MySQL & PHP

Davide Spano

Università di Cagliari

davide.spano@unica.it

Corso di Amministrazione di Sistema

Introduzione

- Nelle precedenti lezioni abbiamo visto come
  - Organizzare i dati
  - Inserirli
  - Leggerli
  - Cancellarli
- Tutto ciò l'abbiamo fatto con delle query SQL
- L'oggetto di queste lezioni è vedere come sia possibile fare le stesse query utilizzando il PHP
- E quindi manipolare il database a partire dalla nostra applicazione

Interazione fra PHP e MySQL

- Come abbiamo visto nelle precedenti lezioni, per accedere ai dati del database abbiamo bisogno di un client
- Questo client si connette al database server
- Da lì possiamo inviare delle query al database
- Che ci restituisce i risultati
- Il processo è lo stesso anche con il PHP
- È bene creare un utente sul db che rappresenti l'applicazione che stiamo sviluppando (o condiviso da più applicazioni)

Interazione fra PHP e MySQL (2)

1. Connessione a MySQL
2. Selezionare il database da utilizzare
3. Creare una query SQL
4. Eseguire la query
5. Ricevere i risultati e utilizzarli
6. Ripetere i passi da 3 a 5 finché non si è soddisfatti del risultato
7. Chiudere la connessione a MySQL

Funzioni di libreria

- Per la connessione al database server, il PHP mette a disposizione una libreria
- Essenzialmente, si tratta di una classe che ha metodi per interagire con il database
- In modo da non doverci preoccupare di effettuare i passi con del codice scritto da noi
- La libreria è un'estensione standard del PHP e si chiama `mysqli`
  - Che sta per MySQL Improved extension
- http://php.net/manual/it/book.mysqli.php

Cosa contiene la libreria

- La libreria mette a disposizione una singola classe `mysqli`
- I metodi di questa classe ci permettono di
  - Connetterci a MySQL
  - Inviare query
  - Ricevere i risultati
  - Ottenere informazioni su eventuali errori
  - Chiudere la connessione a MySQL
- Nelle parti successive di queste lezioni ne vedremo l'utilizzo

Connessione al database

- Il primo passo da effettuare è quello di connettersi al database server
- Come abbiamo visto per gli altri client, per connettersi correttamente al database è necessario conoscere
  - La macchina sul quale questo server si trova
  - Il nome del database
  - Username e password di un utente che abbia il diritto di accedere al database
  - L'accesso deve essere disponibile almeno in lettura
- La classe `mysqli` ci offre dei metodi per farlo
- Vediamo come

Connessione al database: passo 1

- Il primo passo da fare è quello di creare un oggetto della classe `mysqli`
- In modo da poterne invocare i metodi
- Una volta istanziato l'oggetto possiamo effettuare tutte le operazione di cui parleremo di seguito
- ```
  // Istanziare la classe mysqli
  $mysqli = new mysqli();
  ```
  ```

Connessione al database: passo 2

- Una volta istanziata la classe, possiamo connetterci al database
- La classe `mysqli` ha un metodo `connect`, con i seguenti parametri:
  - Nome dell'host dove si trova il dabase (stringa)
  - Username dell'utente utilizzato per accedere ai dati (stringa)
  - Password dell'utente utilizzato per accedere ai dati (stringa)
  - Nome del database da utilizzare per le query
- Una volta invocato questo metodo, l'istanza della classe `mysqli` è un client connesso al database
  - Simile alla riga di comando che abbiamo utilizzato le volte precedenti
- ```
  // Connessione al database di esAMMi esempio
  $mysqli->connect("localhost",  "davide",
  "password",  "esammi");
  ```
  ```

Chiusura di una connessione

- Una volta che abbiamo effettuato le nostre operazioni sul database, è necessario chiudere la connessione
- Tenete presente che il numero di possibili connessioni al database è limitato
- Dunque è importante chiudere la connessione non appena si è finito di accedere o modificare i dati
- La chiusura di una connessione si effettua con un altro metodo della classe `mysqli`
-
```
// Chiusura della connessione
$mysqli->close();
```

Gestione errori

- La connessione, ed in generale tutte le operazioni che si fanno con il database possono fallire
- Ovviamente, se non riuscite nella connessione, è difficile che qualcosa nella vostra applicazione funzioni
- È molto importante gestire questo tipo di errori per evitare che l'applicazione si comporti in modo errato
- Segnalare agli utenti che non è possibile fare qualcosa in modo gradevole
- La classe `mysqli` ci offre dei meccanismi per renderci conto della situazione

Gestione errori (2)

- La classe `mysqli` ha una variabile d'istanza `errno`
  - La variabile è 0 in caso non vi sia nessun errore
  - Altrimenti contiene un codice numerico che identifica l'errore
- Inoltre ne ha anche un'altra dedicata solo per gli errori di connessione: `connect_errno`
- Questi codici numerici è sono molto utili per capire cosa sta succedendo
  - Si può cercare nella documentazione MySQL
  - Molto spesso si trovano online delle soluzioni note al problema
- Es. `mysqli->connect_errno` vale 1045 dopo la connect
  - La connessione non è stata effettuata
  - Cerchiamo l'errore: Access denied for user 'root'@'localhost'
  - Una soluzione nota è (incredibile ma vero) provare un'altra password…

Gestione errori (3)

- Il messaggio di errore si può leggere con la variabile `error`
  - Che restituisce una stringa con la descrizione dell'errore più recente
  - Ricordatevi infatti che si può usare lo stesso oggetto per più operazioni `mysqli` con il database
- Anche in questo caso quelle relative alla connessione stanno su una variabile dedicata: `connect_error`
- Molti lo utilizzano per stamparlo semplicemente sulla pagina
  - Questo approccio è buono per il programmatore, ma non è molto rassicurante per l'utente
- Solitamente alla vista di un messaggio del tipo `ERROR 1045 (28000): Access denied for user 'root'@'localhost' (using password: YES)` l'utente
  - Si spaventa e teme di aver rotto qualcosa
  - Si adira e giura di non utilizzare più la vostra applicazione
- Si consiglia quindi di utilizzare messaggi comprensibili e professionali

Gestione errori (4)

```php
// creo l'istanza della classe mysqli
$mysqli = new mysqli();
// connessione al database
$mysqli->connect("localhost", "davide",
                 "pippo", "prova");
// verifico la presenza di errori
if($mysqli->connect_errno != 0){
   // gestione errore
   $idErrore = $mysqli->connect_errno;
   $msg = $mysqli->connect_error;
   error_log("Errore nella connessione al server
              $idErrore : $msg", 0);
   echo "Errore nella connessione $msg";
}else {
   // nessun errore
   echo "Tutto ok";
}
```

Informazioni sulla connessione

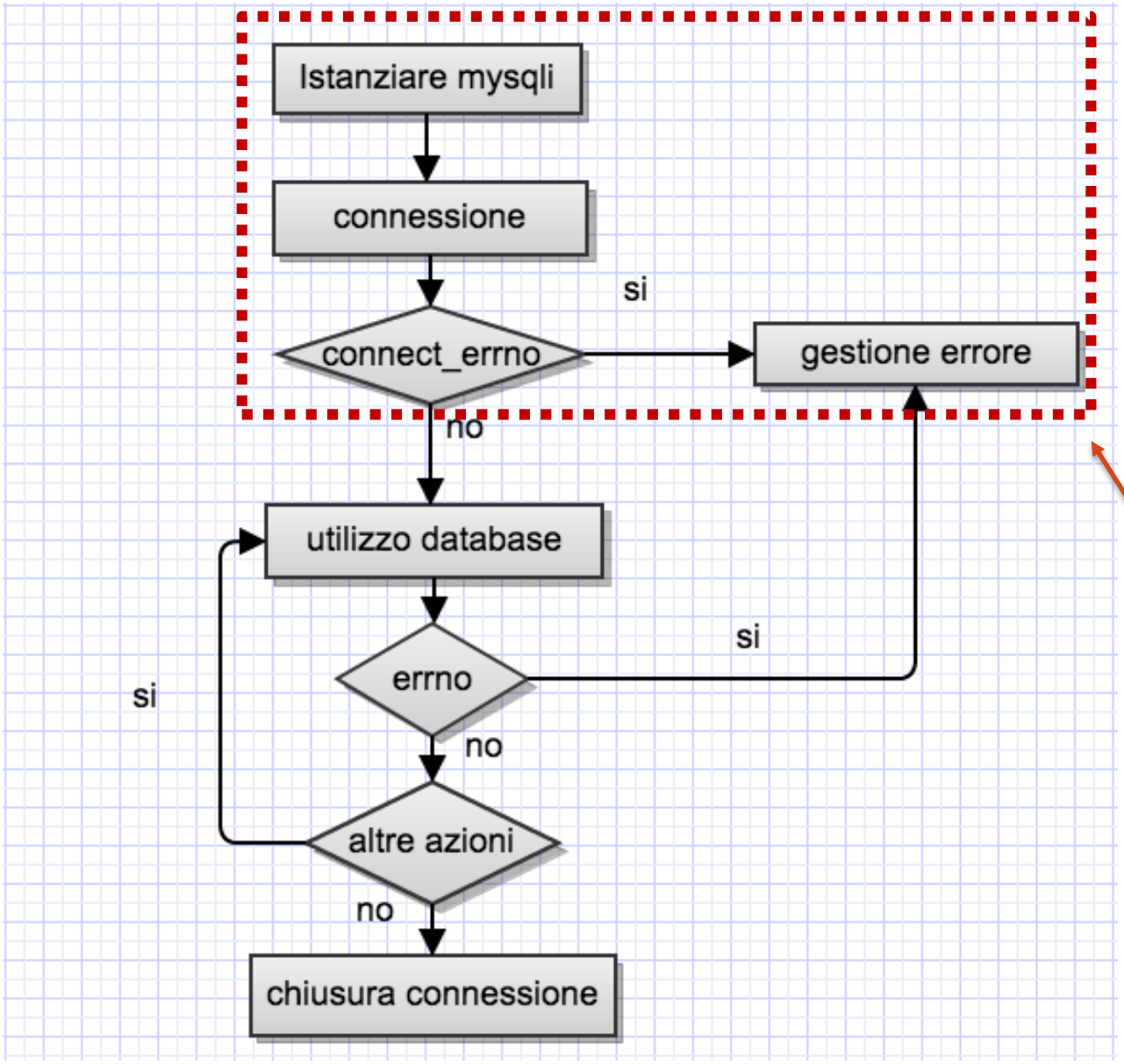
- Le informazioni sulla connessione sono necessarie in ogni punto del codice dove serve accedere al database
- All'interno della applicazione ciò vuol dire che è necessario praticamente in ogni classe del modello
- Ma allora bisogna inserire le informazioni di connessione in ogni classe?
- Ovviamente no, è uno dei casi in cui bisogna utilizzare delle variabili di livello ***applicazione***
- Il modo più facile per implementarle è usare delle variabili o metodi statici

Informazioni sulla connessione (2)

```
class Settings {
    public static $db_host = 'localhost';
    public static $db_user = 'root';
    public static $db_password = 'davide';
    public static $db_name = 'prova';
}
//--------- Classe che usa il db --------------
// creo l'istanza della classe mysqli
$mysqli = new mysqli();

// connessione al database
$mysqli->connect(Settings::$db_host,
            Settings::$db_user,
            Settings::$db_password,
            Settings::$db_name);
```

Schema di accesso al database



Si può isolare in una classe unica

Accesso ai dati

- Per l'accesso ai dati contenuti nel db la classe `mysqli` ci mette a disposizione il metodo `query`
- Il metodo accetta un parametro di tipo stringa che ci permette di inserire la query che vogliamo effettuare sul database
- Inoltre, il metodo restituisce un oggetto di tipo `mysqli_result`, che consente di
  - Leggere i risultati della query effettuata
  - Ottenere informazioni sul numero di righe coinvolte dalla query
- Come detto in precedenza, prima di leggere i risultati della query, è importante verificare l'assenza di errori

Accesso ai dati (1)

```php
// suppongo di aver creato mysqli e di aver chiamato la connect
if($mysqli->connect_errno != 0){
    // gestione errore
    $idErrore = $mysqli->connect_errno;
    $msg = $mysqli->connect_error;
    error_log("Errore nella connessione al server $idErrore : $msg", 0);
    echo "Errore nella connessione $msg";
}else {
    // nessun errore
    $query = "SELECT id, matricola, nome, cognome FROM studenti";
    $result = $mysqli->query($query);
    if($mysqli->errno > 0){
        // errore nella esecuzione della query (es. sintassi)
        error_log("Errore nella esecuzione della query
                    $mysqli->errno : $mysqli->error", 0);
    }else {
        // query eseguita correttamente
        echo "<ul>\n";
        while($row = $result->fetch_row()){
            echo "<li> ($row[0]) $row[1] $row[3] $row[2] </li>\n";
        }
        echo "</ul>\n";
    }
}
```

- (1) 253662 Spano Davide
- (2) 123456 Pallino Pinco
- (3) 654321 Rossi Mario

Accesso ai dati (2)

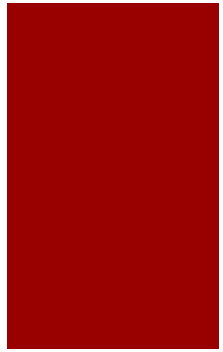
- Oltre ad accedere ai dati con un array indicizzato per posizione è possibile farlo
  - Con una sintassi orientata agli oggetti (cioè con i campi che sono delle variabili d'istanza
  - Con un array associativo
  - Con due differenti metodi della classe `mysqli_result`
- Metodo per ottenere oggetti: `fetch_object()`
- Metodo per ottenere array associativi: `fetch_array()`
- Entrambi i metodi restituiscono **una riga alla volta!**

Accesso ai dati (oggetti)

```
// ... tutti controlli fatti in precedenza

// nessun errore
    $query = "SELECT id, matricola, nome, cognome FROM studenti";
    $result = $mysqli->query($query);
    if($mysqli->errno > 0){
        // errore nella esecuzione della query (es. sintassi)
        error_log("Errore nella esecuzione della query
                    $mysqli->errno : $mysqli->error", 0);
    }else {
        // query eseguita correttamente
        echo "<ul>\n";
        while($row = $result->fetch_object()){
            echo "<li> ($row->id) $row->matricola
                        $row->cognome $row->nome </li>\n";
        }
        echo "</ul>\n";
    }
}
```

Accesso ai dati (array)



```php
// ... tutti controlli fatti in precedenza

// nessun errore
$query = "SELECT id, matricola, nome, cognome FROM studenti";
$result = $mysqli->query($query);
if($mysqli->errno > 0){
    // errore nella esecuzione della query (es. sintassi)
    error_log("Errore nella esecuzione della query
  $mysqli->errno : $mysqli->error", 0);
}else {
    // query eseguita correttamente
    echo "<ul>\n";
    while($row = $result->fetch_array()){
        echo "<li> (";
        echo $row['id'].")";
        echo $row['matricola'].' ';
        echo $row['cognome'].' ';
        echo $row['nome'];
        echo "</li>\n";
    }
    echo "</ul>\n";
}
```

Determinare il numero di righe

- In una query di tipo **SELECT**, spesso è necessario stabilire quante siano le righe che sono state restituite
- La classe `mysqli_result` mette a disposizione la variabile `num_rows` d'istanza per accedere a questa informazione
- In modo simmetrico, è importante anche stabilire quante righe sono state coinvolte in query di tipo **INSERT, UPDATE, DELETE**
- In questo caso si utilizza la variabile d'istanza `affected_rows`
- Vediamo due esempi

Numero di righe: SELECT

```php
// suppongo di aver creato mysqli e di aver chiamato la connect
if($mysqli->connect_errno != 0){
    // gestione errore
    error_log("Errore nella connessione al server
                $mysqli->connect_errno : $mysqli->connect_error", 0);
    echo "Errore nella connessione $mysqli->connect_error";
}else {
    // nessun errore di connessione
    $query = "SELECT id, matricola, nome, cognome FROM studenti";
    $result = $mysqli->query($query);
    if($mysqli->errno > 0){
        // errore nella esecuzione della query (es. sintassi)
        error_log("Errore nella esecuzione della query
                    $mysqli->errno : $mysqli->error", 0);
    }else {
        // query eseguita correttamente
        echo "La query ha restituito $result->num_rows righe";
    }
}
```

La query ha restituito 3 righe

Numero righe: UPDATE

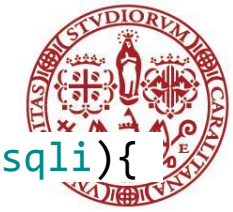
```php
// suppongo di aver creato mysqli e di aver chiamato la connect
if($mysqli->connect_errno != 0){
    // gestione errore
    error_log("Errore nella connessione al server
                $mysqli->connect_errno : $mysqli->connect_error", 0);
    echo "Errore nella connessione $mysqli->connect_error";
}else {
    // nessun errore di connessione
    $query = "UPDATE studenti SET matricola = 253662 where id = 1";
    $result = $mysqli->query($query);
    if($mysqli->errno > 0){
        // errore nella esecuzione della query (es. sintassi)
        error_log("Errore nella esecuzione della query
                    $mysqli->errno : $mysqli->error", 0);
    }else {
        // query eseguita correttamente
        echo "La query ha modificato $mysqli->affected_rows righe";
    }
}
```

La query ha modificato 1 righe

Query con parametri

- Di solito le query che facciamo sul database hanno uno schema ben preciso
- Sappiamo su quali tabelle dobbiamo cercare
- Sappiamo quali colonne dobbiamo selezionare
- Ma il contenuto delle colonne che vogliamo selezionare di solito è variabile
- In soldoni, abbiamo delle clausole WHERE parametriche…
- Ma noi sappiamo come inserire delle variabili nelle stringhe!
- `"SELECT id, matricola, nome, cognome FROM studenti WHERE matricola = $myMatricola"`

Query con parametri: esempio



```
public static function cercaStudente($stud_nome, $stud_cognome, $mysqli){
    $query = "SELECT id, matricola, nome, cognome FROM
              studenti WHERE nome='$stud_nome' and
              cognome='$stud_cognome'";
    $result = $mysqli->query($query);
    if($mysqli->errno > 0){
        // errore nella esecuzione della query (es. sintassi)
        error_log("Errore nella esecuzione della query
        $mysqli->errno : $mysqli->error", 0);
    }else {
        // query eseguita correttamente
        echo "<ul>\n";
        while($row = $result->fetch_object()){
            echo "<li> ($row->id) $row->matricola
                       $row->cognome $row->nome </li>\n";
        }
        echo "</ul>\n";
    }
}
```

Query e input utente

- È molto probabile che le variabili introdotte nelle query provengano dall'input utente
- Questo può creare problemi?
  - In teoria è proprio quello che vorremmo per applicazioni web
  - Ma può essere un'opportunità per qualche malintenzionato
- Perché?
  - Ricordatevi come stiamo inserendo i valori di queste variabili
  - Con una concatenazione di stringhe
  - Chi conosce questi gli strumenti può cercare di sfruttarli
  - Vediamo come

SQL injection

- Supponiamo di autenticare l'utente se questa query ci restituisce una riga. L'utente sarà quello specificato nella colonna user.

- ```
  $user = $_POST['user'];
  $pass = $_POST['pass'];
  $query = "SELECT * FROM users WHERE
  user='$user' AND pass='$pass'";
  ```

- Supponiamo che
  - `$_POST['user']="davide";`
  - `$_POST['pass']="password";`
  - Tutto ok.

- Ma se
  - `$_POST['user']="admin' #";`
  - `$_POST['pass']="";`
  - Che succede ?
  ```

SQL injection (2)

- Sostituiamo i valori delle variabili nella query

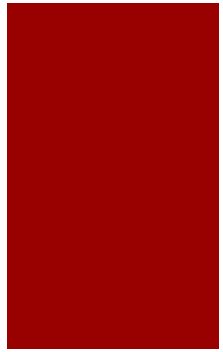
```
SELECT * FROM users WHERE user='admin' #' and pass='
                          $_POST['user']  $_POST['pass']
```

- Ha cambiato totalmente la query
  - Ora seleziona solo l'utente con username admin
  - Che molto probabilmente è l'amministratore del sito
- E siccome la verifica della password è commentata, **lo autentica sempre**
- Il malintenzionato ora amministra il vostro sito!
  - L'utilizzo di altri comandi può anche danneggiare il database
- Inserire stringhe che possono modificare la query del database usando sintassi SQL è un attacco noto
  - **SQL injection**

Come contrastare la SQL injection

- Come si può contrastare questo tipo di attacco?
- **Prepared statements**
  - Sono uno speciale oggetto che rappresenta una query SQL
  - Le parti parametriche vengono specificate esplicitamente
  - La sintassi viene validata **prima di eseguire la query con i valori veri dei parametri**
  - Dunque, se viene inserito altro SQL, non funziona
- Come funziona:
  - Si specifica prima la query che funziona da “template”
  - Una volta che si ottengono i valori veri dei parametri si collegano al template
  - Si esegue la query
  - Nel caso si debbano leggere anche dei risultati, anche loro si collegano a delle variabili

Prepared Statements



```php
//NB ad ogni istruzione andrebbero gestiti gli errori
// inizializzo il prepared statement
$stmt = $mysqli->stmt_init();

$query = "SELECT matricola, nome, cognome
                FROM studenti WHERE nome=? and cognome=?";

// preparo lo statement per l'esecuzione
$stmt->prepare($query);

// collego i parametri della query con il loro tipo
$stmt->bind_param("ss", $nome, $cognome);

// eseguiamo la query
$stmt->execute();

// collego i risultati della query con un insieme di variabili
$stmt->bind_result($res_matricola, $res_nome, $res_cognome);

// ciclo sulle righe che la query ha restituito
while($stmt->fetch()){
    // ho nelle varibili dei risultati il contenuto delle colonne
    echo "Nome: $res_nome, Cognome: $res_cognome, Matricola: $res_matricola";
}

// liberiamo le risorse dello statement
$stmt->close();
```

Prepared Statements

- Per collegare i parametri si utilizzano le seguenti abbreviazioni per i tipi
  - Interi: i
  - Stringhe: s
  - Numeri in virgola mobile (float e double): d
  - Blob: b
- Si crea una stringa che rappresenta i tipi dell'intera lista di parametri
  - Es. Se la stringa dei tipi è `"isd"`
    - Il primo parametro è intero
    - Il secondo è stringa
    - Il terzo è in virgola mobile

Ricerche con pattern

- A volte è necessario fare delle ricerche all'interno di alcuni campi, senza conoscerne l'esatto valore
- Nelle query SQL l'operatore **LIKE** nella **WHERE** permette di "fissare" solo una parte del valore
- Il pattern viene specificato tramite due *wildcards* (caratteri che si sostituiscono ad altri)
  - Il carattere _ sostituisce un solo carattere qualsiasi
  - Il carattere % sostituisce una sequenza di caratteri qualsiasi
  - Si possono inserire prima o dopo una stringa fissa (o anche prima e dopo)
- Esempi
  - **LIKE "Mari_"** seleziona sia Maria che Mario, ma non Mariottide
  - **LIKE "Mari%"** seleziona Maria, Mario e Mariottide
  - **LIKE "%Mari%"** seleziona Caro Mario e Ciao Mariottide

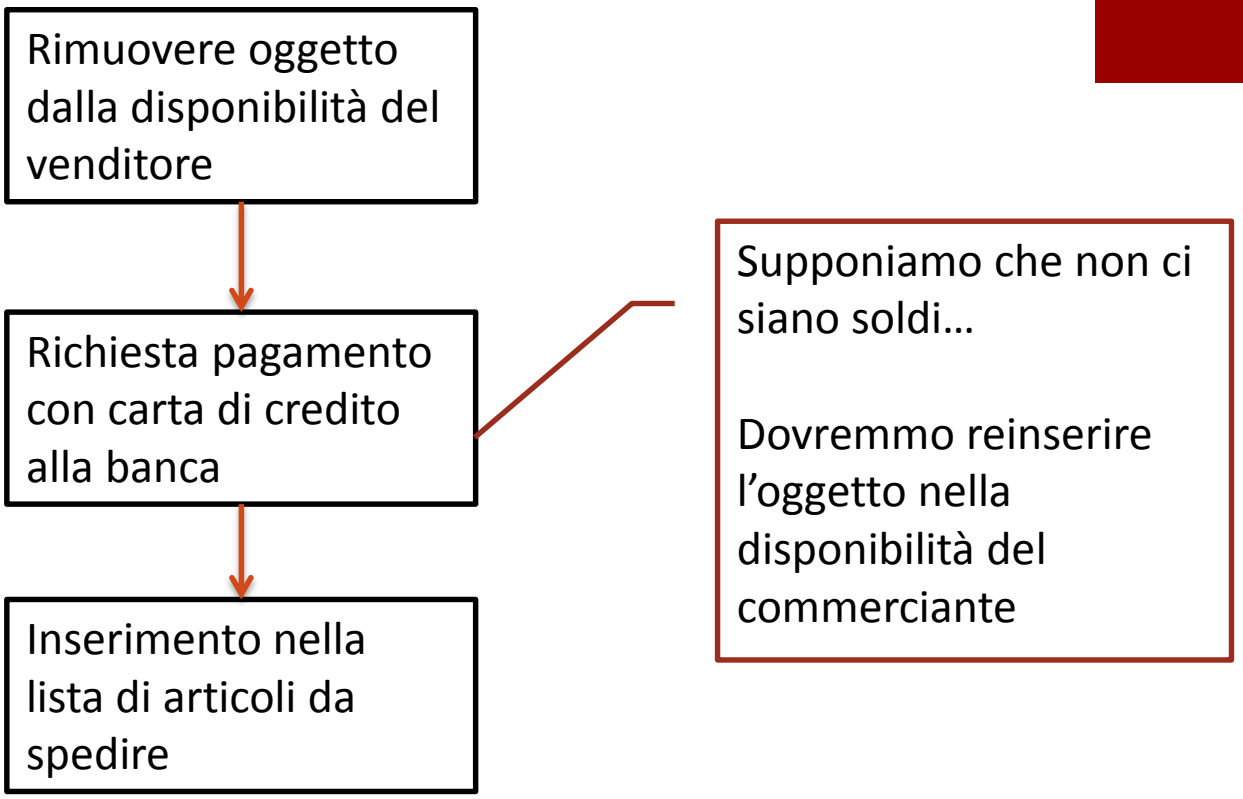
Ricerche con pattern (2)

- Ricerchiamo tutti gli studenti il cui nome contenga una i
- `SELECT * FROM studenti WHERE nome LIKE '%i%';`

```
+----+--------+---------+-----------+
| id | nome   | cognome | matricola |
+----+--------+---------+-----------+
|  1 | Davide | Spano   |    253662 |
|  2 | Pinco  | Pallino |    123456 |
|  3 | Mario  | Rossi   |    654321 |
+----+--------+---------+-----------+
```

Transazioni

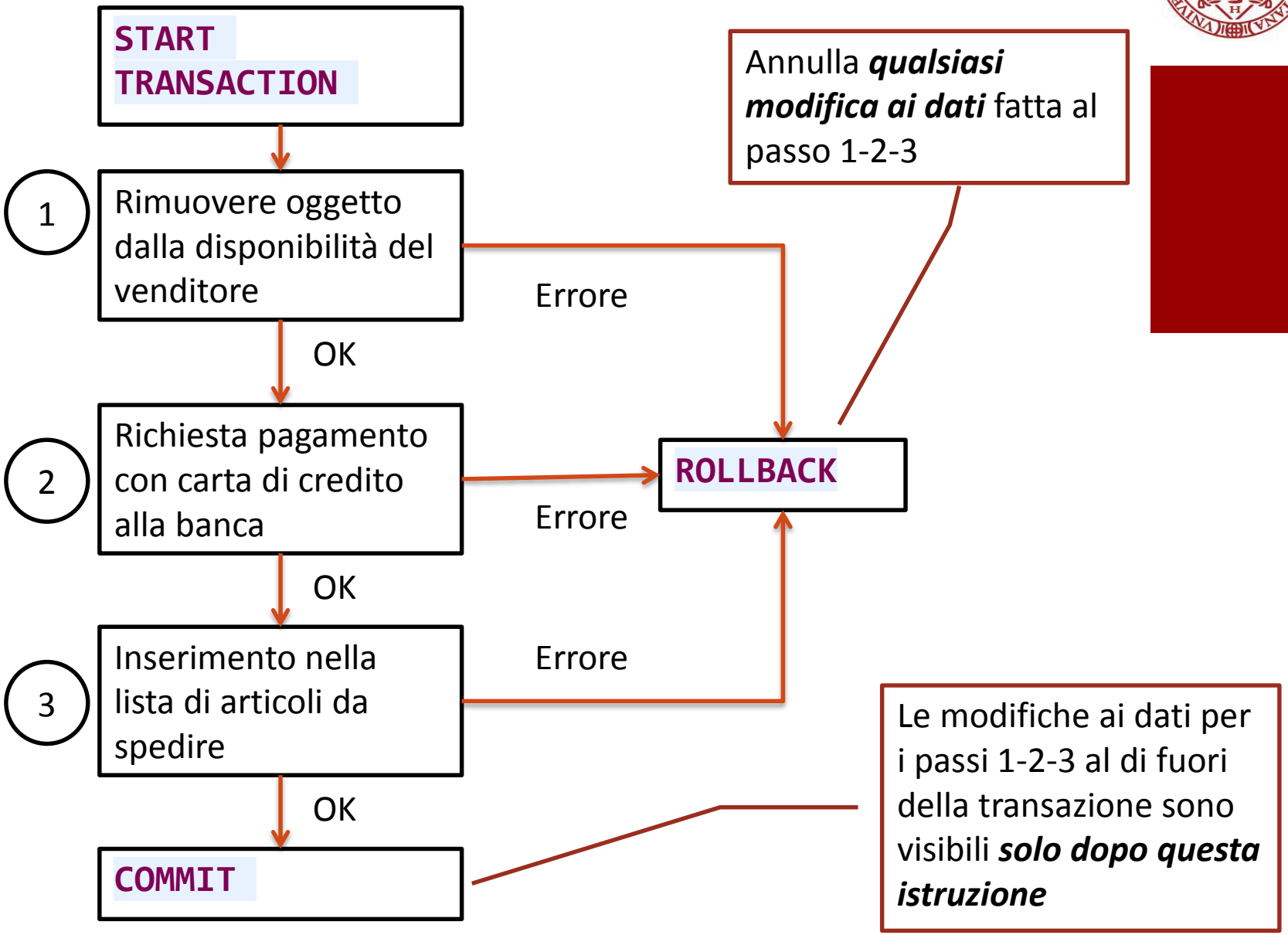
- A volte è necessario utilizzare più passi di modifica sul db per implementare una data funzionalità
- Es. pagamento di un bene



Transazioni (2)

- Invece di compensare manualmente le modifiche precedenti (che potrebbero avere a loro volta problemi di compensazione) il DB ci offre ***le transazioni***
- Si marcano una sequenza di istruzioni SQL con due comandi
  - `START TRANSACTION`
  - `COMMIT`
- La prima inizia una serie di istruzioni che deve essere considerata come "atomica" cioè come se tutte le modifiche siano visibili al di fuori solo quando **tutte le istruzioni sono terminate**
- La seconda rende definitive tutte modifiche
- Nel caso qualcosa vada storto, la `ROLLBACK` permette di riportare il database allo stato precedente alla `START TRANSACTION`

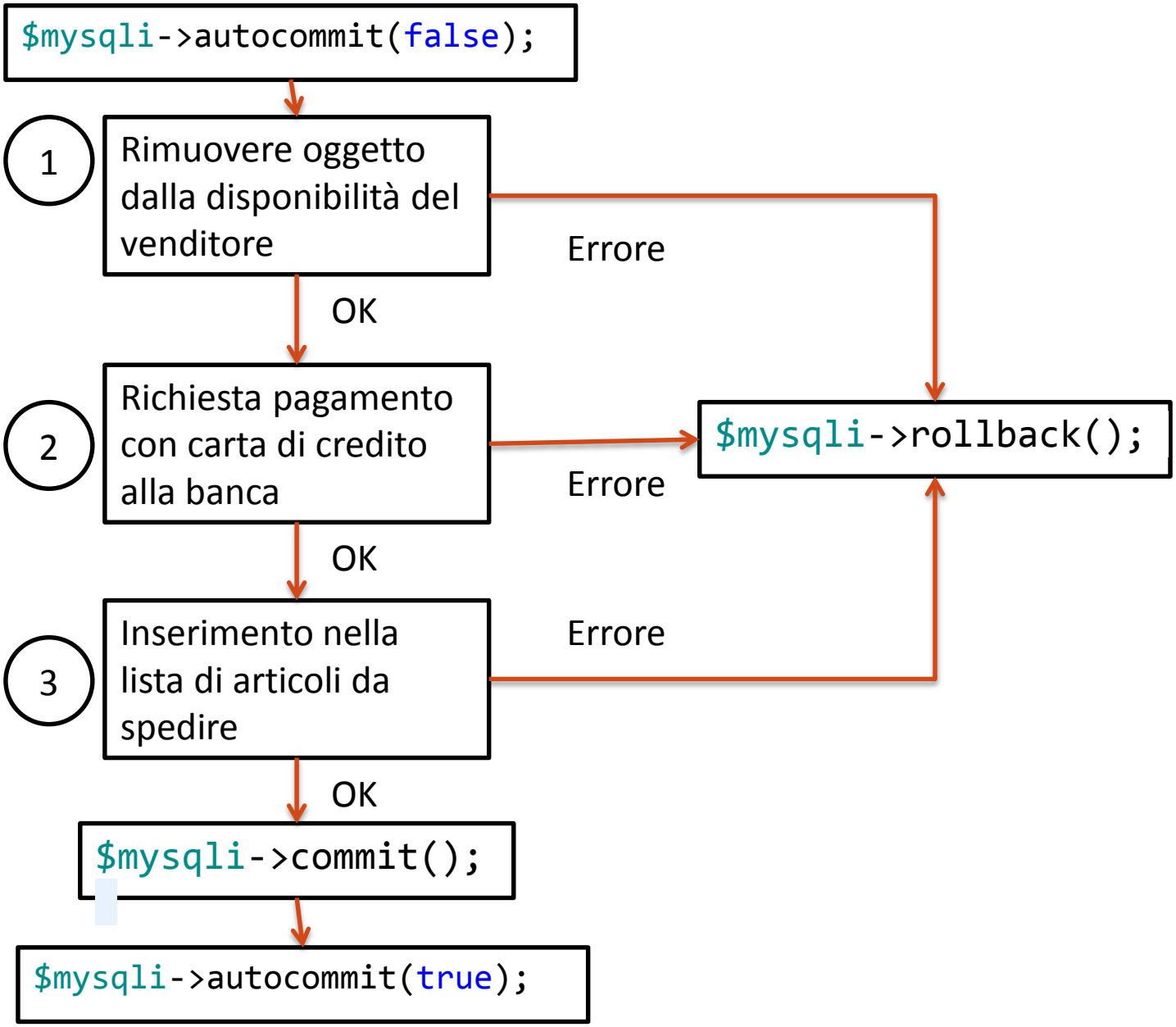
Transazioni (2)



Transazioni in PHP

- Normalmente tutte le query che vengono inviate al db tramite **mysqli** sono automaticamente rese permanenti (commit)
- A meno che non si invochi il metodo **autocommit(false)**
  - Da lì in avanti il commit deve essere fatto esplicitamente
  - Il comportamento normale si ripristina con **autocommit(true)**
- Senza il commit automatico, si può implementare una transazione:
  - Il metodo **commit** di **mysqli** rende permanenti le query
  - Il metodo **rollback** di **mysqli** annulla le modifiche delle query

Transazioni in PHP



Riferimenti

- Robin Nixon *Learning PHP, MySQL, JavaScript and CSS* O'Reilly,
  - Cap. 10
- Jason W. Gilmore *Beginning PHP and MySQL* Apress
  - Cap. 25
  - Cap. 30
  - Cap. 37