APRIL 30, 2021

DATABASE APPLICATION FOR CAJUN PETROLEUM PRODUCING COMPANY (CPPC)

CPPC WELL PRODUCTION LOGGING SYSTEM

Rashid Shaibu (C00422601) And Andrew J Shullaw (C00161818)

CMPS460: DBMS FINAL PROJECT

Section 1: Project Requirements

Cajun Petroleum Producing Company (CPPC) is a limited liability company involved in the production of hydrocarbons (oil and gas) from onshore wells in Louisiana. The **CPPC Well Production Logging System** is designed to keep track of the daily production of oil and gas from all wells owned by CPPC. The aim of the system is to provide a central point for all wells and their details and to ensure production reporting process can be done online without the need of paperwork anymore. Data will be readily accessible anywhere online without need of faxing or postage.

CPPC has 10 oil fields across 8 Parishes in the state of Louisiana. In total, there are 60 wells in these fields. A field can have many wells. The name, measured depth, vertical depth, and production start date of each well is known. Each well produces oil, gas, and unwanted water. CPPC has 3 offices whose address, and phone number are known. Each office oversees production from selected parishes—each office is assigned specific parishes. At the end of each day, designated employees in these offices report production for wells in parishes under their jurisdiction. Occasionally some wells undergo testing to record some reservoir parameters (well pressure, water, oil, gas) which relates to the life of the wells.

The application is designed using XAMPP and runs with the bundle of Apache as HTTP Server, MariaDB as the database management system and PHP as the scripting language used for dynamic web pages and web development.

The following database interactions are expected from the database system:

1. List of wells by field, and Parish where well is located.
2. Daily oil and gas production by well and field
3. Name, location and ID of employee logging production data and time for logging the data.
4. List of wells which have undergone testing.
5. Office location with assigned wells, fields, and parishes

Section 2: Data Modelling

2.1 High level Conceptual data model

Figure 1. shows the structure of the database—a high-level data model (E-R diagram) which consist of the entities, attributes, and associations.

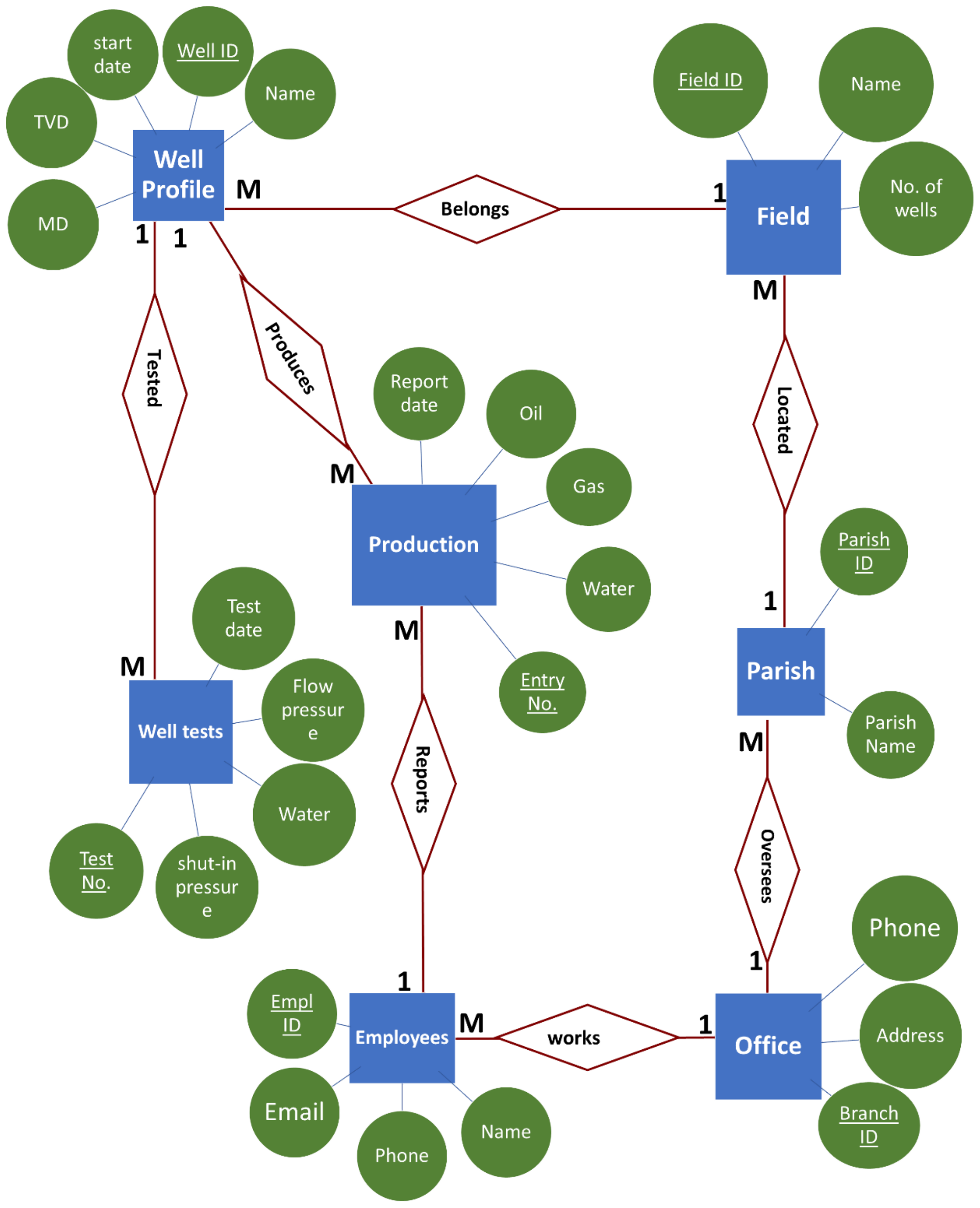
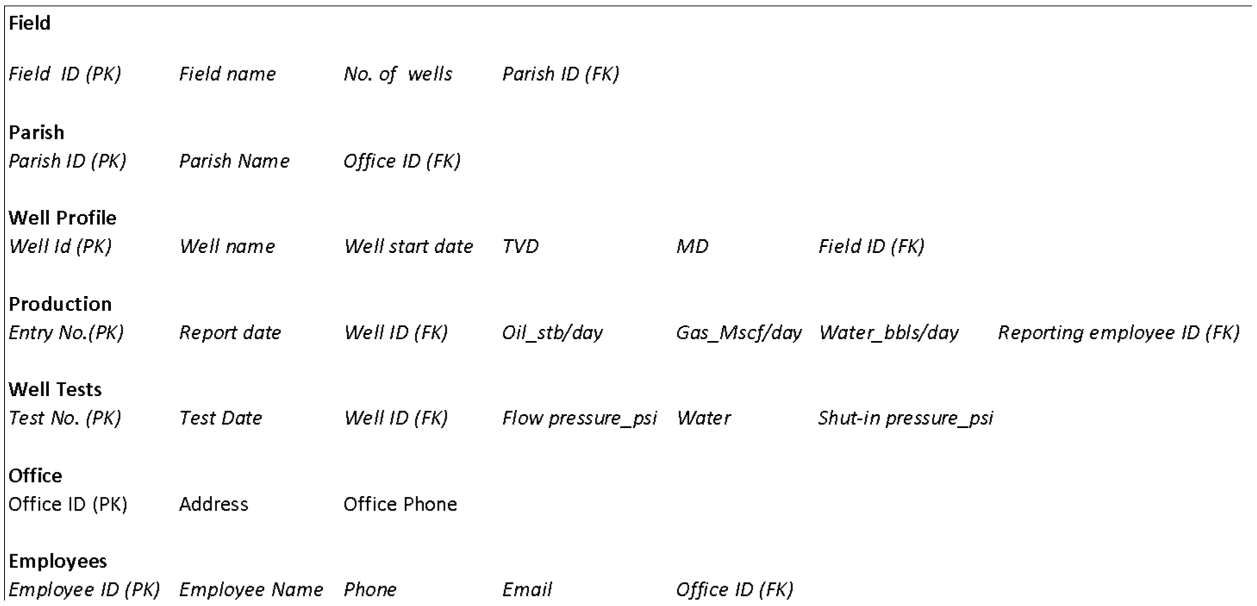


Figure 1: ER Diagram for the CPPC Well Production Logging System

2.2 The Low-level relational data model



Field						
Field ID (PK)	*Field name*	*No. of wells*	*Parish ID (FK)*			
Parish						
Parish ID (PK)	*Parish Name*	*Office ID (FK)*				
Well Profile						
Well Id (PK)	*Well name*	*Well start date*	*TVD*	*MD*	*Field ID (FK)*	
Production						
Entry No.(PK)	*Report date*	*Well ID (FK)*	*Oil_stb/day*	*Gas_Mscf/day*	*Water_bbls/day*	*Reporting employee ID (FK)*
Well Tests						
Test No. (PK)	*Test Date*	*Well ID (FK)*	*Flow pressure_psi*	*Water*	*Shut-in pressure_psi*	
Office						
Office ID (PK)	Address	Office Phone				
Employees						
Employee ID (PK)	*Employee Name*	*Phone*	*Email*	*Office ID (FK)*		

2.2.1 Nomenclature used in the data model.

Abbreviation	Meaning	Description
TVD	Total vertical depth	Vertical depth of a well
MD	Measured depth	Measured depth of a well
psi	pounds per square inch	Unit for pressure
bbls	Barrels	Unit for volume, typically water
stb	Stock tank barrels	Unit for produced oil at standard conditions
Mscf	Million standard cubic feet	Unit for produced gas at standard conditions

2.3 Normalization

All tables in the relational model are in 3NF because.

- There are no composite attributes—the data values are atomic.
- There are no composite primary keys. Each table has a unique primary key. In essence, functional dependency of non-key attributes on subsets of the primary key is absent.
- There is no functional dependency between non-key attributes.

Section 3: Implementation

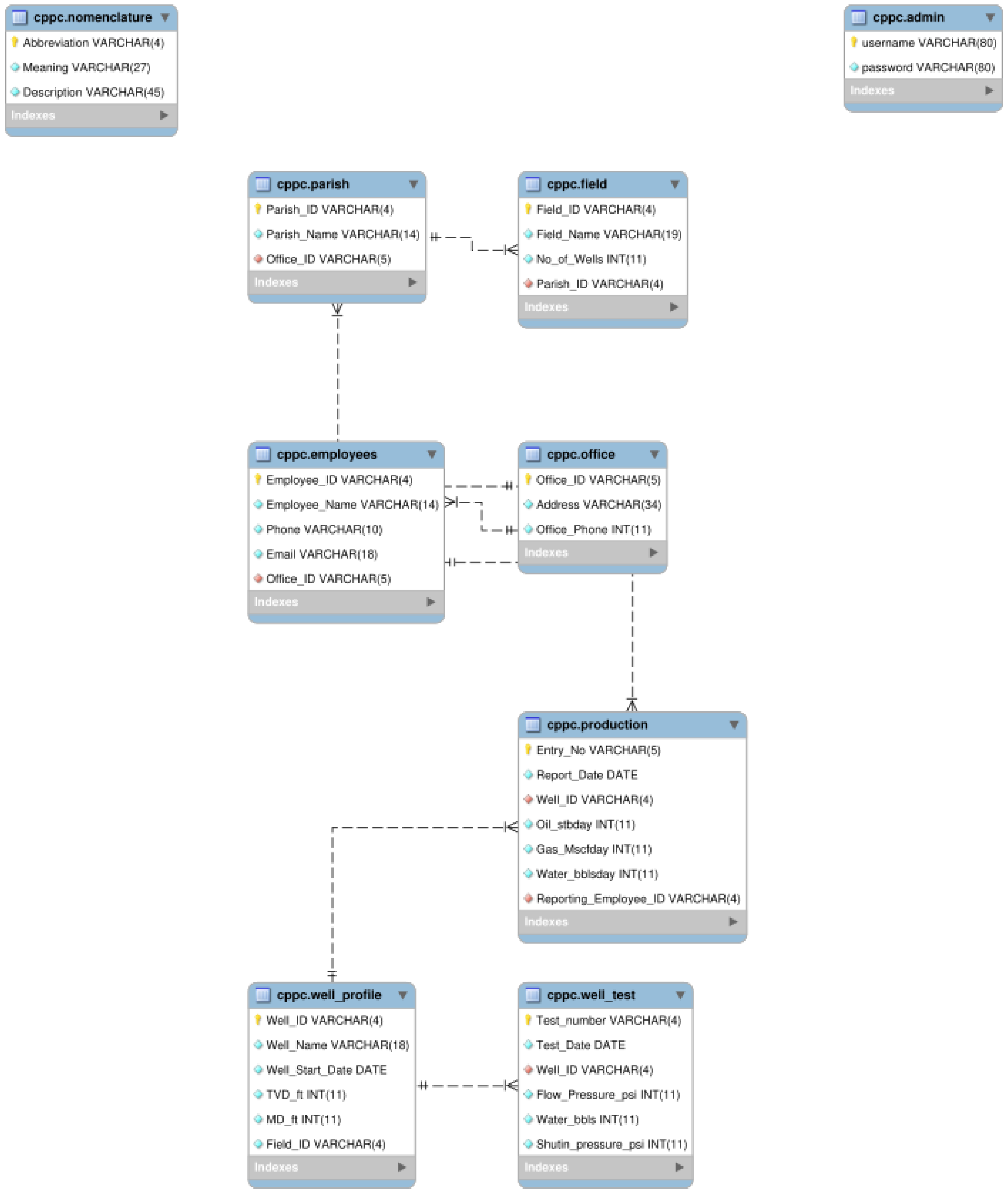
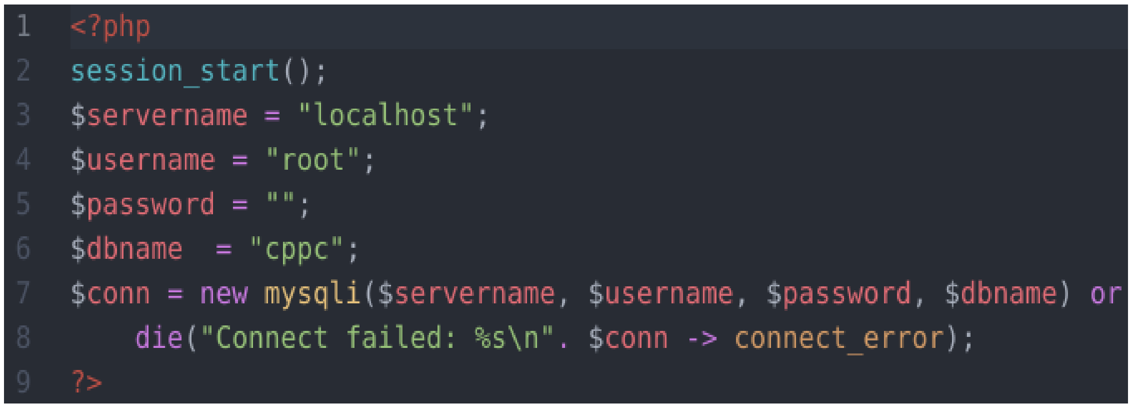


Figure 3.0.1 shows the final layout of the CPPC database produced by MySQL Workbench

3.1 Database Connection

A connection to the database must be established through the server using PHP session variables.



```php
<?php
session_start();
$servername = "localhost";
$username = "root";
$password = "";
$dbname  = "cppc";
$conn = new mysqli($servername, $username, $password, $dbname) or
    die("Connect failed: %s\n". $conn -> connect_error);
?>
```

Figure 3.1.1 config.php

The configuration file is included in every file (page) of the application to ensure a connection is established to the database server. The code creates a *mysqli* object that stores the required credentials to connect to the server. If a connection is not established due to any reason, an error is thrown.

3.2 Credential Security

This application was produced with the intent of being used by the company itself and for public access. Therefore, in addition to the database itself there were tables created to store user credentials (company side) to restrict access to parts of the application that have privileges such as inserting, editing, and deleting information from the database.

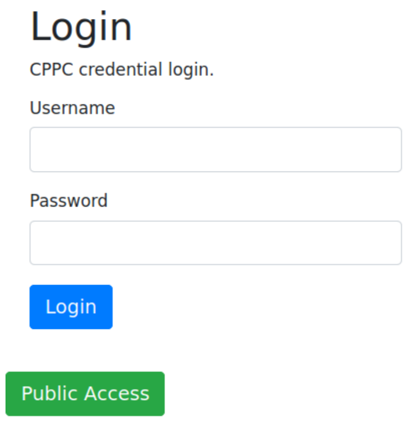
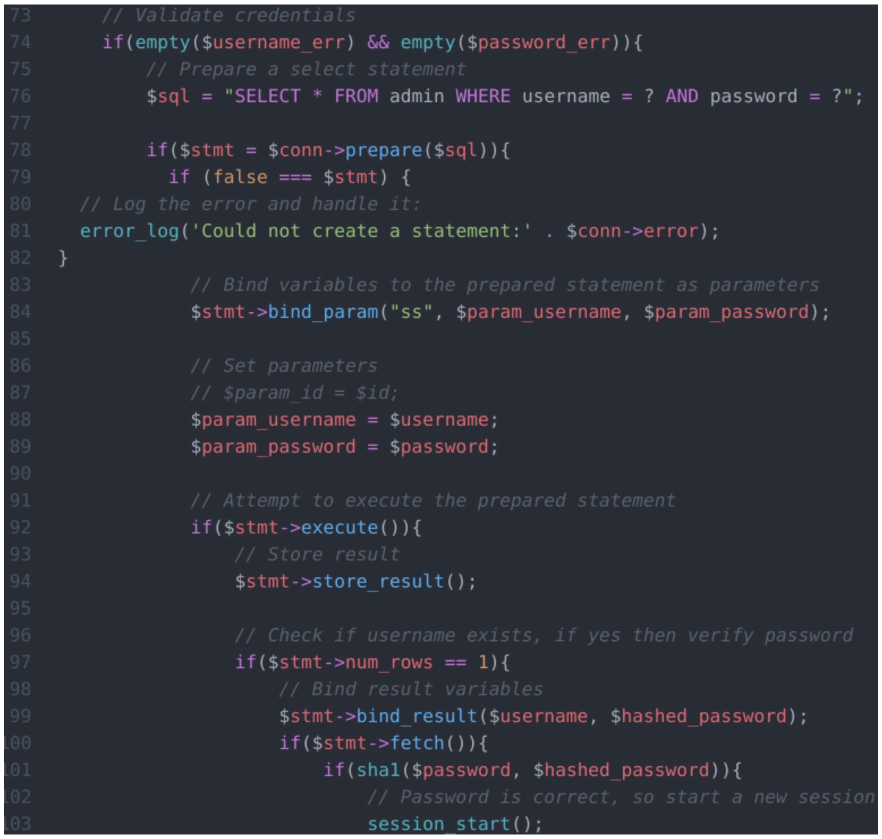


Figure 3.2.1 Login Page

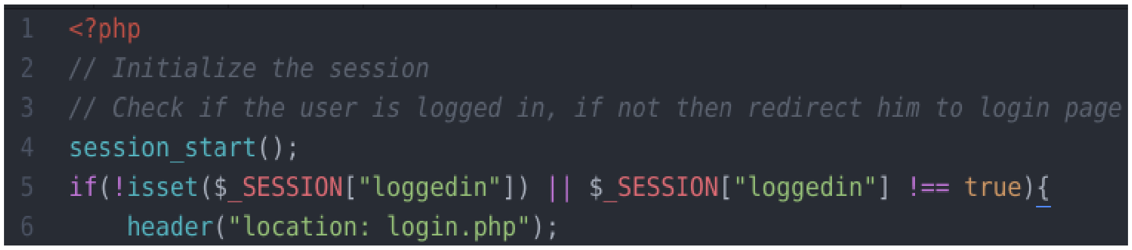
Implementation of SHA1 password hashing, trimming of non-alphanumeric characters, and binding of variables was used to ensure company side users were securely logged into the server and that no injections may occur.



```
    // Validate credentials
    if(empty($username_err) && empty($password_err)){
        // Prepare a select statement
        $sql = "SELECT * FROM admin WHERE username = ? AND password = ?";

        if($stmt = $conn->prepare($sql)){
          if (false === $stmt) {
  // Log the error and handle it:
  error_log('Could not create a statement:' . $conn->error);
}
            // Bind variables to the prepared statement as parameters
            $stmt->bind_param("ss", $param_username, $param_password);

            // Set parameters
            // $param_id = $id;
            $param_username = $username;
            $param_password = $password;

            // Attempt to execute the prepared statement
            if($stmt->execute()){
                // Store result
                $stmt->store_result();

                // Check if username exists, if yes then verify password
                if($stmt->num_rows == 1){
                    // Bind result variables
                    $stmt->bind_result($username, $hashed_password);
                    if($stmt->fetch()){
                        if(sha1($password, $hashed_password)){
                            // Password is correct, so start a new session
                            session_start();
```

Figure 3.2.2 PHP code for password protection

Session variables *username* and *loggedin* are stored after verification in order to ensure that one cannot access a page simply by knowing its location. A user would be returned to the login page if they were accessing a company side page without the system recognizing their credentials.



```
<?php
// Initialize the session
// Check if the user is logged in, if not then redirect him to login page
session_start();
if(!isset($_SESSION["loggedin"]) || $_SESSION["loggedin"] !== true){
    header("location: login.php");
```

Figure 3.2.3 Verification of credentials after being redirected

Once logged in, the user is redirected to the dashboard where they can access each page of the application. To be consistent with the integrity of the verification process, the current session's user is displayed in the upper right corner of every page.

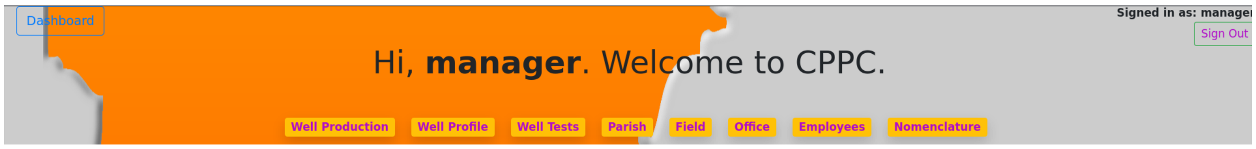
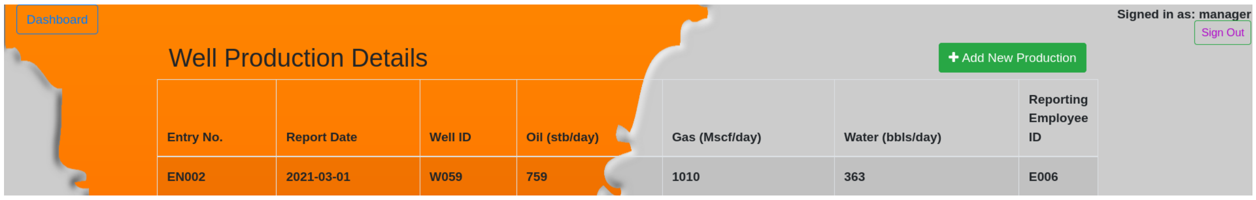


Figure 3.2.4 The dashboard display for a verified user

All rows and columns are selected from the Well Production table and the user has the option to make an Insert query by clicking the Add To Production button shown in the figure below. This button is not shown on the public access version.

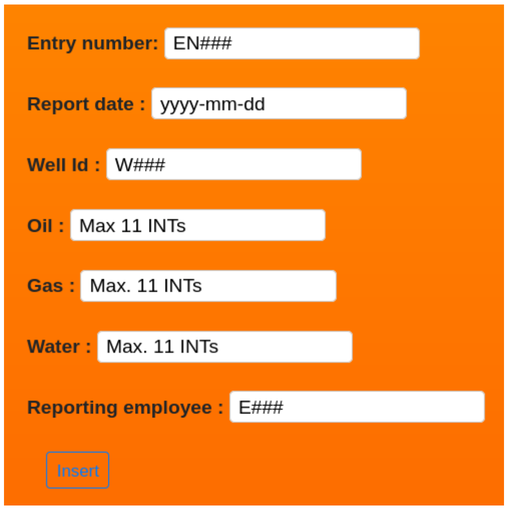


Well Production Details

Entry No.	Report Date	Well ID	Oil (stb/day)	Gas (Mscf/day)	Water (bbls/day)	Reporting Employee ID
EN002	2021-03-01	W059	759	1010	363	E006

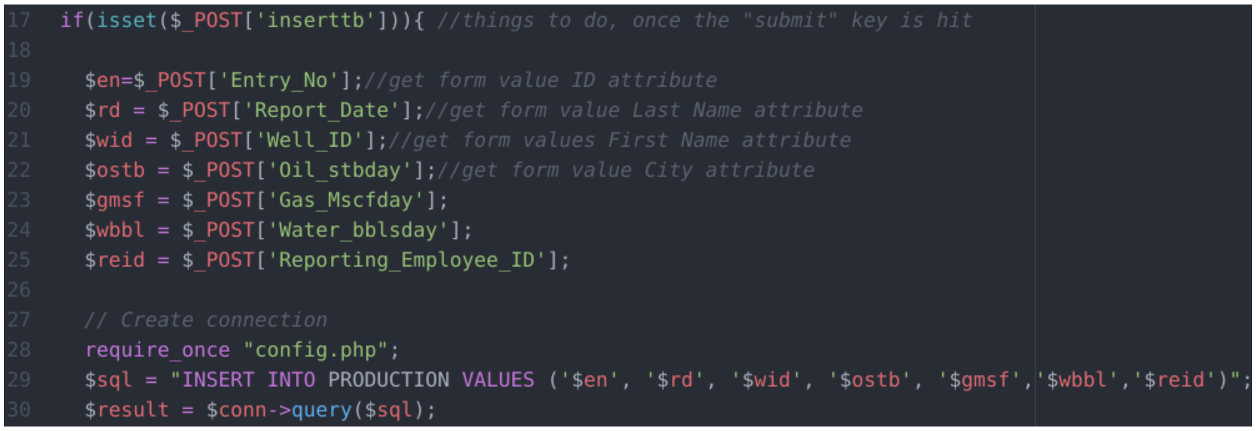
Figure 3.2.5 Well Production view for verified user

The user is given the option to insert a new record into the table, return to the dashboard, or sign out. HTML *values* are set to remind the user what type of input is required for a successful insert and *minlength* and *maxlength* are set to ensure that the user cannot under or overfill the form.



Entry number:	EN###
Report date :	yyyy-mm-dd
Well Id :	W###
Oil :	Max 11 INTs
Gas :	Max. 11 INTs
Water :	Max. 11 INTs
Reporting employee :	E###

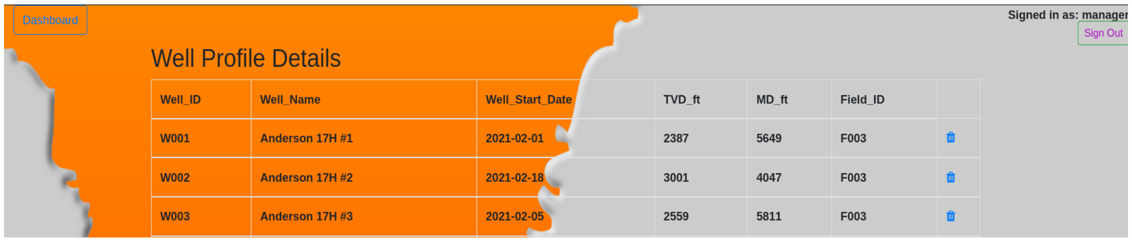
Figure 3.2.6 Insert button redirects to insert_into_production.php



```
if(isset($_POST['inserttb'])){ //things to do, once the "submit" key is hit

  $en=$_POST['Entry_No'];//get form value ID attribute
  $rd = $_POST['Report_Date'];//get form value Last Name attribute
  $wid = $_POST['Well_ID'];//get form values First Name attribute
  $ostb = $_POST['Oil_stbday'];//get form value City attribute
  $gmsf = $_POST['Gas_Mscfday'];
  $wbbl = $_POST['Water_bblsday'];
  $reid = $_POST['Reporting_Employee_ID'];

  // Create connection
  require_once "config.php";
  $sql = "INSERT INTO PRODUCTION VALUES ('$en', '$rd', '$wid', '$ostb', '$gmsf','$wbbl','$reid')";
  $result = $conn->query($sql);
```

Figure 3.2.7 PHP variables used with an embedded MySQL query to Insert into the table

The Well Profile page displays all selected fields and has the option to delete any row in the table.



Well Profile Details

Well_ID	Well_Name	Well_Start_Date	TVD_ft	MD_ft	Field_ID	
W001	Anderson 17H #1	2021-02-01	2387	5649	F003	🗑
W002	Anderson 17H #2	2021-02-18	3001	4047	F003	🗑
W003	Anderson 17H #3	2021-02-05	2559	5811	F003	🗑

Figure 3.2.8 Well Profile page containing links to Delete_wellprofile.php

If the delete button is triggered, the user is redirected to the confirmation prompt contained within *confirm.php*. If cancel is chosen, the user is redirected to the page they came from. Choosing *Go to deletion page* directs the user to the actual Delete Well Profile page where the delete buttons contain links that will select a row from the database to delete.

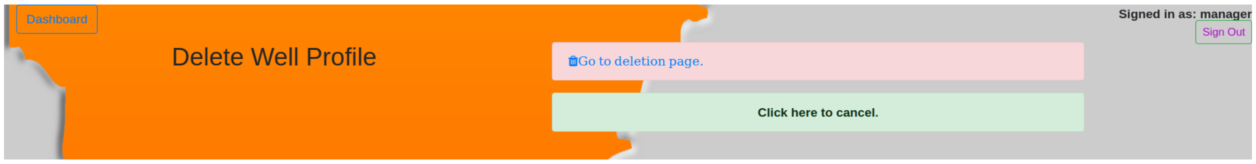
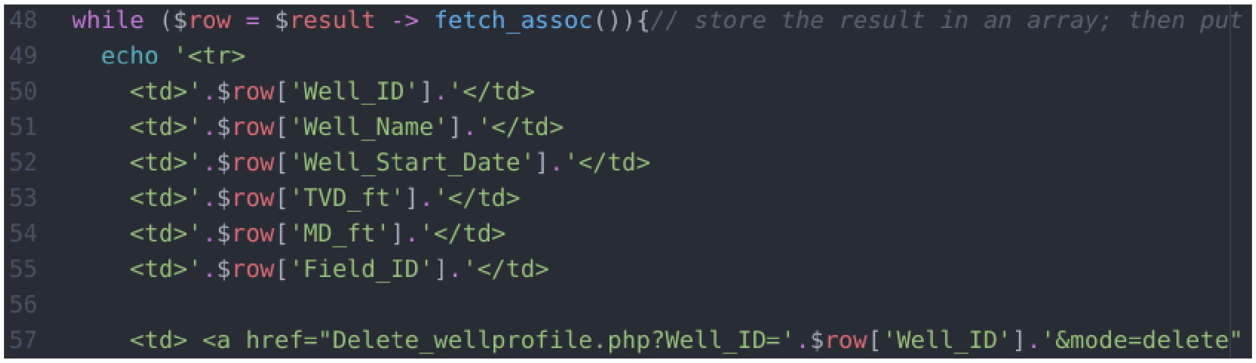


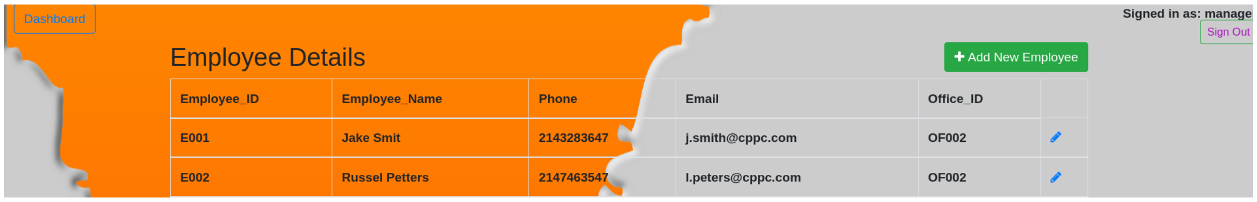
Figure 3.2.9 Prompt before confirming a delete query



```
while ($row = $result -> fetch_assoc()){// store the result in an array; then put
  echo '<tr>
    <td>'.$row['Well_ID'].'</td>
    <td>'.$row['Well_Name'].'</td>
    <td>'.$row['Well_Start_Date'].'</td>
    <td>'.$row['TVD_ft'].'</td>
    <td>'.$row['MD_ft'].'</td>
    <td>'.$row['Field_ID'].'</td>

    <td> <a href="Delete_wellprofile.php?Well_ID='.$row['Well_ID'].'&mode=delete"
```

Figure 3.2.10 *Well_ID* is associated with the row it is in and mode is set to delete

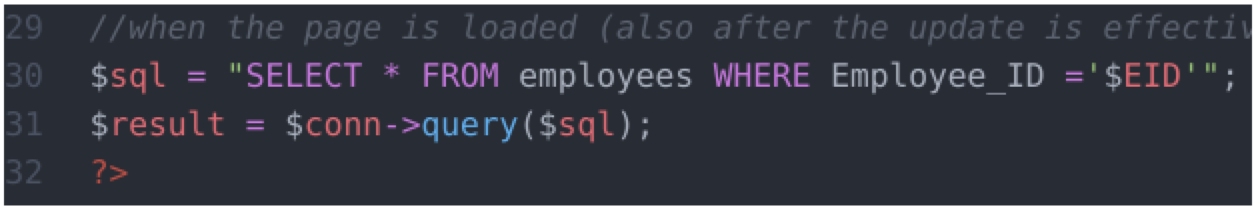
A message is shown if the delete is successful and the record will no longer exist in the table. For a database this size, it would be a good idea to create an archive for deleted records in case of an accidently delete even after confirmation. Other than the Employees table, the remaining tables in the database have no query access other than Select.



Employee_ID	Employee_Name	Phone	Email	Office_ID	
E001	Jake Smit	2143283647	j.smith@cppc.com	OF002	✎
E002	Russel Petters	2147463547	l.peters@cppc.com	OF002	✎

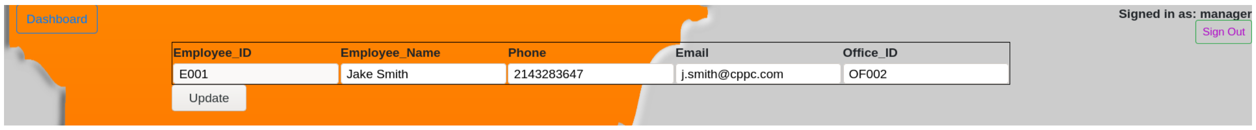
Figure 3.2.11 Employee table containing links to *alter* and *insert*

The edit link in the final cell of the Employees table redirects the user to *Edit_employee.php* specific to the row that was chosen for editing. Differing from the *insert_*.php* files, the new table that appears will be prefilled with the information from that specific row.



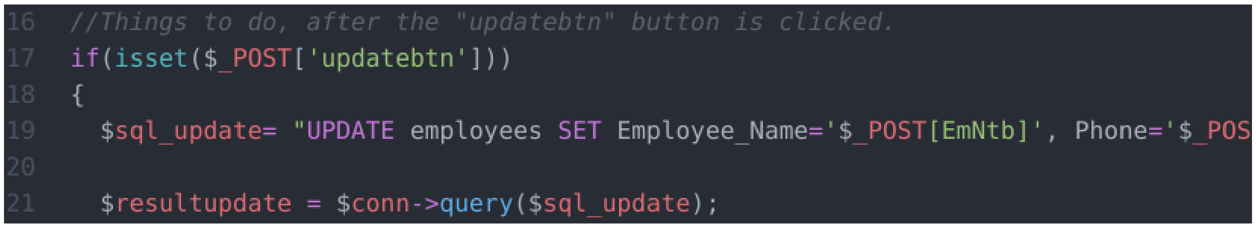
```
//when the page is loaded (also after the update is effecti
$sql = "SELECT * FROM employees WHERE Employee_ID ='$EID'";
$result = $conn->query($sql);
?>
```

Figure 3.2.12 Embedded MySQL query for a specific row



Employee_ID	Employee_Name	Phone	Email	Office_ID
E001	Jake Smith	2143283647	j.smith@cppc.com	OF002

Figure 3.2.13 View of when user redirected to Edit_employee.php



```
//Things to do, after the "updatebtn" button is clicked.
if(isset($_POST['updatebtn']))
{
  $sql_update= "UPDATE employees SET Employee_Name='$_POST[EmNtb]', Phone='$_POS

  $resultupdate = $conn->query($sql_update);
```

FIgure 3.2.14 Embedded MySQL for updating record inside of *Edit_Employee.php*

The total sum security features implemented in this application were:

1. Credential Login, with the ability for public viewing with no credentials
2. SHA1 hashed passwords
3. Binded PHP variables when taking input from users
    a. Null
    b. tab
    c. newline
    d. vertical tab
    e. carriage return
    f. ordinary white space
4. Storage and display of user session variables to ensure credentials were used to login
5. Prompts before dangerous actions such as a delete query
6. Prefilling of forms and use of *minlength* & *maxlength* to aid in the proper input to the database