Criterion C: Development

Table Creation

Success criteria fulfilled:

- To have all the required tables set up beforehand.
- To have a form for each table for the user to easily input the desired records into the database.
- Constraints to be set on database fields to minimise data entry errors and prevent database errors from occurring.

Concepts and techniques used:

- Basic table creation (1).
- Foreign key relationships (2).
- Data validation through SQL constraints (3).
- Use of different data types (4).
- Basic mathematical operations for auto-calculated fields (5).
- Primary key ID with prefixing and sequencing through triggers and a separate table (6).

Example:

```
CREATE TABLE Cost_Option (
    CO_ID VARCHAR(7) NOT NULL PRIMARY KEY DEFAULT '0', -- PRIMARY KEY
    Product_ID VARCHAR(6) NOT NULL, -- FOREIGN KEY
    Product_Name VARCHAR(50) NOT NULL,

    -- Integer fields (units + costs)
    Units INTEGER NOT NULL,
    Printing INTEGER NOT NULL,
    Design INTEGER NOT NULL,
    Plate INTEGER NOT NULL,
    Die_Cut_Mould INTEGER NOT NULL,

    -- Decimal fields (cost rates)
    Die_Cutting DECIMAL(4,3) NOT NULL,
    Lamination DECIMAL(4,3),
    Emboss DECIMAL(4,3),
    Hot_Stamping DECIMAL(4,3),
    Gluing DECIMAL(4,3),

    -- Integer fields (costs)
    Packing INTEGER NOT NULL,
    Transportation INTEGER NOT NULL,
    Packing_Material INTEGER NOT NULL,
```

```
    -- Calculated fields
    Unit_Cost DECIMAL(7,2) GENERATED ALWAYS AS
    ((Printing + Design + Plate + Die_Cut_Mould +
    (Die_Cutting * Units) + (Lamination * Units) + (Emboss * Units) +
    (Hot_Stamping * Units) + (Gluing * Units) +
    Packing + Transportation + Packing_Material) / Units),

    Total_Cost DECIMAL(7,2) GENERATED ALWAYS AS
    (Printing + Design + Plate + Die_Cut_Mould +
    (Die_Cutting * Units) + (Lamination * Units) + (Emboss * Units) +
    (Hot_Stamping * Units) + (Gluing * Units) +
    Packing + Transportation + Packing_Material),

    FOREIGN KEY(Product_ID) REFERENCES Product(Product_ID) ON DELETE CASCADE
);
```

Figure 1. Cost_Option Table Creation

```
-- Cost Option ID Sequencing Table
CREATE TABLE Cost_Option_Seq (
    ID INT NOT NULL AUTO_INCREMENT PRIMARY KEY
);

-- Cost Option ID Trigger
DELIMITER $$
CREATE TRIGGER TG_Cost_Option_Insert
BEFORE INSERT ON Cost_Option
FOR EACH ROW
BEGIN
    INSERT INTO Cost_Option_Seq VALUES (NULL);
    SET NEW.CO_ID = CONCAT('CO-', LPAD(LAST_INSERT_ID(), 4, '0'));
END$$
```

Figure 2. CO_ID Configuration

Justification and Explanation:

Firstly, in terms of concepts (1) and (2), these figures depict the simple mechanic of table creation using the CREATE TABLE statement. There is also the use of a foreign key relationship, to create a link between this table and another table (Product). This was done using the FOREIGN KEY constraint.

For concepts (3) and (4), these act as data validation for the database, which helps to maintain data integrity. Other constraints such as NOT NULL, PRIMARY KEY and DEFAULT are used. Data types such as VARCHAR, INTEGER and DECIMAL have been used to ensure that the correct types of data are entered.

For concept (5), the last columns are basically automatically generated through mathematical calculations. Basic mathematical operators such as * and + are used with GENERATED ALWAYS AS *(expression)*.

Finally, concept (6) was used to ensure that the primary key IDs are not only differentiable but also auto incrementing. This was done with the help of a separate table for sequencing and a BEFORE INSERT trigger that work together to automatically and correctly set the CO_ID.

Views Creation

Success criteria fulfilled:

- To have preset views that merge together the desired tables that are needed to assemble the required fields and records together for the PDF document generation.

Concepts and techniques used:

- Basic view creation (1).
- Merging fields and records using common IDs (2).
- Basic mathematical operations for auto-calculated fields (3).

Example:

```
CREATE VIEW Complete_Invoice AS
SELECT i.*, -- all invoice fields

c.Client_Name, c.Office_Address,
c.Office_Number, c.Email_Address, -- client fields

-- cost option fields
co1.Product_Name Item_One_Name, co1.Units Item_One_Units, co1.Unit_Cost Item_One_Unit_Cost, co1.Total_Cost Item_One_Total_Cost,

co2.Product_Name Item_Two_Name, co2.Units Item_Two_Units, co2.Unit_Cost Item_Two_Unit_Cost, co2.Total_Cost Item_Two_Total_Cost,

co3.Product_Name Item_Three_Name, co3.Units Item_Three_Units, co3.Unit_Cost Item_Three_Unit_Cost, co3.Total_Cost Item_Three_Total_Cost,

-- calculated fields
ROUND(((co1.Total_Cost + co2.Total_Cost + co3.Total_Cost) * 0.1), 2) Sales_Tax_Amount,
ROUND(((co1.Total_Cost + co2.Total_Cost + co3.Total_Cost) + (co1.Total_Cost + co2.Total_Cost + co3.Total_Cost) * 0.1), 2) Total_Price

FROM Invoice i
JOIN Client c ON c.Client_ID = i.Client_ID
JOIN Cost_Option co1 ON i.Item_One_ID = co1.CO_ID
JOIN Cost_Option co2 ON i.Item_Two_ID = co2.CO_ID
JOIN Cost_Option co3 ON i.Item_Three_ID = co3.CO_ID; -- joining fields based on common IDs.
```

Figure 3. Complete_Invoice View Creation

Justification and Explanation:

Separate views are created to be able to gather the full, complete information of each e.g. invoice. For example, here I merged together fields from the Cost_Option table with the Invoice table to tell us more about the items of the Invoice table. This shows concepts (1) and (2). The CREATE VIEW statement is used with JOIN … ON … to merge the fields of the Invoice table with other tables based on the corresponding FKs and PKs.

Concept (3) has already been explained in the “Table Creation” section but is also used here in a similar way.

Database Viewer

Success criteria fulfilled:

- To be able to filter records from tables by inputting the value to be searched for in the table.

Concepts and techniques used:

- To create and concatenate SQL statements in PHP. (1)
- Retrieve inputs in PHP from HTML forms. (2)
- Create a HTML form for user input. (3)
- Set out a HTML table to display our database tables. (4)
- Use the mysqli_fetch_array function to retrieve rows from the database and store them in an array. (5)
- Integrate PHP within HTML tags to display data retrieved from the database. (6)

Example:

```php
$error = '';

include "config.php";

$sql = "SELECT * FROM Client ";

if (isset($_POST['search'])) {

  // data validation
  if(empty($_POST['search-box'])) {
    $error = 'You have not inputted a value to search.';
  } else {
    // search for value. If there are no matching values, present headings only.
    $result = mysqli_real_escape_string($mysqli, $_POST['search-box']);
    $sql .= "WHERE Client_ID = '{$result}'";
    $sql .= " OR Client_Name = '{$result}'";
    $sql .= " OR Email_Address = '{$result}'";
    $sql .= " OR Office_Address = '{$result}'";
    $sql .= " OR Office_Number = '{$result}'";
    $sql .= " OR First_Manager_Name = '{$result}'";
    $sql .= " OR First_Manager_Number = '{$result}'";
    $sql .= " OR Second_Manager_Name = '{$result}'";
    $sql .= " OR Second_Manager_Number = '{$result}'";
  }
}

$query = mysqli_query($mysqli, $sql) or die(mysqli_error($mysqli));
```

Figure 4. PHP Code for Viewing Client Table

```
<!-- form to input value to be searched for -->
<form name="search-form" method="POST" action="client.php">
    <input class="form-control form-input" type="text" name="search-box" value="" placeholder="Search for record(s) with specified value here.">
    <div class="red-text"><?php echo $error; ?></div>
    <input class="search-button" type="submit" name="search" value="Search">
</form>

<!-- HTML to display table -->
<table width="80%" cellpadding="8" cellspace="7">
    <tr class="field-names">
        <td>Client_ID</td>
        <td>Client_Name</td>
        <td>Email_Address</td>
        <td>Office_Address</td>
        <td>Office_Number</td>
        <td>First_Manager_Name</td>
        <td>First_Manager_Number</td>
        <td>Second_Manager_Name</td>
        <td>Second_Manager_Number</td>
    </tr>

    <?php while ($row = mysqli_fetch_array($query, MYSQLI_ASSOC)) { ?>
    <tr>
        <td><?php echo $row['Client_ID']; ?></td>
        <td><?php echo $row['Client_Name']; ?></td>
        <td><?php echo $row['Email_Address']; ?></td>
        <td><?php echo $row['Office_Address']; ?></td>
        <td><?php echo $row['Office_Number']; ?></td>
        <td><?php echo $row['First_Manager_Name']; ?></td>
        <td><?php echo $row['First_Manager_Number']; ?></td>
        <td><?php echo $row['Second_Manager_Name']; ?></td>
        <td><?php echo $row['Second_Manager_Number']; ?></td>
    </tr>
    <?php } ?>

</table>
```

Figure 5. HTML & PHP Code for Viewing Client Table

Justification and Explanation:

Concept (1) involves storing SQL statements as strings within PHP variables, then concatenating them with WHERE clauses. This is then used with available preset functions to retrieve data from the database. From the first figure above, the $sql variable stores the SELECT statement that gets the entire Client table. When the HTML form is submitted with an input, this $sql variable is extended with a WHERE clause that filters the records by the user's input. Then, the mysqli_query function is used to actually perform the $sql query against the database, and this command is stored in a separate $query variable which we will use later.

Concepts (2) and (3) work together to get the user's input, which we use to filter the records. (2) can be seen in the first figure above, where $result stores the value from the form retrieved by the mysqli_real_escape_string function. Then (3) is seen in the second figure, where there is a simple HTML form.

Concepts (4), (5) and (6) work together to display the database tables on our webpage. (4) is self-explanatory, and is seen in the second figure. (5) is also shown in the second figure, where the mysqli_fetch_array function is used to store the records from the database in an array. This function is called with the $query variable from earlier, to store only records identified by our $sql statement. (6) then finally displays the data on our webpage. We use php within the *td* tags to echo the data from the array.

Documentation Generator

Success criteria fulfilled:

- To be able to automatically generate the selected documentation by inputting the appropriate document ID.
- To make sure the records and data in the generated PDF fit within the margins and borders (especially with longer record values), and do not overlap with any of the other elements of the PDF.

Concepts and techniques used:

- Retrieve required user input using forms and data validation (1).
- Use of TCPDF to generate and layout PDF documentation (2).

Example:

```php
<?php
$error = '';

include "config.php";

if (isset($_GET['pdf_quotation_generate'])) {
  // data validation
  if (empty($_GET['Quotation_ID'])) {
    $error = 'You have not inputted an ID to search.';
  } else {
    $result = mysqli_real_escape_string($mysqli, $_GET['Quotation_ID']);

    if(!preg_match('^Q-\d{4}^', $result)) {
      $error = 'You must input a value in the form of Q-0001';
    } else {
      header('Location: generatequotationpdf.php?Quotation_ID='.$result.'&pdf_quotation_generate=');
    }
  }
}

?>
```

```php
<form method="get" action="quotation.php">
  <div class="form-group">
    <label for="Quotation_ID">Enter Quotation ID</label>
    <input type="text" class="form-control form-input" id="Quotation_ID" name="Quotation_ID" placeholder="Please input
    quotation ID in the form of Q-0001.">
    <div class="red-text"><?php echo $error; ?></div>
  </div>

  <button type="submit" name="pdf_quotation_generate">Generate</button>
</form>
```

Figures 6. Getting User Input

```php
<?php
require 'config.php';
require_once('TCPDF-main/tcpdf.php');

if (isset($_GET['pdf_quotation_generate'])) {
    $Quotation_ID = $_GET['Quotation_ID'];

    $select = "SELECT * FROM db_nippon_printing.complete_quotation WHERE
    db_nippon_printing.complete_quotation.Quotation_ID = '$Quotation_ID'";

    $query = mysqli_query($mysqli, $select);

    while ($row = mysqli_fetch_array($query)) {
        $Quotation_ID = $row['Quotation_ID'];
        $Quotation_Creation_Date = $row['Quotation_Creation_Date'];
        $Lead_Time = $row['Lead_Time'];
        $Tolerance_Of_Quantity = $row['Tolerance_Of_Quantity'];
        $Terms_Of_Payment = $row['Terms_Of_Payment'];
        $Quotation_Validity_Period = $row['Quotation_Validity_Period'];
        $Client_Name = $row['Client_Name'];
        $Office_Address = $row['Office_Address'];
        $First_Manager_Name = $row['First_Manager_Name'];
        $First_Manager_Number = $row['First_Manager_Number'];
        $Second_Manager_Name = $row['Second_Manager_Name'];
        $Second_Manager_Number = $row['Second_Manager_Number'];
        $Product_Name = $row['Product_Name'];
        $CO1_Units = $row['CO1_Units'];
        $CO1_Unit_Cost = $row['CO1_Unit_Cost'];
        $CO2_Units = $row['CO2_Units'];
        $CO2_Unit_Cost = $row['CO2_Unit_Cost'];
        $CO3_Units = $row['CO3_Units'];
        $CO3_Unit_Cost = $row['CO3_Unit_Cost'];
    }
}
```

Figure 7. Getting Data from the Database

```php
$pdf->SetFont('Times', 'B', 11);
$pdf->SetFillColor(220, 220, 220);

$pdf->Cell(60, 7, 'Product Name', 1, 0, 'C', 1);
$pdf->Cell(60, 7, 'Units', 1, 0, 'C', 1);
$pdf->Cell(60, 7, 'Unit Cost', 1, 1, 'C', 1);
$pdf->SetFont('Times', '', 11);

$pdf->Multicell(60, 21, $Product_Name, 1, 'C', false, 0);
$pdf->Multicell(60, 7, $CO1_Units, 1, 'C', false, 0);
$pdf->Multicell(60, 7, 'RM '.$CO1_Unit_Cost, 1, 'C', false, 1);

$pdf->Multicell(60, 7, '', 0, 'C', false, 0);
$pdf->Multicell(60, 7, $CO2_Units, 1, 'C', false, 0);
$pdf->Multicell(60, 7, 'RM '.$CO2_Unit_Cost, 1, 'C', false, 1);

$pdf->Multicell(60, 7, '', 0, 'C', false, 0);
$pdf->Multicell(60, 7, $CO3_Units, 1, 'C', false, 0);
$pdf->Multicell(60, 7, 'RM '.$CO3_Unit_Cost, 1, 'C', false, 1);
```

Figure 8. Designing the PDF Documentation

Justification and Explanation:
Concept (1) focuses on the front-end of the documentation generator, collecting user input and checking its validity. The basic input collection was done through simple HTML form elements and data validation was done in PHP. This concept is shown in figures 6.

Concept (2) focuses on the actual generation of the documentation in a PDF format. I used TCPDF, which is an open source software PHP class for generating PDF documents. I was able to use TCPDF to easily format the documentation on the PDF. Figure 7 fetches the required data from the database with respect to the inputted ID and the last figure shows the use of TCPDF functions to actually display the data on the PDF. The Multicell function has been used so longer values actually wrap within the borders.

Website UI

Success criteria fulfilled:

- To have a responsive website UI that scales with the window size.

Concepts and techniques used:

- Use of relative CSS length units (1).

Example:

```css
.container {
    background-color: rgb(24, 24, 24);
    width: 40vw;
    height: 70vh;
    margin-top: 13vh;
    border-radius: 20px;
    box-shadow: 2px 1px 3px #b8b8b8;
    border: 3px solid rgb(226, 226, 226);
}
```

```css
button {
    margin: 3%;
    padding: 0.8em;
    width: 20vw;
    border-radius: 20px;
    font-weight: 800;
    font-size: 17px;
    border: 3px solid rgb(199, 199, 199);
    color: rgb(24, 24, 24);
    background: rgb(255, 255, 255);
    transition: 0.3s;
}
```

Figures 9. CSS Properties

Justification and Explanation:
Concept (1) is about using relative CSS length units to design a responsive web user interface. For example, *vw* and *vh* have been used, which are relative to the height and width of the window. *em* is also used, which is relative to the size of the current font. Finally, % is used and is relative to the size of the parent element.

Data Validation

Success criteria fulfilled:

- Data validation for both the forms on the web-based application and forms in the MySQL database.

Concepts and techniques used:

- MySQL data validation through REGEX and triggers. (1)
- Application-level PHP data validation through empty() function, preg_match and REGEX. (2)

Example:

```
-- Email_Address
DELIMITER $$
CREATE TRIGGER TG_Client_Email_Check_BI BEFORE INSERT ON Client
FOR EACH ROW
BEGIN
IF (NEW.Email_Address REGEXP "^[a-z0-9!#$%&'*+/=?^_`{|}~-]+(\.[a-z0-9!#$%&'*+/=?^_`{|}~-]+)*@([a-z0-9]+[a-z0-9-]*)*[a-z0-9]+(\.([a-z0-9]+[a-z0-9-]*)*[a-z0-9]+)*\.[a-z]{2,6}$") = 0 THEN
  SIGNAL SQLSTATE '45000'
     SET MESSAGE_TEXT = 'Enter a valid email address.';
END IF;
END$$
DELIMITER ;
```

```
-- Office_Number
DELIMITER $$
CREATE TRIGGER TG_Client_Office_Number_Check_BI BEFORE INSERT ON Client
FOR EACH ROW
BEGIN
IF (NEW.Office_Number REGEXP "0[0-9]*-[0-9]* [0-9]*") = 0 THEN
  SIGNAL SQLSTATE '45000'
     SET MESSAGE_TEXT = 'There must be a dash and space to separate numbers in the phone number. (e.g. 012-3456 7890)';
END IF;
END$$
DELIMITER ;
```

```
-- Units, printing, design, plate, die_cut_mould
DELIMITER $$
CREATE TRIGGER TG_CO_Integer1_Check_BI BEFORE INSERT ON Cost_Option
FOR EACH ROW
BEGIN
IF (NEW.Units < 0 OR NEW.Printing < 0 OR NEW.Design < 0 OR NEW.Plate < 0 OR NEW.Die_Cut_Mould < 0) THEN
  SIGNAL SQLSTATE '45000'
     SET MESSAGE_TEXT = 'Error: there is a negative value in units, printing, design, plate or die_cut_mould.';
END IF;
END$$
DELIMITER ;
```

```
-- Die_cutting, lamination, emboss, hot_stamping, gluing
DELIMITER $$
CREATE TRIGGER TG_CO_Decimal_Check_BI BEFORE INSERT ON Cost_Option
FOR EACH ROW
BEGIN
IF (NEW.Die_Cutting < 0 OR NEW.Die_Cutting > 1 OR
    NEW.Lamination < 0 OR NEW.Lamination > 1 OR
    NEW.Emboss < 0 OR NEW.Emboss > 1 OR
    NEW.Hot_Stamping < 0 OR NEW.Hot_Stamping > 1 OR
    NEW.Gluing < 0 OR NEW.Gluing > 1) THEN
  SIGNAL SQLSTATE '45000'
     SET MESSAGE_TEXT = 'Die_cutting, lamination, emboss, hot_stamping and gluing rates cannot be < 0 OR > 1.';
END IF;
END$$
DELIMITER ;
```

```
-- Supplier_Type
DELIMITER $$
CREATE TRIGGER TG_Supplier_Type_Check_BI BEFORE INSERT ON Supplier
FOR EACH ROW
BEGIN
IF (NEW.Supplier_Type = 'Paper' OR NEW.Supplier_Type = 'Printing' OR NEW.Supplier_Type = 'Finishing') = 0 THEN
  SIGNAL SQLSTATE '45000'
     SET MESSAGE_TEXT = 'Invalid supplier type - options are paper, printing and finishing.';
END IF;
END$$
DELIMITER ;
```

Figures 10. Database-layer Data Validation Rules

```php
<?php

$error = '';

include "config.php";

$sql = "SELECT * FROM Client ";

if (isset($_POST['search'])) {

  // data validation
  if(empty($_POST['search-box'])) {
    $error = 'You have not inputted a value to search.';
  } else {
    // search for value. If there are no matching values, present headings only.
    $result = mysqli_real_escape_string($mysqli, $_POST['search-box']);
    $sql .= "WHERE Client_ID = '{$result}'";
    $sql .= " OR Client_Name = '{$result}'";
    $sql .= " OR Email_Address = '{$result}'";
    $sql .= " OR Office_Address = '{$result}'";
    $sql .= " OR Office_Number = '{$result}'";
    $sql .= " OR First_Manager_Name = '{$result}'";
    $sql .= " OR First_Manager_Number = '{$result}'";
    $sql .= " OR Second_Manager_Name = '{$result}'";
    $sql .= " OR Second_Manager_Number = '{$result}'";
  }
}
```

```php
if (isset($_POST['search-date'])) {

    // data validation
    if (empty($_POST['search-date-box'])) {
        $errordate = 'You have not inputted a date value to search.';
    } else {
        $result = mysqli_real_escape_string($mysqli, $_POST['search-date-box']);

        if(!preg_match('^\d{4}\-(0[1-9]|1[012])\-(0[1-9]|[12][0-9]|3[01])$^', $result)) {
            $errordate = 'You must enter the date in YYYY-MM-DD.';
        } else {
            $sql .= "WHERE Product_Creation_Date = '{$result}'";
        }
    }
}
```

```php
<?php
$error = '';

include "config.php";

if (isset($_GET['pdf_quotation_generate'])) {
  // data validation
  if (empty($_GET['Quotation_ID'])) {
    $error = 'You have not inputted an ID to search.';
  } else {
    $result = mysqli_real_escape_string($mysqli, $_GET['Quotation_ID']);

    if(!preg_match('^Q-\d{4}^', $result)) {
      $error = 'You must input a value in the form of Q-0001';
    } else {
      header('Location: generatequotationpdf.php?Quotation_ID='.$result.'&pdf_quotation_generate=');
    }
  }
}

?>
```

Figures 11. Application-layer Data Validation Rules

Justification and Explanation:

Concept (1) involves data validation to again ensure data integrity, correctness and consistency. All these data validations use a similar format with the help of triggers. These data validations have been set to occur before the update and insertion of data. Some of these validations use simple expressions with mathematical operators but there are some that use REGEX to check if the entered data fits within a certain format. Some of the regular expressions have been taken from the internet (e.g. email addresses) due to their great complexity, some manually written (e.g. phone numbers). Anyhow, all the triggers will output their respective error messages if the IF condition is met.

For concept (2), it involves implementing data validation in the web-based application forms.

There are basically two types of validations in this application-layer. The first being a presence check using the empty() function. The next level is a format check using REGEX, checking either that the value entered is in the YYYY-MM-DD format or in a specific ID format.