modo la connessione verrà chiusa dopo l'intervallo di tempo definito in mancaza di istruzioni al DBMS;
- **MYSQLI_CLIENT_SSL**: utilizza il protocollo crittografico **SSL** (*Secure Sockets Layer*) che consente una comunicazione sicura e l'integrità dei dati su reti TCP/IP.

N.B. Per questioni di sicurezza, la **funzione mysqli_real_connect()** non supporta il flag **MULTI_STATEMENT** per l'invio simultaneo di più interrogazioni al DBMS, a questo scopo sarà invece possibile utilizzare come alternativa una funzione apposita, denominata **mysqli_multi_query()** (o il relativo metodo **mysqli::multi_query**).

Connessione a MySQL con MySQLi

Esempio: Funzione mysqli_real_connect() da usare con approccio procedurale

```
<?php
// connessione con il metodo mysqli_real_connect()
 // inizializzazione di MySQLi
$link = mysqli_init() or die ("<br> Errore nell'inizializzazione di MySQLi " . $mysqli->error . " ".
      $mysqli->errno );

 // definizione delle variabili
$host = "localhost";
$user = "username";
$pass = "password";
$db = "data";
 // connessione al DBMS
```

.....Vedi slide seguente.....

Connessione a MySQL con MySQLi

Esempio: Funzione mysqli_real_connect() da usare con approccio procedurale

```
// connessione al DBMS
mysqli_real_connect($link, $host, $user, $pass, $db)
 or die ("<br>Errore di connessione" . mysqli_error($connessione) . " ". mysqli_errno($connessione));

//resto del programma
.........


// chiusura della connessione
mysqli_close($link)
  or die ("<br>Errore di chiusura" . mysqli_error($connessione) . " ". mysqli_errno($connessione));
?>
```

Connessione a MySQL con MySQLi

Esempio: Funzione mysqli_real_connect() da usare con approccio procedurale

Si analizzi il codice proposto:

- viene inizializzato MySQLi tramite la funzione **mysqli_init()** che crea una risorsa ("link") da passare come argomento all'istruzione di connessione;
- vengono definite le variabili da utilizzare come parametri a **mysqli_real_connect();**
- viene stabilita la connessione a meno di eventuali errori;
- viene chiusa la connessione tramite la funzione **mysqli_close()** a cui viene passata come argomento la risorsa creata in inizializzazione.

Connessione a MySQL con MySQLi

Metodo mysqli::real_connect da usare con approccio Object Oriented

```
<?php
// connessione con il metodo mysqli_real_connect()
// inizializzazione di MySQLi
$link = mysqli_init() or die ("<br> Errore nell'inizializzazione di MySQLi " . $mysqli->error . " ".
      $mysqli->errno );

 // definizione delle variabili
$host = "localhost";
$user = "username";
$pass = "password";
$db = "data";
```

.... Vedi slide seguente

Connessione a MySQL con MySQLi

Metodo mysqli::real_connect da usare con approccio Object Oriented

```
// connessione al DBMS
$mysqli->real_connect($host, $user, $pass, $db)
    or die("<br>Errore di connessione" . $mysqli->error . " ". $mysqli->errno );

//resto del programma
.....

// chiusura della connessione
$mysqli->close()
     or die ("<br>Chiusura connessione fallita " . $mysqli->error . " ". $mysqli->errno );?>
```

Connessione a MySQL con MySQLi

Metodo mysqli::real_connect da usare con approccio Object Oriented

Le differenze tra i due codici proposti non sono tante: anche in questo caso MySQLi viene inizializzato tramite **mysqli_init()**, ma la risorsa creata non deve essere passata al metodo di connessione come parametro, perché disponibile per istanza all'interno della **classe MySQLi**; di conseguenza non sarà necessario passare questo argomento neanche al metodo **mysqli::close** che chiuderà la connessione

Eseguire una query con MySQLi

Indipendentemente dall'approccio che si intenderà utilizzare (procedurale o Object Oriented) in generale ciascuna query è composta di 4 fasi (3 + 1):

- **preparazione** = costruzione di una stringa contenente il comando SQL da eseguire
- **esecuzione** = invio del comando SQL al DBMS, generazione e ricezione del result set
- **estrazione** = lettura dei dati presenti nel result set
- **formattazione** = costruzione del codice HTML per visualizzare i risultati nella pagina

L'estrazione e formattazione solitamente sono organizzate in un ciclo (while o for)

La formattazione è opzionale, dipende dalle esigenze (presentazione o uso per altre funzioni)

Eseguire una query con MySQLi

Query: preparazione

- **comando SQL fisso**

esempi

$query = "SELECT login FROM utente";

$query = "INSERT INTO utente (login, password) VALUES ('pippo', 'xyz')";

- **comando SQL con valori variabili**

esempio:

$query = "SELECT login FROM utente WHERE password = ' . $password . ' " ;

- si usa l'operatore di concatenazione (.) per inserire variabili PHP nella query SQL (N.B. usare gli apici singoli '…' intorno ad ogni stringa)

N.B. la fase di preparazione della query è la stessa in entrambi gli approcci in quanto si tratta di costruire, tramite il PHP, un comando che rispetti la sintassi del linguaggio SQL

Eseguire una query con MySQLi

Query: esecuzione

- **una volta preparata, è sufficiente inviare la query SQL al DBMS, usando la connessione attiva ed il database attualmente selezionato:**
- **per le query di tipo INSERT, DELETE o UPDATE il lavoro è terminato**
- **per le query di tipo SELECT occorre estrarre ed analizzare il result set**

  (-) il valore di ritorno è un "riferimento" al result set attraverso cui si arriva al risultato vero e proprio

  (-) liberare la memoria una volta esaminato il risultato

Eseguire una query con MySQLi

Query: esecuzione (approccio procedurale)

```
//apertura di una connessione
....
$query  = "SELECT * FROM utente“;
$result  = mysqli_query ($connessione, $query) or
            die ("Query fallita " . mysqli_error($connessione) . " " . mysqli_errno($connessione));

// operazioni sul result set
...
// rilascio della memoria associata al result set solo per le query di selezione
...
// chiusura connessione
....
```

Eseguire una query con MySQLi

Query: estrazione (approccio procedurale)

- **si basa sull'utilizzo di una delle due funzioni**

  mixed mysqli_fetch_array (mysqli_result $result [, int $resulttype = MYSQLI_BOTH])
  oppure array mysqli_fetch_assoc (mysqli_result $result)

- **result è il result set ottenuto da una precedente chiamata a mysqli_query**
- **il risultato della chiamata a mysqli_fetch_array è un array o solo numerico (**$resulttype **=MYSQLI_NUM), o solo associativo (**$resulttype **=MYSQLI_ASSOC), oppure entrambi gli array (default** $resulttype **=MYSQLI_BOTH) oppure NULL quando è terminato il result set**
- **il risultato della chiamata a mysqli_fetch_assoc è ESCLUSIVAMENTE un array associativo dove gli indici = nomi dei campi o attributi oppure NULL quando è terminato il result set**
- **per ottenere il numero di elementi nel result set si utilizza la funzione**

  int mysqli_num_rows (mysqli_result $result)

Eseguire una query con MySQLi

Query: estrazione (e formattazione) MODO 1 (approccio procedurale)

```
... // apertura connessione
... // esecuzione query

while ($row = mysqli_fetch_array ($result, MYSQLI_NUM))    //solo numerico
  {
 echo "<p> ID: $row[ 0] COGNOME: $row[ 1] NOME: $row[ 2]   </p>";
  }

... // chiusura connessione
```

Eseguire una query con MySQLi

Query: estrazione (e formattazione) MODO 1 (approccio procedurale)

```
... // apertura connessione
... // esecuzione query

while ($row = mysqli_fetch_array ($result, MYSQL_ASSOC))  //solo associativo
  {
 echo “<p> ID: $row[ 'id' ] COGNOME: $row[ 'surname' ] NOME: $row[ 'name' ]   </p>”;
  }

... // chiusura connessione
```

Eseguire una query con MySQLi

Query: estrazione (e formattazione) MODO 1(approccio procedurale)

```
... // apertura connessione
... // esecuzione query

while ($row = mysqli_fetch_array ($result, MYSQL_BOTH))     //entrambi -default
  {
  echo “<p> ID: $row[ 0] COGNOME: $row[ 1] NOME: $row[ 2]   </p><br>”;
  echo “<p> ID: $row[ 'id' ] COGNOME: $row[ 'surname' ] NOME: $row[ 'name' ]   </p>”;
  }

... // chiusura connessione
```

Eseguire una query con MySQLi

Query: estrazione (e formattazione) MODO 2 (approccio procedurale)

```
... // apertura connessione
... // esecuzione query

while ($row = mysqli_fetch_assoc ($result))     //solo associativo
  {
 echo “<p> ID: $row[ 'id' ] COGNOME: $row[ 'surname' ] NOME: $row[ 'name' ]   </p>);
  }

... // chiusura connessione
```

Eseguire una query con MySQLi

Query: estrazione (e formattazione) MODO 3 (approccio procedurale)

```
... // apertura connessione
... // esecuzione query

$ num_righe = mysqli_num_rows($result);
for ($i=1; $i<=$num_righe; $i++)
   {
   $row = mysqli_fetch_assoc($result);
   $id = $row[' id' ];
   $surname = $row[' surname' ];
   $name = $row[' name' ];
   echo "<p> ID: $id COGNOME: $surname NOME: $name</p>";
   }... // chiusura connessione
```

Eseguire una query con MySQLi

Query: esecuzione (approccio Object-Oriented)

```
//apertura connessione
.....
$query  = "SELECT * FROM utente“;
$result= $mysqli->query($query)
    or die ("<br>Query fallita " . $mysqli->error . " ". $mysqli->errno );

// operazioni eventuali sul result set
..
// liberazione risorse
.....
// chiusura connessione
```

Eseguire una query con MySQLi

Query: estrazione (approccio Object-Oriented)

- **si basa sull'utilizzo di di uno dei due metodi**

  mixed mysqli_result::**fetch_array** ([int $resulttype = MYSQLI_BOTH])

  oppure array mysqli_result::**fetch_assoc** ( )

- **il risultato della chiamata al metodo fetch_array è un array $result o solo numerico (**$resulttype **=MYSQLI_NUM), o solo associativo (**$resulttype **=MYSQLI_ASSOC), oppure entrambi gli array (default** $resulttype **=MYSQLI_BOTH) oppure NULL quando è terminato il result set**
- **il risultato della chiamata al metodo fetch_assoc è ESCLUSIVAMENTE l'array associativo $result dove gli indici coincidono con i nomi dei campi (attributi) oppure NULL quando è terminato il result set**
- **per ottenere il numero di elementi nel result set si utilizza l'attributo**

  int $result->**num_rows**;

Eseguire una query con MySQLi

Query: estrazione (e formattazione) MODO 1 (approccio Object-Oriented)

```
... // apertura connessione
... // esecuzione query

while ($row = $result->fetch_array (MYSQLI_NUM))    //solo numerico
  {
 echo "<p> ID: $row[ 0] COGNOME: $row[ 1] NOME: $row[ 2]   </p>";
  }

... // chiusura connessione
```

Eseguire una query con MySQLi

Query: estrazione (e formattazione) MODO 1 (approccio procedurale)

```
... // apertura connessione
... // esecuzione query

while ($row = $result->fetch_array (MYSQL_ASSOC))  //solo associativo
  {
 echo “<p> ID: $row[ 'id' ] COGNOME: $row[ 'surname' ] NOME: $row[ 'name' ]   </p>”;
  }

... // chiusura connessione
```

Eseguire una query con MySQLi

Query: estrazione (e formattazione) MODO 1(approccio procedurale)

```
... // apertura connessione
... // esecuzione query

while ($row = $result->fetch_array (MYSQL_BOTH))     //entrambi - default
  {
  echo “<p> ID: $row[ 0] COGNOME: $row[ 1] NOME: $row[ 2]   </p><br>”;
  echo “<p> ID: $row[ 'id' ] COGNOME: $row[ 'surname' ] NOME: $row[ 'name' ]   </p>”;
  }

... // chiusura connessione
```

Eseguire una query con MySQLi

Query: estrazione (e formattazione) MODO 2 (approccio Object-Oriented)

```
... // apertura connessione
... // esecuzione query

while ($row = fetch_assoc())     //solo associativo
  {
 echo “<p> ID: $row[ 'id' ] COGNOME: $row[ 'surname' ] NOME: $row[ 'name' ]   </p>);
  }

... // chiusura connessione
```

Eseguire una query con MySQLi

Query: estrazione (e formattazione) MODO 3 (approccio Object-Oriented)

```
... // apertura connessione
... // esecuzione query

$nrow = $result>num_rows;
for ($i=1; $i<=$num_righe; $i++)
   {
   $row = $result->fetch_array (MYSQL_BOTH);
   $id = $row[' id' ];
   $surname = $row[' surname' ];
   $name = $row[' name' ];
   echo "<p> ID: $id COGNOME: $surname NOME: $name</p>";
   }... // chiusura connessione
```

Risultati di una query con MySQLi

Nel caso di entrambi gli approcci 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:

- **Approccio procedurale:** la funzione **mysqli_num_rows ($result)**
- **Approccio Object-Oriented: la proprietà $result>num_rows**

che in entrambi io casi ci **restituisce** il numero di righe contenute dall'identificativo del risultato che le passiamo.

Risultati di una query con MySQLi

Se invece abbiamo eseguito una query di aggiornamento

INSERT, UPDATE, DELETE

e vogliamo sapere quante righe sono state modificate, possiamo usare:

- **Approccio procedurale:** la funzione **mysqli_affected_rows ($result)**
- **Approccio Object-Oriented: la proprietà $result>affected_rows**

Chiudere una connessione con MySQLi

Per la chiusura di una connessione secondo **l'approccio procedurale** , l'estensione MySQLi mette a disposizione la funzione **mysqli_close()** che accetta come parametro il link alla connessione che si desidera chiudere, nel caso in cui sia attiva un'unica connessione non sarà necessario passarne il link alla funzione

mysqli_close ($connessione) **or die** ("
Chiusura connessione fallita " . mysqli_error ($connessione) . " ". mysqli_errno($connessione));

In alternativa, nel caso **dell'approccio OOP**, sarà possibile utilizzare il metodo **mysqli::close()** come istanza della classe di connessione:

$mysqli->close();

Prepared statement con MySQLi

La possibilità di definire **prepared statements** (o, meno comunemente, *parameterized statements*) è uno dei maggiori vantaggi nell'utilizzo dell'estensione MySQLI.

Sono query SQL con struttura fissa ma dati variabili (di input e/o di output)

I **prepared statements** **costituiscono infatti una protezione contro gli attacchi basati su SQL injections**, ciò è una conseguenza del fatto che in tali dichiarazioni l'SQL e i dati rimangono separati; in pratica, anche in presenza di tentativi di **SQL injections** questi non potranno avere successo, in quanto le istruzioni malevole non entreranno a far parte del template e i dati verranno manipolati al di fuori delle istruzioni basate su SQL.

SQL injection è una tecnica di **code injection**, usata per attaccare applicazioni di gestione dati, con la quale vengono inseriti delle stringhe di codice SQL malevole all'interno di campi di input in modo che vengano eseguiti (es. per fare inviare il contenuto del database all'attaccante).

Prepared statement con MySQLi

- Per **formulare una semplice query** finalizzata all'estrazione di dati da una tabella tramite la sintassi di MySQLi, potrebbe essere sufficiente per quanto visto finora un'istruzione come le seguenti :

(approccio procedurale)

$result = **mysqli_query** ($connessione, "SELECT nome, cognome FROM agenda WHERE record_id = 5"));

(approccio Object-Oriented)

$result = $mysqli**->query** ("SELECT nome, cognome FROM agenda WHERE record_id = 5");

N.B. In entrambi i casi non sono stati utilizzati i **prepared statements,** come si può notare infatti la logica dell'SQL non è stata separata dai dati, il valore relativo alla condizione della clausola WHERE non è rappresentato da un **placeholder** e al DBMS non è stato passato il **template** di una dichiarazione ma un'istruzione SQL vera e propria.

Prepared statement con MySQLi

- Volendo parametrare la query proposta in precedenza, **utilizzando l'approccio Object-Oriented**, si potrebbe riformularla in questo modo :

```
// preparazione del template
$pst = $mysqli->prepare ("SELECT cognome, nome FROM agenda WHERE
    record_id = ?");
// definizione della variabile per la sostituzione del placeholder
$pst->bind_param('i', $record_id);
// valorizzazione della variabile per l'esecuzione
$record_id = 5;
$pst->execute();                          // esecuzione dell'istruzione
$pst->bind_result ($cogn, $nom);          //effettuo bind variabili prepared statement
$pst->fetch();                            //prelevo i valori delle variabili dopo il bind
$pst->close();                            //chiusura dell'istruzione
```

Prepared statement con MySQLi

- In pratica, l'esempio appena riportato mostra la fase relativa alla preparazione con la relativa definizione del template, ad essa faranno seguito l'associazione del parametro per la sostituzione del placeholder ("?") e, infine, l'esecuzione dell'istruzione sulla base del modello memorizzato da MySQL.
- Mentre il funzionamento de metodi **prepare()** ed **execute()** dovrebbe risultare sufficientemente intuitivo, **bind_param()** **merita un ulteriore approfondimento**; quest'ultimo infatti si occuperà di effettuare il binding delle variabili accettando due argomenti: **il primo** (nel caso specifico "i") **destinato a definire il tipo di dato associato ad un valore atteso, il secondo** indicante **la variabile a cui tale valore dovrà essere associato**.
- Ecco perché, nell'esempio, il placeholder "?" verrà sostituito dal valore "5" utilizzato per inizializzare la variabile **intera $record_id**.

Prepared statement con MySQLi

L'utilizzo di '**i**' per l'indicazione di un tipo di dato non è il frutto di una scelta arbitraria o di una semplice convenzione, ma di uno schema di corrispondenze previsto a livello sintattico che vede anche:

'**d**' corrispondere a variabili associate al tipo di dato double;

'**s**' a variabili associate al tipo di dato string;

'**b**' corrispondere a blob, cioè a dati in formato binario.

I caratteri per la specifica del tipo di dato dovranno essere ripetuti tante volte quante sono le variabili da valorizzare e ordinati sulla base di queste ultime, come nel caso concreto proposto nella slide successiva

Prepared statement con MySQLi

- Volendo parametrare la query proposta in precedenza, **utilizzando l'approccio procedurale**, si potrebbe riformularla in questo modo :

```
// preparazione del template
$stmt = mysqli_prepare ( $conn, ("SELECT nome, cognome FROM agenda
    WHERE record_id = ?"));
// definizione della variabile per la sostituzione del placeholder
mysqli_stmt_bind_param($stmt, "i", $record_id);
// valorizzazione della variabile per l'esecuzione
$record_id = 5;
mysqli_stmt_execute($stmt);                    // esecuzione dell'istruzione
mysqli_stmt_bind_result($stmt, $cogn, $nom);  //Bind delle variabili
mysqli_stmt_fetch($stmt);    // prelevo i valori delle variabili dopo il bind
mysqli_stmt_close($stmt);                      //chiusura dell'istruzione
```

Prepared statement con MySQLi

Funzione per gli errori ¶

- Stile procedurale

int mysqli_stmt_errno (mysqli_stmt $stmt)

- Stile orientato agli oggetti

int $mysqli_stmt->errno;

Returns the error code for the most recently invoked statement function that can succeed or fail.

Client error message numbers are listed in the MySQL errmsg.h header file, server error message numbers are listed in mysqld_error.h. In the MySQL source distribution you can find a complete list of error messages and error numbers in the file Docs/mysqld_error.txt.

Prepared statement con MySQLi

Funzioni di gestione degli errori

- Stile procedurale

string mysqli_stmt_error (mysqli_stmt $stmt)

- Stile orientato agli oggetti

string $mysqli_stmt->error;

Returns a string containing the error message for the most recently invoked statement function that can succeed or fail.

Ulteriori informazioni con MySQLi

Riepilogo delle funzioni e dei metodi estensione MySQLI

http://php.net/manual/en/mysqli.summary.php

La classe MySQLi

http://php.net/manual/it/class.mysqli.php