**SQL Queries and Helpers**

**1). Update Entity Relationship Diagram:**

* Identify the entities involved: **Product, Category, and Subcategory.**
* Add a new entity **Subcategory** that is related to **Category**. This will represent the subcategories.
* Establish relationships:
* **Product** is related to **Category** with a many-to-one relationship (each product belongs to a single category).
* **Product** is related to **Subcategory** with a many-to-many relationship (each product can belong to multiple subcategories, and each subcategory can have multiple products).

**2). Modify Tables and Relationships:**

* In the database schema, add a new table for **Subcategory**:
* **Subcategory** table with fields like **subcategory\_id** (primary key) and **subcategory\_name**.
* Update the **Product** table to include a foreign key referencing the **Subcategory** table:
* Add a foreign key column **subcategory\_id** in the Product table.

**3). Implement Relationships:**

* The relationship between **Product** and **Category** remains unchanged.
* Create a junction table to represent the many-to-many relationship between **Product** and **Subcategory**:
* **Product\_Subcategory** table with fields like **product\_id** and **subcategory\_id**.
* Each row in this table represents a product's association with a **subcategory**.



**orders: Contains order details including order\_uuid, client\_id, status\_id, and payment\_id.**

**order\_products: Contains details about products in each order.**

**clients: Contains client information.**

**payments: Contains payment information.**

**status: Contains status details.**

***CREATE OR REPLACE VIEW daily\_order\_details AS***

***SELECT***

***o.order\_uuid,***

***op.product\_count AS number\_of\_products,***

***op.order\_amount\_in\_cents,***

***c.client\_name,***

***p.payment\_type,***

***s.status\_name***

***FROM***

***orders o***

***JOIN order\_products op ON o.order\_id = op.order\_id***

***JOIN clients c ON o.client\_id = c.client\_id***

***JOIN payments p ON o.payment\_id = p.payment\_id***

***JOIN status s ON o.status\_id = s.status\_id***

***WHERE***

***DATE\_PART('week', o.order\_date) = DATE\_PART('week', CURRENT\_DATE)***

***ORDER BY***

***o.order\_date;***

The DATE\_PART('week', o.order\_date) function extracts the week number from the order date, and we compare it with the week number of the current date to filter the orders for the current week.

***CREATE OR REPLACE VIEW weekly\_report AS***

***SELECT***

***order\_uuid,***

***CONCAT\_WS('::',***

***TO\_CHAR(order\_date + INTERVAL '0 days', 'DY'),***

***TO\_CHAR(order\_date, 'YYYY-MM-DD')) AS day\_and\_date,***

***product\_count AS number\_of\_products,***

***order\_amount\_in\_cents***

***FROM***

***daily\_order\_details***

TO\_CHAR is used to format the order date as day and date. The CONCAT\_WS function concatenates the formatted day and date using the double colon separator "::".

**HTML email template**

**Analyze the PDF Document:**

Take a close look at the provided PDF document to understand its structure, layout, and elements. Identify the placeholders for dynamic data that need to be replaced with actual data from the Pet Shop API.

**Set Up HTML Template:**

Create an HTML template using Handlebars syntax. You'll define placeholders for the dynamic data that you'll later replace with actual data from the API.

**Fetch Data from Pet Shop API:**

Use the Pet Shop API to retrieve the necessary data for the document, such as order details, customer information, and more. You'll use this data to populate the placeholders in the HTML template.

**Render HTML Template:**

Integrate Handlebars into your application to render the HTML template with the dynamic data fetched from the API.

<!DOCTYPE html>

<html>

<head>

<title>Order Confirmation</title>

</head>

<body>

<div>

<h1>Order Confirmation</h1>

<p>Thank you for shopping at Pet Shop!</p>

<p>Order Date: <span class="order-date">{{orderDate}}</span></p>

<p>Order Number: <span class="order-number">{{orderNumber}}</span></p>

<p>Customer Name: <span class="customer-name">{{customerName}}</span></p>

<table>

<thead>

<tr>

<th>Product</th>

<th>Quantity</th>

<th>Price</th>

</tr>

</thead>

<tbody>

{{#each orderItems}}

<tr>

<td>{{productName}}</td>

<td>{{quantity}}</td>

<td>{{price}}</td>

</tr>

{{/each}}

</tbody>

</table>

<p>Total Amount: <span class="total-amount">{{totalAmount}}</span></p>

<p>Payment Status: <span class="payment-status">{{paymentStatus}}</span></p>

<button><a href="{{checkoutLink}}">PAY NOW</a></button>

</div>

</body>

</html>

In this template, I've added placeholders using Handlebars syntax like {{orderDate}}, {{orderNumber}}, etc. Now, let's move on to the data fetching step. You'll need to use JavaScript to fetch data from the Pet Shop API and then render it using Handlebars.

// Assuming you have a function to fetch data from the API

function fetchOrderData(orderUuid) {

// Fetch order data using the API

// Replace this with your actual API call

return fetch(`/api/v1/order/${orderUuid}`)

.then(response => response.json());

}

// Assuming you have the Handlebars library loaded

// Compile the Handlebars template

const source = document.querySelector('#template').innerHTML;

const template = Handlebars.compile(source);

const orderUuid = 'your\_order\_uuid\_here'; // Replace with the actual order UUID

fetchOrderData(orderUuid)

.then(orderData => {

// Render the template with the fetched data

const renderedHtml = template(orderData);

document.querySelector('#output').innerHTML = renderedHtml;

})

.catch(error => {

console.error('Error fetching order data:', error);

});

In this example, we assume you have a template defined in an HTML script tag with id="template", and an output container with id="output" where the rendered HTML will be placed.

**Client Issue Resolution**

**Case 1: Issue with Final Price Calculation**

**Gather Information:**

* Review the customer's purchase details and their reported issue.
* Verify the prices of the individual items and the quantities purchased.
* Cross-check the calculated final price with the customer's claim of 85Kn.

**Troubleshooting:**

* Calculate the total price manually based on the provided item prices and quantities.
* Identify any discrepancies between the manual calculation and the displayed price.
* Check for any applied discounts, taxes, or shipping charges that might affect the final price.

**Resolution:**

* If the issue is due to a technical glitch, work with the development team to fix the calculation error.
* If the issue was caused by a misunderstanding or miscommunication, provide clear explanations to the client.
* Update the customer with the corrected final price.

**Collaboration:**

* Collaborate with the Development and Quality Assurance teams to identify and rectify any software-related calculation issues.

**Case 2: Login and Password Reset Issues**

**Gather Information:**

* Collect the client's username and account details.
* Confirm if the client's credentials are accurate and that there are no typing errors.

**Troubleshooting:**

* Attempt to log in with the provided credentials to verify the issue.
* Check the email system to ensure that recovery emails are being sent.
* Investigate if the recovery email is being flagged as spam or blocked by the email provider.

**Resolution:**

* If the issue is due to a technical glitch, work with the development team to resolve login and email delivery issues.
* If the recovery emails are being blocked, advise the client to check their spam folder or adjust email settings.
* Provide the client with temporary login assistance if needed.

**Collaboration:**

* Collaborate with the Development and Email Management teams to address technical issues and email delivery problems.

**Case 3: Payment Processing Freezing**

**Gather Information:**

* Obtain details about the customer's attempted payment methods and the stage at which the screen freezes.
* Confirm if the issue is happening consistently and across different devices.

**Troubleshooting:**

* Replicate the customer's scenario to identify the freezing point.
* Test various payment methods to determine if the issue is specific to a certain method.

**Resolution:**

* If the issue is system-related, collaborate with the development team to resolve payment processing glitches.
* Advise the customer to try clearing their browser cache or using a different device/browser.
* Offer alternative payment methods if the freezing issue persists.

**Collaboration:**

* Collaborate with the Development and Quality Assurance teams to diagnose and address payment system issues.

**Case 4: Incorrect Amount Due Email Content**

**Gather Information:**

* Review the email content the customer received and the placeholder "{amunt\_due}".

**Troubleshooting:**

* Confirm if the email template contains the correct placeholders for the actual amount due.
* Verify if the email was generated correctly and if there were any errors during the process.

**Resolution:**

* Work with the email management team to correct the email template and replace the placeholder with the actual amount due.
* Send a corrected email to the customer.

**Collaboration:**

* Collaborate with the Email Management team to correct the email template and ensure proper email generation.

JsonLogic

{

"rules": [

{

"name": "Condition A",

"condition": {

"all": [

{ "var": "due\_amount" },

{ "var": "purchase\_date", "after": "2021-01-01" },

{

"any": [

{ "===": [{ "var": "shipping\_country" }, "Croatia"] },

{ "===": [{ "var": "shipping\_country" }, "Italy"] }

]

}

]

}

},

{

"name": "Condition B",

"condition": {

"all": [

{ "var": "due\_amount" },

{ "var": "purchase\_date", "after": "2021-01-01" },

{

"any": [

{ "===": [{ "var": "another\_variable" }, "SomeValue"] }

// Add more conditions using the relevant variable(s) from the purchase data

]

}

]

}

},

{

"name": "Condition C",

"condition": {

"at\_least": 2,

"conditions": [

{ "===": [{ "var": "custom\_condition\_1" }, true] },

{ "===": [{ "var": "custom\_condition\_2" }, true] },

// Add two more conditions based on your choice of custom conditions

]

}

}

]

}