

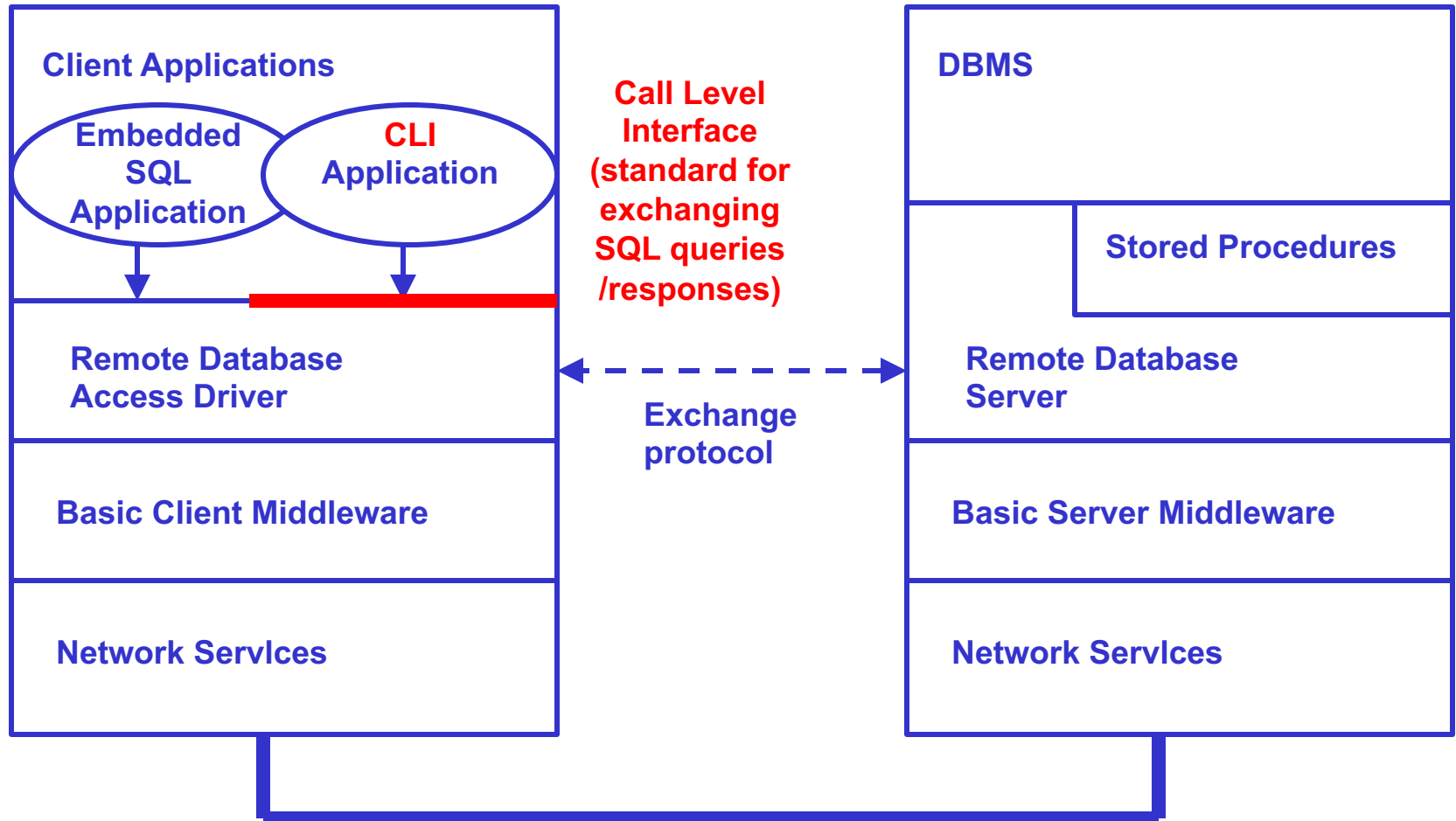
Accesso a basi di dati

- ❑ Tipicamente le soluzioni per database forniscono meccanismi di accesso remoto
- ❑ Obiettivo: dare accesso a dati distribuiti/replicati come se fossero una singola copia locale
- ❑ Problemi da risolvere:
 - ❖ Eterogeneità dei formati di dati e rappresentazioni
 - ❖ Accesso concorrente ai dati
 - ❖ Failures e recupero da crash
 - ❖ Sicurezza

Differenti tipi di servizi

- ❑ Single-site Remote Database Access (SRDA)
 - ❖ Accesso remoto a un singolo database (tipicamente fornito da un middleware proprietario specifico del singolo database)
- ❑ Distributed Query Processing (DQP)
 - ❖ Accesso remoto sola-lettura a database multipli (query su tabelle distribuite su più databases)
- ❑ Distributed Transaction Processing (DTP)
 - ❖ Transazioni distribuite su più database con possibilità di replicazione dei dati

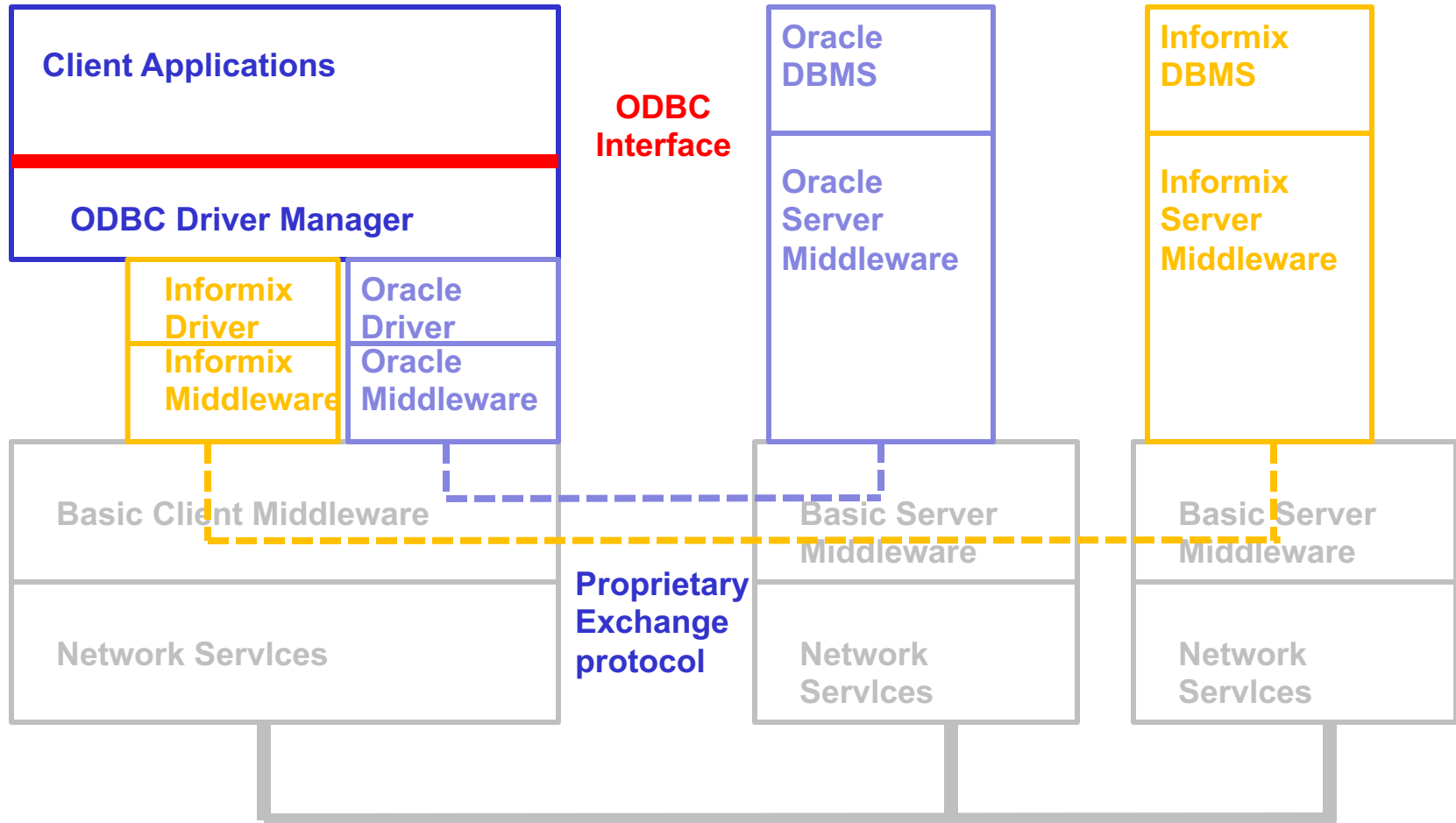
Tipica organizzazione client/server di SRDA SQL middleware



Standardizzazione delle API

- ❑ Gli sviluppatori di database forniscono i loro protocolli proprietari e le loro API
- ❑ La standardizzazione ha condotto a:
 - ❖ APIs aperte (es. ODBC e derivati)
 - ❖ Protocolli aperti (es. RDA)

Example: ODBC



Accesso a basi di dati

- ❑ PHP fornisce accesso sia tramite ODBC, sia tramite librerie specifiche per specifici software
- ❑ Nello specifico, vediamo MySQLI (I = improved)
 - ❖ prodotto molto diffuso, open source
 - ❖ basato sul linguaggio standard SQL
 - ❖ <http://www.mysql.com>
 - ❖ utilizza il protocollo binario per MySQL
 - ❖ Permette di avere transazioni e operazioni pre-compile

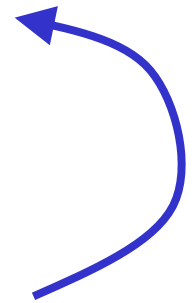
Libreria MySQLI

- ❑ Tecnicamente è un'estensione di PHP
- ❑ Molte funzioni (solo alcune sono presentate qui)
 - ❖ aprire e chiudere la connessione al DB server e selezionare il database;
 - ❖ inviare una istruzione SQL (che può essere una query, la creazione di un record, ecc.)
 - ❖ interpretare il risultato (può essere complesso, es. composto da decine di record con molti campi ciascuno):
 - ❖ operazioni predefinite («prepared statements») e transazioni
- ❑ Interfaccia procedurale o orientata ad oggetti
 - ❖ qui si presenta l'interfaccia procedurale

Uso della libreria mysqli

□ Procedura tipica

- ❖ Connessione al DB server
- ❖ Selezione del database da usare
- ❖ Preparazione della stringa SQL
- ❖ Invio della stringa SQL al DB server
- ❖ Recupero del risultato e
processamento/visualizzazione in pagina web
- ❖ Disconnessione dal DB server



Connessione

- ❑ Prima di eseguire qualsiasi operazione, bisogna stabilire una connessione
- ❑ **mysqli_connect (.....)** parametri (opzionali)
 - ❖ Ritorna un ID di connessione con il DB
 - ❖ **host** ind. IP o nome macchina con DB, se manca è localhost, se preceduto da **p**: è connessione persistente
 - ❖ **username**
 - ❖ **passwd**
 - ❖ **dbname** nome DB default per le operazioni
 - ❖ **port** n. porta per il server MySQL
 - ❖ **socket** socket cliente da utilizzare

Esempio

```
<?php
    $conn = mysqli_connect('localhost',
                           'my_user', 'my_password', 'my_db');

    if (!$conn) {
        die('Errore nella connessione('
            .mysqli_connect_errno().') '
            .mysqli_connect_error());
    }
    echo 'Successo'.mysqli_get_host_info($conn)."\n";
    ...
    mysqli_close($conn);
?>
```

- ❑ NB: Non usare die() o exit() tranne che per prove ed esperimenti, perché interrompe l'esecuzione della pagina (incluso HTML successivo): gestire l'errore in modo più appropriato

Note

- ❑ Si possono aprire anche più connessioni contemporaneamente
- ❑ I risultati di `mysqli_connect_errno()` e `mysqli_connect_error()` si riferiscono all'ultima operazione tentata (interfaccia procedurale)
- ❑ Nell'interfaccia ad oggetti, ogni connessione ha i suoi attributi di errore

Connessioni persistenti e non

- ❑ L'operazione di connessione è alquanto lunga
- ❑ Connessioni persistenti consentono il re-uso (pooling)
- ❑ Prima di essere riusate è necessario:
 - ❖ rollback delle transazioni attive
 - ❖ tabelle temporanee chiuse e cancellate
 - ❖ unlock delle tabelle
 - ❖ reset delle variabili di sessione
 - ❖ statement preparati vengono chiusi
 - ❖ rilascio di lock effettuati con `get_lock`
 - ❖ chiusura handler

Selezionare un DB

- Da eseguirsi dopo la connessione

`bool mysqli_select_db(conn., nome_db)`

`conn.` id. di connessione precedentemente
ottenuto

`nome_db` stringa con il nome del DB

- Può servire per cambiare DB senza aprire una nuova connessione

Esempio di selezione di un DB

```
<?php
    $conn = mysqli_connect("localhost",
        "my_user", "my_password", "test");
    if (mysqli_connect_errno()) {
        echo "Connessione fallita: ".
            mysqli_connect_error();
        exit();
    }
    ..... //operazioni su DB "test"
    mysqli_select_db($conn, "world");
    /*cambio DB su "world"*/
    ..... //operazioni su "world"
    mysqli_close($conn);
?>
```

Interrogazioni SQL (query)

`mixed mysqli_query (conn. , query[, modo])`

- ❖ *conn.* id. conness. precedentemente ottenuto
- ❖ *query* stringa con interrogazione SQL (select, show, describe, explain,.....)
- ❖ *modo*
 - `MYSQLI_USE_RESULT` risultato non in un buffer (grandi dimensioni, dati richiesti ogni volta (es. ogni riga))
 - `MYSQLI_STORE_RESULT` (default) risultato in un buffer
- ❖ Ritorna:
 - `FALSE` (error),
 - object se select, show, describe, explain
 - `TRUE` per altre queries

Esempio 1: con buffer (default)

```
<?php
    $conn = mysqli_connect("localhost","user","pass","dbname");

    if ($ris = mysqli_query($conn,
        "SELECT Name FROM progetti LIMIT 25")) {

        echo "Select ha prod. un num. di righe
            pari a ". mysqli_num_rows($ris));

        //altre operazioni su $ris permesse

        //libera la memoria del buffer
        //dovrebbe essere sempre richiamata quando si termina
        //l'elaborazione del buffer
        mysqli_free_result($ris);
    }
    else die ("Operazione non riuscita");
?>
```


Esempio 2: senza buffer

```
<?php
    $conn = mysqli_connect("localhost","user","pass","dbname");

    if ($ris = mysqli_query($conn,
        "SELECT nome FROM progetti",MYSQLI_USE_RESULT)) {

        //non si possono fare altre op. sulla tabella da PHP
        //perche' i dati sono caricati di volta in volta

        //libera buffer ed accesso al DB: necessaria!
        mysqli_free_result($ris);

        //qui altre operazioni sono possibili sul DB

    }
?>
```

Estrarre dati di una query

- ❑ Mettere in un array la prossima riga (record) di un risultato (NULL se siamo al fondo)

```
array mysqli_fetch_array(risultato,  
                        tipo)
```

risultato ottenuto prec. da una interrog.

tipo tipo di array che rappresenta la singola riga:

MYSQLI ASSOC array associativo (chiavi=colonne)

MYSQLI_NUM array con indici numerici
 (ogni indice corrisponde a una
 colonna della tabella)

MYSQLI **BOTH** entrambi (default)

Esempi di lettura di righe

```
<?php
    ... ..
    // operazioni di apertura e selezione DB
    $query = "SELECT ID, Name FROM progetti WHERE
                citta='Roma' LIMIT 3";
    $ris = mysqli_query($conn,$query);

    $riga = mysqli_fetch_array($ris,MYSQLI_NUM);
    echo "<br>Primo progetto".$riga[0]."    ".$riga[1];

    $riga = mysqli_fetch_array($res,MYSQLI_ASSOC);
    echo "<br>Ancora: ".$riga["ID"].$riga["Name"]);

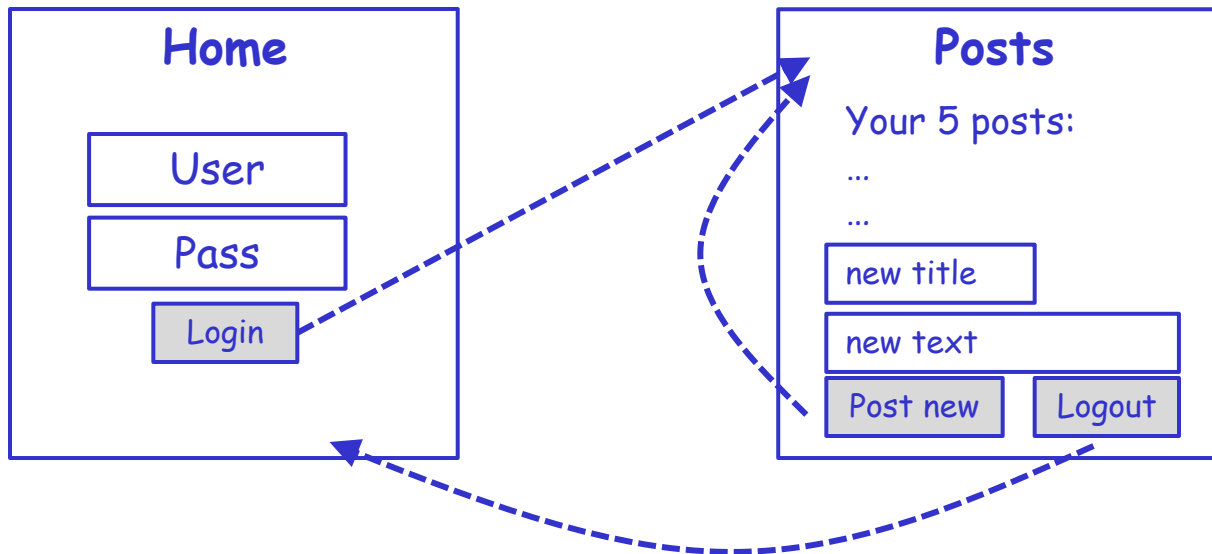
    $riga = mysqli_fetch_array($res);
    // Default: sia NUM sia ASSOC: MYSQLI_BOTH
    echo "<br>Terza volta ".$riga[0], $riga["Name"]);

    $mysqli_free_result($ris);

?>
```

Esempio: blog

- ❑ Si vuole costruire un sito web per un blog
- ❑ Funzionalità di:
 - ❖ Login / Logout
 - ❖ Inserimento post



Blog: database

- Il database utilizzato sarà costituito da due tabelle:
 - ❖ **post(id, data, utente, titolo, testo)**, destinata a memorizzare i post pubblicati sul blog;
 - ❖ **utenti(utente, password)**, tabella di supporto, utilizzata per contenere i dati di autenticazione degli utenti

SQL per creare il DB

```
CREATE TABLE posts (  
    id INTEGER NOT NULL AUTO_INCREMENT PRIMARY KEY,  
    data DATETIME NOT NULL,  
    utente VARCHAR(20),  
    titolo VARCHAR(255),  
    testo TEXT  
);
```

```
CREATE TABLE utenti (  
    utente VARCHAR(20) NOT NULL PRIMARY KEY,  
    password VARCHAR(32)  
);
```

```
INSERT INTO utenti VALUES ('CapitanBlogger',  
    md5('segreta'));
```



Immagazzina l'MD5 della password

Nota: memorizzare password

- ❑ L'ultima istruzione aggiunge un utente alla tabella degli utenti, indicando username e password
- ❑ La password però non viene memorizzata così come è, ma cifrata utilizzando l'algoritmo di hash standard MD5 (o altro algoritmo simile, es. SHA1)
 - ❖ È una funzione in SQL ma esiste anche in PHP
- ❑ In questo modo nel DB non ci sono password in chiaro, ed aumenta la sicurezza perché anche chi ha accesso al DB non può conoscere le password

Nota: password e SQL injection

- ❑ Utilizzare password non in chiaro è una buona pratica di programmazione
- ❑ NON si deve modificare la password immessa dall'utente, perché si rischia di indebolirla
 - ❖ NO strip_tags, NO mysqli_real_escape o altro
 - ❖ Memorizzare l'hash e non la password automaticamente previene problemi di SQL injection (l'unico problema possibile, perché non si visualizza mai la password tramite una risposta lato server)
 - ❖ NO query con MD5 o altro fatto dal DB
 - NO: `INSERT INTO utenti VALUES ($user, MD5($pass));`
 - SI: `$hash=md5($pass); // In php
INSERT INTO utenti VALUES ($user, $hash);`

Nota: password con «sale»

- ❑ Se un utente malintenzionato viene in possesso della lista degli hash di **molte password** (sito con molti utenti) può tentare di indovinarle con un attacco «a dizionario»:
 - ❖ crea l'hash di un insieme di possibili password (dizionario)
 - ❖ confronta l'hash di ognuna con gli hash di tutti gli utenti
- ❑ Per prevenire: quando si memorizza una nuova password, aggiungere una stringa casuale («sale»), di lunghezza prefissata e sufficiente (es. 10 char), e memorizzarla in chiaro, nel DB, oltre all'hash
 - ❖ Il malintenzionato deve rifare l'hash di ogni possibile password per ogni utente perché il sale è diverso
 - ❖ Password uguali non sono riconoscibili dall'hash

Logica di funzionamento

❑ Dobbiamo implementare:

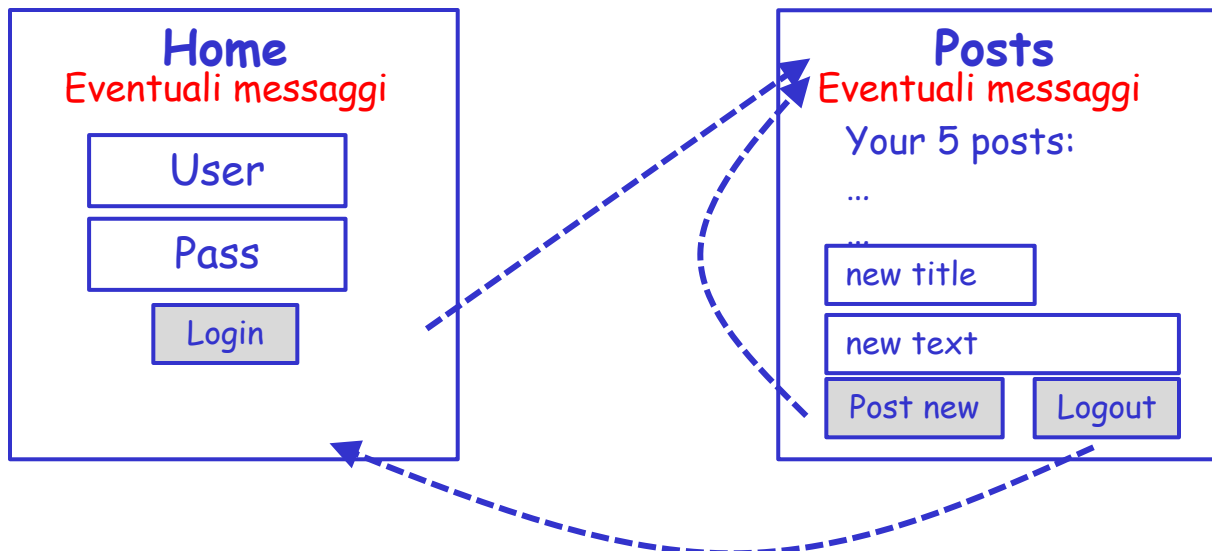
❖ Validazione utente

- Verificato in ogni pagina che richiede autenticazione

❖ Recupero post precedenti (es. numero, contenuto)

❖ Inserimento nuovo post

❖ Uscita (logout)



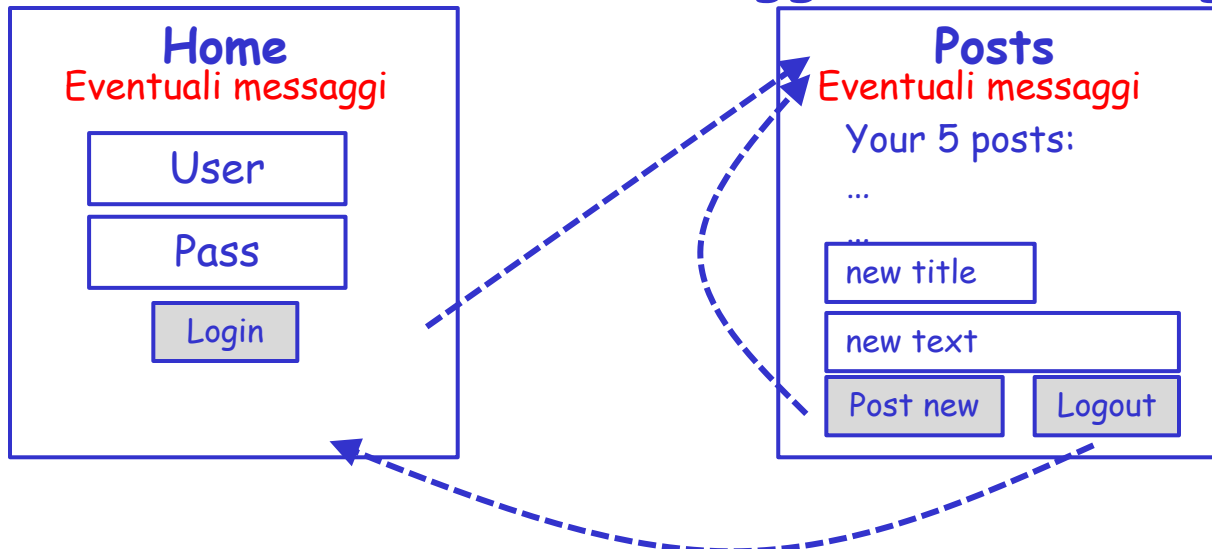
Interfaccia tra le pagine

□ Post:

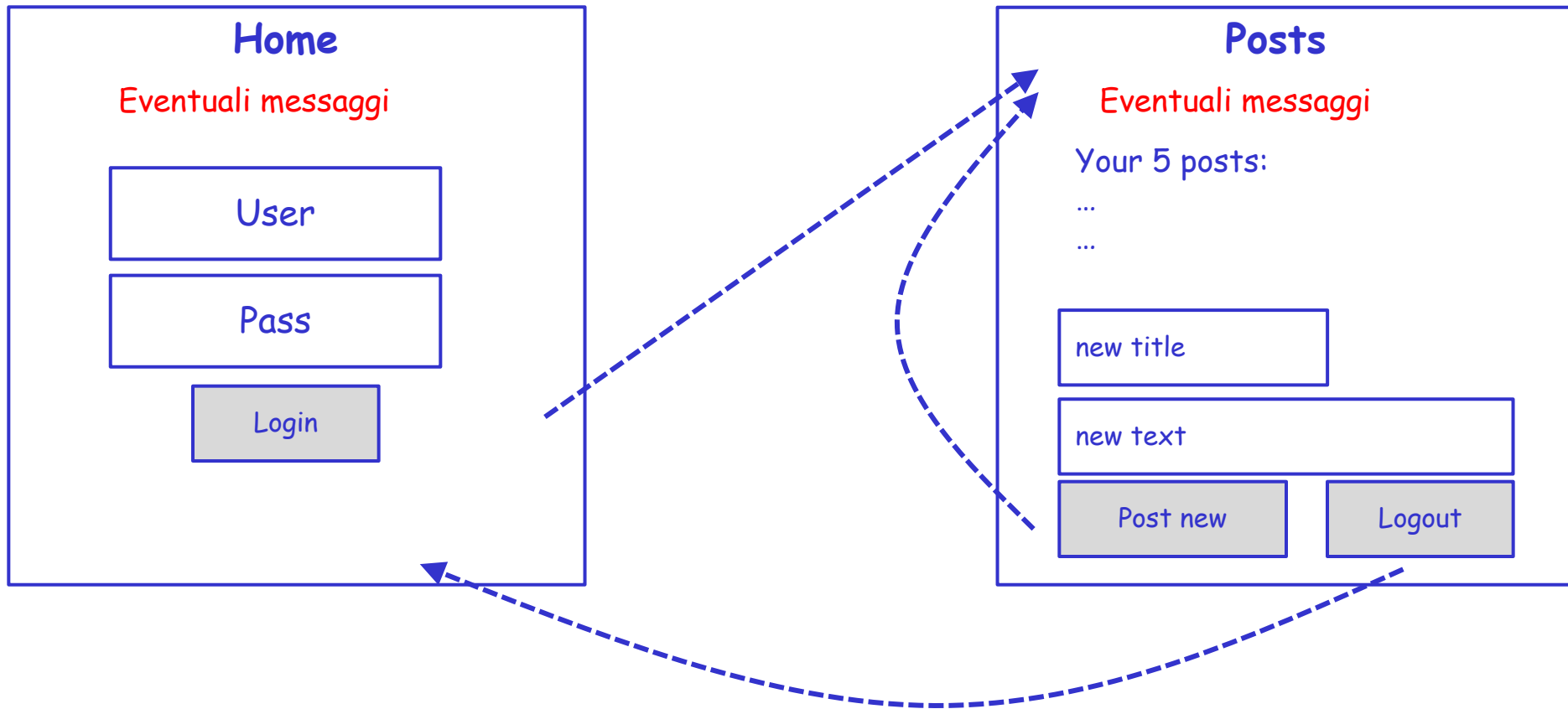
- ❖ Azione da attuare: login, nuovopost, o logout
- ❖ Parametri specifici: user e pass, o titolo e testo
- ❖ ~~Parametri di servizio: messaggio errore~~
 - Non necessario: prodotto al momento

□ Home:

- ❖ Parametri di servizio: messaggio errore (migliorabile)



Interfaccia tra le pagine



Home

```
<html><head>...</head><body>
<div class="message" ...
<?php
    if (isset($_REQUEST['msg'])) {
        $msg = $_REQUEST['msg'];
        // leave only alphanumeric characters to avoid
        // unwanted html and javascript code execution
        $msg = my_sanitize_string($msg);
        echo $msg;
    }
?>
</div> <div>
<form ... action="posts.php" method="post">
<input type="text" ... > ...
</form>
</div>
</body></html>
```

Posts

```
<?php // A inizio file, potrebbe iniziare sessione
    include('myfunctions.php');
    do_action(); ?>
<html><head>...</head><body>
<div class="message" ...
<?php echo $msg; // mia var. globale ?>
</div> <div class="mainpage">
<?php
    echo formatted_posts(0); // from 0 ?>
<form ... action="posts.php" method="post">
<input type="text" name="title" ... >
...
</form>
</div>
</body></html>
```

Funzione do_action()

In myfunctions.php:

```
function do_action() {  
    /* switch depending on $_REQUEST['action'] */  
    login :  
        login(...); // this redirects in case of error  
    logout:  
        my_destroy_session(); redirect su home  
    nuovopost:  
        crea nuovo post // verrà già visualizzato  
    default:  
        nulla; //i posts verranno visualizzati dopo, se  
        //l'utente è stato precedentemente autenticato  
}  
function formatted_posts($da) {  
    return "<div>".leggiPosts($da)."</div>";  
}
```

Funzioni con accesso a DB

□ Dobbiamo implementare:

- ❖ `newPost($titolo, $testo)`
- ❖ `leggiPosts($da, $quanti)`
- ❖ `login($username, $password)`

□ Più funzioni di supporto:

- ❖ `userLoggedIn()`
- ❖ `dbConnect()`
- ❖ `myRedirect()`
- ❖ `myDestroySession()`

userLoggedIn() e myRedirect()

```
function userLoggedIn() {  
    if (isset($_SESSION['myuser'])) {  
        return ($_SESSION['myuser']);  
    } else {  
        return false;  
    }  
}
```

```
function myRedirect($msg="") {  
    header('HTTP/1.1 307 temporary redirect');  
    // L'URL relativo è accettato solo da HTTP/1.1  
    header("Location: home.php?msg=".urlencode($msg));  
    exit; // Necessario per evitare ulteriore  
         // processamento della pagina  
}
```

myDestroySession() e dbConnect()

```
function myDestroySession() {
    $SESSION=array();
    if (ini_get("session.use_cookies")) {
        $params = session_get_cookie_params();
        setcookie(session_name(), '', time()-3600*24,
            $params["path"],$params["domain"],
            $params["secure"], $params["httponly"]);
    }
    session_destroy(); // destroy session
}

function dbConnect() {
    $conn = mysqli_connect("localhost","blog","segreta");
    if(mysqli_connect_error())
        { myRedirect("Errore di collegamento al DB"); }
    return $conn;
}
```

Note

- ❑ Nella connessione possono accadere errori:
 - ❖ Un errore nella connessione al server (a cui passiamo uno username ed una password),
 - Es. server giù, utente non autorizzato o inesistente, utente non autorizzato da quella macchina
 - ❖ Un errore nella selezione del DB
 - DB non esistente, utente non autorizzato ad accedere a quel db

newPost()

```
function newPost($titolo, $testo)
{
    converte \n in <br>
    global $msg;
    if (!userLoggedIn()) { myRedirect(); exit; }
    $conn = dbConnect();
    // Validazione input, contro HTML injection e
    // javascript (elimina tutti i tag, es. <script> o
    // <... onclick="javascript:..." )
    $titolo = nl2br(htmlentities($titolo));
    $testo = nl2br(htmlentities($testo));
    $data = date("Y-m-d H:i:s");

    $sql = "INSERT INTO posts(data, utente, titolo, testo)
    VALUES('".$data."', '".$user."', '".$titolo."', '".$testo."')";
    if(!mysqli_query($sql)) {
        $msg = "Errore nell'inserimento del post, riprovare";
    }
    mysqli_close($conn);
}
```

leggiPosts()

```
function leggiPosts($da, $quanti = 10) {
    global $msg;
    $user = userLoggedIn();
    $conn = dbConnect();
    $risultato = array();
    $da=$da-1;
    $sql="SELECT * FROM posts WHERE user='". $user. "' ORDER BY
        data DESC LIMIT ".$da.", ".$quanti;
    if(! $risposta = mysqli_query($sql))
    {$msg="Errore nella query: ".$sql."<br>".mysqli_error());
        // Solo per debug, rimuovere info in versione finale }
    $risultato="<ul class=...>";
    while ($riga = mysqli_fetch_array($risposta)) {
        $risultato.= <li>$riga['titolo']</li>;
    } ...
    mysqli_close($conn);
    return $risultato;
}
```

login()

```
function login($utente, $password) {  
    $conn = dbConnect();  
    $sql = "SELECT password FROM utenti WHERE utente = '"  
           . $utente . "'";  
    if(! $risposta = mysqli_query($conn,$sql))  
        { myRedirect("Errore di collegamento al DB"); }  
    if (mysqli_num_rows($risposta) == 0) {  
        myDestroySession(); // logout  
        myRedirect("Utente o password errata"); }  
    $riga = mysqli_fetch_array($risposta);  
    /* check password ... */  
    mysqli_close($conn);  
    session_start();  
    $_SESSION['myuser']=$utente;  
}  
cosa succede se la stringa $utente non è quello che ci si aspetta?  
(es. operatori logici ecc.)
```

utenteValido()

```
function login($utente, $password) {
    $conn = dbConnect();
    // Validation
    $utente = mysqli_real_escape_string($conn,$utente);
    $sql = "SELECT password FROM utenti WHERE utente = '"
            . $utente . "'";
    if(! $risposta = mysqli_query($conn,$sql))
        { myRedirect("Errore di collegamento al DB"); }
    if (mysqli_num_rows($risposta) == 0) {
        myDestroySession(); // logout
        myRedirect("Utente o password errata"); }
    $riga = mysqli_fetch_array($risposta);
    mysqli_close($conn);
    session_start();
    $_SESSION['myuser']=$utente;
}
```

SQL Injection

- ❑ Quando si accede al DB tramite query SQL, si deve fare attenzione a come l'input dell'utente si passa alla query SQL

- ❖ C'è il rischio di SQL injection da attaccanti

- ❖ Esempio:

```
$user = $_POST['user'];  
$pass = $_POST['pass'];  
$query = "SELECT * FROM users WHERE user='$user'  
        AND pass='$pass'";  
$result = conn->query($query);
```

- ❖ Se si invia la stringa **admin' #** come username, la query SQL diventa:

```
SELECT * FROM users WHERE user='admin' #' AND ...
```


Prevenire SQL Injection

- ❑ Sanificare l'input usando escaping della stringa SQL, es. facendo escaping di
NULL(0), LF, CR, \, ', " e ^z
- ❑ Si può ottenere chiamando la funzione
- ❑ `string mysqli_real_escape_string(
 connessione, stringa)`
 - ❖ *connessione* riferimento a connessione
 - ❖ *stringa* stringa da trasformare
 - ❖ Ritorna la stringa trasformata
- ❑ Alternativa: usare gli statement pre-compilati, in cui i caratteri sono sempre interpretati per come sono scritti (non esistono delimitatori)
 - ❖ Verificarne la disponibilità prima di scrivere l'app.

Passaggio di parametri

- ❑ Tutto l'input dall'esterno deve essere validato
 - ❖ Evitare l'inserimento di codice HTML o Javascript (es. da link su altre pagine verso la nostra applicazione)
- ❑ Come evitare comportamenti indesiderati dell'applicazione?
 - ❖ Esempio: ricarica di pagina con parametro msg=... visualizza di nuovo il messaggio
- ❑ Possibili soluzioni:
 - ❖ Realizzare un solo template di pagina e riempirlo di conseguenza
 - Si evitano redirect
 - ❖ Mantenere le informazioni nella sessione
 - Svantaggio: richiede mantenere la sessione per tutte le pagine anche se non è necessario (es. home)

Esempio: blog con singola pagina

- ❑ Svolgere il processamento dei parametri e tutte le operazioni all'inizio (prima dell'HTML)

- ❖ Validazione parametri
- ❖ Inizializzazione sessione se necessario, verifica sessione (es. scadenza, ecc.)
- ❖ Accesso a database

- ❑ Comporre la pagina sulla base di un template

- ❖ Es. menu, corpo
- ❖ In ogni elemento richiamare funzioni PHP che forniscono gli elementi desiderati sulla base del precedente processamento



- ❑ Reinviare sempre alla stessa pagina

- ❖ Il contenuto viene deciso all'inizio sulla base dello stato e dell'esito delle operazioni

Eccezioni

- ❑ In PHP5 la gestione degli errori può essere effettuata tramite le eccezioni, similmente ad altri linguaggi (es. C++, Java)
- ❑ Come negli altri linguaggi è utile per snellire il codice di controllo degli errori
- ❑ E' possibile utilizzare la classe **Exception** fornita dal PHP o crearne di proprie, ereditando da questa

Eccezioni: esempio

```
<?php
try {
    if (!mysqli_query($conn,comando1))
        throw new Exception("Query fallita");

    if ($error)
        throw new Exception("Query fallita");

    ...
} catch (Exception $e) {
    $errmsg = e->getMessage();
    ...
    echo $errmsg;
}

?>
```

Transazioni

- ❑ La gestione esplicita delle transazioni è necessaria per transazioni composte da più query
- ❑ Le transazioni garantiscono le proprietà "ACID"
 - ❖ Atomicity - no esecuzione parziale (tutto o niente)
 - ❖ Consistency - vincoli rispettati prima e dopo la transazione
 - ❖ Isolation - esecuzione concorrente equivalente a esecuzione serializzata
 - ❖ Durability - gli effetti della transazione sono persistenti (anche in caso di crash ecc.)
- ❖ Meccanismi di lock sono usati in modo trasparente dal DBMS per ottenere le proprietà ACID (es. Una riga è bloccata ("locked") prima di essere aggiornata)

Transazioni

- ❑ Per default, la connessione è in modalità autocommit (commit automatico dopo ogni statement SQL). Per disabilitare il commit automatico si usa la function:

```
mysqli_autocommit($conn, false);
```

- ❑ Per eseguire manualmente il commit (o il rollback) si usano i comandi:

```
boolean mysqli_commit($conn)
```

```
boolean mysqli_rollback($conn)
```

Per poter essere eseguite devono essere supportate
(altrimenti si genera un errore)

Tipica transazione

```
<?php
try {
    autocommit($conn,false);
    if(!mysqli_query($conn,comando1))
        throw new Exception("comando1 fallito");
    // ... operazioni
    if(!mysqli_query($conn,comando2))
        throw new Exception("comando2 fallito");
    // ... Operazioni
    if(!mysqli_query($conn,comando3))
        throw new Exception("comando3 fallito");

    .....
    if (!mysqli_commit($conn)) {
        // per avere il corretto messaggio di errore
        throw Exception("Commit fallita");
    }
} catch (Exception $e) {
    mysqli_rollback($conn);
    echo "Rollback ".e->getMessage();
    autocommit($conn,true);
    .....
}
autocommit($conn,true);
?>
```


Transazioni e Lock

- ❑ Alcune volte le proprietà ACID (garantite dalle transazioni) non sono sufficienti a risolvere tutti i problemi di concorrenza
- ❑ Esempio: svolgere in maniera atomica lettura, processamento (in PHP) e update
 - ❖ Mettere lettura e update in una transazione non garantisce che i dati letti non siano modificati prima dell'aggiornamento
- ❑ Possibile soluzione
 - ❖ Riunire tutte le operazioni in una singola query (sfruttando l'atomicità dell'esecuzione della query SQL)
 - ❖ Usare **SELECT ... FOR UPDATE** (che forza il lock sui dati letti dalla select)

Transazioni e Lock: quando usare

- ❑ Usare tutte le volte che la logica applicativa lo richiede
- ❑ Diversi utenti possono accedere all'applicazione contemporaneamente
 - ❖ Le applicazioni web sono applicazioni distribuite
 - ❖ Eliminare tutte le «race conditions»
- ❑ Esempi:
 - ❖ Prenotazione di risorse (lettura e aggiornamento)
 - ❖ Svolgimento di più operazioni logicamente unite (es. trasferimento di denaro elettronico, spostamento di oggetto da magazzino a carrello/ordine, ecc.)

Statement preparati

- ❑ Permettono di creare statement parametrizzati da eseguire con parametri attuali ogni volta diversi
- ❑ Bisogna:
 - ❖ inizializzare lo statement
 - ❖ associare una operazione SQL parametrizzata con lo statement
 - ❖ dare un valore al parametro
 - ❖ eseguire l'operazione
 - ❖ associare il risultato a delle variabili
 - ❖ caricare (fetch) del risultato
 - ❖ eventuale chiusura dello statement
- ❑ NB: Potrebbero non essere disponibili in certe implementazioni

Funzioni per statement (1)

- ❑ `mysqli_stmt_init(connessione)`
 - ❖ restituisce un riferimento ad uno *statement*
- ❑ `mysqli_stmt_prepare(statement, SQL)`
 - ❖ *SQL* contiene una unica operazione SQL con parametri indicati da ?
 - ❖ restituisce un risultato booleano
- ❑ `mysqli_stmt_bind_param(statement, tipo, var1[, var2[, var3...]])`
 - ❖ associa le variabili date ai param. dello statement
 - ❖ *tipo* indica il tipo delle variabili "s" stringa, "i" intere, "b" blob e "d" double . Es: "sssi" = 3 stringhe e 1 intero
 - ❖ restituisce un booleano

Funzioni per statement (2)

- ❑ `mysqli_stmt_execute(statement)`
 - ❖ esegue lo statement
- ❑ `mysqli_stmt_bind_result(var1[, var2[, var3...]])`
 - ❖ associa delle variabili alle colonne del risultato
 - ❖ restituisce un valore booleano
- ❑ `mysqli_stmt_fetch(statement)`
 - ❖ carica 1 nuova riga del risultato nelle var. scelte
 - ❖ restituisce NULL se non vi sono più righe
 - ❖ restituisce FALSE se vi è stato un errore

Funzioni per statement (3)

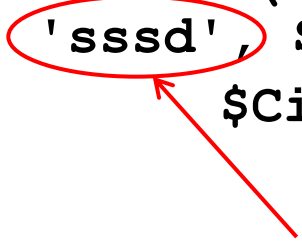
- ❑ `mysqli_stmt_store_result(statement)`
 - ❖ memorizza in un buffer il risultato di uno stmt
- ❑ `mysqli_stmt_free_result(statement)`
 - ❖ libera il buffer con i risultati dello statement
- ❑ `mysqli_stmt_close(statement)`
 - ❖ chiude (distrugge) lo statement

Esempio d'uso di prepared stmt

```
<?php
...// apertura connessione, ecc.
$stmt = mysqli_prepare($conn, "INSERT INTO Projects
VALUES (?, ?, ?, ?)");
mysqli_stmt_bind_param($stmt, 'sssd', $ID, $Name,
$City, $Value);

$ID = 'J33';
$Name = 'Aircraft';
$City = "Athens";
$Value = 11.2;
// esegue il prepared statement
mysqli_stmt_execute($stmt);
echo "Row inserted.\n";
echo mysqli_stmt_affected_rows($stmt);

// chiude lo statement
mysqli_stmt_close($stmt);
?>
```



Tipi dei dati

Esempio 2: con bind_result

```
<?php
... // apertura connessione
$query = "SELECT Name, City FROM Projects ORDER by
                                                ID DESC LIMIT 150,5";
if ($stmt = mysqli_prepare($conn, $query)) {
    mysqli_stmt_execute($stmt); // esegue lo statement
    mysqli_stmt_bind_result($stmt, $name, $city);
    while (mysqli_stmt_fetch($stmt)) {
        // Fetch di una riga alla volta
        echo $name." ".$city."<br>\n";
    }
    // chiude lo statement
    mysqli_stmt_close($stmt);
}
else {die('Errore nello statement preparato');}
// chiude la connessione
mysqli_close($conn);
?>
```


Esempio 3: con store_result

```
<?php
...// apertura connessione
$query = "SELECT Name, City FROM Projects ORDER by
                                                ID DESC LIMIT 150,5";
if ($stmt = mysqli_prepare($conn, $query)) {
    mysqli_stmt_execute($stmt); /*esegue lo
                                statement*/

    mysqli_stmt_store_result($stmt);
    mysqli_stmt_bind_result($stmt, $name, $city);
    while (mysqli_stmt_fetch($stmt)) {
        echo $name." ".$city."<br>\n";
    }
    mysqli_stmt_free_result($stmt);
    /* chiude lo statement */
    mysqli_stmt_close($stmt);
}
else {die("errore nello statement preparato");}
/* chiude la connessione */
mysqli_close($conn);
?>
```

Osservazioni

- ❑ I parametri non possono apparire ovunque nell'operazione SQL
 - ❖ non possono essere nomi di colonne
 - ❖ gli operandi di un confronto non possono essere entrambi parametri
 - ❖ ...
- ❑ I risultati memorizzati in un buffer non possono cambiare durante la lettura
- ❑ Si può liberare un buffer ed eseguire nuovamente lo statement

Oggetti in PHP

- In PHP si possono definire delle classi

```
class contatto {  
    protected $nome, $tel, $note;  
    public function __construct($nome) metodo  
costruttore  
    { this->nome=$nome; }  
    public function set_tel($tel)  
    { this->tel=$tel; }  
    public function set_note($note)  
    { this->note=$note; }  
    public function stampa_contatto()  
    { $out="Nome: ".this->nome."<br>Tel.: ".  
        this->tel."<br>Note: ".this->note;  
        return $out; }  
}
```

Visibilità

- ❑ Ogni proprietà di una classe può essere preceduta da:
 - ❖ **public** indica che la proprietà è accessibile da tutti
 - ❖ **private** indica che la proprietà è accessibile solo dal codice della stessa classe in cui è definita
 - ❖ **protected** indica che la proprietà è accessibile solo dalla stessa classe in cui è definita, da quelle predecessori e da quelle derivate
- ❑ Se si usa **var** la variabile è pubblica

Creare oggetti

- ❑ Si possono creare istanze di una classe già definita
- ❑ Quando si crea l'istanza, viene invocato il metodo costruttore
`$contact=new contatto("Mario");`
- ❑ Il costruttore può anche avere lo stesso nome della classe (deprecato)
- ❑ Non è obbligatorio avere un costruttore
- ❑ Può esistere un metodo distruttore, chiamato `__destruct()`, invocato dal garbage collector

Ereditarietà

```
class contatto_lavoro extends contatto
{
    protected $azienda, $ruolo;
    public function __construct($a,$b)
    {
        contatto::__construct($a);
        $this->ruolo=$b;
    }
    public function set_az($azienda)
    {
        $this->azienda=$azienda;
    }
    public function stampa_contatto()
    {
        $out="Nome: ".$this->nome."<br>Tel.: ".
            $this->tel."<br>Note: ".$this->note.
            "<br>Azienda: ".$this->azienda.
            "<br>Ruolo: ".$this->ruolo";
        return $out;
    }
}
```

nome classe

Esempio di utilizzo

```
$a=new contatto("Mario");  
$b=new contatto_lavoro("Giovanni",  
                        "Software Engineer");
```

```
$a->set_tel("+39011543789");
```

```
$b->set_az("ACME Gadgets");
```

```
echo $a->stampa_contatto();
```

```
echo $b->stampa_contatto();
```

Chiamate di
metodi

```
$b->fax="+390113874690";
```

proprietà definita
"al volo": è possibile!

Altre caratteristiche

- ❑ La classe deve essere definita in un solo blocco `<?php.....?>`
- ❑ Si può ereditare da **una sola classe**
- ❑ Si può sfruttare nel costruttore la possibilità di chiamata con numero di parametri variabile
- ❑ Si possono definire **classi astratte** precedute da **abstract**
 - ❖ contengono metodi di cui si dà solo il prototipo
 - ❖ la definizione del metodo è lasciata alle classi derivate
 - ❖ metodi privati non possono essere astratti

Interfacce

- E' una classe completamente astratta

 - ❖ Keyword **interface**

```
interface Storable {function getContentsAsText();}
```

- Le interfacce sono utili per separare completamente l'interfaccia dall'implementazione

Esempio

□ Simile a molti linguaggi ad oggetti

```
class Document implements Storable {
    public function getContentsAsText() {
        return "This is Text of the Document\n";
    }
}

class Indexer {
    public function readAndIndex(Storable $s) {
        $textData = $s->getContentsAsText();
        echo $textData;
    }
}

$p = new Document();
$i = new Indexer();
$i->readAndIndex($p);
```

Classi astratte vs interfacce

Classe astratta

- ❑ Può definire alcune funzioni e lasciarne altre alle classi derivate
- ❑ Le classi derivate possono o meno ridefinire i metodi
- ❑ Ci deve essere una relazione logica fra la classe base e quelle derivate

Interfaccia

- ❑ Non può contenere elementi procedurali
- ❑ La classe che implementa l'interfaccia **deve** definire i metodi relativi
- ❑ Le classi che implementano la stessa interfaccia possono essere scorrelate

Serializzazione

- ❑ Si può serializzare il contenuto di un oggetto (es. per salvarlo su un file)

```
$a=new contatto("Mario");  
$a->set_tel("+39011543789");  
$a->set_note("Da conservare");  
$str=serialize($a);  
/* $str viene scritto in un file */  
/* si rilegge dal file in $str */  
$b=unserialize($str); //ripristino
```

Magic functions

- ❑ Funzioni che quando ridefinite in una classe hanno un significato particolare. Il loro nome inizia per `__`
- ❑ Esempi:
 - ❖ `__clone()` definisce cosa fare quando si vuole duplicare un oggetto
 - ❖ `__get`, `__set` definiscono cosa fare (es. controlli) quando si dà un valore (anche "al volo") ad una variabile dell'oggetto o la si legge
 - ❖ `__call` definisce cosa fare quando si invoca un metodo definito "al volo"
 - ❖ `__sleep`, `__wakeup` definiscono cosa salvare di un oggetto che viene serializzato, e come ripristinarlo
 - ❖ `__isset`, `__unset` invocate quando si applica `isset` o `unset` ad una proprietà dell'oggetto
 - ❖

Esempio: accesso a MySQL DB con API ad oggetti

□ Classi principali:

- ❖ **mysqli** classe che rappresenta una connessione a MySQL, contiene tutti i metodi applicabili alle connessioni
- ❖ **mysqli_stmt** classe che definisce uno statement preparato
- ❖ **mysqli_result** classe che rappresenta un result set di una operazione su DB
- ❖

□ Per eseguire una operazione, si crea un oggetto oppure si invoca un metodo

Esempio

```
// apre e controlla connessione
$conn=new mysqli("localhost",$user,$passwd,$db) ;
if (mysqli_connect_errno()) {
    die("Connect failed: <br>".
        mysqli_connect_error())
}

$stmt = $conn->prepare("INSERT INTO Projects
                        VALUES (?, ?, ?, ?)");

$ID = 'J33';
$Name = 'Aircraft';
$City = "Athens";
$Value = 11.2;
```

Esempio (2)

```
$stmt->bind_param('sssd', $ID, $Name, $City, $Value) ;  
// esegue il prepared statement  
$stmt->execute() ;  
echo "Row inserted.<br>" ;  
echo $stmt->affected_rows() ;  
  
// chiude lo statement  
$stmt->close() ;  
  
//chiude la connessione  
$conn->close() ;
```


Altro esempio (1)

```
$conn = new mysqli("localhost", $user, $passwd, "world");
if (mysqli_connect_errno()) {
    die("Connect failed: ".mysqli_connect_error());
}

$query = "SELECT Name, City FROM Projects
          ORDER by ID LIMIT 3";

if (!$result = $conn->query($query)) {
    die("Errore nell'interrogazione" . $query."<br>");
}

// lettura in array numerico
if ($riga = $result->fetch_array(MYSQLI_NUM)) {
    echo $riga[0]."    ".$riga[1]."<br>\n";
}
```

Altro esempio (2)

```
// lettura in array associativo
if($riga = $result->fetch_array(MYSQLI_ASSOC)) {
    echo $row["Name"]." ". $row["City"]."<br>\n";
}

// lettura in array associativo e numerico
if($riga = $result->fetch_array(MYSQLI_BOTH)) {
    echo $riga[0]." ".$riga["City"]."<br>\n";
}

// libera il result set, equiv. di free_result
$result->close();

// chiude la connessione */
$conn->close();
```