You're on the right track in wanting to manage the logic related to queries and updates primarily on the PostgreSQL server, as this approach enhances security, maintainability, and performance. Here's how you can structure this workflow:

### Workflow Overview:

1. \*\*Load Records via a View\*\*:

- Initially, load and filter records using a view. This view can combine data from multiple tables (e.g., `mrl\_line\_items` and `fulfillment\_items`) for easy display and filtering in your Access form.

2. \*\*Select an Individual Record\*\*:

- When a user selects an individual `mrl\_line\_item` record from the filtered list, you can open a detailed form that shows both the `mrl\_line\_item` and its related `fulfillment\_items`.

3. \*\*Edit and Save Changes\*\*:

- For editing, the form should directly interact with the underlying tables (e.g., `fulfillment\_items`), not the view. This ensures that any changes made are correctly saved back to the database.

4. \*\*Use Stored Procedures for Logic\*\*:

- Implement stored procedures (or functions) in PostgreSQL to handle the update, insert, and delete operations. This centralizes the business logic on the server side, making it easier to maintain and secure.

### Step-by-Step Implementation:

#### 1. \*\*View for Loading and Filtering Data\*\*

- Use your existing view (`combined\_line\_items\_fulfillments\_view`) to load and filter data in the Access form. This view should join the `mrl\_line\_items` table with the `fulfillment\_items` table (and any other relevant tables).

#### 2. \*\*Selecting and Displaying an Individual Record\*\*

- When the user selects a record, pass the `mrl\_line\_item` ID to a new form that loads the specific `mrl\_line\_item` and its associated `fulfillment\_items`.

Example VBA to open the detailed form:

```vba

Private Sub btnOpenDetailForm\_Click()

Dim selectedID As Long

selectedID = Me.List.SelectedItemID ' Replace with the actual method to get the selected item ID

' Open the detail form and pass the selected ID

DoCmd.OpenForm "frmMRLDetail", acNormal, , "id = " & selectedID

End Sub

```

#### 3. \*\*Editing and Saving Edits\*\*

- In the `frmMRLDetail` form, load the `mrl\_line\_item` and its related `fulfillment\_items` directly from their respective tables.

Example SQL for the form’s RecordSource:

```sql

SELECT \* FROM mrl\_line\_items WHERE id = [Forms]![frmMRLDetail]![txtMRLID]

```

- Use subforms or linked list boxes to display the related `fulfillment\_items`.

- For saving edits, create stored procedures on the PostgreSQL server to handle updates. This way, all business logic (validations, calculations, etc.) is centralized.

Example VBA to call a stored procedure when saving:

```vba

Private Sub btnSave\_Click()

On Error GoTo ErrorHandler

Dim cmd As ADODB.Command

Set cmd = New ADODB.Command

cmd.ActiveConnection = g\_conn

cmd.CommandType = adCmdStoredProc

cmd.CommandText = "update\_fulfillment\_item"

' Pass parameters to the stored procedure

cmd.Parameters.Append cmd.CreateParameter("p\_item\_id", adInteger, adParamInput, , Me.txtItemID)

cmd.Parameters.Append cmd.CreateParameter("p\_fulfillment\_status", adVarChar, adParamInput, 50, Me.txtFulfillmentStatus)

' Add more parameters as needed

cmd.Execute

MsgBox "Record updated successfully.", vbInformation

Exit Sub

ErrorHandler:

MsgBox "Error " & Err.Number & ": " & Err.Description, vbCritical, "Update Error"

End Sub

```

Example Stored Procedure in PostgreSQL:

```sql

CREATE OR REPLACE FUNCTION update\_fulfillment\_item(

p\_item\_id INTEGER,

p\_fulfillment\_status VARCHAR

)

RETURNS VOID AS $$

BEGIN

UPDATE fulfillment\_items

SET fulfillment\_status = p\_fulfillment\_status

WHERE item\_id = p\_item\_id;

END;

$$ LANGUAGE plpgsql;

```

#### 4. \*\*Advantages of Using Stored Procedures\*\*:

- \*\*Centralized Logic\*\*: Business rules and data validation are maintained in one place, reducing redundancy and improving maintainability.

- \*\*Security\*\*: Direct access to tables is limited; the application interacts through controlled stored procedures, minimizing the risk of SQL injection.

- \*\*Performance\*\*: Complex operations are performed on the server, reducing the amount of data transferred between the client and the server.

### Summary:

- \*\*Views\*\* are used for initial data loading and filtering.

- \*\*Forms\*\* interact directly with tables for editing.

- \*\*Stored Procedures\*\* handle updates, ensuring that all logic and validation are centralized on the PostgreSQL server.

By structuring your Access application this way, you keep your client-side lightweight while leveraging PostgreSQL's power for data processing and business logic.