Doing More with the Database

pluralsight
hardcore developer training

Lesson Objectives

- **In this lesson we will explore additional techniques using PDO:**
  - Using prepared statements
  - Execute non-queries (`INSERT`, `DELETE`, …)
  - Execute "summary" queries
  - Create stored procedures and functions
  - Execute stored procedures and functions

Using Prepared Statements

- **Prepared statements define a "template" for an SQL statement**
  - Created using the `prepare()` method of the PDO class
  - Include placeholders for parameters that will be inserted when the statement is executed
- **Offers better performance**
  - Query is only parsed once, may be executed many times
- **Parameters are automatically quoted**
  - Eliminates SQL injection risk
- **Parameters can be associated with the placeholders in two ways:**
  - By position
  - By name

Binding Parameters by Position – Method 1

- Statement templates are prepared using "?" as a placeholder

```
$stmt = $db->prepare("select * from books " .
        "where title like ? and author like ? ");
```

Place holders

- To execute the statement, provide the parameter values as an array:

```
$stmt->execute(array( "%$searchtitle%" ,
                      "%$searchauthor%"));
```

Binding Parameters by Position – Method 2

- **Parameters can be bound explicitly:**

```
$stmt = $db->prepare("select * from books " .
        "where title regexp ? and author regexp ? ");
```

```
$stmt->bindParam( 1, $searchtitle );
$stmt->bindParam( 2, $searchauthor );
$searchtitle = "Potter";
$searchauthor = "Rowling";
$stmt->execute();
```

Parameters are bound *by reference*
You cannot pass a literal (like "Dickens")
or an expression here

Binding Parameters by Name – Method 1

- Statement templates are prepared using named placeholders

```
$stmt = $db->prepare("select * from books " .
        "where title like :title " .
        "and author like  :author");
```

Place holders

- To execute the statement, provide the parameter values as an associative array:

```
$stmt->execute(array(
                ":title"  => "%$searchtitle%",
                ":author" => "%$searchauthor%"));
```

Binding Parameters by Name – Method 2

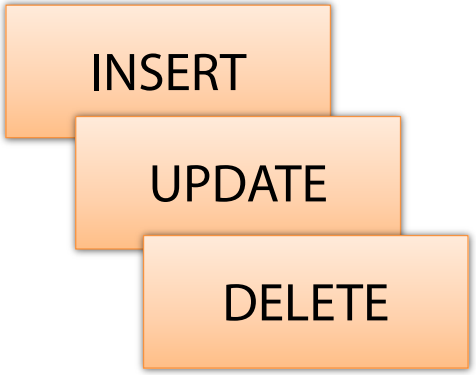
- **Parameters can be bound explicitly:**

```
$stmt = $db->prepare("select * from books " .
        "where title regexp :title " .
        "and author regexp :author ");
```

```
$stmt->bindParam(':title', $searchtitle);
$stmt->bindParam(':author', $searchauthor);
$stmt->execute();
```

Executing non-queries

- **Not all SQL operations return a result set**
- **Some simply modify the database**
  - Return the number of rows affected



Inserting into the Database

- Insertions can be efficiently performed using prepared statements

```
$stmt = $db->prepare("insert into borrowers " .
          (name, address) values (:name, :address)");
```

```
$stmt->execute(array(":name" => "Harold Wilson",
               ":address" => "10 Downing Street"));
$stmt->execute(array(":name" => "Bill Clinton",
               ":address" => "1600 Pennsylvania Ave"));
```

The prepared statement can be executed repeatedly with different values

Deleting from the Database

- **Deletions can be efficiently performed using prepared statements**

```
$stmt = $db->prepare(
"delete from borrowers where address = ?");
```

```
$stmt->execute(array('10 Downing Street"));
printf("%d rows deleted\n", $stmt->rowCount());
```

The `rowCount()` method
returns the number of rows deleted

Summary Functions

- **SQL provides a number of "summary" or "aggregate" functions**
  - Calculate and return a single value from a result set
  - Null values are ignored
  - Answer returned as a result set with one row and one column
- **Common summary functions include:**

Function	Description
`count()`	**Counts the number of values in the result set**
`max(), min()`	**Returns the maximum / minimum of the values in a selected column for selected rows**
`avg()`	**Returns the average of the values in a column**
`stddev()`	**Returns the standard deviation of the values in a selected column**
`sum()`	**Returns the sum of the values in a selected column**

Summary Functions Example

- **Summary functions are executed like any other query:**

```
$stmt = $db->query("select count(*) from books " .
                   "where author like '%Dickens'");
  printf("We have %d books by Dickens\n",
          $stmt->fetchColumn());
```

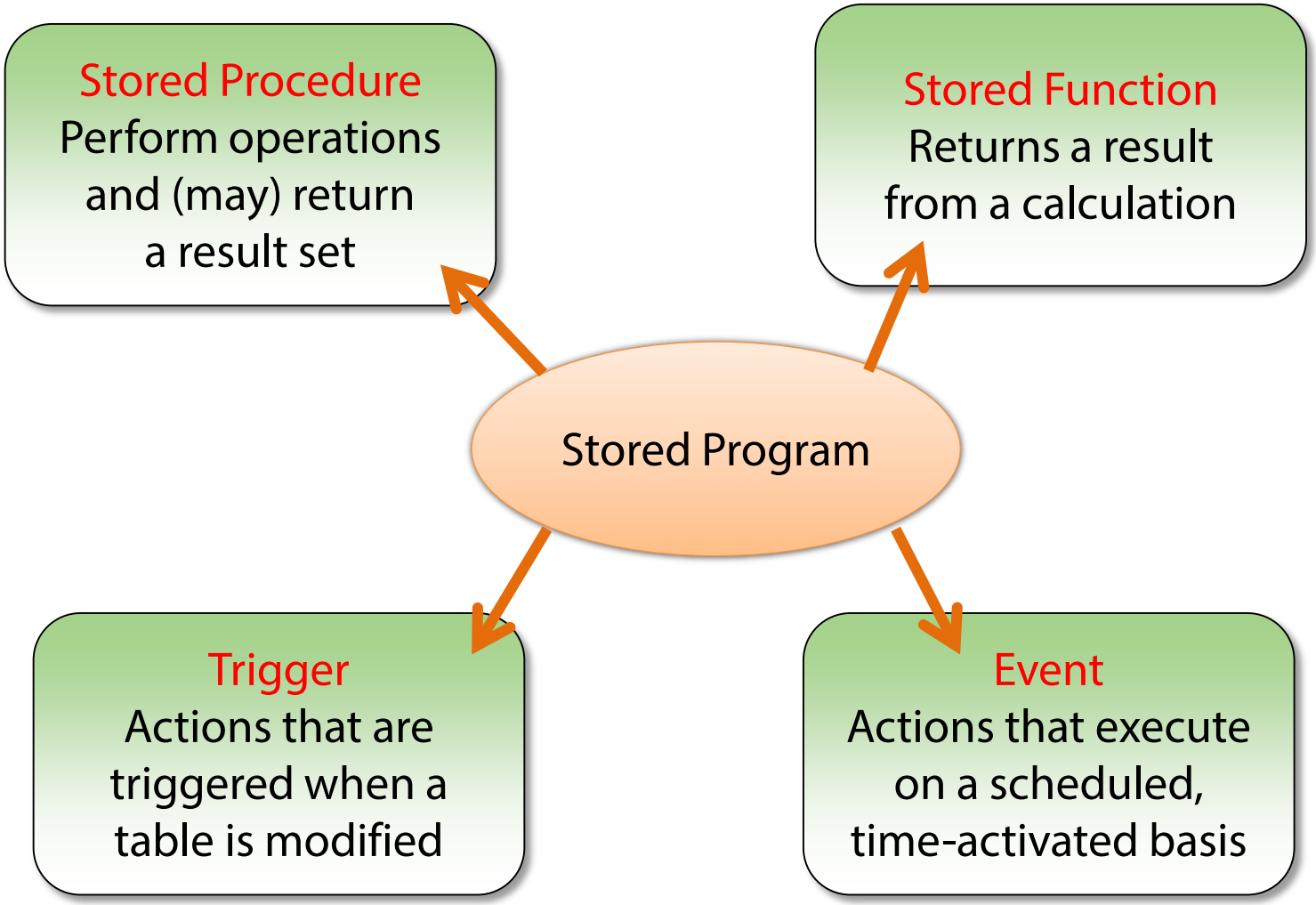
Fetches a specified column (by default, the first)
from the current row of the result set
In this case, there is only one row!

Stored Programs

- **A stored program is a piece of SQL code**
  - Pre-compiled and stored in the database

- **Advantages**
  - Efficient: do not need to transmit and compile the code for every query
  - Maintained separately from the applications
  - Re-usable: can be used across many applications
  - Extended SQL syntax: loops, branches etc.

Types of Stored Program



Creating a Stored Procedure

- Stored procedures can be created from a `mysql` command prompt:

```
mysql> delimiter $$
mysql> create procedure `overdue_books`()
 -> begin
 -> select title from books where duedate < current_date;
 -> end
 -> $$
Query OK, 0 rows affected (0.00 sec)
```

Calling a Stored Procedure

- **Use the SQL "call" command to call a stored procedure**
  - If the procedure returns a result set it can be retrieved like any other

```
$stmt = $db->query( "call overdue_books()" );
while ($row = $stmt->fetch(PDO::FETCH_ASSOC)) {
   printf("%s\n", $row["title"]);
}
```

Creating a Stored Function with MySQL Workbench

Change delimiter so can use ';' within function body

Count how many books are overdue by more than this number of days

```
1   USE `library`;
2   DROP function IF EXISTS `count_overdue_books`;
3
4   DELIMITER $$
5   USE `library`$$
6   CREATE FUNCTION `count_overdue_books` (days integer)
7   RETURNS INTEGER
8   BEGIN
9    return (select count(*) from books
10   where duedate < date_sub(current_date(), interval days day));
11  END$$
12
13  DELIMITER ;
14
15
```

Restore delimiter

Calling a Stored Function

- **Use the SQL "select" command to call a stored function**
  - Retrieve the result just as you would for a built-in function

```
$stmt = $db->prepare("select count_overdue_books( ? )");
  $stmt->execute(array( 2 ));
  printf("No. of overdue books = %d\n",
          $stmt->fetchColumn());
```

Calling a Stored Function

- **Use the SQL "select" command to call a stored function**
  - Retrieve the result just as you would for a built-in function

```
$stmt = $db->prepare("select count_overdue_books( ? )");
  $stmt->execute(array( 2 ));
  printf("No. of overdue books = %d\n",
          $stmt->fetchColumn());
```

Lesson Summary

- **We have learned some additional techniques for using PDO**
  - How to use prepared statements (and why they are a Good Thing)
  - How to execute "non-queries" (such as INSERT and DELETE)
  - Execute "summary" functions
  - How to create stored procedures and functions (and why you might want to)
  - How to execute stored procedures

Coming up in Lesson 7:

Maintaining state in web applications

Why "state" is an issue

The SESSION array

Cookies

Hidden form fields