

2221956 ACCA5026

by Ryan Hopkins

Submission date: 14-Jan-2025 02:43PM (UTC+0000)

Submission ID: 248425409

File name: 51220_Ryan_Hopkins_2221956_ACCA5026_630034_991022639.docx (193.93K)

Word count: 848

Character count: 4328



DATABASES AND APPLICATION DEVELOPMENT

ACCA5026

Abstract

This report outlines, the database design, development and testing for the contracted database from Adventureworks cycles

Ryan Hopkins (2221956)
2221956@student.uwtsd.ac.uk

Contents

Introduction..... **Error! Bookmark not defined.**

Conclusion 4

Bibliography 4

Part 1- sql injection

SQL injection theory

When developing a website using php the developer should include a number of protections that defend the data from being stolen using injections. This should include password hashing, ssmt. Protections against SQL leaks prevent databreaches from happening and harming the reliability of the company that created it, as Luke Noonan Says, [1]"The reputational damage resulting from a data breach can be devastating for a business. Research has shown that up to a third of customers in retail, finance and healthcare will stop doing business with organisations that have been breached. Additionally, 85% will tell others about their experience, and 33.5% will take to social media to vent their anger."

When including passwords on a website Password hashing protect the individual accounts of customers/ users from becoming a target from hackers, as this will encrypt the users password, using a hashing index, overall it doesn't keep the password from being cracked after being leaked, but once a databreach has happened typically the company would send out a mass email to the affected clients, or prompt the user the next time they've signed in to update passwords.

The most common type sql injections is an in-band injection, this will be a simple string of text which can accidentally miss read by the database as a command, such as OR '1'='1', This acts like a regular command without the proper protection in place, where the injection will just print all rows into common text, if passwords are unhashed they will be in plain text which will allow the hackers to steal personal information and the account if they choose.

SQL injection in my code

The php I developed assisted in removing the in-band injection as a potential threat as, as the malicious "OR 1==1" gets thrown out and requests the user to go back to the index page, this removes that threat however the database is still weak to a error based sql injection, this would be the insertion of php code, which created a fault, this poses a problem as it shows the table, names attributes, from this the bad actor can use a more advanced and specialised query which attacks the specific table and attributes, potentially revealing the data from the customer table

```
mysqli_stmt_prepare($stmt,$query);
mysqli_stmt_bind_param($stmt,"sssss",$fn,$sn,$em,$un,$pw);
mysqli_stmt_execute($stmt);
$result = mysqli_stmt_get_result($stmt);
```

Figure 1 insert into code in the file create.php

CRUD

Crud is a major part in the creation of a database, it allows web developers to implement segments to a database through their website, the typical end user will only ever interact with two or three of these functions, those being, the Create and read most of the time, and rarely the update, where they might update information about themselves.

Create

The create function in my code provides the user with anti-SQL injections, as previously stated in part 1, The reason why I entered it here as its important that the customer database

should be protected with everything that the website developer understands and can implement, as the customer database is never read from, then the only place it can be accessed is the creation of a new query or the login field,

```
?include "coninfo.php";

$fn = $_POST["fname"];
$sn = $_POST["lname"];
$em = $_POST["Email"];
$un = $_POST["userN"];
$pw = $_POST["PassW"];

$query = "INSERT INTO customer(FirstName,SecondName,Email,UserName>Password)
VALUES(?,?,?,?,?)";

$stmt = mysqli_stmt_init($link);

mysqli_stmt_prepare($stmt,$query);
mysqli_stmt_bind_param($stmt,"sssss",$fn,$sn,$em,$un,$pw);
mysqli_stmt_execute($stmt);
$result = mysqli_stmt_get_result($stmt);
>
Record added!
a href="home.php"> back to home page</A>
```

Figure 2 create data, inside the create.php file

Whilst the code is incredibly simple it took a number of hours to pull off, as I saw unforeseen errors that I haven't dealt with in an incredibly long time, when using bind parameters, I had set them to specific variables, which caused an error however, after troubleshooting,



Read

The Read function is just pulling the record out of the database, this could be to see how it would display without editing the data within the database, this worked for the Watch database where it pulled from the database the name, description, dialsize, and price, however due to unforeseen circumstances the images didn't print the way In the future I would've changed the way the database interacts with images to a blob file, while being unstable it would provide a unique way of inserting the image, instead of storing the file path as some data could produce errors in the code.

```

<?include "coninfo.php"?>

<h2>Results:</h2>
<?

$query = "SELECT * FROM Watch";
$result = mysqli_query($link, $query);

while($r = mysqli_fetch_array($result))
{
    $wn = $r["WatchName"];
    echo $wn." ";
    $de = $r["Description"];
    echo $de." ";
    $ds=$r["DialSize"];
    echo $ds." ";
    $img = $r["Images"];
    <img src=
    ".image/<?php echo $data['filename']; ?>" />
    <?
    echo $img;

    $pr = $r["Price"];
    echo $pr." <br><br>";
}
?>

```

Figure 3 read data inside the read.php file

Conclusion

Whilst there is a lot of incredible techniques that could've been used to protect the database from bad actors, the use of CRUD to allow the users to update the website for with new data allows the database to grow without getting relying on the database administrator to update it. Whilst my database included Segments of CRUD and anti sql injection, I should've included the all the parts of CRUD and included password hashing inside the customer database. As my website did not end up like I wanted it too, I have uploaded raw files which should've been added onto the website rather than being loose in the same folder.

Bibliography

- [1] L. Noonan, " 5 Damaging Consequences of Data Breach: Protect Your Assets," MetaCompliance, 1 Jan 2025. [Online]. Available: <https://www.metacompliance.com/blog/data-breaches/5-damaging-consequences-of-a-data-breach>. [Accessed 14 Jan 2025].

[2] G. Andrus, "Normalization," DePaul University, 01 01 2025. [Online]. Available: <https://condor.depaul.edu/gandrus/240IT/accesspages/normalization3.htm>. [Accessed 03 01 2025].

FINAL GRADE

48/100

GENERAL COMMENTS

Database Structure and Design - A reasonable design with 3 tables. Normalisation is limited to Manufacturer - I think I'd have kept Country in the main table and normalised it in its own right - there are lots of repeating groups here and it makes it more difficult to search with it locked up in the manufacturers table. Status "Operational" is too long - lots of repeating text here. You could just use 1 for Operational, 2 for Non-Operational and sort out the text for the front end in code.

6/10

Range of back-end coding - Good to see you've used index.php so that the front page on the site loads automatically.

Login is OK, but it would be better to filter "login" from the nav bar once you're logged in as this is less confusing to the user. The login does not seem to be used across pages - once I navigate away from the Orders page I can't get back to it (without logging in again)

Register needs validation as I could add a blank user and also add users with the same username (which should be unique)

There are some attempts at delete in the code albeit unlinked to the actual site, but no edit (that I can find)

No search other than see all the watches on one page.

22/50

Overall integration of website - Nice layout on the front of the site on the whole and good to see navigation. bar. Contact us page doesn't seem to work. Some pages are not styled.

Some of the navigation tries to take the user to home.php which doesn't exist. Login section has strange navigation, but I have commented on that above.

6/10

SQL Injection overview - A bit on the brief side on the explanation of what SQL is. - you need some general examples here to illustrate the scope of the problem - you do have the 1=1 attack covered in class but there are more complex ones. You seems a bit off topic here, I only asked for a review of SQL Injection, not password hashing etc 6/15

SQL Code implementation/proposal - You've added some prepared statements, although a bit limited 8/15

General comments:

You must avoid the use of "I", "my" and so on in academic reports - particular at Levels 5 and 6.

Overall mark for PHP assignment: 48%

Overall mark for module (including Divya's assignment) - 53.5%

PAGE 1

PAGE 2

PAGE 3

PAGE 4



Comment 1

??

PAGE 5

PAGE 6
