Course team
Developer: Jenny Lim, Consultant
Designer: Chris Baker, OUHK
Coordinator: Dr Li Tak Sing, OUHK
Member: Dr Andrew Lui Kwok Fai, OUHK

External Course Assessor
Prof. Mingshu Li, Institute of Software, Chinese Academy of Sciences

Production
ETPU Publishing Team
# Contents

Overview 1

Objectives 2

Introduction 3

**Database concepts** 4
  - Relational databases 6
  - Database management systems (DBMS) 9
  - Structured Query Language (SQL) 13

**PHP and MySQL** 20
  - ABC Books: our case study business 20
  - PHP MySQL functions 20
  - Open Database Connectivity (ODBC) 23
  - Database abstraction 29

**Session management mechanisms** 32
  - Client-side techniques 33
  - Server-side techniques 38

**Issues in state management** 45
  - Ensuring security 45
  - Managing sessions across multiple servers 45

Summary 47

Suggested answers to self-tests 48

References 52
Overview

Databases are used in any information system which needs to store, organize and manage large amounts of data. They also play an important role in making an organization’s information resources available online. Product catalogues, customer orders and user registrations are just some of the types of data that can be stored in back-end databases which are connected to the Web.

In the first part of this unit, we’ll briefly look at the history of databases. We also look at the most common data model used by database systems today: the relational model. We then discuss the ways in which users can access Web databases and will look at the basic SQL commands that are used for retrieving and updating information in a database. We will also look at how these commands can be incorporated within a server-side script written in a language such as PHP.

The second half of the unit deals with session management. In general, a session begins when a user visits the first page on the site, and ends when he/she clicks onto a new site or remains inactive for a long period of time. There are many situations when a Web server needs to keep track of what has been going on within a user session. For example, search engine sites must know the page you are currently viewing so that they can retrieve the previous or next page of results correctly. Shopping sites need to match online shoppers with their virtual shopping baskets through all the pages that they visit.

However, HTTP is a stateless protocol, which means that a Web server processes each request for a new webpage as a discrete unit, without retaining any knowledge of previous pages requested within the same user session. This design approach leads to better performance and scalability. However, many Web applications depend on the ability to track individual requests from distinct users and store information about the current state of a session for each client.

We will look at a number of techniques which are used to store session information for HTTP clients. We will also discuss the pros and cons of each of these methods. These techniques are all implemented within the application level due to the lack of support for it in the underlying protocol.

Using database technology and the session management techniques you’ve learned, you will implement a shopping cart application for ABC Books. This will allow shoppers to browse and select books from a product catalogue database running behind their website. Session management techniques will be used to identify each user and to keep track of the items in their shopping cart. Once again, we will use PHP and MySQL to implement this application.

This unit should take you about five weeks or 40–45 hours to complete. Please plan your time carefully and remember to contact your tutor if you have problems.
Objectives

1. **Describe** how databases can be integrated into Web applications.

2. **Discuss** the necessary technologies for implementing Web-database connectivity.

3. **Examine** how session management can be performed within a stateless protocol like HTTP.

4. **Construct** a Web application which interfaces with databases and maintains state over a series of HTTP requests.

5. **Explain** relevant issues when planning and adopting a session management strategy for a website.
Introduction

Static HTML pages are adequate for publishing small amounts of information that does not change frequently. By contrast, dynamic HTML pages are required for publishing data which changes very often or when your users need to enter and retrieve live data from a database using a form. Most e-commerce sites nowadays make use of back-end databases to store their content. Server-side programs are then used to assemble the data from the database in many different ways in response to specific user requests.

Relational database management systems are the most common technology used for storing, organizing and managing Web databases. Server-side programs can issue calls against the database using standard technologies such as Open Database Connectivity (ODBC) or through built-in functions provided by the scripting language. Requests for retrieving, updating, or adding records to the database are written using another standard known as SQL (Structured Query Language). The Web server interfaces with server-side programs and application servers which, in turn, issue the requests to back-end databases.

To implement shopping carts, user preferences, or detailed visitor tracking, the server must be able to recognize a user throughout an entire session, from one page to the next. This is the essence of session management. As mentioned in the ‘Overview’, session management is not so straightforward under HTTP and must be implemented within the Web application itself.

We will discuss both client-side and server-side techniques for managing user sessions. Client-side techniques, such as cookies, involve storing the session data on the client machine. Server-side techniques involve storing the session state within a file or database on the Web server itself. That’s right, databases are used not just for storing and organizing website content, they also play a very important role in session management!

Implementing the session management techniques discussed in this unit will help you deliver a richer, more interactive Web environment to your users.
**Database concepts**

We’ll start this section with a brief history of databases. We will also look at the most common data model used by database systems today: the relational model. We’ll discuss the core concepts of the relational model and show you how data is stored and organized based on this model. Towards the end of the section, you will get a chance to perform data administration and manipulation tasks using MySQL, a very popular open-source database management system which uses the relational data model.

**Flat files**

The business world started adopting computers for their data processing needs after World War II. Using the same approach as scientists and engineers, businesses originally stored data in flat files where data was organized in tabular format. Rows in the table were called ‘records’ and columns were called ‘fields’. (In Figure 5.2, records give information about individual students while fields deal with categories such as students’ names, subjects or grades.)

<table>
<thead>
<tr>
<th>Student Name</th>
<th>Semester</th>
<th>Course</th>
<th>Grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>Marvin Kwok</td>
<td>2003 Semester 1</td>
<td>Chemistry 101</td>
<td>C</td>
</tr>
<tr>
<td>Alice Tam</td>
<td>2003 Semester 2</td>
<td>Physics 102</td>
<td>B-</td>
</tr>
<tr>
<td>Betty Cheung</td>
<td>2003 Semester 2</td>
<td>Math 150</td>
<td>B</td>
</tr>
</tbody>
</table>

**Figure 5.2**  A subset of a file containing student results

In order to improve the performance of flat files, indexes were built on selected fields which are commonly used to search the file. These index files allowed applications to access records directly rather than searching through the entire file from the very beginning.

Flat files were not well-suited for the storage and management of significant volumes of data. A flat-file structure only contains a single table and is not practical for most business applications, which must store information about multiple entities (e.g. Customer, Order, Invoice, Supplier) that may have relationships with each other.

In addition, early data-processing systems tended to grow independently without an overall plan to guide their development. Departments such as accounting, finance, manufacturing, human resources, sales and marketing would develop their own programs and data files in isolation from other functional areas. This approach led to multiple master files which were created, maintained and operated by separate divisions or departments. The next figure illustrates how data is organized within the traditional flat-file environment.
Several problems result from a flat-file approach:

1. **Data redundancy and confusion**

   The same employee, customer or product record may appear in several files belonging to different departments. This makes it difficult to keep the same information consistent throughout the organization.

2. **Program-data dependence**

   There is a tight relationship or coupling between data stored in flat files and the specific programs required to update and maintain those files. Any change in the file structure, such as adding a new field or increasing the length of an existing field, requires a change in all programs that access the data.

3. **Lack of flexibility**

   End-users did not have any tools to access flat files directly. The specialized skills of a programmer were needed in order to write a program which could extract the data and generate the necessary reports. This made it difficult to deliver ad-hoc reports or respond to unanticipated information requirements.

Database technology can solve many of the problems created by the traditional flat-file approach. A database is actually a collection of data organized to serve many applications efficiently by centralizing the data and minimizing redundant data. Databases may contain one or more tables. In effect, a single database can serve multiple applications simultaneously. The next figure illustrates the shared database approach.
Databases can be classified by the model or data structures associated with them. Some models have been in wide use for some time, such as the hierarchical and network models. However, the dominant model today is the relational model, used together with the SQL query language. This is the model that we will adopt for the ABCBooks database.

Relational databases

Relational databases are based on the relational model, which was invented by an IBM researcher named Ted Codd in 1970. The relational model is based on relational calculus and algebra, and was meant to allow non-technical users to store and retrieve large amounts of information. Ted Codd envisioned a system where the user would be able to access information with English-like commands, and where information would be stored in tables.

Relational databases actually represent data as a collection of two-dimensional tables. A table can be visualized as a group of records which, in turn, are made up of fields. Tables are also referred to as files, since each table can be viewed as equivalent to a flat file. However, databases may contain one or more tables, while flat files only hold one table at a time. The next figure illustrates this hierarchy.
Each table stores data about a single entity, which could be a person, place, object, event or concept of interest to the business. Books, customers and orders are just some of the entities that will be stored in the ABCBooks database.

The rows within a table correspond to the records that contain information about a single instance of an entity. For example, there will be one record within the ABCBooks Book table for *A Dream of Red Mansions* by Tsao Hsueh-Chin, which is one particular instance of a book title.

The fields within a table contain the characteristics or attributes recorded for each entity. Databases generally support three types of fields: numeric types, date and time types, and string or character types. There is a special field known as the primary key, which serves to identify each record uniquely within a table. *book ID,* *title,* *description,* *unitPrice* and *monthPublished* and *yearPublished* are just some of the fields that may be stored for each book, with the *book ID* serving as the primary key.

The next figure shows an example of the tables and fields for our initial database design for ABC Books. The primary keys are underlined for each table.
Relational databases can also record the relationships or linkages between tables through common fields which occur in the related tables. In the example above, there is a relationship between the Order table and the Customer table through the field customerID. We can therefore retrieve all the orders for a particular customer by looking for all Order records that contain a specific customerID. We can also look up the customer details for a particular order by using the same ID. customerID is considered a foreign key within the Order table, because it is a primary key in another table. Foreign keys are used to look up information in another table to avoid keeping the same data in more than one place.

Here are the three basic operations used to extract data from a relational database:

1. Select — creates a subset of records from a file which meet a certain criteria. For example, we may want to select all books written by Mark Twain from the Product table.

2. Join — combines fields from more than one table. For example, we may want to join the Product table with the Order Line table so that we can see all the products within an order, along with their descriptions, number of pages and date of publication.

3. Project — permits the creation of new tables or views that contain only the information required from the underlying table or tables. For example, some programs may not need all the fields in the Customer table, so a view of this table containing only the fields required may be created and made accessible to these programs.

The actual commands for these operations are expressed in Structured Query Language, which you’ve used briefly in Activity 4.11 (about full-text searching with MySQL). The users and application programs who issue these commands do not need to know how these operations will
actually be carried out. They just need to issue the right instructions and
the database will take care of servicing the request.

Because of this, relational databases are much more flexible in
accommodating ad-hoc queries, or queries which combine information
from different tables and sources. The looser coupling between programs
and data also allows the addition of new tables, columns, and records
without the need to rewrite the programs that access the database. This
has made relational databases very popular with businesses.

Self-test 5.1

1 Briefly describe the problems caused by using the flat-file structure
   for storing and organizing data.

2 How is data organized using the relational data model?

3 What is the difference between a primary key and a foreign key?

4 Referring to Figure 5.6, identify two other foreign keys aside from
   CustomerID. What relationships are established by these keys?

Database management systems (DBMS)

A database management system is simply the software used to create,
access and manipulate databases. The DBMS acts as an interface
between application programs and the physical data files. It relieves the
programmer or end user from the task of understanding where and how
the data is actually stored by separating the logical and physical views of
the data.

A database management system has three major components:

1 Data Definition Language (DDL) — language used to define the
   content and structure of the database, such as creating and dropping
   databases and tables.

2 Data Manipulation Language (DML) — language used to extract,
   insert, update and delete data.

3 Data dictionary — inventory of all the data elements or fields within
   the database; it contains metadata or data about the database itself.
   This allows programs and users to query the server and find out what
   data structures (e.g. databases, tables, indexes, fields) are stored
   within.

Many popular database management systems operate using the client-
server model, with a database server responding to client requests over a
network connection. However, it is also possible for the client and database server to be on the same machine.

![Diagram](image)

**Figure 5.7** DBMS serves as an interface between applications, users and the physical data files

We will be using MySQL to implement the back-end database for user registration, catalogue search and online ordering on the ABC Books website. The next reading gives you a quick overview of the programs that you will use to administer and manipulate MySQL databases. These programs make up the MySQL DBMS, and are classified as server, client or utility programs. You will use some of these programs in the following activities.

### Reading 5.1


These programs can be invoked at the command line by entering the program name, followed by any options or other arguments needed to instruct the program. Please note that mysqld is the name of the MySQL database server, and this program must be running before clients can connect to it.

In the next activity, we will explore some of the administrative functions that can be performed with MySQL.

### Activity 5.1

We will now perform some common database administration tasks such as starting and stopping the database server, creating a new database and adding a new user account.

We will use three MySQL client programs in this activity — *mysql*, *mysqladmin*, and *mysqlshow*. They were all briefly described in Reading 5.1. While you’re working through the examples below, you can
also refer to this URL as a more detailed reference for these and many other useful programs:

‘MySQL client and utility programs’,

Just a quick reminder before you start typing commands into the interactive MySQL client programs. You’re free to type your SQL commands in upper or lower case, but if you’re using MySQL on a UNIX-based system, be aware that database and table names are case-sensitive. This is because they correspond to directories and files in the MySQL data directory. So if you defined a table as `Order_Detail` and then try to query it using the name `order_detail`, MySQL will tell you that the table doesn’t exist.

Part 1 — Starting and stopping a server

The recommended way to start a MySQL server is to use the `mysqld_safe` startup script. This method is preferred over starting the `mysqld` daemon directly, because it adds some safety features such as restarting the server when an error occurs. It also logs run-time information to an error log file. In prior versions of MySQL (before 4.0), this startup script was known as `safe_mysqld`.

There are also a number of ways to stop the server, but we will use the `mysqladmin` client program to do this.

1 Check whether MySQL is already running:
   
   shell> mysqladmin status

2 If MySQL is already running, you can shut it down:
   
   shell> mysqladmin shutdown

3 You can restart MySQL using the `mysqld_safe` script. You must include the ampersand (&) at the end to run it as a background task:
   
   shell> mysqld_safe &

4 Now use `mysqladmin` to check the status of the MySQL server after restarting it:
   
   shell> mysqladmin status

Part 2 — Creating a new database

1 We will now create a new database called `ABCBooks` using the `mysqladmin` client program:
   
   shell> mysqladmin create ABCBooks

2 Next, verify that the database has been created using the `mysqlshow` command. This command will give you a listing of all the existing databases on the server. You should see `ABCBooks` among the databases listed:
Part 3 — Adding a new user account

MySQL is initially installed with default accounts for the ‘root’ user. These accounts have blank passwords, so anyone can log in to the database as long as they know the account name. For security reasons, however, you do not want your Web server scripts to connect to the database server as ‘root’. We will create a new user account to be used by our scripts for accessing and manipulating the ABCBooks database.

1. You will use the mysql client to connect to the MySQL server as ‘root’. The mysql client is an interactive program that allows you to connect to a MySQL server, run queries, and view the results:

   shell> mysql –u root

   Note that if you have already assigned a password to ‘root’ (which is a good idea), you should include the password using the –p option, with no space between the option switch and the password (i.e. –pmypassword).

2. Once you’ve logged in as ‘root’, you can select which database you’re going to work on from the several databases that the server currently maintains. Since we’re going to create a new database user, we need to use the system database called mysql where user accounts are stored:

   mysql shell> use mysql;

3. User accounts are stored in a table called user within the MySQL database. We will insert a record for our new user into this table. After adding the record, you must tell the server to reload the user account information with FLUSH_PRIVILEGES. Without issuing the FLUSH_PRIVILEGES command, the server will not recognize the newly-added user until after the server has been restarted.

   Lastly, we will use the PASSWORD function to encrypt the password before storing it in the user table:

   mysql shell> insert into user (Host, User, Password) values (‘localhost’, ‘webuser’, PASSWORD(‘mt834’));
   mysql shell> flush privileges;
   mysql shell> grant select, insert, update, delete, create, drop on ABCBooks.* TO ‘webuser’@‘localhost’ identified by ‘mt834’;

We have just created a new user called ‘webuser’ who is allowed to perform data manipulation (e.g. insert, update, select and delete) and administration (e.g. create and drop) tasks on the ABCBooks database.
4. You can verify that the user has been added correctly by logging into the MySQL client once more using this new account.

Exit from the MySQL client command shell with:

```
mysql shell> \q
```

Then from the operating system command line, you can log in as the new user. Please note that there must be no space between the `-p` option switch and the password. You can also specify the database that you want to use (e.g. `ABCBooks`) as part of the options in the command line up front:

```
shell> mysql -u webuser -pmt834 ABCBooks
```

Our `ABCBooks` database is empty for the moment, but it won’t stay that way for long!

Now do the following self-test to revise the database management concepts you’ve learned in this section.

### Self-test 5.2

For questions 2 and 3 below, you may want to consult this reference: http://www.mysql.com/doc/en/Client-Side_Scripts.html.

1. Describe the steps involved in adding a new database user account to MySQL.

2. Describe four values that are displayed when you run the `mysqladmin status` command.

3. Write the `mysqlshow` command that will display information about the `Author` table in the `ABCBooks` database.

### Structured Query Language (SQL)

SQL (pronounced ‘ess-que-el’) is a standard, declarative language used to communicate with a relational database. It is declarative because it describes what must be done rather than how it must be done. Application programmers who use SQL statements do not have to worry about how the database will carry out their request. They just have to issue commands using SQL, send the commands to the database server, and the DBMS will carry them out.

SQL is used to implement the DDL (Data Definition Language) and DML (Data Manipulation Language) of many popular relational DBMSs such as MySQL, Oracle, Sybase, Microsoft SQL Server, Microsoft Access, Postgress, etc.
Be aware that although most database systems use SQL, they may also have their own additional SQL extensions that are available only on their system. This means that different DBMSs may not provide the complete set of SQL functions or may include functions which are not part of the standard.

In this unit, we will discuss the standard SQL commands such as Select, Insert, Update, Delete, Create, and Drop. This small set of commands is adequate for accomplishing most of the actions that we will perform on the ABCBooks database.

The next reading goes through each of these SQL commands in detail. You can also try submitting your own SQL statements in an online form at the end of each SQL command discussed.

**Reading 5.2**

You may skim through these readings first, just to get a fair idea of what’s involved before you start entering SQL commands interactively in the MySQL server on your local machine:


In the next activity, we will issue SQL commands to create and manipulate the tables within the ABCBooks database. The commands will be entered via the MySQL client program, which is an interactive program that allows you to connect to a MySQL server, run queries, and view the results.

**Activity 5.2**

The files needed for this activity can be downloaded from the course website. If you need help while completing the tasks, you can refer to the following portions of the MySQL online documentation:

- ‘SQL statement syntax’,
- ‘Column types (or types of data supported by MySQL)’,
1 Use the mysql client to log in to the MySQL server as `webuser'.
Specify that you will be working on the ABCBooks database. (Refer to Activity 5.1 for more information on how to do this.)

2 Now type the following SQL statement into the mysql client command shell. This will create a table called `Book', which stores information about the titles sold by ABC Books. The semicolon on the last line denotes the end of the statement. Pressing the Enter key after typing the semicolon will cause the shell to execute the statement:

```sql
create table Book (
    bookID int NOT NULL AUTO_INCREMENT PRIMARY KEY,
    title varchar(200) NOT NULL,
    author_lastName varchar(150) NOT NULL,
    author_firstName varchar(150) NOT NULL,
    author_middleName varchar(150),
    imageURL varchar(100),
    price numeric(8,2),
    numPages numeric(5),
    pub_Month numeric(2),
    year_Month numeric(4),
    description mediumtext,
    primary key (bookID)
);```

This statement creates a table named `Book' with 11 fields. `bookID' is used as the primary key, which means that an index will be created for this field and no two rows of the table will have the same `bookID'. A unique number will automatically be generated by MySQL and stored in the `bookID' field for every new record inserted into the table.

title, auth_fname, auth_lname, auth_mname and imageURL are strings of variable lengths. The longest strings that can be stored within these fields are indicated by the numbers within the parentheses. `listPrice' can contain a value with two decimal places. `numPages', `monthPublished' and `yearPublished' can only contain integers with the specified maximum number of digits, while `description' can store a very long block of text.

3 Aside from the book description, ABC Books also wants to store short biographies of their authors in the database. We will create a new table called `Author' so that we can store author-related information separately from the books they’ve written. It is more efficient to store one copy of the author’s record in a separate table called `Author', and then link the `Book' and `Author' table through the foreign key fields — auth_firstName, auth_lastName and auth_middleName.

Let’s create the `Author' table next:
create table Author(
  auth_lastName varchar(150) NOT NULL,
  auth_firstName varchar(150) NOT NULL,
  auth_middleName varchar(150) NOT NULL,
  imageURL varchar(100),
  auth_bio mediumtext,
  primary key (auth_lastName, auth_firstName, auth_middleName)
);
Once again, we will use the MySQL password function to encrypt the password in the database. We also keep track of the date when the customer registered on the site with the MySQL now function.

7 Now, you can retrieve the record you just added by using the SELECT command:

```
select * from User where email='jennylim@freemail.com';
```

The asterisk(*) denotes that all fields within the table should be retrieved. You can also specify the field names that must be returned, separated by commas.

8 The file load_abc.sql contains more INSERT statements for loading entries into the Book, Author and Customer table. You can run this script file using the source command:

```
source load_abc.sql
```

MySQL would treat the contents of load_abc.sql as commands to be executed.

9 Now, you can check the contents of the two tables by typing the following commands:

```
select * from Book;
select * from Author;
select * from Customer;
```

10 Another interesting thing to try is the JOIN operation, which allows you to query information from more than one table. Let’s create a SELECT statement which displays the author’s biography along with the rest of the information about a book:

```
select bookID, title, auth_firstName, auth_lastName,
auth_middleName, listPrice, monthPublished,
yearPublished, numPages, description, auth_bio
from Book, Author
where Book.author_lastName = Author.auth_lastName and
Book.auth_firstName = Author.auth_firstName and
Book.auth_middleName = Author.auth_middleName;
```

11 You can also create a view table that contains the result of joining the Book and Author tables from the previous step. A view table never actually exists physically. Instead, it is formed from the rows of the underlying base table(s) when the view table is specified in an SQL statement:

```
CREATE VIEW Book_with_Author_Info AS
select bookID, title, auth_firstName, auth_lastName,
auth_middleName, listPrice, monthPublished,
yearPublished, numPages, description, auth_bio
from Book, Author
where Book.author_lastName = Author.auth_lastName
and Book.auth_firstName = Author.auth_firstName
and Book.auth_middleName = Author.auth_middleName;
```
And now you can query the view as follows:

```sql
SELECT * from Book_with_Author_Info where auth_firstName = 'Mark' and auth_lastName = 'Twain';
```

12 Next, let’s try the UPDATE and DELETE commands. You can verify that the commands were executed correctly by using SELECT to query the tables before and after issuing the UPDATE and DELETE commands:

```sql
UPDATE Book SET listPrice = 15.50 where bookID = 5;
UPDATE Book SET listPrice = price * .75 WHERE bookID = 10;
DELETE FROM Author WHERE auth_firstName = 'Eileen' and auth_lastName = 'Chang';
```

13 Now, let’s try creating indexes on some fields other than the primary key. Indexes can speed up searching on fields which are commonly used as search criteria, such as the book title and the customer’s first and last name:

```sql
CREATE INDEX idx_title ON Book (title);
```

where `idx_title` is the name of the index, `Book` is the table and `title` is the field being indexed.

14 Here’s another command which indexes the Book table on the Description field. This field is used for full-text searching, so we have an additional keyword FULLTEXT in the command:

```sql
CREATE FULLTEXT INDEX idx_description ON Book (description);
```

15 Another command that can be used to create an index is the ALTER TABLE command:

```sql
ALTER TABLE Book ADD INDEX idx_author (auth_lastName, auth_firstName, auth_middleName);
```

16 The file `index_abc.sql` contains more statements for creating indexes for the tables in the ABCBooks database. You can run this script file using the `source` command:

```sql
source index_abc.sql
```

17 You can exit the MySQL command shell by typing quit or \q.

There are many other useful SQL commands we haven’t tried yet. However, the commands you’ve learned in this activity should be enough for the majority of the database operations you’ll need to perform.
Please refer to the SQL statement syntax reference provided at the start of this activity for a complete list of SQL commands available in MySQL.

As a summary, here are the SQL statements discussed in this activity:

1 Data definition (DDL)
   - CREATE TABLE;
   - CREATE VIEW; and
   - ALTER TABLE (changing field type and length, adding an index).

2 Data manipulation (DML)
   - INSERT;
   - UPDATE;
   - DELETE;
   - SELECT (single table); and
   - SELECT (join).

You’ve gone through a lot of database operations in this section! You’ve also built some of the data structures and loaded all the necessary data into the ABCBooks database. In the next section, you will learn how to issue requests to the database from within an application program. Please complete the following self-test to assess your understanding of SQL.

**Self-test 5.3**

1 What will happen if you issue the DELETE statement without specifying any criteria? Example: DELETE FROM AUTHORS;

2 Write an SQL statement that will create an index on the email field of the Customer table.

3 Write an SQL statement which retrieves all records from the Book table where the title contains the string Earth.

4 Identify two column types that may be used when defining date/time fields in a table. You can use this URL as a reference: http://www.mysql.com/doc/en/Column_types.html.
PHP and MySQL

Previously, you were issuing commands directly to the database server from a command-line client. In this section, we will incorporate the SQL commands within a PHP program which acts as a client to the database server. The PHP program runs as a server-side script on our Web server. It will process inputs from our Web users and generate dynamic pages from the database in response to their requests.

We will start the section by discussing the native PHP functions for accessing MySQL databases. Later on, we will also talk about the Open Database Connectivity (ODBC) programming interface and how this can allow the same program to work with many different databases (not just MySQL). We will end the section with a brief introduction to a programming approach which uses a database abstraction layer to centralize all the database access code within your server-side scripts.

ABC Books: our case study business

Here’s a breakdown of the functions which we will implement for ABC Books.

1  Searching for a book using a variety of methods:
   • by title — entire title or part of the title;
   • by author — various combinations of first name, last name and middle name (if any); and
   • by keywords — full-text search on book titles, book descriptions, author names and author descriptions.

2  Maintaining a shopping cart:

   Users can add a book to their shopping cart by clicking on an Add to Cart button. The quantity for each book is set to 1 by default but can be updated. Setting the quantity to 0 will remove the book from the cart.

   Once users have selected all the books they wish to purchase, they can proceed to Order Checkout. At this point, they will have to log in with their email address and password before they can continue processing the order. The User Login/Registration and Order Checkout functions will be covered later when we talk about session management.

PHP MySQL functions

PHP comes with an extensive set of built-in functions that allow it to interface with a wide range of DBMSs, including functions for MySQL,
mSQL, PostgreSQL, Oracle, Sybase and Microsoft SQL Server. The next reading describes the most commonly used MySQL functions in PHP.

**Reading 5.3**

‘Build your own database driven website using PHP & MySQL — Part 4: publishing MySQL data on the Web’, 

*Note:* You only need to go through pages 2 to 5.

Here are the relevant functions from the previous reading, along with sample code fragments:

1. **mysql_connect** — establishes a connection to a database. It returns the database connection handler if the connection is successful. If you want to reuse the connection even after the script has ended, you should use **mysql_pconnect** instead (*p* stands for persistent).

   The following sample code can be used to connect to the ABCBooks database with our webuser account. If the connection is unsuccessful, the **die** function will display the error message and terminate the program afterwards:

   ```
   $conn = mysql_connect(“localhost”, “webuser”, “mt834”) 
   or die(“Could not connect to MySQL”);
   ```

   Reading 5.3 also provides another way to catch errors as a result of MySQL calls. The author puts the ‘at’ sign (@) in front of the function name. Then he uses an **IF** statement to check whether the function returns a ‘true’ or ‘false’ value before displaying the error statement. This approach is fine, too.

2. **mysql_select_db** — selects the database to be used. It accepts two parameters — the name of the database and the database connection handler:

   ```
   $conn = mysql_connect(“localhost”, “webuser”, “mt834”) 
   or die(“Could not connect to MySQL”);
   $selected_db = mysql_select_db(“ABCBooks”, $conn) or 
   die(“Could not select database”);
   ```

3. **mysql_query** — sends an SQL query string to the database. It will return a result handler if the request is successfully executed:

   ```
   $conn = mysql_connect(“localhost”, “webuser”, “mt834”) 
   or die(“Could not connect:” . mysql_error());
   $selected_db = mysql_select_db(“ABCBooks”, $conn) or 
   die(“Could not select:” . mysql_error());
   ```
$result = mysql_query("SELECT * from Book");

4  mysql_fetch_object — fetches the next row in the result set as an object. The object’s properties correspond to the record’s fields. This function is usually used in a loop:

$conn = mysql_connect("localhost", "webuser", "mt834") or die("Could not connect:" . mysql_error());

$selected_db = mysql_select_db("ABCBooks", $conn) or die("Could not select database" . mysql_error());

$result = mysql_query("SELECT * from Book");

while ($row = mysql_fetch_object($result)) {
    echo ("ID " . $row->bookID . ", Title: " . $row->title . ", Date Published: " . $row->monthPublished . "/" . $row->yearPublished . "&lt;br&gt;");
}

5  mysql_close — closes non-persistent connections to the MySQL server and returns ‘true’ or ‘false’, depending on its success:

$conn = mysql_connect("localhost", "webuser", "mt834") or die("Could not connect:" . mysql_error());

mysql_close($conn);

In the next activity, you will install these two scripts on your local Web server: search_catalog.php and maintain_cart.php.

Activity 5.3

The files needed in this activity can be downloaded from the course website.

1  Open search_catalog.php and maintain_cart.php in a text editor. These programs will both connect to the MySQL server and perform operations on it. search_catalog.php will issue SELECT commands for retrieving records from the Book table, while maintain_cart.php will issue SELECT, INSERT, UPDATE and DELETE commands that will allow users to add, update and remove items from their shopping cart (Cart and Cart_Detail tables).

    Skim through the comments in these two programs to find out how they work. Comments in PHP code begin with //.

2  Now store these programs on your local Web server and make them executable using the chmod command.
3 Access `search_catalog.php` from your Web browser. This will display a search form which accepts two kinds of search criteria from the user — by title and by keyword. Try both kinds of searches and verify that it all works correctly.

4 You can add some books to your shopping cart by clicking the Add to Cart button next to the book within the search results. You can also try changing the quantity of a book.

5 Now, you will be asked to add more functionality to the two scripts:

   `search_catalog.php` — adds the capability to search on the author’s last name, first name and middle name.

   `maintain_cart.php` — removes the book from the shopping cart when its quantity is set to 0. Currently, the book remains in the cart even when the quantity is zero.

6 Here’s an overview of what you must do for `search_catalog.php`:

   • If any of these three fields were entered by the user, author first name, author last name or author middle initial, you must include them in the selection criteria that will be used for searching the Book table.

   • The selection criteria is built by appending fields to a string as new search fields are read from the user input.

7 Here’s an overview of what you must do for `maintain_cart.php`:

   • If the user enters a quantity of zero for any item in the cart, you must issue a DELETE command against the Cart_Detail table instead of simply updating the quantity field to zero.

8 There are comments within the PHP code which tell you where these commands should be inserted in the PHP script. Just look for any comments that begin with Activity 5.3. The model answer is also available on the course website.

Database programming in PHP is easy and convenient given the wide variety of built-in database functions that are available. For MySQL, just use the MySQL functions, for Oracle, use the Oracle functions, and so on.

**Open Database Connectivity (ODBC)**

The functions discussed in the previous section provide a highly efficient way to access MySQL databases from your PHP scripts. However, they come with one major limitation. Code written using PHP’s native
functions for MySQL will no longer work when you switch to another database product.

Consider the Search and Shopping Cart functions written for ABC Books in the Activity 5.3. If they were to switch to a PostgreSQL database in the future, the `mysql_connect` and `mysql_query` function calls would have to be rewritten using the PHP functions that are specific to PostgreSQL, i.e. `pg_connect` and `pg_query`, respectively.

ODBC makes it possible for applications to access databases in a standard and uniform way, regardless of the DBMS or operating platform used by the application. It achieves this by inserting a middle layer, called a database driver, between an application and the DBMS.

![Figure 5.8](image)

Figure 5.8 ODBC serves as a middle layer between an application and the DBMS

In the figure above, the application program acts as a client to the database server. An ODBC Data Source Name (DSN) (as shown in figure 5.9) containing the connection parameters for a particular data source must be defined on the client. The configuration file `odbc.ini` contains information about DSNs available to all users, and this is where you will add an entry for each new data source.
The connection parameters for a DSN are then passed to the appropriate ODBC driver, which communicates with the database server itself. Because the ODBC database drivers are loaded at run time, a user only has to install a new driver to access a new DBMS; it is not necessary to recompile or re-link the application. This approach allows a single application to access different DBMSs with the same source code.

You will also need to edit `odbcinst.ini` after installing a new database driver. This configuration file contains information about ODBC drivers available to all users.

In general, ODBC simplifies connectivity to a database because applications only need to support a single, common API rather than different APIs from various database vendors. However, this approach can increase processing time due to the intermediate step needed to translate between the native database calls and the ODBC API. If performance and speed are key issues for an application, then developers may have to resort to native database APIs at the expense of portability.

The next reading describes the process of installing the iODBC library package on your Linux machine. iODBC is an open source, platform-independent implementation of both the ODBC and X/Open SQL specifications. The reading also comes with sample scripts for issuing ODBC calls from PHP.
Reading 5.4

Using PHP with ODBC — A User Guide,

Note: Start reading from the section called ‘Downloading iODBC for Linux/Unix’.

Here are the relevant functions from the previous reading, along with sample code fragments:

1  odbc_connect — establishes a connection to an ODBC dataset name (DSN). If you want to reuse the connection even after the script has ended, you should use odbc_pconnect instead (p stands for persistent):

   $conn = odbc_connect("localhost", "webuser", "mt834")
   or die(odbc_error());

   odbc_error is an ODBC function which returns the error state and error message.

2  odbc_prepare — parses and compiles the statement, and then the odbc_execute function (see below) is used to run it later on. This is recommended for statements which will be executed several times (e.g. within a loop), as it can boost performance:

   $conn = odbc_connect("localhost", "webuser", "mt834")
   or die("Could not connect: " . odbc_error());

   $sql = "select * from Book where listPrice > 10.00";

   $result = odbc_prepare($conn, $sql) or die("Couldn't prepare statement:" . odbc_error());

3  odbc_exec and odbc_do — these two functions mean the same thing — they will execute an SQL statement on the specified database connection.

   odbc_exec and odbc_do can also be used to parse, compile and then immediately execute an SQL statement in one step. This approach is fine for SQL statements which will only be run once. Statements which will be run multiple times should be prepared beforehand via odbc_prepare so that they do not have to be parsed and compiled repeatedly for each execution.

Example 1  odbc_execute with odbc_prepare

   $conn = odbc_connect("localhost", "webuser", "mt834")
   or die("Could not connect: " . odbc_error());

   $sql = "select * from Book where listPrice > 10.00";
$result = odbc_prepare($conn, $sql) or die("Could not prepare: ". odbc_error());

odbc_exec ($result) or die("Couldn't execute SQL: ". odbc_error());

**Example 2**  odbc_exec without a previous odbc_prepare

$conn = odbc_connect("localhost", "webuser", "mt834") or die("Could not connect: ". odbc_error());

$sql = "select * from Book where listPrice > 10.00";

$result = odbc_exec($conn, $sql) or die("Couldn't execute SQL: ". odbc_error());

4  odbc_fetch_row — fetches the next row from a result set that was returned by odbc_do or odbc_exec. It returns false when there are no more rows left. The resulting fetched row can be accessed via the odbc_result function:

$conn = odbc_connect("localhost", "webuser", "mt834") or die(odbc_error());

$sql = "select * from Book where listPrice > 10.00";

$result = odbc_exec($conn, $sql) or die("Couldn't execute SQL: ". odbc_error());

while (odbc_fetch_row($result)){
    $field1 = odbc_result($result, "title");
    $field2 = odbc_result($result, "auth_firstName");
    $field3 = odbc_result($result, "auth_lastName");
    $field4 = odbc_result($result, "auth_middleName");
    $field5 = odbc_result($result, "listPrice");

    echo "Title: $field1, Author: $field2 $field4 $field3, Price: $field3
"
}

5  odbc_close — takes a specific connection ID and closes this ODBC connection. You are not required to include this function since any open database connections are automatically closed when your PHP program finishes execution.

In the next activity, we will rewrite the two scripts from Activity 5.3 — search_catalog.php and maintain_cart.php — using ODBC function calls instead of MySQL-specific function calls.
Activity 5.4

The files needed for this activity can be downloaded from the course website.

1. You can verify whether ODBC has already been installed on your local Web server by accessing the `phpinfo.php` server-side script from your Web browser. This script was originally mentioned in Activity 3.6 of Unit 3.

If all goes well, part of your output should look like Figure 5.11.

<table>
<thead>
<tr>
<th>ODBC Support</th>
<th>enabled</th>
</tr>
</thead>
<tbody>
<tr>
<td>Active Persistent Links</td>
<td>0</td>
</tr>
<tr>
<td>Active Links</td>
<td>0</td>
</tr>
<tr>
<td>ODBC library</td>
<td>unixODBC</td>
</tr>
<tr>
<td>ODBC_INCLUDE</td>
<td><code>-l/usr/local/include</code></td>
</tr>
<tr>
<td>ODBC_LFLAGS</td>
<td><code>-L/usr/local/lib</code></td>
</tr>
<tr>
<td>ODBC_LIBS</td>
<td><code>-lodbc</code></td>
</tr>
</tbody>
</table>

Figure 5.11 Output if ODBC is already installed on your local Web server

2. If ODBC has not yet been installed, you must download and install the iODBC library package on your local machine. Detailed instructions can be found at http://www.iodbc.org/index.php?page=languages/php/odbc-phpHOWTO.

You can verify that the installation was successful by running `phpinfo.php` on your Web browser again.

3. Next, edit the following ODBC configuration files using any text editor:

   - `odbcinst.ini` — must contain a section for the MySQL database driver; and
   - `odbc.ini` — must contain a section for your new data source name (ABCBooks).

   For more information on what goes into these two configuration files, you can consult http://www.unixodbc.org/odbcinst.html.

   Your entries should look similar to the samples shown in Figure 5.9 and 5.10, but perhaps with different file paths.

4. Open `search_catalog_odbc.php` and `maintain_cart_odbc.php` in a text editor. These scripts are similar to the ones from Activity 5.3, but all database calls have been
ported to ODBC. Skim through the comments in these two programs to find out how they work.

5 Store these two programs on your Web server and make them executable using the `chmod` command. Test these two scripts by searching for books and adding them to your cart.

6 Now modify `maintain_cart.php` so that it calls the `odbc_prepare()` function to precompile any SQL statements which will be executed repeatedly (i.e. within a loop). There are comments within the scripts which will guide you on how to do this.

7 Test your changes by searching for books and adding them to your cart.

The model answer is also available on the course website.

We’ve discussed how databases can be accessed from PHP by using native database functions and the ODBC Standard API. With ODBC, the same program will work regardless of the back-end database used. However, ODBC could result in slower and less efficient performance due to the middle layer that is inserted between the database and the application program.

In the next section, we will briefly introduce another programming approach that can be used for database access within PHP.

**Database abstraction**

The main reason for using ODBC is to avoid having to rewrite and retest your programs if you should ever switch to a new database. Another way to avoid this unpleasant situation is to create a database abstraction layer. Database abstraction will create a centralized body of code which handles all interaction with the database. All SQL statements are contained in this centralized body, so any changes only need to be made to this small set of code rather than all over the place. Ideally, a programmer who uses a database abstraction layer does not need to know the details of the underlying database.

Even if you do not anticipate switching to another database in the future, a database abstraction layer is still a good idea. Centralizing database access results in cleaner and easier-to-manage code.

One way to abstract the database is to wrap each API-specific function in a more generic function called `numRows`. In the example below, the `mysql_num_rows` function is wrapped in another function called `numRows`. PHP application programs would then call the `numRows` function instead of using the API-specific function directly.
If we were to switch to PostgreSQL in the future, we would only have to port the function call from `mysql_num_rows` to `pg_numrows` in one place.

```php
function numRows($result) {
    //Return the number of rows in the result set.
    return (@pg_numrows($result));
}
```

Aside from writing your own database abstraction layer, there are several that can be downloaded from the Web for free. One of them is PEAR DB, which consists of a set of classes that contains our database functions. It also comes with an object-oriented style query interface. Read the following optional reading if you wish to know more.

**Reading 5.5 (Optional)**


This concludes our section on accessing MySQL databases from PHP. Do the following self-test to check your understanding of this topic.

**Self-test 5.4**

1 Refer to the MySQL documentation for details on the `mysql_fetch_assoc` function. This can be found under the ‘Table of contents’ section at http://hk.php.net/manual/en/ref.mysql.php.

   Now rewrite the following code using `mysql_fetch_assoc` instead of `mysql_fetch_object`:

   ```sql
<table>
<thead>
<tr>
<th>Line number</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$result = mysql_query(&quot;SELECT * from Book&quot;);</td>
</tr>
<tr>
<td>2</td>
<td>while ($row = mysql_fetch_assoc($result)) {</td>
</tr>
<tr>
<td>3</td>
<td>echo (&quot;ID &quot;, $row-&gt;bookID , &quot; , Title: &quot;, $row-&gt;title , &quot;, Date Published: &quot;, $row-&gt;monthPublished , &quot;/&quot; , $row-&gt;yearPublished , &quot;&lt;br&gt;&quot;);</td>
</tr>
<tr>
<td></td>
<td>}</td>
</tr>
</tbody>
</table>
   ```
2 Rewrite the code above so that it checks whether `mysql_query` was executed successfully and whether at least one record was found before fetching the result rows.

3 List one advantage and one disadvantage of using ODBC over using native database function calls.

4 Which ODBC system configuration files must be edited whenever a new data source name (DSN) is defined or whenever a new database driver is installed?

5 Will using a database abstraction layer offer better performance than ODBC? Why or why not?
Session management mechanisms

Session management refers to the ability to recognize a user throughout all the pages that they visit on a website. This is achieved by allowing the state of a user session to be saved and remembered. A user session begins when a user clicks onto a site for the first time, and ends when they move on to another site. A session will end if the site has not ‘heard’ from the user from a while, for example, within 20 or 30 minutes.

In this section, we will describe some common techniques for storing session state within a Web application.

One of the challenges faced by Web developers is how to create a coherent application out of a series of independent HTML pages. This is particularly important in Web development because HTTP is a stateless protocol. Each browser request to a Web server is discrete and independent, and the server does not retain any knowledge about previously serviced requests from the same browser.

This is illustrated in following figure.

When the same user connects again with another request a few seconds later, the server regards it as a totally new interaction. If necessary, the Web application has to reconstruct the user’s state information all over again. Any data previously supplied by the user (through online forms, for example) is not automatically available on subsequent requests.

The stateless, connectionless nature of HTTP is fine for static webpages. However, many applications deployed on the Web require a user-specific state to function properly. Although maintaining a client state is not officially part of the HTTP protocol, Web programmers often have to implement session management techniques in order to provide consistent user sessions on the Web.
Session management takes care of correlating an HTTP request with other previous requests generated from the same session. It also keeps track of user-specific information that needs to persist throughout a session. In effect, session management allows the Web server to maintain session state on top of HTTP, a sessionless and stateless protocol.

Session state consists of all the information needed to reconstruct what has taken place so far during the user’s session. For a shopping cart application, this might consist of a shopping cart ID which correlates to a shopping basket containing the list of items selected on the server. For a Web-based email application, this may consist of the username that was entered during the log-in process. Unique session identifiers (IDs) could be generated in order to track users across several page requests.

Developers must decide which variables need to be persistent throughout a session. Another critical decision is where to store the session information. You can choose to store it on the client-side, the server-side, or a combination of the two.

**Client-side techniques**

Client-side techniques maintain the session state on the client machine. The data is appended to both the HTTP request and HTTP response messages that are exchanged between the client and server.

**Hidden forms and embedding data with URLs**

When a user interacts with a website, the server can store the session data in hidden form elements or as part of the URL query string. The data is sent to the client as part of the HTML page.

The next time the user clicks the **Submit** button for the form or clicks on the link containing the embedded data, the session state will be passed back to the server as part of the HTTP request submitted by the browser. The server can then use the session data to remember what has gone on previously for this particular user session.

http://www.ourstore.com/shop/cart.pl?id=097f81a043a905e8fb6e4c100f0c5411&mode=R

**Figure 5.13** Session state is embedded in the URL contained within the HTML page
This approach can be used for simple data that needs to be exchanged between the client and server. Passing values as part of the URL works for small amounts of information, but will be inadequate for larger quantities of data.

For security reasons, hidden form fields and URL rewriting should not be used for passing confidential state information. These values can easily be stolen by anyone who views the HTML source. You can combine this technique with one of the server-side techniques so that the confidential portion of your session data is stored on the server side only.

**Cookies**

A cookie is a piece of information stored on the client on behalf of the server. Cookies originate from the Web application running on the server. They are sent to the client as part of the server's response to an HTTP request. The client sends the cookie back to the server along with every HTTP request, so the Web server can use cookies to retain state information about a particular session.

Cookies are an ideal way to distinguish a series of HTTP requests that come from the same user. When a user establishes a session for the first time, the HTTP request is received without a cookie. The server application generates a new unique identifier and puts this in a cookie, which becomes part of the response sent back to the client. The identifier, or session ID, is then sent with every subsequent request.

Similar to hidden form fields and URL rewriting, cookies come with size and security constraints. A cookie can only contain up to 4,096 bytes of data, and each Web domain can only store up to a maximum of 25 cookies on the client. For security reasons, cookies can only be read by the website which stored them on the machine.
Cookies are also stored in a known location on the client’s local drive. For example, Internet Explorer uses the C:\Documents and Settings\<username>\Cookies folder. As you can see, cookies should not be used to store sensitive information which may be sniffed by potential intruders.

Another problem with cookies is that not all browsers support them, and support for cookies can be disabled within the browser as well. Session management techniques that rely extensively on cookies will not work on such browsers.

Cookies have gained a certain notoriety with the public because they are perceived as a threat to personal privacy. Some users may prefer to be totally anonymous when they are online, and they do not feel comfortable knowing that a website is storing information about them every time they visit. Technology also allows companies to develop highly detailed user profiles by tracking the same user as they visit multiple websites or by combining information from cookies with other data sources.

E-commerce sites and marketers argue that cookies are necessary to provide a more interactive and personalized experience. If you want to know what kind of information is being stored in your cookie file, you can check out the Privacy Policy or Notice on a website.

PHP comes with built-in functions for handling cookies. Here’s some of the most important cookie-related functions, along with sample code fragments:

```php
setcookie — sets a new cookie and initializes it with a given value if no cookie with the same name exists. If a similarly-named cookie already exists, its value will be updated. In the following example, the program keeps track of the number of times a user has visited the page through a cookie named ‘count’.
```
<?php
//Read the cookie from the superglobal array $HTTP_COOKIE_VARS.
$count = $HTTP_COOKIE_VARS['count'];
if (!$count) {
    //If this is the first time, then the cookie will be initialized to one.
    $count = 1;
} else {
    //Otherwise, increment the cookie by 1.
    $count++;
}
//Store the value of count in a cookie called 'count'.
setcookie("count", $count);
?>
<html>
<head><title>MT834 Activity 5.5</title></head>
<body>
Thank you for visiting! You've seen this page

<?php
//Display the number of times a user has accessed this page.
echo($count);
if ($count == 1) {
    echo (" time.<br/>");
} else {
    echo (" times.<br/>");
}
?>

Click <a href="reset_cookie.php">here</a> to reset the counter or <a href="init_cookie.php">reload</a> the page to update the counter.
setcookie can also be used to delete a cookie. Just pass the name of the cookie that is to be removed to it without including a value. The next example shows how the cookie named ‘count’ can be deleted.

**Program 5.2 reset_cookie.php**

```php
<?php
    //Delete the cookie named "count".
    setcookie("count");
?>
</html>
<head><title>MT834 Activity 5.5</title></head>
<body bgcolor="#FFFFFF">
    The counter has been reset to zero.
    Click <a href="init_cookie.php">here</a> to set the cookie again.
</body>
</html>
```

In the next activity, you will revisit the Search and Shopping Cart functions from Activity 5.4. These programs make use of a cookie to keep track of a user’s shopping cart ID.

---

**Activity 5.5**

Download the two scripts — init_cookie.php and reset_cookie.php — from the course website. These two scripts were explained in Programs 5.1 and 5.2 above.

Install these scripts into your local Web server and test them out yourself.

Now modify the two scripts so that we are also able to keep track of the date and time when the user last accessed the page.

1. **init_user.php** — assign the date/time when the page was last accessed to a cookie called n_last. Modify the message shown to the user so that it also displays the date/time when the page was last accessed. Use the `date()` function to retrieve the current date and time as shown below.

Thank you for visiting! You've seen this page 9 times.
The last time you accessed this page was on 16:05:14 21/04/2004. Click here to reset the counter or reload the page to update the counter.
reset_cookie.php — delete these two cookies: count and n_last.

The model answer can be viewed on the course website.

We’ve looked at three client-side methods for session management in this section: URL embedding, hidden form fields and cookies. Answer the following self-test to assess your understanding of these session management techniques.

Self-test 5.5

1. Why is it not advisable to remember confidential information in a hidden form field or through a URL query string?

2. Describe the reasons why some users may refuse to accept cookies.

Server-side techniques

Cookies are not always dependable for managing sessions because users may run cookie-blocking programs or configure their browsers to refuse cookies. To get around this difficulty, PHP provides us with the capability to store cookie files on the server instead of the browser.

Using PHP sessions

A PHP session is similar to a cookie, but it is stored in a temporary file on the server (e.g. /tmp/ directory on UNIX systems). The cookie file name is the same as the user’s session identifier (ID). Each user session can store variables that must be preserved from one page to another in this file. An example of a session variable is the first name and last name of the user who has just logged in. This information can then be carried throughout all the pages accessed by this user on the site.

In order to propagate the session ID to the Web client, the PHP interpreter will attempt to send the ID through a cookie. If the browser does not accept cookies, PHP’s built-in session management system will automatically embed the ID in all URLs within the returned page.

Session functions in PHP

In this section, we’ll take a closer look at some of the PHP functions which handle session management.

The session_start function is used to specify the start of a PHP session. The function tells PHP to check whether the user request
contains a session ID, either as a cookie or as a query string. If a session ID is found, PHP will reload into main memory all the variables that belong to this session. If a session ID has not yet been assigned (i.e. this is the first page accessed by the user), then a session ID is assigned and a new cookie file is created with the same name as the session ID. Please note that you should call the `session_start()` function before anything is output to your Web browser, or you’ll get a nasty error message.

The `session_register()` function is used to register a variable with a given name. All variables registered for a session are stored in a superglobal array called `$_SESSION`. In the example below, a session variable called `user` is initialized with the value `duddy`. This variable will now be available to other pages coming from the same browser, as long as they also call `session_start()` to indicate that they are part of the session.

Program 5.3 `init_user.php` starts a new session, initializes and registers a session variable called `user` and then reads this variable from the `$_SESSION` superglobal array.

```
1  <
2     session_start();
3     $user = "duddy";
4     if (session_register("user")) {
5         echo ("User field set to " . $_SESSION["user"] . ".");
6     } else {
7         echo ("Could not set the session variable.");
8     }
9  ?>
10 Click <a href="show_user.php">here</a> for the next page.
```

From `init_user.php`, you can click on a link to visit `show_user`. Then `php.show_user.php` will display the name of the user that was originally registered in `init_user.php` by retrieving it from a session variable.

Program 5.4 `show_user.php`

```
1  <
2     session_start();
3     $user = $_SESSION["user"];  
4     echo("Welcome to the user area, $user!");
5  ?>
```

Here are the actions performed by `init_user.php`:
• Line 1 initializes the session. Thus, if the incoming request already contains session information, the old session will be restored. Otherwise, a new session is created.

• Line 4 registers the variable user, which has been assigned the value ducky.

• Line 5 confirms that the variable was successfully stored as a session variable by reading it from the $_SESSION superglobal array.

• Line 10 generates a link to show_user.php. Note that if the browser is set to reject cookies, then the interpreter will attach a session ID at the end of this URL. The interpreter will only do this for relative URLs such as this.

The operations of show_user.php are:

• Line 2 starts a new session if the incoming request does not contain session information, otherwise it restores the old session. This keeps the ‘connection’ going between the two programs — init_user.php and show_user.php.

• Line 3 reads the session variable user from the $_SESSION superglobal array. Since it is part of a session, it is able to recover the value that was previously saved by init_user.php in this variable.

---

**Activity 5.6**

Download the two scripts — init_user.php and show_user.php — from the course website. These two scripts were explained in Programs 5.3 and 5.4 above.

Install these scripts into your local Web server and test them out yourself.

Now modify the two scripts so that the user’s email address becomes a session variable as well:

1. init_user.php — assign the value test@test.com to a variable called email. Register this as a session variable, along with user.

2. show_user.php — retrieve and display the email address along with the user name from the $_SESSION superglobal array.

The model answer can be viewed on the course website.

Next, we’ll look at three more functions that you’ll often need when using PHP sessions: removing a session variable, destroying a session
(and all the variables associated with it), and retrieving the user’s session ID.

PHP gives you the ability to remove a single session variable by assigning a blank or a ‘false’ Boolean value to it, as shown in the following code fragment.

**Program 5.5  remove_user.php**

```php
<?php
session_start();
$_SESSION['user'] = FALSE;
if($_SESSION['user']){
    echo "The session is still registered.<br /><br/>";
} else {
    echo "Ok, the session is no longer registered! <br />"
    echo "<a href="init_user.php">Start Session Again</a><br/><br/>";
}
?>
```

Session files are deleted when the user closes his/her browser. However, the entire session may need to be removed when users log out of your application, even if they have not yet closed the browser. The `session_destroy()` function deletes all traces of a user session.

In the code fragment below, the values within the `$_SESSION` array are cleared out first before the call to `session_destroy()` is made. We must still run `session_start()` so that PHP knows which session to destroy.

**Program 5.6  destroy_session.php**

```php
<?
session_start();
$_SESSION = array();
session_destroy();
?>
```

Lastly, there’s a function in PHP called `session_id()` that allows you to display the current session ID. Your session ID consists of a long string of alphanumerics such as 097f81a04a905e8f6e4c100f0c5411. This corresponds to the name of the temporary file where the session variables for a particular user is stored.
### Program 5.7 show_sessid.php

```
<?
session_start();
echo "Your session ID is <strong>" . session_id() . "</strong>";
?>
```

We’ve just covered the basics of PHP sessions such as starting a session, registering session variables, assigning values to session variables, unregistering a session variable and destroying a session. In the next session, we’ll look at another server-side technique used for session management: databases.

### Databases

Size and durability of session data are two important issues in session management. The techniques we’ve seen so far, such as cookies, URL embedding and PHP sessions, only allow limited amounts of state information to be persisted. These techniques also do not ensure the durability of session data. For example, PHP session data kept in the Web server’s process memory can be lost when the Web server shuts down. Cookies and URL query strings can be deleted or tampered with on the client-side.

Databases are a good solution whenever size and durability of data are a primary concern. Tables can be created within the database to store session data within record fields. For cases where session data is too simple to warrant the additional processing overheads incurred by databases, flat-files could be used instead.

In the next activity, you will implement the remaining functions on the ABC Books website: User Registration and Order Checkout. These functions make use of PHP session variables to remember the user’s first name and email address throughout all pages.

The contents of a user’s shopping basket also need to be saved as part of the session. However, it’s not a good idea to store these as PHP session variables because the contents of a user’s shopping cart can grow very quickly, and each item comes with associated fields such as book ID, quantity and price. This type of data can be handled better by storing it in a table within the database.

Here are the two tables which will be added to the ABCBooks database for storing the shopping cart details for each user: Cart_Header and Cart_Detail. When the customer enters the Order Checkout process, further details will be added to Cart_Header, such as their billing address, shipping address and payment details.

The PHP session ID is stored as a foreign key in the `sessionID` field of the Cart_Header table. This is how we will relate returning users with their respective baskets.
You’ve previously used `maintain_cart.php` and `maintain_cart_odbc.php` in Activities 5.3 and 5.4, respectively. These two scripts make use of the `Cart_Header` and `Cart_Detail` tables to store, retrieve and update the shopping cart contents.

In the next activity, we will complete the session management functions for the User Login and Order Checkout functions.

### Activity 5.7

The files needed for this activity can be downloaded from the course website.

1. Download the three scripts `maintain_cart.php`, `user_login.php`, and `order_checkout.php` from the course website. You’ve seen `maintain_cart.php` before. The version you just downloaded will redirect the user to `user_login.php` if they are trying to add an item to their cart without logging in first.

   The other two scripts are new, and they will perform the following functions:

   - `user_login.php` — will be executed whenever a user tries to add an item to the shopping cart for the first time. The user is asked to log in with a username and password, or they can be redirected to a registration page if they have not yet registered on the site.

   - `order_checkout.php` — from the shopping cart page (`maintain_cart.php`), the user can click on an Order Checkout button which will begin the order checkout process. The user will enter his billing address, shipping address and payment information. Once the user has confirmed the order, the order is generated in the database and the shopping cart is deleted.

   Skim through the comments in these two files to find out how they work.

2. Install these scripts into your local Web server and test them out yourself. After you have logged in, add items to your cart and then go through Order Checkout, all the way to confirming the order. You can enter this test VISA card number as payment information: 4111111111111111.

---

**Cart_Header** (cartID, customerId, sessionID, start_date, start_time, bill_address1, bill_address2, bill_district, bill_region, bill_phone, ship_address1, ship_address2, ship_district, ship_region, ship_phone, ccard_no, ccard_name, ccard_expiration)

**Cart_Detail** (cartID, cartLineNo, bookID, quantity, listPrice, discount)

---

**Figure 5.16** The user’s shopping cart will be stored in the `Cart_Header` and `Cart_Detail` tables. The primary keys for each table are underlined.
3 Once you have confirmed the order, the session is supposed to be destroyed. There is a call to the `session_destroy()` function in `Order_Checkout.php`, however, this will only delete the session variables stored in the server-side cookie. You must modify `Order_Checkout.php` so that it also deletes the Cart Header and Details from the database. There are comments within the code that guide you on how to do this.

4 Save and test your changes. You can query the database through an interactive SQL session to verify that the user’s shopping cart has been deleted along with the session.

The model answer can be viewed on the course website.

We’ve covered several session management techniques in this section. Under client-side techniques, we talked about URL embedding, hidden form fields and cookies. Under server-side techniques, we discussed PHP sessions and databases. Complete the following self-test to assess how well you understand these various techniques.

**Self-test 5.6**

Describe the main differences between propagating a session ID to the client by URL embedding, cookies, PHP sessions and databases.
Issues in state management

Now, let’s examine more closely the critical issues that must be considered when planning and implementing session management: the security, availability, scope and durability of state information throughout an entire session.

You’ve already gotten a glimpse of the security issues that exist with session management. We need to ensure the security of session data at all times, whether it is stored on the server or the client, and while it is being transmitted between the server and the browser.

Session management techniques must also be robust enough to ensure the availability and durability of session data throughout an entire session. There must be a facility to ensure the continued availability of session data, even when a Web server goes offline. Session management should also work well in distributed server environments.

Ensuring security

Secure sockets layer (SSL) technology can be used to encrypt all the information in both the HTTP request and HTTP response. This could prevent sniffing of confidential data by eavesdroppers as it travels over the Internet. Encryption and decryption of data on both ends of the secure communication channel will add a noticeable processing overhead to your website, so you may want to choose only a subset of your webpages to be SSL-enabled.

Session hijacking is another security issue. Hackers could potentially capture the session ID from a cookie or from embedded data in URL strings. They could then submit valid HTTP requests containing this session ID and gain access to the user’s private data. This could be prevented by keeping the lifetime of session data as short as possible. Session information should not be stored longer than necessary. Idle sessions should be discarded after a specified timeout period, typically set to 30 minutes. Session IDs can also be changed periodically to reduce exposure, but this should be as transparent as possible to the user.

The Web server itself must also be kept secure using techniques such as firewalls, packet filtering, intrusion detection and others. Additionally, sensitive data that is persisted in files and databases must never be stored in cleartext.

Managing sessions across multiple servers

High-traffic websites often require a cluster of Web servers, or a Web farm, to handle huge volumes of incoming requests. Session management becomes an even bigger challenge in this case because user requests will not always be routed to the same server. Instead, load-balancing software sends a request to whichever server is free. And yet, browsers need to
access pages from the same Web server where the session information was stored.

One approach is to ensure that all requests within a user session are directed to the same Web server within a Web farm. The user will initially access an application by requesting the general URL, for example, http://www.ourstore.com. The load balancer then routes the request to a specific server, for example, server3.ourstore.com. The session data is created on this server, and the browser can be redirected to the homepage on this particular server. If all links in the application pages are relative, future requests will be routed to the same server.

In adopting this approach, however, scalability is compromised since the workload may no longer be uniformly distributed among multiple servers.

A more robust alternative is to perform the session management using a back-end database server accessible to all the servers in the farm. Databases allow you to maintain more state information in a durable format. Each user will be given a unique identifier that will serve as a key to the user's information in the database. The identifier could be stored in a cookie on the user’s machine. This is a good example of using combined client and server-side techniques for session management.

**Figure 5.17** Using a common back-end database for all the servers in the Web farm

---

**Self-test 5.7**

1. Why is it challenging to implement session management techniques on a Web farm?

2. Identify two ways to ensure the security of data that is persisted within a Web session.
Summary

In this unit, we initially discussed the history and evolution of databases. We focused particularly on the relational data model since this is the dominant model in use on the Web today. We also talked about relational database concepts such as entities, attributes, tables, fields, primary keys and foreign keys. We gained hands-on experience in administering and manipulating a MySQL database using interactive SQL commands.

Next, we discussed how PHP programs can be written to access a MySQL database over the Web. Here are the three methods that were covered:

- PHP MySQL functions;
- ODBC functions; and
- database abstraction.

PHP MySQL functions are the fastest and most efficient method, but are the least portable. ODBC provides a standard API that can be used to access databases so that no recoding is needed to allow the same program to work with many different databases. However, the extra translation step performed by the ODBC layer makes this method less efficient. Database abstraction is not exactly a method for accessing databases, rather, it is a good programming practice which attempts to centralize all database function calls in one place. This results in easy maintenance of code since you only have to make changes in a single location.

In this unit, we also presented various ways of propagating session information to the client:

- embedding it in a URL;
- embedding it in a hidden form field;
- cookies;
- PHP sessions; and
- databases.

The first three methods involve storing session information on the client machine, while the last two methods store the data on the server. In reality, a combination of client and server-side techniques are often used, such as cookies plus databases.

Lastly, we looked at issues that must be kept in mind when implementing session management, such as security and the ability to manage sessions across multiple servers in a Web farm. As you can see, designing a session management solution depends largely on the type of Web application you’re building, the kind of session information to be stored, and on critical issues such as size, security, data availability and durability.

In the next unit, we will look at the many factors that could affect website performance, and how caching and optimization can alleviate these performance problems.
Suggested answers to self-tests

Self-test 5.1

1 The problems encountered when using the flat-file structure for organizing data on secondary storage include:

- data redundancy and confusion — the same data may be recorded in several places;
- program-data dependence — data about the file itself (e.g. metadata such as field lengths and data types) are encoded into the programs themselves; and
- lack of flexibility — end users do not have the capability to access and manipulate the file directly.

2 Using the relational model, data is organized into tables which contain information about one specific entity (e.g. Customer, Order, Book). Tables are made up of records, and each record consists of fields.

3 A primary key is made up of a field or a combination of fields that is unique for each record in a table. Each table is required to have a primary key. A foreign key is a field or combination of fields which is common to two or more tables, and is the primary key in one of these tables. Foreign keys are used to look up information in another table.

4 bookID is a foreign key which links the Order Detail table to the Book table. auth_firstName, auth_lastName and auth_middleName link the Book table to the Author table.

Self-test 5.2

1 Here are the steps for adding a new database user account to MySQL:

- Use the SQL Insert command to add the new user record in the user table of the mysql database.
- Use the FLUSH PRIVILEGES command to instruct the database server to recognize this new user immediately.
- Use the SQL Grant command to grant database access rights to this new user.

2 Here are four values displayed by the mysqladmin status command:

- uptime — number of seconds MySQL has been running since the last server restart;
- threads — number of active clients;
• questions — number of questions (queries) from clients since the server was started; and

• opens — number of tables the server has opened.

3 Here is the mysqlshow command which shows information about the Author table only:

mysqlshow ABCBooks Author

Self-test 5.3

1 Issuing the DELETE statement without specifying any criteria will delete all the rows in a table.

2 Here’s an SQL statement that will create an index on the email field of the Customer table:

    CREATE INDEX idx_email ON Customer (email);

Here’s another possible SQL command:

ALTER TABLE Customer ADD INDEX idx_email (email);

3 Here’s an SQL statement which retrieves all records from the Book table where the title contains the string Earth:

    SELECT * FROM Book where Title like ‘%Earth%’

4 Here are two column types that can be used for MySQL date or time fields:

    DATE — supports date values in ‘YYYY-MM-DD’ format, with a range of ‘1000-01-01’ to ‘9999-12-31’.

    TIME — supports time values in ‘HH:MM:SS’ format, with a range of ‘-838:59:59’ to ‘838:59:59’.

Self-test 5.4

1 The mysql_fetch_assoc function will return a record from the result set as an associative array, or an array indexed by field names rather than by numbers. Here is the code rewritten using mysql_fetch_assoc instead of mysql_fetch_object:

<table>
<thead>
<tr>
<th>Line number</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$result = mysql_query(&quot;SELECT * from Book&quot;);</td>
</tr>
<tr>
<td>2</td>
<td>while ($row = mysql_fetch_assoc($result)) {</td>
</tr>
<tr>
<td>3</td>
<td>echo (&quot;ID &quot;. $row[‘bookID’] . &quot;, Title: &quot;. $row[‘title’] . &quot;, Date Published: &quot;. $row[‘pub_month’] . &quot;/&quot; . $row[‘pub_year’] . &quot;&lt;br&gt;&quot;);</td>
</tr>
</tbody>
</table>

} |
The newly added IF statement in line 2 is used to test whether $result is true and whether there was at least one row returned by the SQL command (via mysql_num_rows function call).

<table>
<thead>
<tr>
<th>Line number</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$result = mysql_query(&quot;SELECT * from Book&quot;);</td>
</tr>
<tr>
<td>2</td>
<td>if ($result &amp;&amp; (mysql_num_rows($result) &gt; 0)) {</td>
</tr>
<tr>
<td>3</td>
<td>while ($row = mysql_fetch_assoc($result)) {</td>
</tr>
<tr>
<td>4</td>
<td>echo (&quot;ID &quot; . $row-&gt;{'bookID'} . &quot;, Title: &quot;. $row-&gt;{'title'} . &quot;, Date Published: &quot;. $row-&gt;{'pub_month'} . &quot;/&quot; . $row-&gt;{'pub_year'} . &quot;&lt;br&gt;&quot;);</td>
</tr>
<tr>
<td>5</td>
<td>} else {</td>
</tr>
<tr>
<td>6</td>
<td>echo &quot;No matches were found.\n&quot;;</td>
</tr>
<tr>
<td>7</td>
<td>}</td>
</tr>
</tbody>
</table>

An advantage of ODBC is that programs do not have to be rewritten in order to work with another database.

A disadvantage of ODBC is that performance is slower and less efficient due to the extra translation step.

odbc.ini must be edited whenever a new data source name (DSN) is defined. This file contains information about all the data source names that can be used by the users on this machine.

odbcinst.ini must be edited whenever a new database driver is installed. This file contains information about all the database drivers that are available on the system.

A database abstraction layer inserts an extra layer of code around the native database API functions used. This will add slightly to the amount of processing that occurs with every database call, but the processing will not be as extensive as the translation step (ODBC API <-> native database API) that occurs with ODBC.

**Self-test 5.5**

1. Information passed through a hidden form field or through a URL query string can be viewed in the HTML source of a webpage.

2. Users may refuse to accept cookies due to security and privacy concerns:
   - Security — they may suspect, rightly or wrongly, that cookies contain viruses and other malicious code. Cookies stored in unencrypted form on disk may also be vulnerable to hackers.
• Privacy — they may not want websites to form detailed profiles from their surfing patterns, in combination with files collected from other sources (e.g. advertising networks and third-party data brokers).

Self-test 5.6

The main differences are:

• URL embedding — the session ID is passed through the URL query string. It’s possible for the same user to have different sessions on the same machine since a new session ID can be assigned to the user whenever the session ID is not passed in the query string. If a session is configured to time out after a very long time, then the bookmarked URL containing the session ID can be used to reenter the old session immediately.

• Cookies — the session ID is saved in a cookie file on the client machine and is passed back to the server whenever the browser makes a request. Cookies are specific to one machine, not to each user, so only one session can be active for any machine at a given time. Cookies with a very long expiry date can be used to allow users to reenter an existing session even after they’ve been inactive for a while.

• PHP sessions — a set of session variables are stored in a server-side cookie. The server-side cookie file name is the same as the session ID assigned to the user. PHP sessions can be set up and destroyed through built-in PHP session functions.

• Databases — a set of session variables are stored in the database. The PHP session ID is used to link the database record with a returning user. Session variables must be deleted from the database when the user session ends.

Self-test 5.7

1 There is no guarantee that a user will always be routed to the same Web server on a Web farm, which makes it hard for session data to be preserved on the server-side.

2 Encryption and session timeouts are two ways to ensure the security of data within a Web session.
References


