DUMPSARENA

Zend Framework Certification (Version 4.1)

Zend ZF-100-500

Version Demo

Total Demo Questions: 18

Total Premium Questions: 182

Buy Premium PDF

https://dumpsarena.co

sales@dumpsarena.co

sales@dumpsarena.co
dumpsarena.co

Topic Break Down

Topic	No. of Questions
Topic 1, Zend Framework Basics	20
Topic 2, Zend Framework MVC	27
Topic 3, Zend Framework Forms	8
Topic 4, Zend Framework Database	35
Topic 5, Zend Framework Authentication and Authorization	9
Topic 6, Zend Framework Web Services	13
Topic 7, Zend Framework Internationalization	16
Topic 8, Zend Framework Other Components	46
Topic 9, Mix Questions	8
Total	182

QUESTION NO: 1

You have created a table based on the following data:

EmpID NUMBER (5) PRIMARY KEYEmpName VARCHAR2 (35) NOT NULL Salary NUMBER (9, 2) NOT NULL CommissionNUMBER (4, 2) ManagerName VARCHAR2 (25) ManagerID NUMBER (5) Now, you wantto display the names of employees and their managers, using a self join.

Which of the following SQL statements can you use to accomplish this? Each correct answer represents a complete solution. Choose two.

A. SELECT e.EmpName, m.ManagerName
FROM Employees e, Employeesm
WHERE e.EmpID = m.ManagerID;

B. SELECT e.EmpName, m.ManagerName
FROM Employees e SELF JOIN Employeesm
ON e.EmpID = m.ManagerID;

C. SELECT e.EmpName, m.ManagerNam e
FROM Employees e INNER JOIN Employeesm
ON e.EmpID = m.ManagerID;

D. SELECT e.EmpName, m.ManagerName
FROM Employees e LEFT OUTER JOIN Employees m
ON e.EmpID = m.ManagerID;

E. SELECT e.EmpName, m.EmpName
FROM Employees e, Employees m
WHERE e.ManagerID = m.EmpID;

F. SELECT e.EmpName, m.EmpName
FROM Employees e INNER JOIN Employees m
ON e.ManagerID = m.EmpID;

ANSWER: E F

Explanation:

A self join uses the `Employees` table twice, assigning a separate alias to each logical role. One alias represents the employee row and the other represents that employee's manager row. The relationship is established by matching the employee's `ManagerID` to the manager row's primary key, `EmpID`: `e.ManagerID = m.EmpID`. The selected columns should therefore be `e.EmpName` for the employee and `m.EmpName` for the manager, because both names are stored in rows of the same table. This is the standard hierarchical employee/manager self-reference model.

Both the comma-separated join with a `WHERE` predicate and ANSI `INNER JOIN ... ON` syntax express the same inner self join and return employees for whom a matching manager row exists. Aliases are essential because they distinguish the two references to `Employees`. Oracle documents that a table can be joined to itself by using aliases, and that join conditions determine which rows are related. See the Oracle SELECT documentation and the Oracle documentation on joins.

QUESTION NO: 2

Consider the following XML file:

<!DOCTYPE html

PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"

"http://www.w3.org/TR/xhtml1/DTD/xhtml1 -transitional.dtd" >

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en" >

<head>

<title> SimpleXML Example</title>

</head>

<body>

<h1>

Please go href="http://www.PassGuide.com" >http: //www.PassGuide.com

</h1>

</body>

</html>

Which of the following statements will display the HREF attribute on the anchor tag if the SimpleXML object is $sxml?

A. $sxml->body->h1->a->href

B. $sxml->body->h1->a

C. $sxml->body->h1->a['href']

D. $sxml->h1->a->href

E. echo (string) $sxml->children('http://www.w3.org/1999/xhtml')->body->h1->a['href'];

ANSWER: E

Explanation:

`echo (string) $sxml->children('http://www.w3.org/1999/xhtml')->body->h1->a['href'];` is the appropriate statement because the document declares `http://www.w3.org/1999/xhtml` as its default XML namespace on the root `html` element. Consequently, the unprefixed XHTML elements—including `body`, `h1`, and `a`—belong to that namespace even though their tags do not visibly contain a prefix. SimpleXML requires namespace-aware navigation for elements in a default namespace. Calling `children()` with the XHTML namespace URI establishes the relevant child-element context, allowing the chain to reach the anchor element. XML attributes do not inherit an element's default namespace, so the unprefixed `href` attribute is then obtained with SimpleXML's array-style attribute syntax, `['href']`. Casting the resulting SimpleXML attribute node to a string and passing it to `echo` outputs its value, which is the URL. PHP documents namespace-aware child access through `SimpleXMLElement::children()` at PHP: SimpleXMLElement::children; the default-namespace behavior follows the W3C Namespaces in XML specification.

QUESTION NO: 3 - (SIMULATION)

Fill in the blank with the appropriate term. _____is used to implement a classic Two-Step View pattern that allows a user to wrap the application content within another view.

ANSWER: See the explanation for the answer

Explanation:

Answer: `Layout` (the `Zend\View\Helper\Layout` view helper).

The Layout helper implements the classic Two-Step View pattern by rendering application content first and then wrapping it in a layout/template view.

QUESTION NO: 4

Which of the following are Zend_Cache frontend classes? Each correct answer represents a complete solution. Choose all that apply.

A. Zend_Cache_Core

B. Zend_Cache_Frontend_Output

C. Zend_Cache_Frontend_Function

D. Zend_Cache_Frontend_Class

E. Zend_Cache_Backend_File

ANSWER: A B C D

Explanation:

`Zend_Cache_Core`, `Zend_Cache_Frontend_Output`, `Zend_Cache_Frontend_Function`, and `Zend_Cache_Frontend_Class` are Zend Framework 1 cache frontends. A frontend determines what type of data or operation is being cached and provides the API through which application code interacts with the cache.

The Core frontend provides general-purpose caching of arbitrary data. The Output frontend captures generated output, the Function frontend caches function-call results, and the Class frontend provides caching support for public methods of a class. In contrast, `Zend_Cache_Backend_File` is a backend. A backend determines where cached records are stored, such as the filesystem, memory, SQLite, or another persistence mechanism. See the Zend_Cache frontends documentation and the Zend_Cache backends documentation.

QUESTION NO: 5

Which of the following keywords is necessary for all the switch statements?

A. Case

B. Default

C. Final

D. View

ANSWER: A

Explanation:

Case is the appropriate answer because a switch statement performs its selection by comparing the value supplied to `switch (...)` against one or more `case` labels. Each `case` introduces a possible matching value and the statements that should run when that value matches the switch expression. For example, `switch ($status) { case 'active': ...; break; }` uses `case` to define the active branch of the control structure. This is the keyword that gives a switch statement its normal multi-branch decision-making behavior and is therefore the expected keyword in this question. PHP evaluates switch cases using loose comparison semantics in traditional switch statements, and execution begins at the matching case label; a `break` is commonly used to stop execution from continuing into subsequent case blocks. The PHP manual describes the switch construct and shows `case` as the labels used to match values and select code paths. See the PHP switch documentation and the PHP break documentation for the syntax and execution flow.

QUESTION NO: 6

Which of the following can be used as a countermeasure against the SQL injection attack?

Each correct answer represents a complete solution. Choose two.

A. session_regenerate_id()

B. Prepared statement

C. mysql_escape_string()

D. mysql_real_escape_string()

ANSWER: B D

Explanation:

Prepared statement is a primary defense against SQL injection because it separates SQL program structure from supplied data. The database parses the statement template first, and values are then bound as parameters rather than concatenated into the query text. Consequently, input supplied for a value cannot alter the intended SQL syntax. PHP's PDO documentation describes preparing a statement and binding values with placeholders, which is the preferred modern approach.

`mysql_real_escape_string()` was also historically usable as a countermeasure when applications had to construct SQL strings directly. It escapes special characters according to the active MySQL connection character set, helping ensure that a string value remains data when placed within appropriately quoted SQL string literals. This is an older API and has been removed from modern PHP versions, so current applications should use prepared statements through PDO or MySQLi instead. In the context of this question's legacy PHP/MySQL functions, however, it is a recognized SQL-injection mitigation and constitutes a complete solution when applied correctly to every relevant string value. See the PHP PDO::prepare documentation and the OWASP SQL Injection Prevention Cheat Sheet.

QUESTION NO: 7

Which of the following steps will you use to create a multi -lingual Website? Each correct answer represents a complete solution. Choose all that apply.

A. Creating the View and integrate Zend_Translate into the code

B. Deciding which adapter to use

C. Translating the source file to the desired language

D. Putting the Zend_Translate into session

E. Creating the source file from the code

ANSWER: A B C E

Explanation:

A multilingual Zend Framework application is built around a translation source and an appropriate `Zend_Translate` adapter. "Deciding which adapter to use" is a necessary design step because the adapter determines the supported translation-data format, such as array, CSV, gettext, TMX, or others. The application must then have translation data, so "Creating the source file from the code" establishes the source strings or message identifiers that need localization. "Translating the source file to the desired language" supplies the locale-specific equivalents for those messages; each supported locale needs its own translated data in the format understood by the selected adapter.

"Creating the View and integrate Zend_Translate into the code" completes the application-facing part of the process. The translator must be made available where messages are rendered, allowing views to retrieve the localized text for the active locale rather than displaying hard-coded source strings. Zend Framework's translation component is designed to load translation data through adapters and use message identifiers to return translated content. The legacy Zend Framework manual describes adapter-based translation sources and usage in Zend_Translate documentation; the maintained successor's corresponding concepts are documented in the Laminas Translator documentation.

QUESTION NO: 8

You have given the following XML data in the tasks.XML file:

Validate data

String Validation

Secure data

Encryption

Now, you run the following PHP script:

```
$objDOM = new DOMDocument();
$objDOM->load("tasks.xml");
$note = $objDOM->getElementsByTagName("note");
foreach( $note as $value )
{
$tasks = $value->getElementsByTagName("tasks");
$task = $tasks->item(0)->nodeValue;
$details = $value->getElementsByTagName("details");
$detail = $details->item(0)->nodeValue;
echo "$task :: $detail
";
}
?>
```

What should be displayed when this script is executed?

A. The contents of the whole XML document

B. The XML of every tasks and details nodes

C. The contents of every tasks and details nodes

D. The XML of whole XML document

E. No task/detail lines are displayed because tasks.xml is not well-formed XML.

ANSWER: E

Explanation:

No task/detail lines are displayed because the supplied file is not well-formed XML. The `tasklist` element is closed immediately, before the `note` elements, and a second closing `</tasklist>` appears later without a corresponding open element. XML requires elements to be properly nested and each document to have exactly one document element. Consequently, `DOMDocument::load()` cannot parse this file successfully and returns `false`. Since no valid document containing `note` elements has been loaded, the node list used by the `foreach` loop contains no usable note nodes, so the script's `echo` statement does not produce the intended task and detail output. Depending on the PHP and libxml error-display configuration, XML parsing diagnostics may also be reported, but those diagnostics are not output from the `echo` statement. See the PHP documentation for DOMDocument::load() and the XML well-formedness requirements in the W3C XML specification.

QUESTION NO: 9

You have created a table based on the following data:

EmpID NUMBER (5) PRIMARY KEYEmpName VARCHAR2 (35) NOT NULL Salary NUMBER (9, 2) NOT NULL CommissionNUMBER (4, 2) ManagerName VARCHAR2 (25) ManagerID NUMBER (5) Now, you wantto display the names of employees and their managers, using a self join.

Which of the following SQL statements can you use to accomplish this? Each correct answer represents a complete solution. Choose two.

A. SELECT e.EmpName, m.EmpName FROM Employees e, Employees m WHERE e.ManagerID = m.EmpID

B. FROM Employees e SELF JOIN Employeesm

C. SELECT e.EmpName, m.EmpName FROM Employees e INNER JOIN Employees m ON e.ManagerID = m.EmpID

D. FROM Employees e LEFT OUTER JOIN Employees m

ANSWER: A C

Explanation:

A self join uses the `Employees` table twice, assigning a distinct alias to each logical role. One alias represents the employee row, while the other represents the manager row. The relationship is established by matching the employee's `ManagerID` to the manager's `EmpID`: `e.ManagerID = m.EmpID`. This lets the query select `e.EmpName` as the employee name and `m.EmpName` as that employee's manager name.

The comma-separated form is the traditional join syntax. It places both aliased table references in the `FROM` clause and puts the relationship predicate in a `WHERE` clause. The `INNER JOIN` form expresses the same relationship using explicit ANSI join syntax, with the predicate in an `ON` clause. Both forms return rows for employees whose `ManagerID` identifies an existing employee row, making them complete self-join solutions for listing employees and their managers.

Oracle documents the traditional and ANSI join forms in its SQL language reference: Oracle Database SQL Language Reference: SELECT and Oracle Database SQL Language Reference: Joins.

QUESTION NO: 10

Which of the following methods will you use to get the actual set language in Zend_Translateclass?

A. setLocale()

B. getList()

C. getLocale()

D. isAvailable()

ANSWER: C

Explanation:

`getLocale()` is the appropriate method because Zend_Translate uses the locale to represent the language currently selected for translation. After an application configures a translator and sets or changes its locale, calling `getLocale()` retrieves that active locale value. The returned value can be used wherever the application needs to inspect the translator's current language setting, such as selecting locale-specific presentation details, logging the translation context, or confirming which translation catalogue is being used.

In Zend Framework's translation component, a locale can be supplied during translator configuration or changed at runtime. The component then resolves translations according to that locale and configured fallback behavior. Reading the active setting through `getLocale()` is therefore the direct counterpart to configuring the translator's language context. Zend Framework documentation describes locale-aware translation and translator configuration in the Zend_Translate manual. The legacy API reference for the translator class is also available through the Zend i18n Translator documentation, which documents the general purpose of obtaining the translator's locale when working with internationalized messages.

QUESTION NO: 11

Consider the following script:

echo 'This is some sample text';

?>

Which of the following tags is used in the php script?

A. ASP tag

B. Short tag

C. Script tag

D. Standard tag

ANSWER: D

Explanation:

The correct answer is **Standard tag**. The PHP code in the script begins with `<?php` and ends with `?>`. This is the standard PHP opening-and-closing tag syntax used to embed PHP instructions within an HTML document. Everything outside these delimiters is treated as ordinary output, such as the page's HTML markup and text, while the statement `echo 'This is some sample text';` is interpreted and executed by PHP because it appears between the PHP tags.

The standard opening tag, `<?php`, is the portable and recommended syntax for PHP source files. It clearly identifies the start of PHP code and is consistently supported in PHP environments. The closing tag `?>` marks the end of the PHP block, after which the parser resumes treating content as non-PHP output. PHP's documentation describes PHP tags as the delimiters that switch the parser into and out of PHP mode. See the PHP manual on PHP tags and the PHP manual on escaping from HTML.

QUESTION NO: 12

Which of the following Zend Framework methods will you use to convert a PHP value into a JSON string?

A. Zend_Json::decode()

B. Zend_Json::encode()

C. Zend_Json::serialize()

D. Zend_XmlRpc_Value::encode()

ANSWER: B

Explanation:

`Zend_Json::encode()` is correct because it serializes a PHP value into its JSON representation. Arrays and objects can be converted into JSON text suitable for an HTTP response, a JavaScript client, or another JSON-consuming service. For example, `Zend_Json::encode(array('status' => 'ok'))` produces a JSON object containing the supplied status value.

The corresponding `Zend_Json::decode()` method performs the reverse operation by parsing a JSON string into a PHP value. `Zend_XmlRpc_Value` is related to XML-RPC value handling, not JSON encoding. See the Zend_Json documentation and the Zend_Json API reference.

QUESTION NO: 13

Which of the following statements correctly explain the working of Zend_Sera ch_Lucene?

Each correct answer represents a complete solution. Choose all that apply.

A. It supports ranked searching, phrase queries, wildcard queries, and proximity queries.

B. It is a text search engine.

C. It requires the Zend_Db class to connect to a database server.

D. It can also be used to search by any specific field, such as, title, author, etc.

ANSWER: A B D

Explanation:

Zend_Search_Lucene is a general-purpose, file-based text-search engine implemented in PHP and modeled on the Lucene search approach. It builds and searches indexes rather than requiring a database connection, making it suitable for indexing application content and retrieving relevant documents efficiently. Its search results are ranked: each matching document receives a score that represents its relevance to the submitted query, and results can be ordered using that score.

The component supports the query capabilities named in the correct statements, including phrase queries for terms appearing together in a specified sequence, wildcard queries for pattern matching, and proximity queries for terms occurring near one another. Zend_Search_Lucene also indexes documents as named fields. Consequently, an application can limit a search expression to a particular field, such as `title` or `author`, instead of searching every indexed field. This permits more precise queries and supports document structures containing distinct metadata and content fields. The Zend Framework reference documentation describes the component, its indexing model, query syntax, result scoring, and field-oriented searching. See the Zend Framework 1 Zend_Search_Lucene manual and its query API documentation.

QUESTION NO: 14

Which of the following classes will you use to create a navigation container from an array configuration in Zend Framework?

A. Zend_Navigation

B. Zend_View_Navigation

C. Zend_Controller_Navigation

D. Zend_Route_Container

ANSWER: A

Explanation:

`Zend_Navigation` is correct because it is the standard navigation container class in Zend Framework 1. It can be instantiated with an array or configuration object describing pages and their properties, such as labels, routes, URIs, controller names, actions, visibility, and access-control requirements.

The resulting container can be supplied to navigation view helpers such as `navigation()->menu()`, `breadcrumbs()`, or `sitemap()`. Individual entries are represented by navigation page classes, but `Zend_Navigation` is the container that organizes the page hierarchy. See the Zend_Navigation documentation.

QUESTION NO: 15 - (SIMULATION)

Fill in the blank with the appropriate method name. The_____ method is used to retrieve headers when the storage has been opened.

ANSWER: See the explanation for the answer

Explanation:

The correct method is `getHeaders()`.

When a Zend mail storage object has been opened, use `getHeaders()` to retrieve the message headers.

QUESTION NO: 16

Which of the following steps will you use to create a multi -lingual Website? Each correct answer represents a complete solution. Choose all that apply.

A. Creating the View and integrating Zend_Translate into the code

B. Deciding which adapter to use

C. Translating the source file to the desired language

D. Putting Zend_Translate into session

E. Creating the source file from the code

ANSWER: A B C E

Explanation:

Creating a multilingual site with Zend Framework's `Zend_Translate` begins by deciding which adapter to use, because the adapter determines the supported translation-source format and how messages are loaded. Zend_Translate supports several source formats, such as arrays, CSV files, gettext files, TMX, and translation-memory formats. The application's translation data must then be prepared by creating the source file from the code: this means extracting or collecting the message identifiers used by the application into the format required by the selected adapter. Translating the source file to the desired language supplies the locale-specific message values that Zend_Translate will resolve at runtime. Finally, creating the View and integrating Zend_Translate into the code makes translated messages available where they are rendered, commonly through the view translation helper or an appropriately configured translator instance. These activities form the normal translation workflow: select storage format, create message catalogs, localize them, and invoke translation while producing output. Zend Framework's documentation describes adapter-based translation sources and their use in applications. See the Zend_Translate manual and the translation source creation documentation.

QUESTION NO: 17

Which of the following functions can you use to mitigate a command injection attack? Each correct answer represents a complete solution. Choose all that apply.

A. strip_tags()

B. escapeshellarg()

C. htmlentities()

D. escapeshellcmd()

ANSWER: B D

Explanation:

`escapeshellarg()` and `escapeshellcmd()` are PHP functions specifically intended for contexts where data could otherwise alter a shell command. `escapeshellarg()` escapes one complete argument so that shell metacharacters contained in supplied data are treated as literal content rather than shell syntax. It is the appropriate protection when an application constructs a command from a fixed executable and fixed arguments while incorporating an untrusted value as an individual argument. PHP documents that the function adds surrounding quotes and escapes embedded quote characters in a platform-appropriate manner.

`escapeshellcmd()` escapes characters that may be used to manipulate a shell command, making it relevant when escaping a complete command string before it is passed to a shell-executing API. PHP's documentation notes that it should be used with care: escaping a command does not replace input validation or the safer design of avoiding a shell entirely. In practice, applications should use strict allowlists for expected values, avoid concatenating untrusted input into executable commands, and prefer process APIs that accept an argument array when available. Still, both listed functions are recognized PHP shell-escaping controls and can mitigate command-injection risk when applied to the correct command-construction context. See the PHP escapeshellarg() documentation and PHP escapeshellcmd() documentation.

QUESTION NO: 18

Fill in the blank with the appropriate term. _____is used to implement a classic Two-Step View pattern that allows a user to wrap the application content within another view.

A. Zend_Layout

ANSWER: A

Explanation:

`Zend_Layout` is the Zend Framework component designed to implement the classic Two-Step View pattern. In this pattern, an action first renders its own view script to produce the page-specific content. That content is then inserted into a separate layout script, which provides the shared outer structure of the response, such as the document markup, header, navigation, footer, and common asset references. This separation lets an application keep reusable presentation chrome in one place while allowing each controller action to supply only its unique content.

When enabled, `Zend_Layout` captures the action view output and makes it available to the layout through the layout content placeholder. Layouts may be configured through application bootstrap resources or used directly, and the selected layout can be changed when a request requires a different page structure. This behavior is precisely the "wrapping" of application content described in the question. The Zend Framework manual identifies Zend_Layout as providing a two-step-view pattern implementation and documents its layout-script rendering workflow. See the Zend Framework documentation and the Zend Framework 1 source repository.