Medical Inventory Management

College Name: Pollachi College of Arts & Science

College Code: bruaf

TEAM ID: NM2025TMID21332

TEAM MEMBERS:

Team Leader Name: JANANI S

Email : 23nmcsjanani@gmail.com

Team Member : SHANMUGAPRIYA G

Email : 23nmcsshanmugapriya@gmail.com

Team Member : NITHISH S

Email : 23nmcsnithish@gmail.com

Team Member : DHARSHAN R

Email : 23nmcsdharshan@gmail.com

1.INTRODUCTION

1.1 Project Overview

This project is a comprehensive Salesforce application to streamline and manage various operational aspects of medical inventory. The system aims to efficiently maintain supplier details, manage purchase orders, track product details and transactions, and monitor the expiry dates of products. Maintain detailed records of suppliers, including contact information. Catalog product information, including descriptions, stock levels. Monitor and track product expiry dates to avoid using expired items. Comprehensive reports to track supplier performance, and purchase orders.

1.2 Purpose:

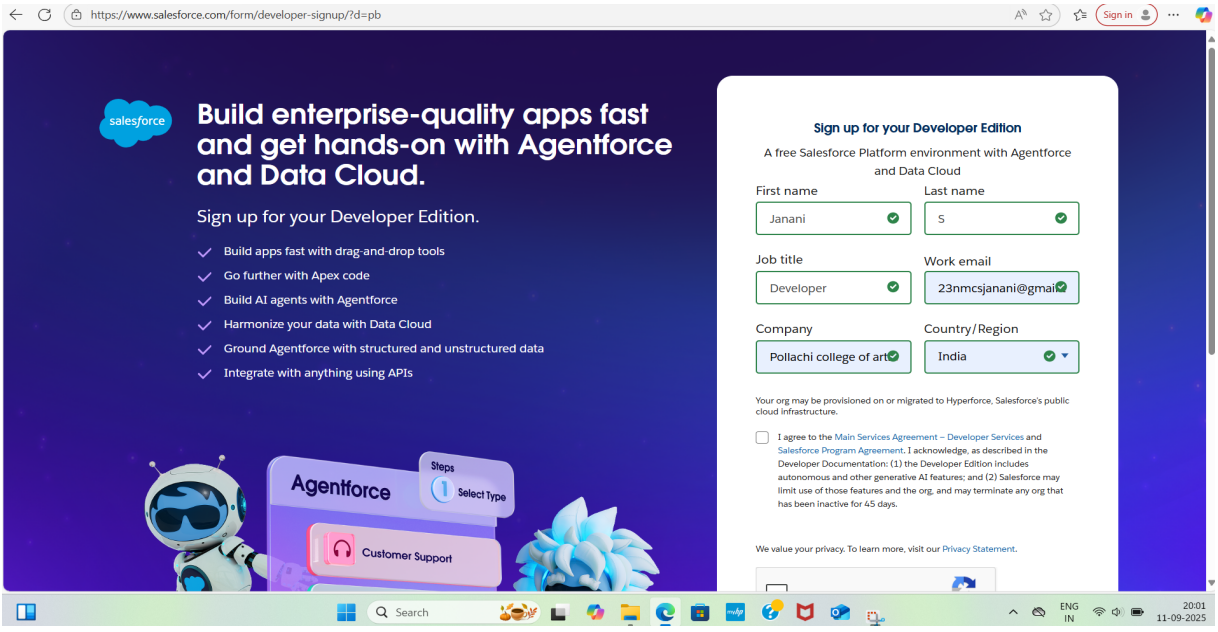
The main objective of the project is to provide a centralized system that helps healthcare organizations efficiently track, control, and optimize their medical supplies and equipment. It ensures real-time visibility of stock levels, prevents shortages or overstocking, and reduces waste by monitoring expiry dates and usage patterns. By automating processes such as reordering, alerts, and reporting, it saves time and lowers costs while also supporting regulatory compliance through accurate record-keeping. Ultimately, effective medical inventory management improves patient care by ensuring that essential medicines and equipment are always available when needed.

DEVELOPMENT PHASE

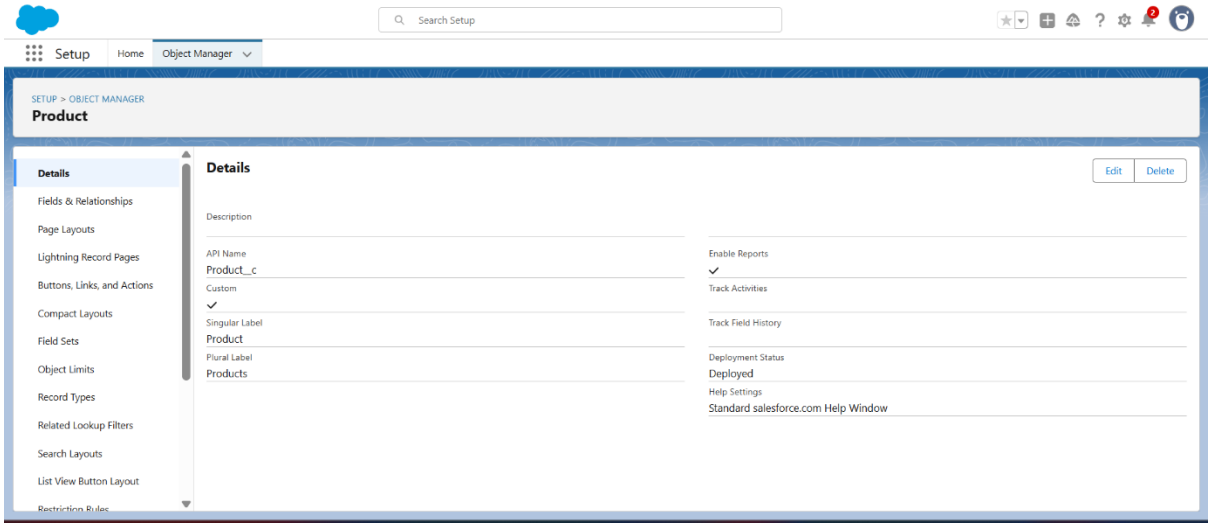
Creating Developer Account:

By using this URL –

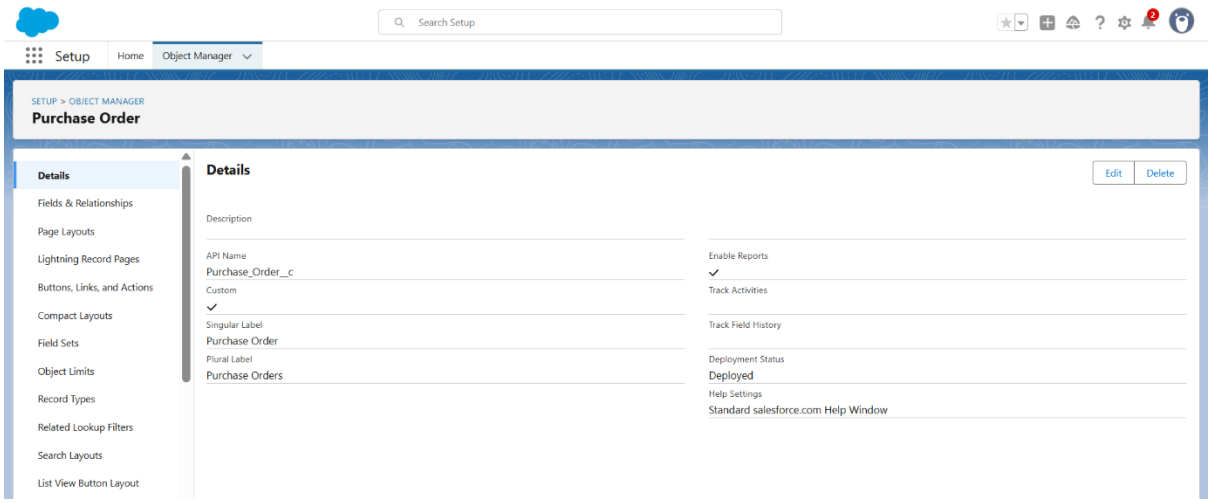
https://www.salesforce.com/form/developer-signup/?d=pb



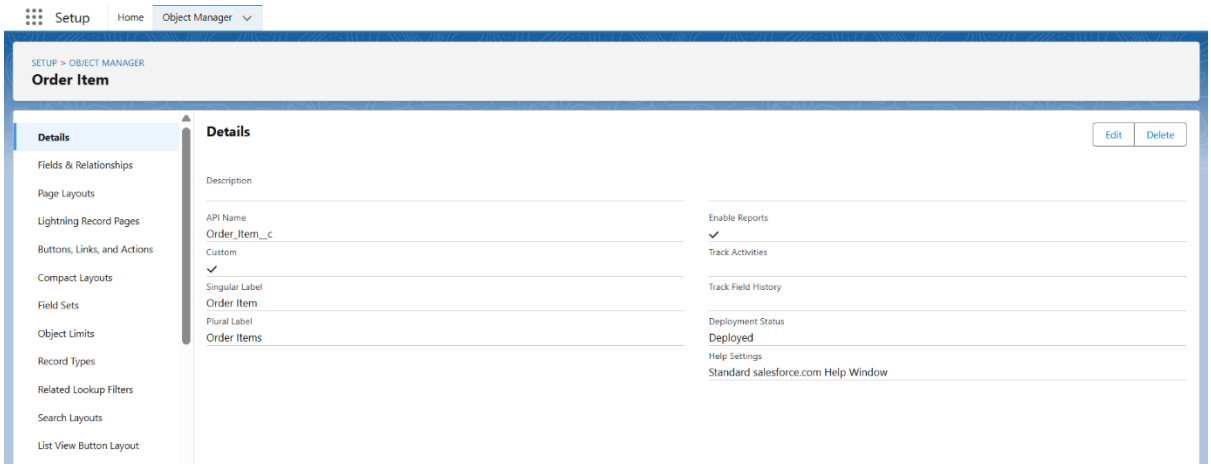
- Created objects: Product, Purchase Order, Order Item, Inventory Transaction and Supplier objects.



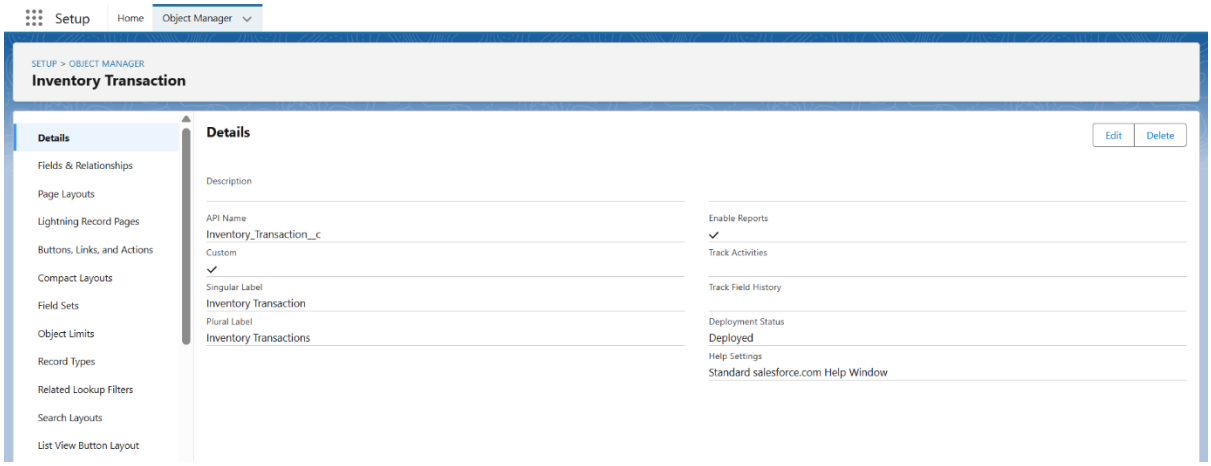
- Purchase Order Object



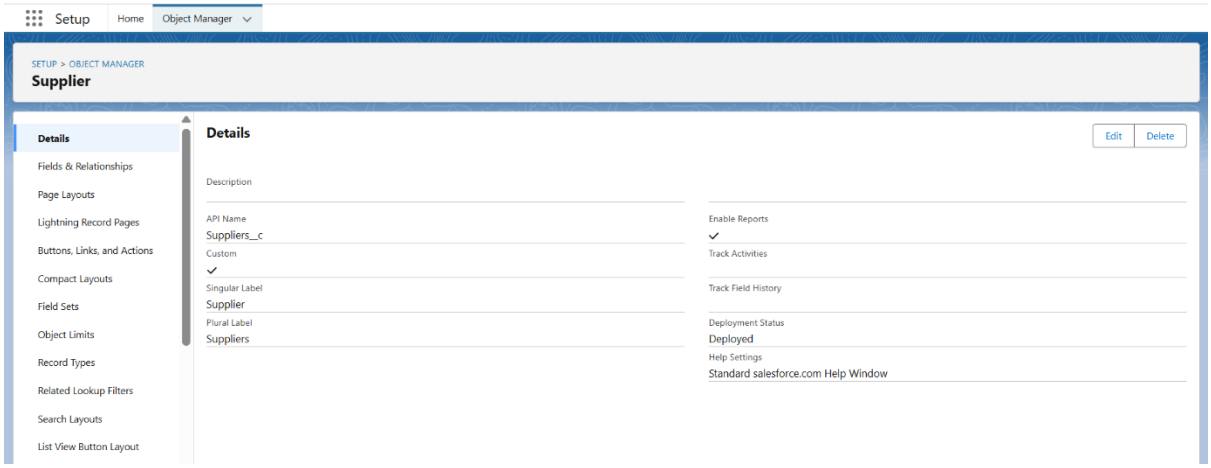
Order Item Object



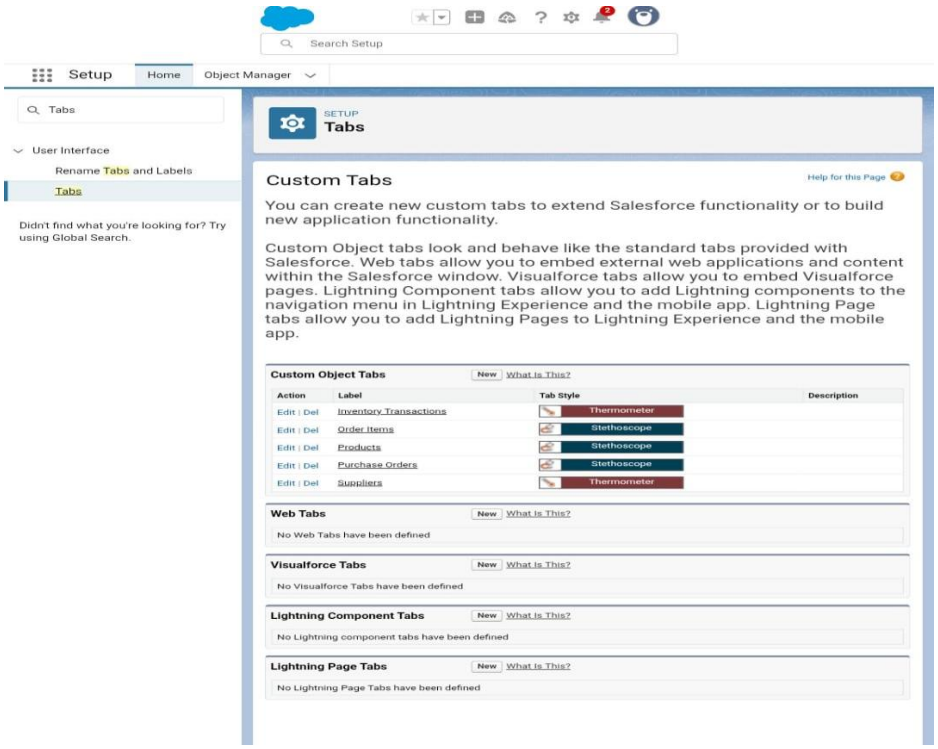
Inventory Transaction Object



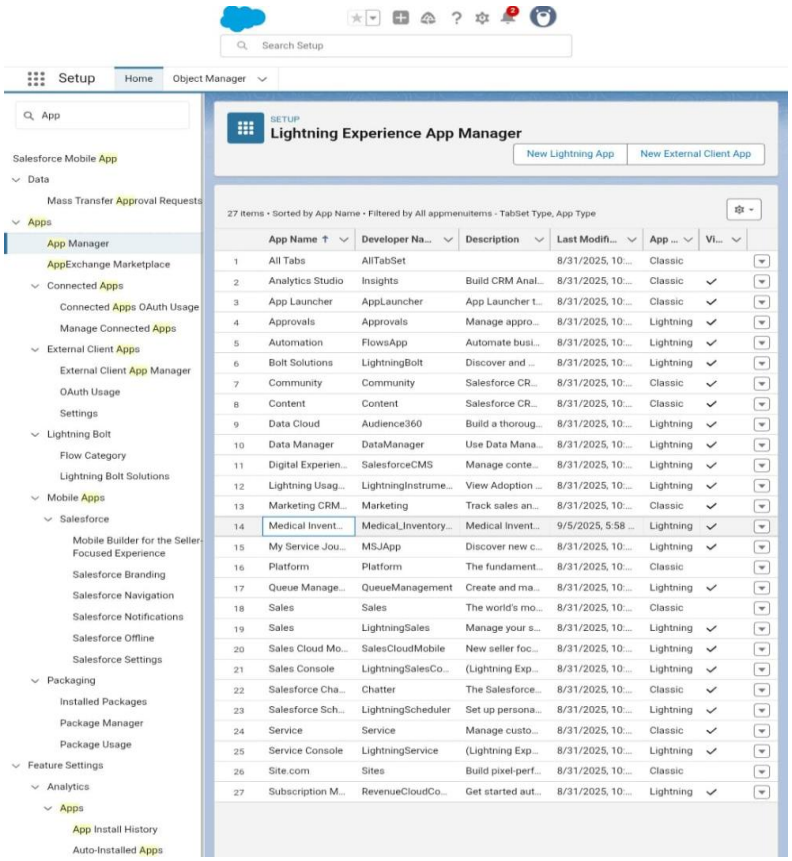
Supplier Object



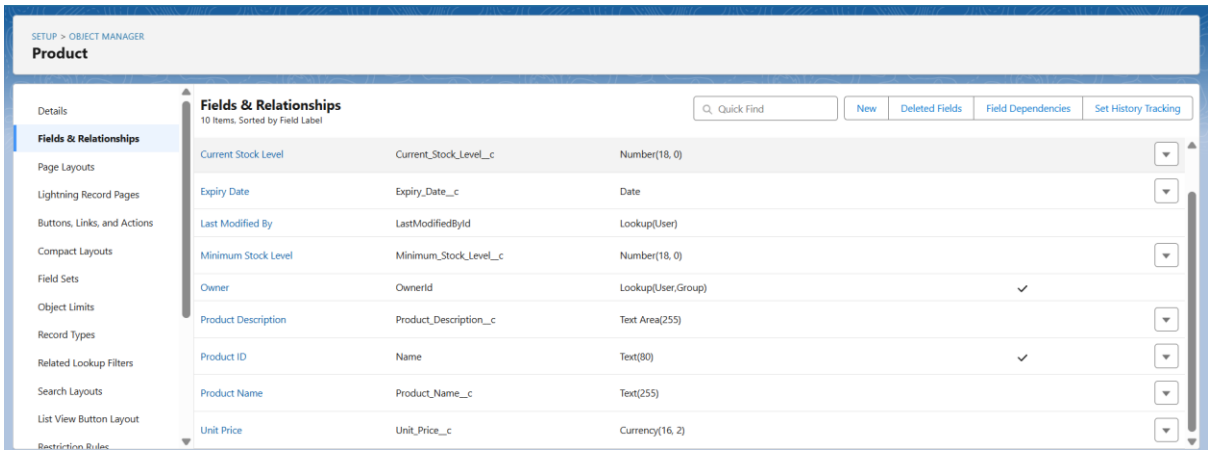
- Created tabs: Product, Purchase Order, Order Item, Inventory Transaction and Supplier objects.



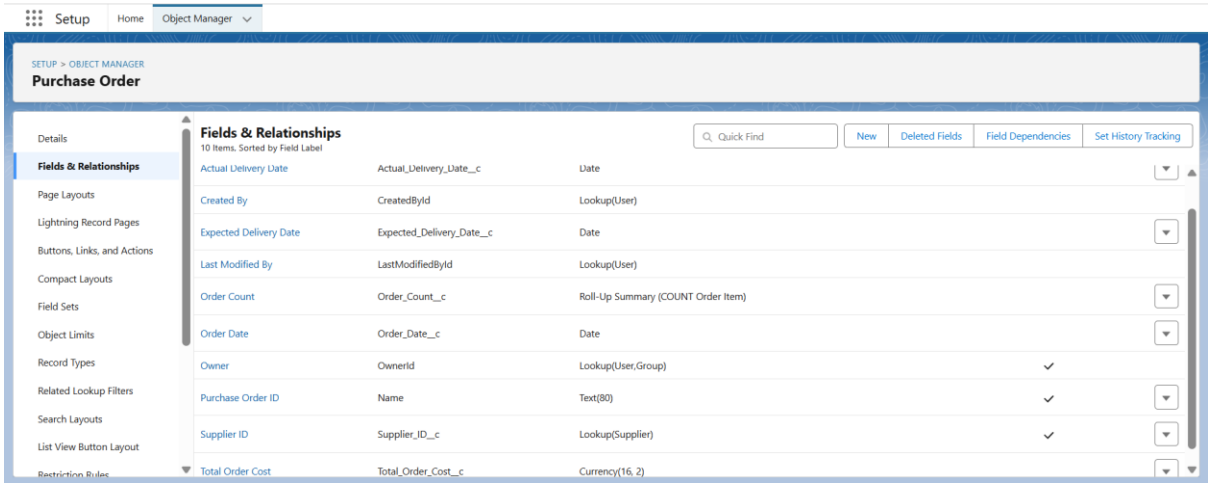
- Creating a Lightning App for Medical Inventory Management.



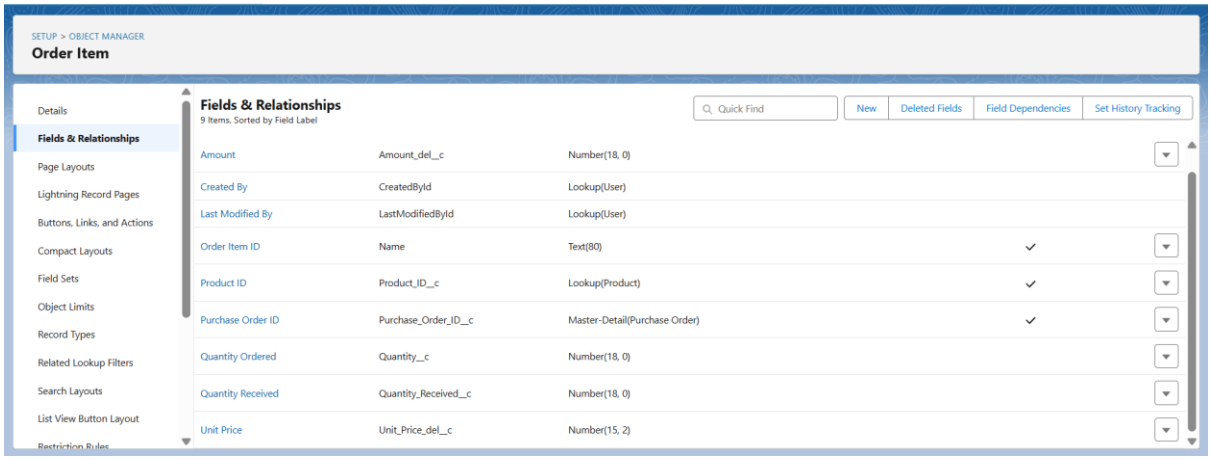
➢ Created a Text Field, Text Area Field, Number Field, Currency Field, Date Field in Product Object.



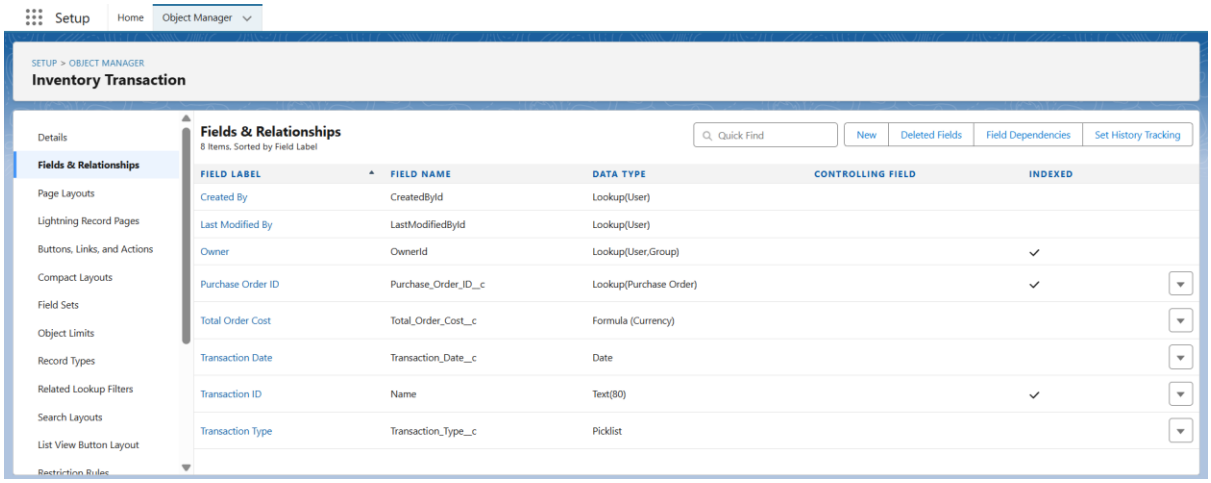
- Created a Text Field, Lookup relationship, Date Field, Roll-up summary Field, Currency field in Purchase Order Object.



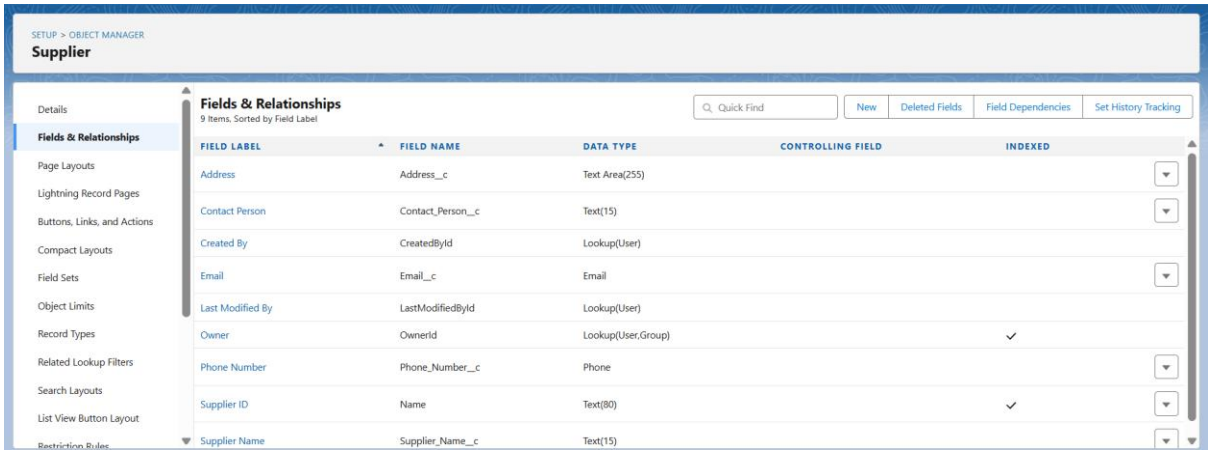
- Created a Text Field, Lookup Relationship, Master-detail, Number Field, Formula Field in Order Item Object.



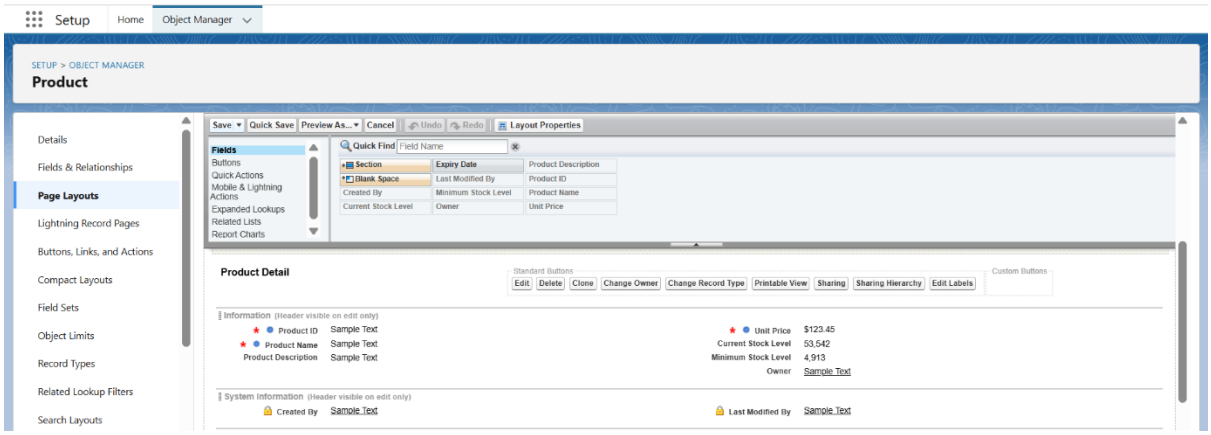
- Created a Text Field, Lookup Relationship, Date Field, Picklist Field, Formula Field in Inventory Transaction Object.

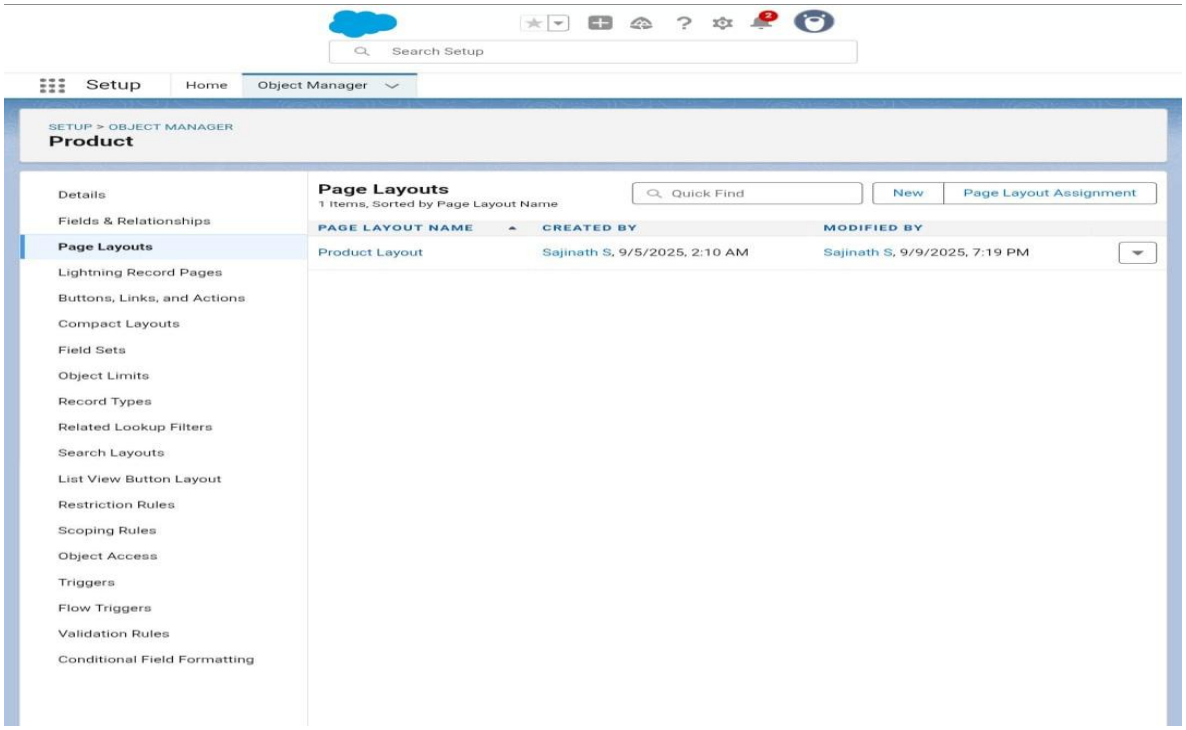


➢ Created a Text field, Phone Field, Email Field, TextArea Field in Supplier Object.

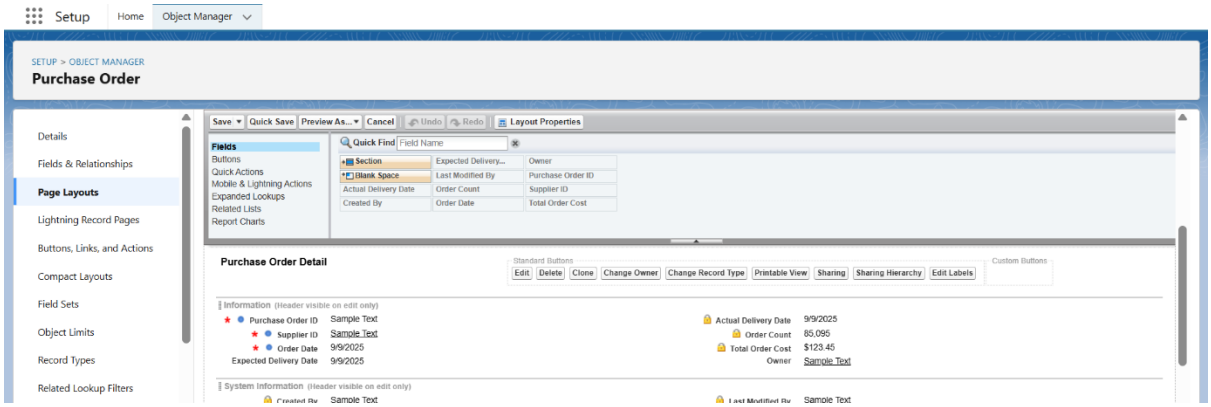


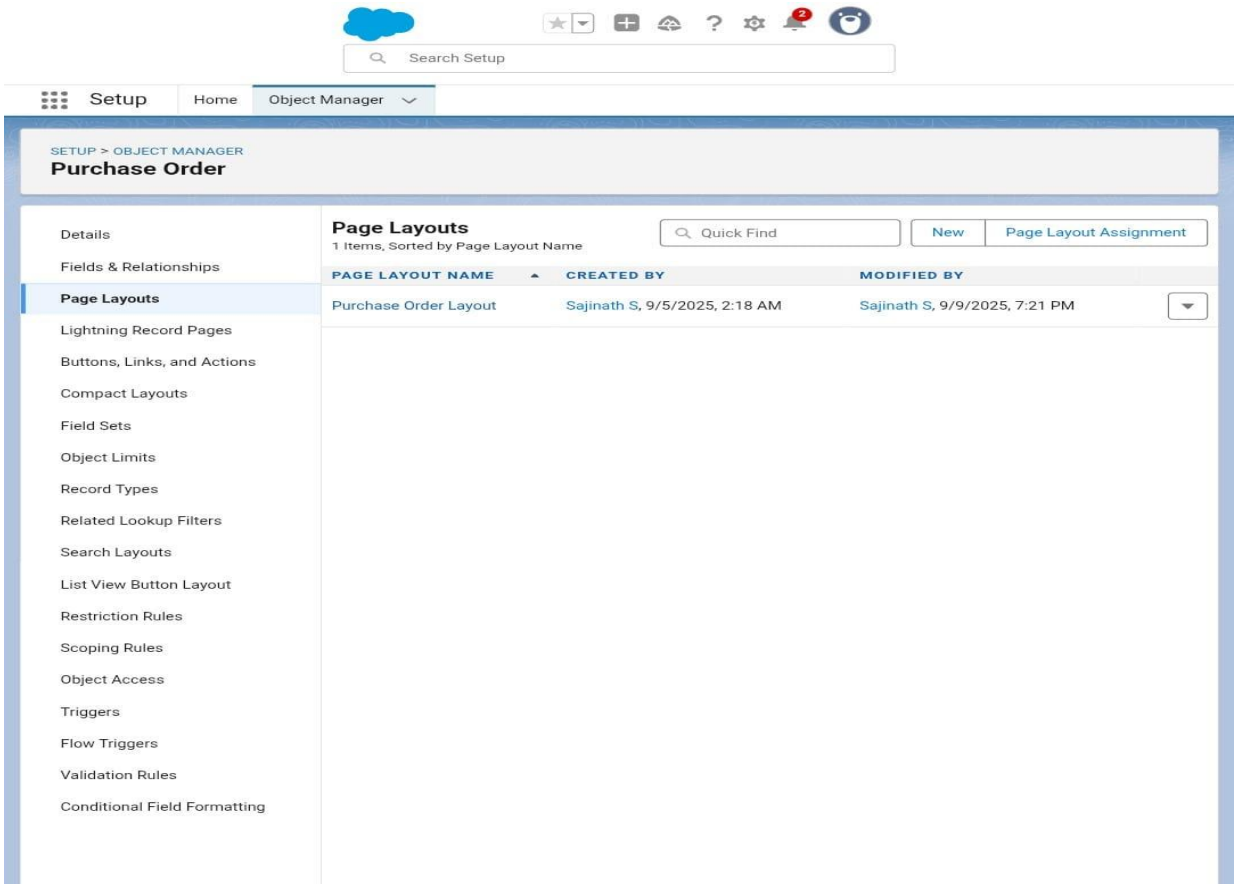
- Edit a Page Layout: Product Object, Purchase Order Object, Order Item Object, Inventory Transaction Object, Supplier Object



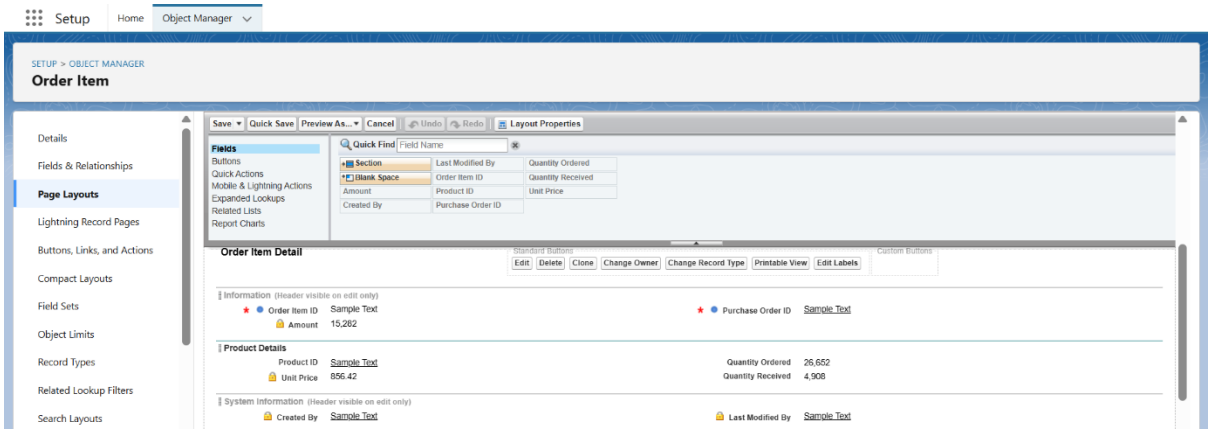


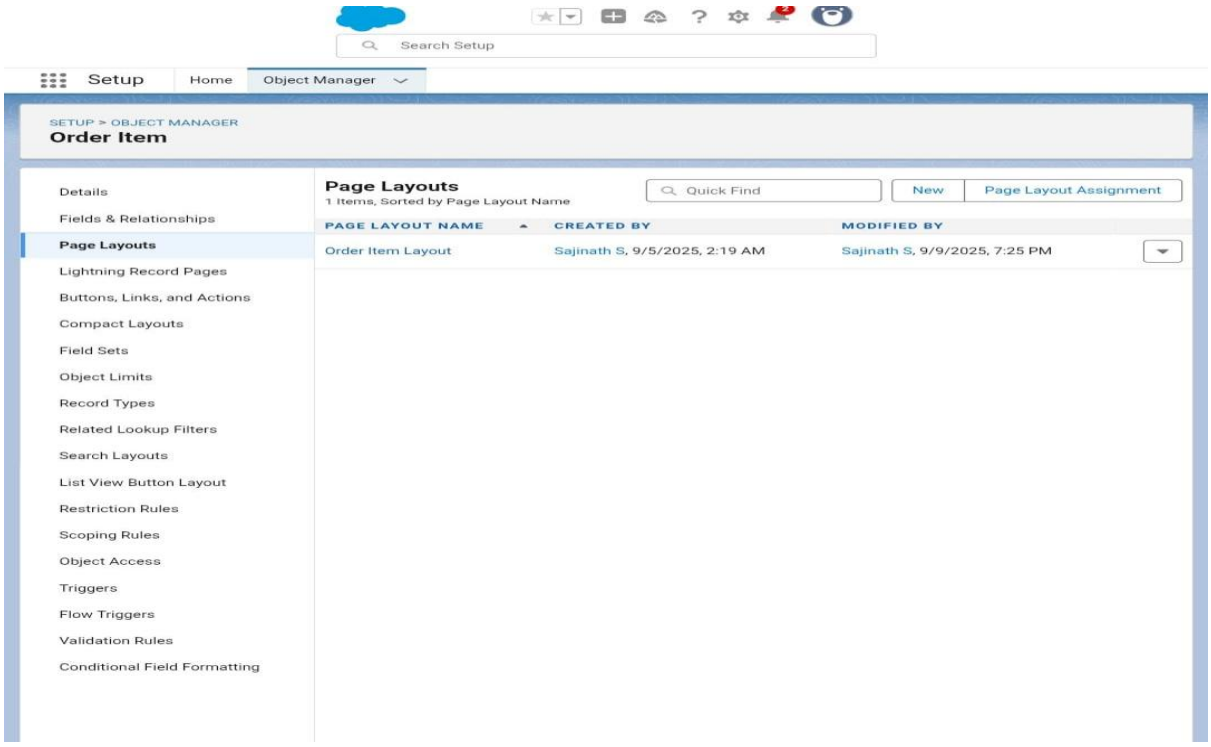
- Purchase Order Object



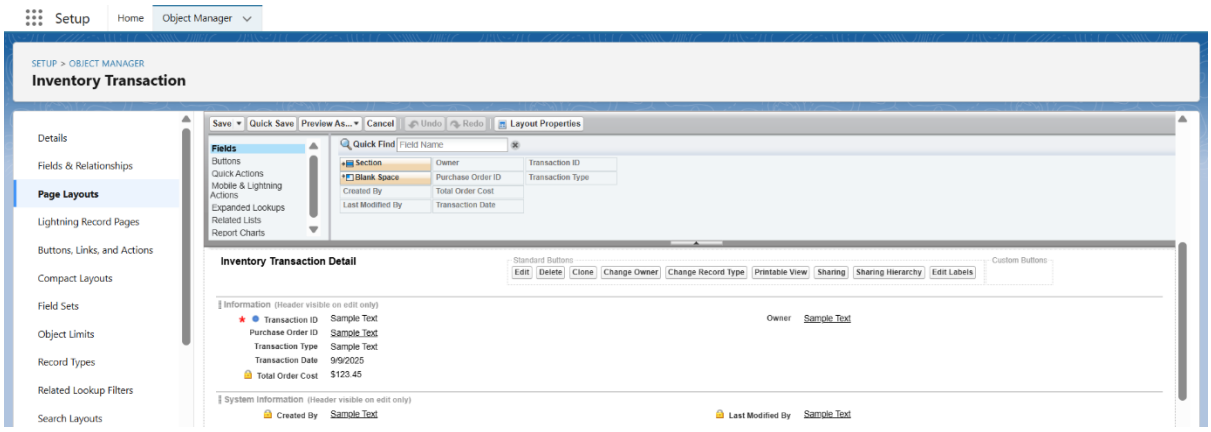


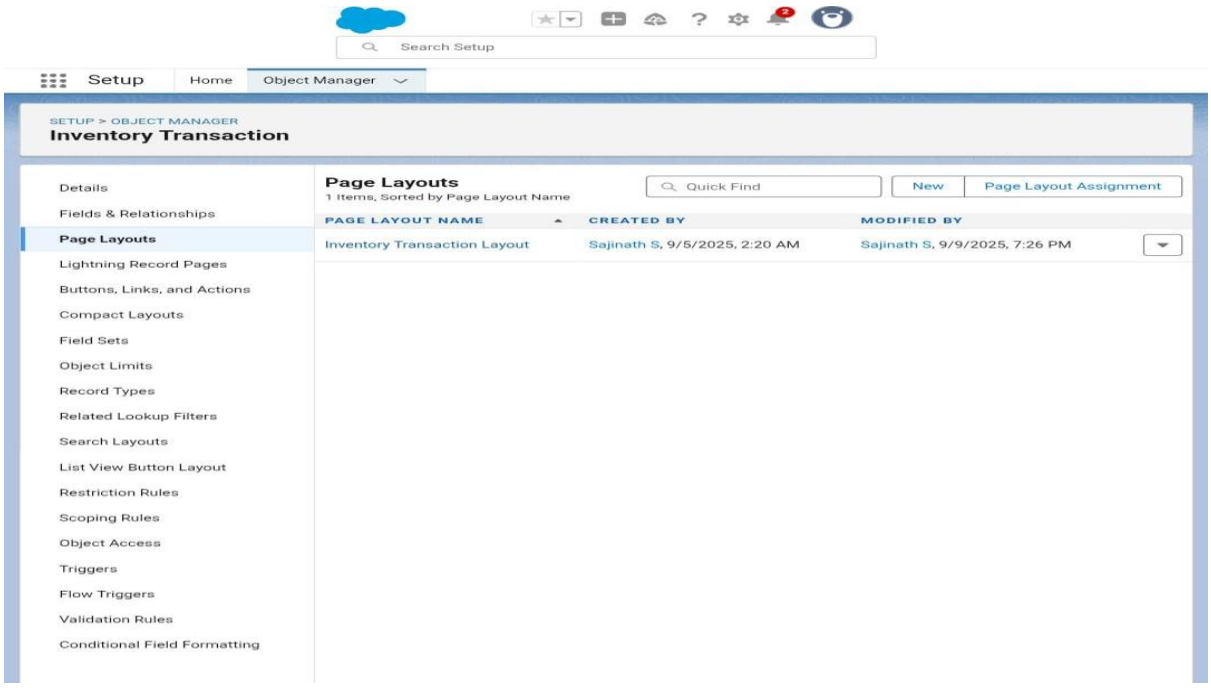
➢ Order Item Object



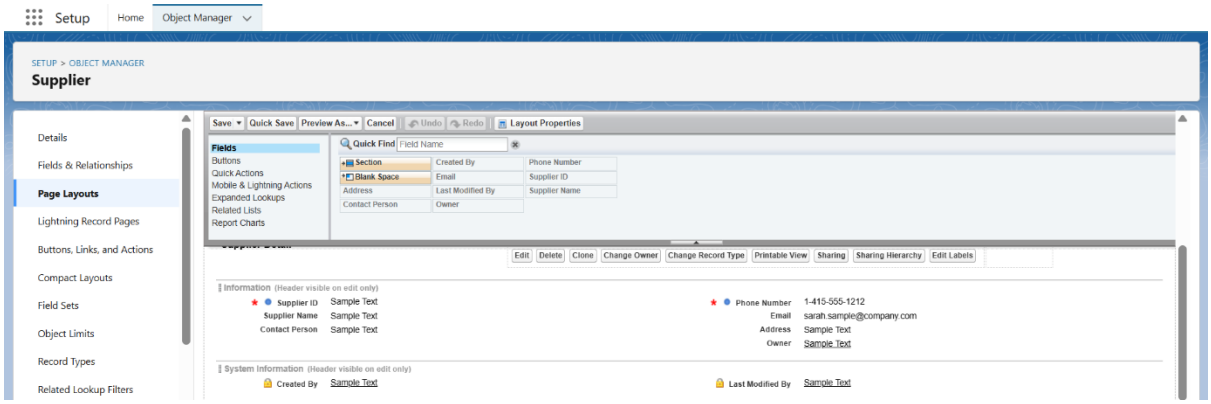


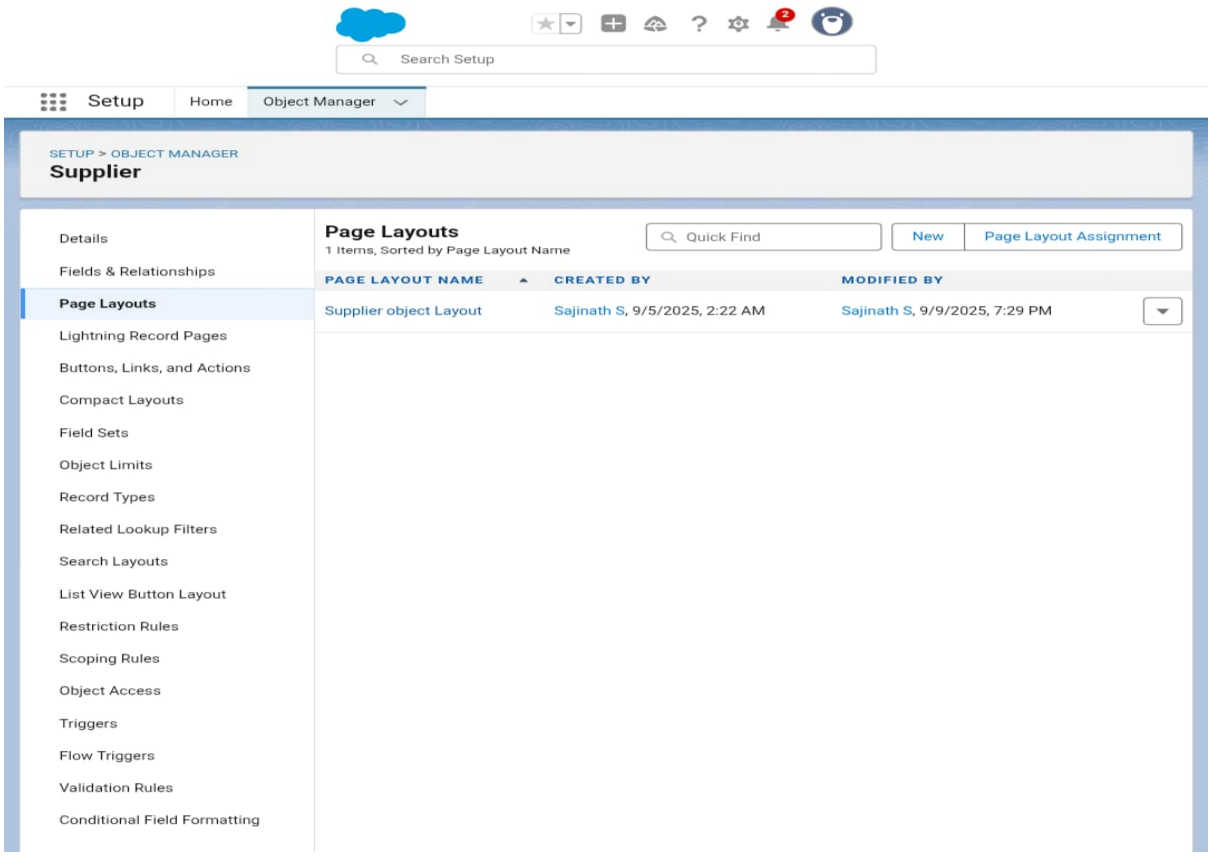
➢ Inventory Transaction Object



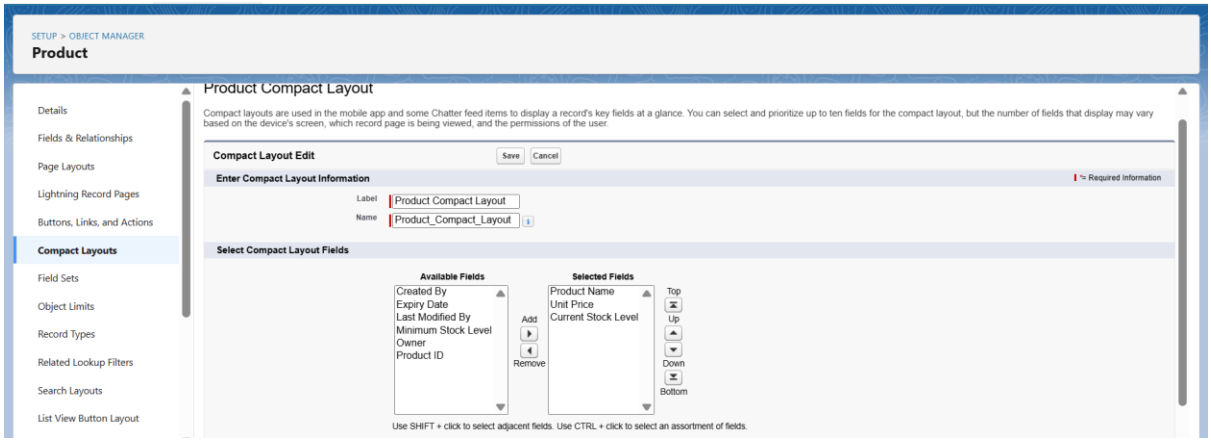


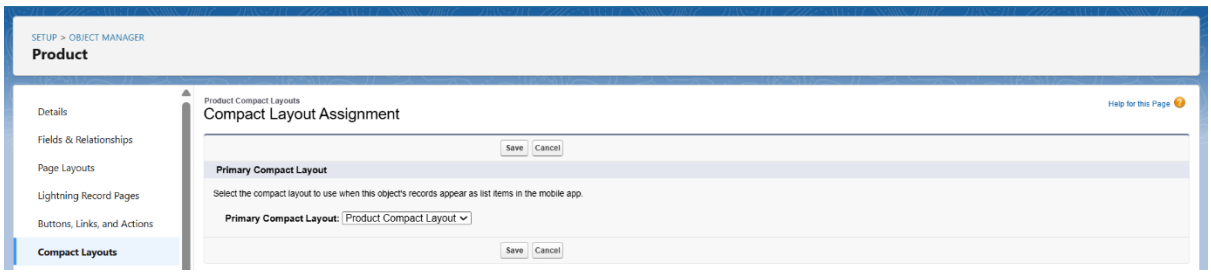
➢ Supplier Object

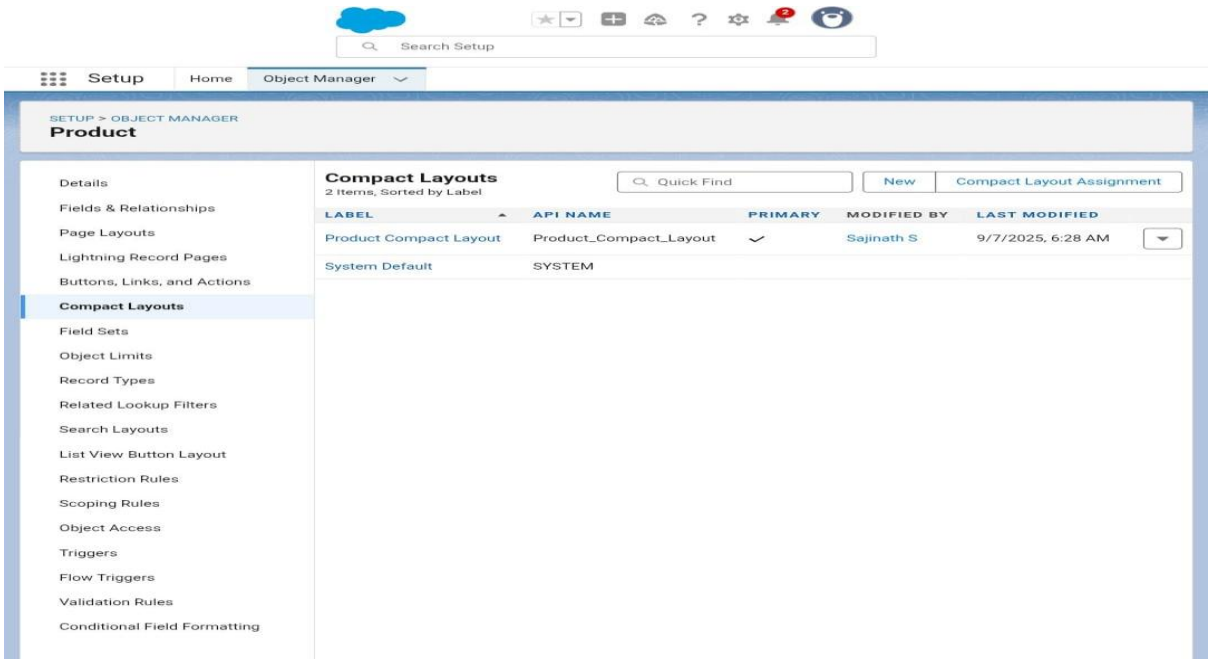




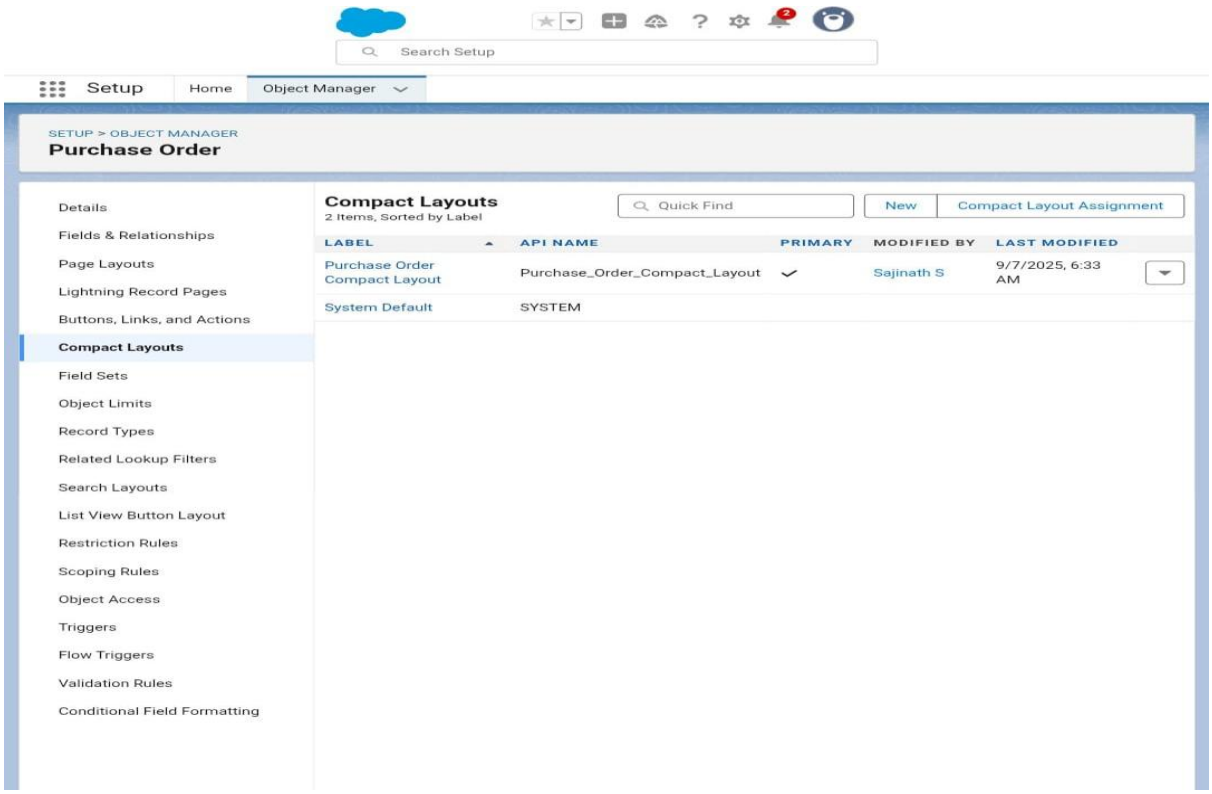
➢ Creating a Compact Layout: Product Object, Purchase Order Object

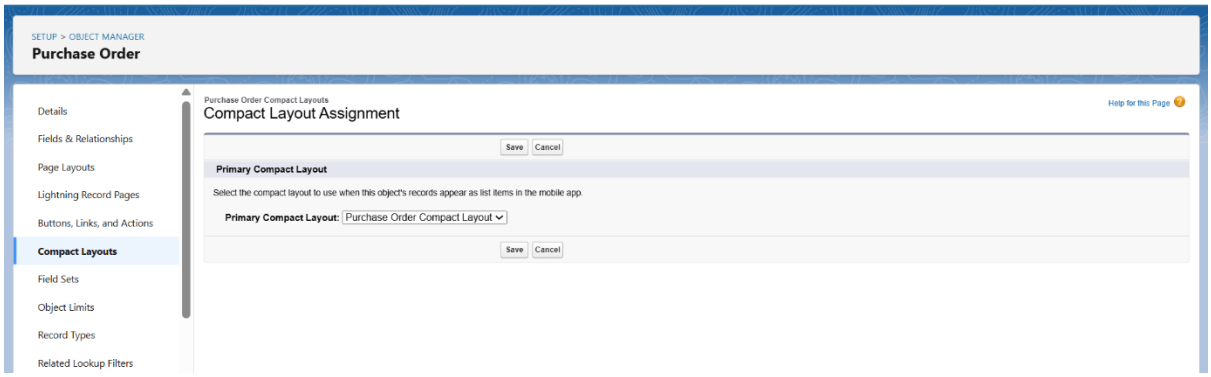


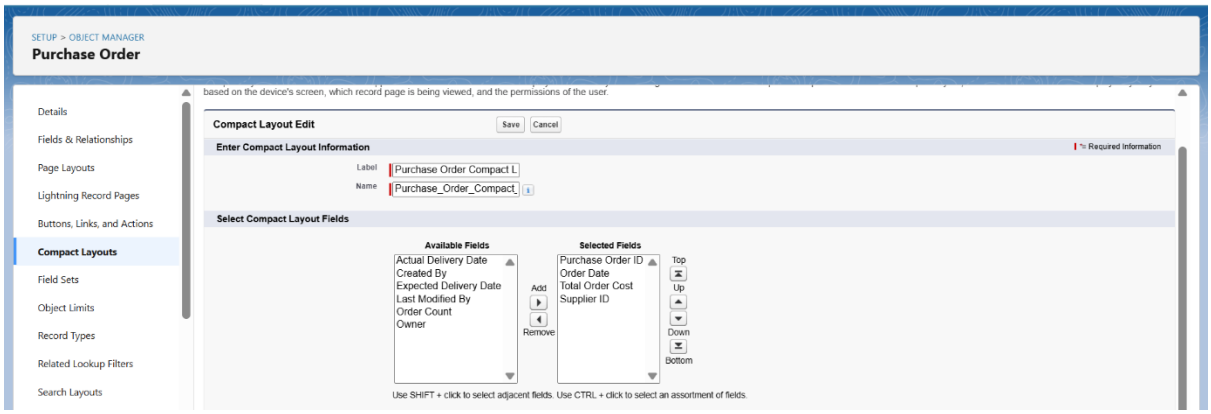




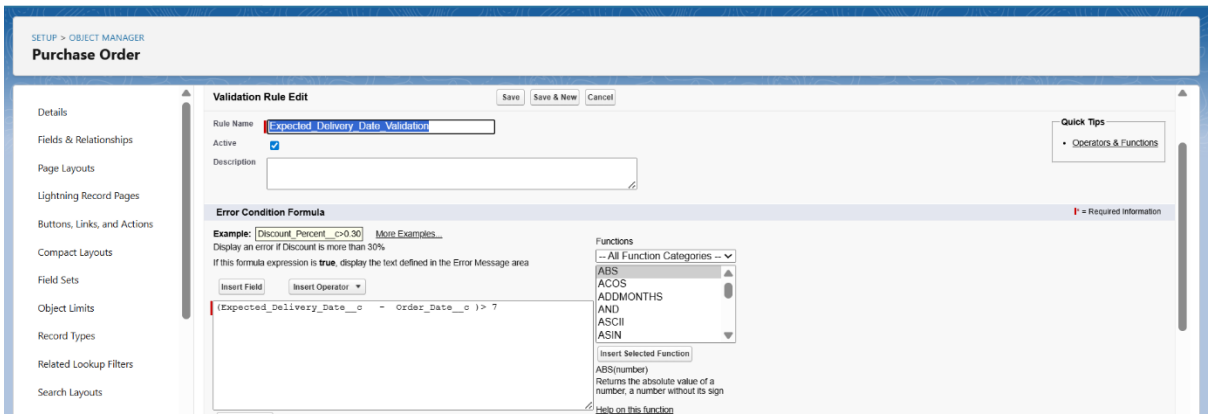
➢ Purchase Order Object

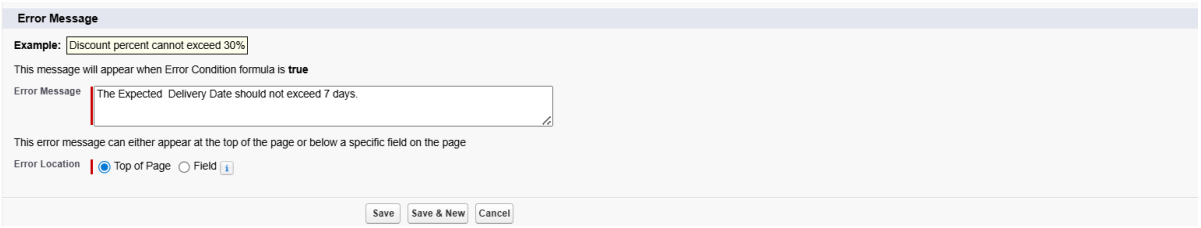


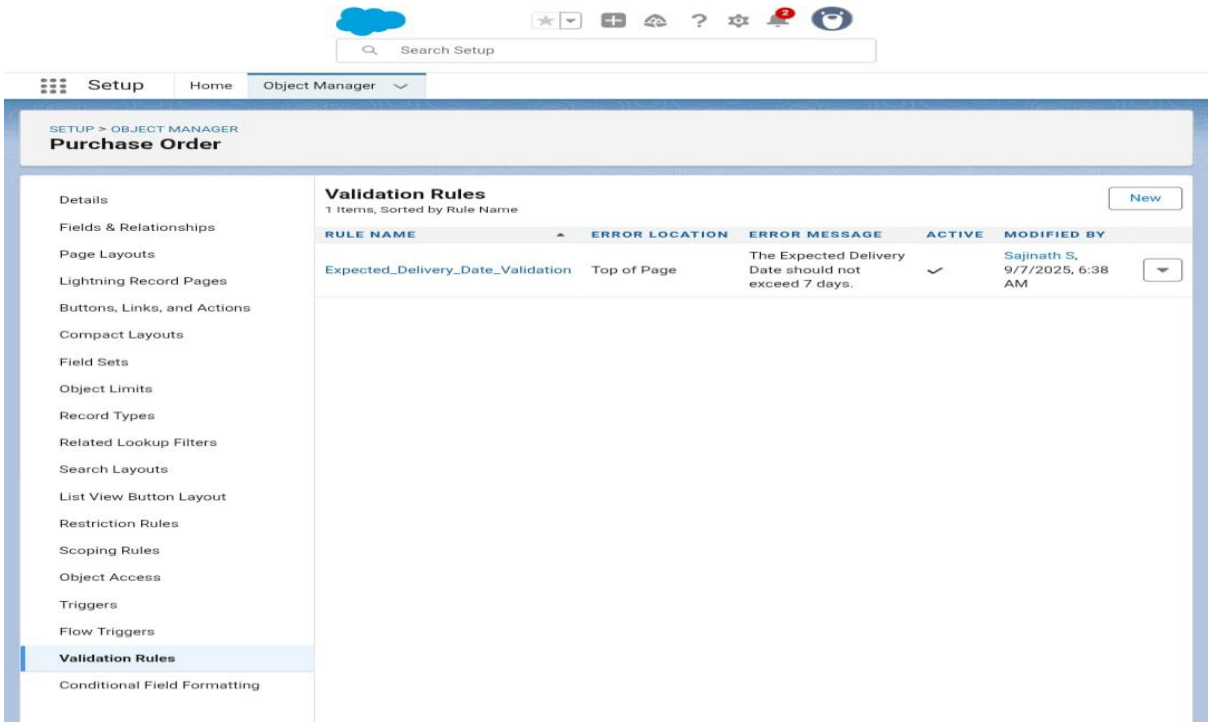




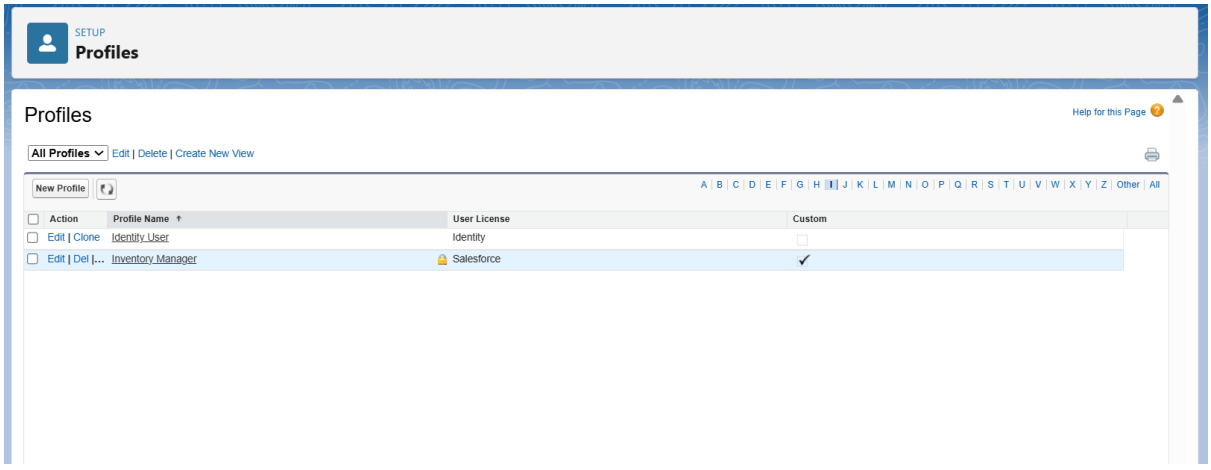
➢ Creating an Expected Delivery Date Validation rule to a Employee Object



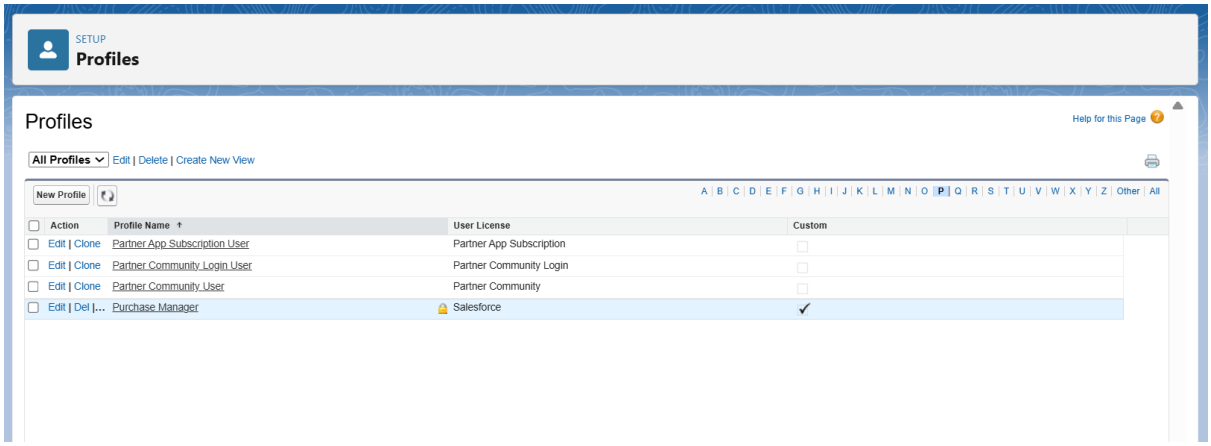




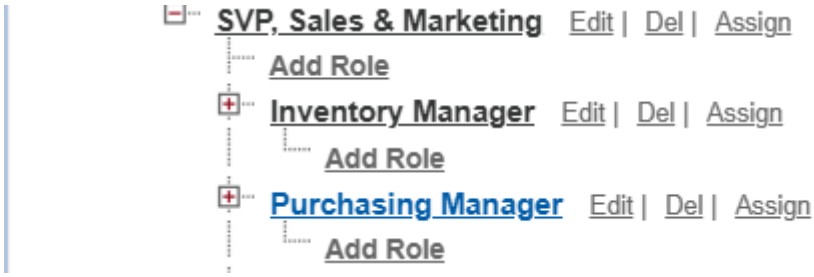
➢ Created a Profile: Inventory Manager Profile, Purchase Manager Profile



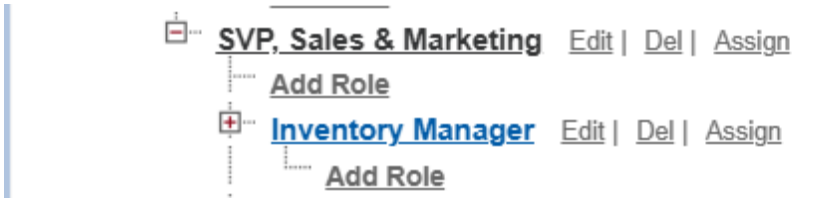
➢ Purchase Manager Profile



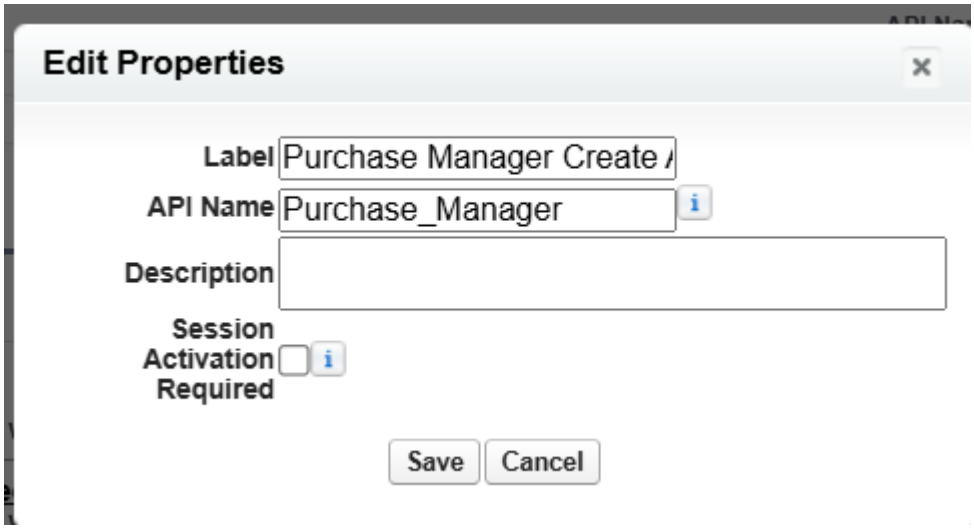
- Created a Role: Purchasing Manager, Inventory Manager.

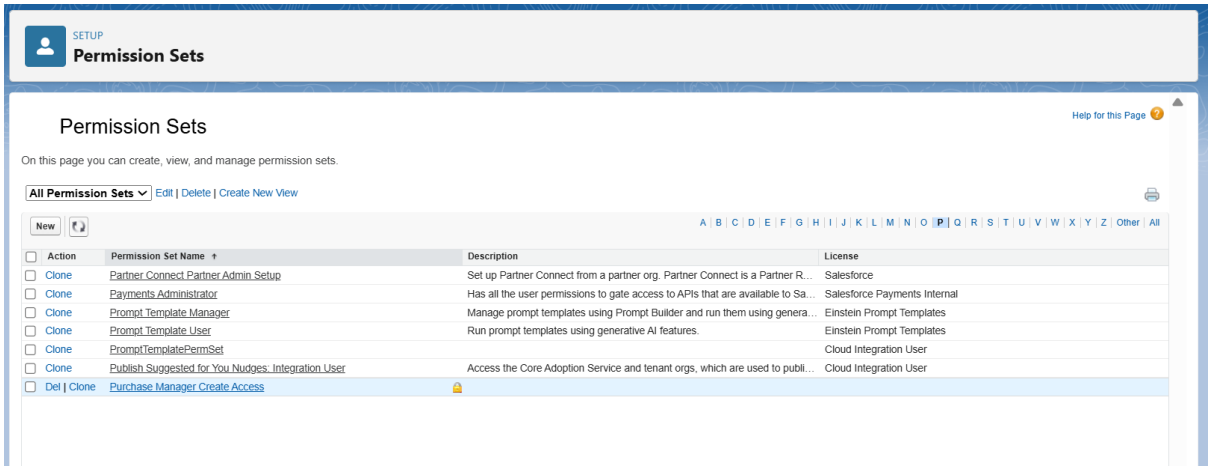


- Inventory Manager

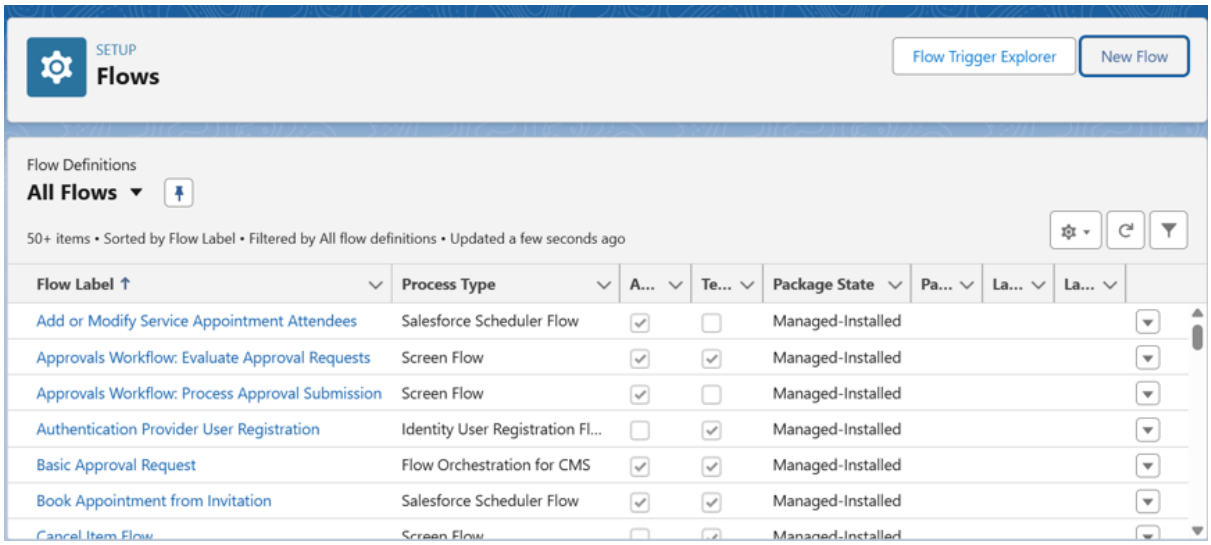


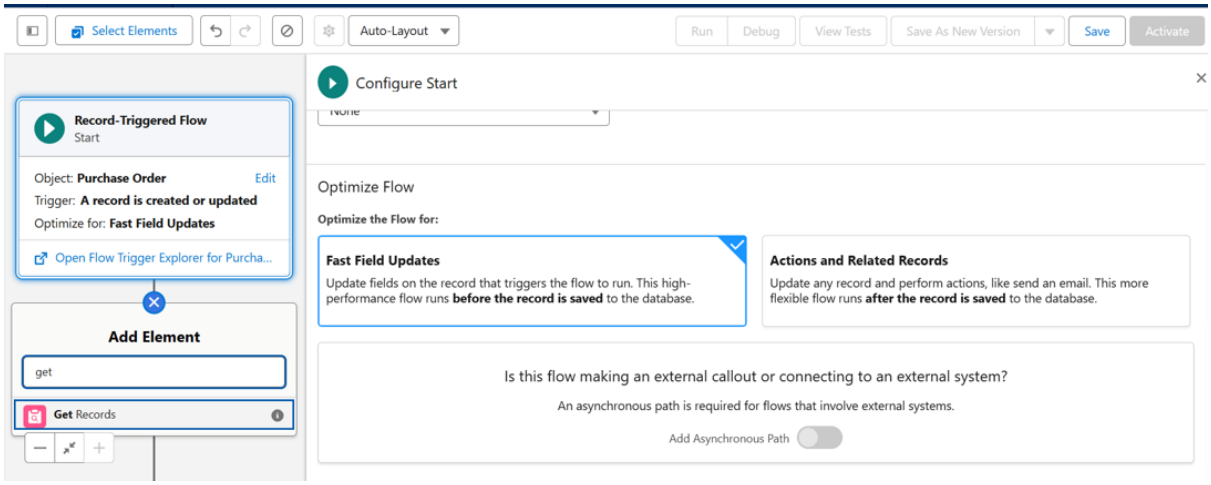
Creating a Permission Set.

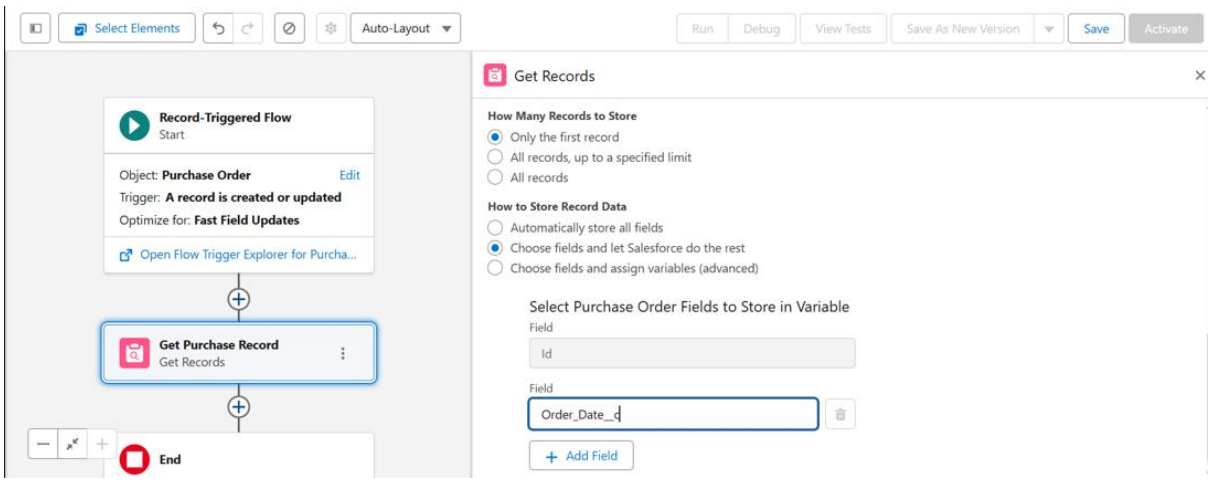


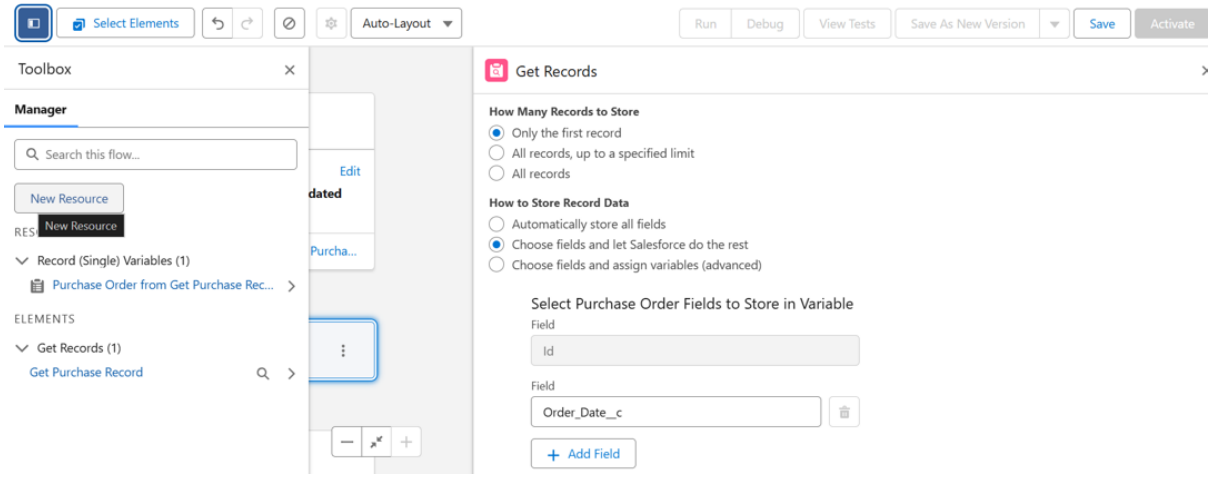


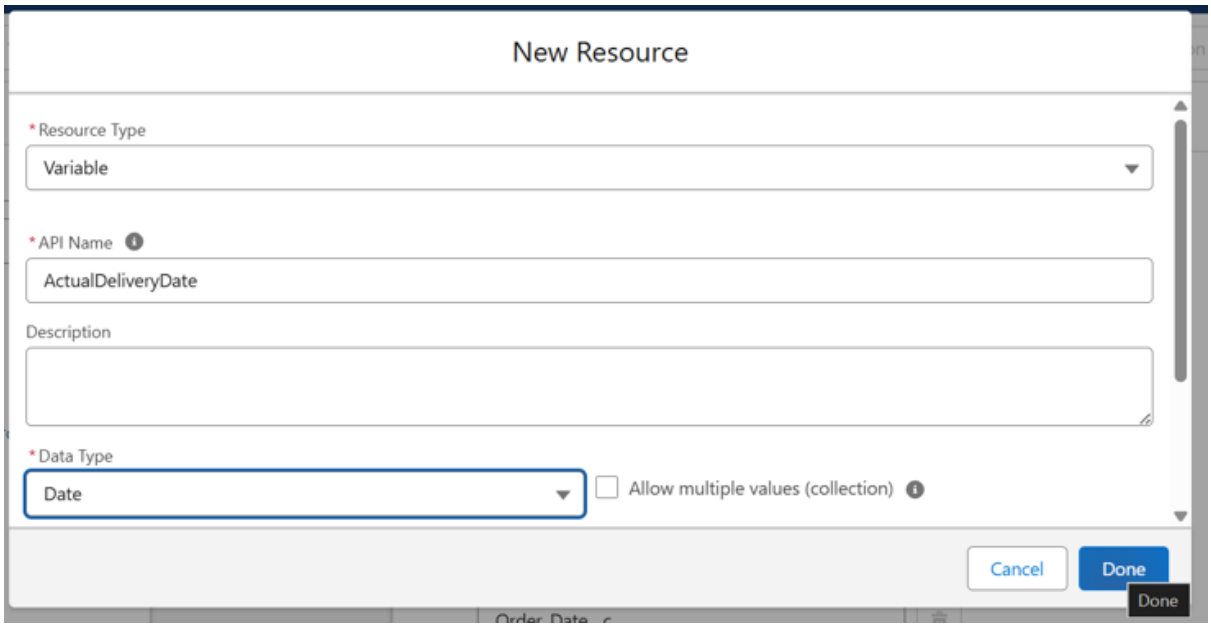
➢ Creating Flow to update the Actual Delivery Date.

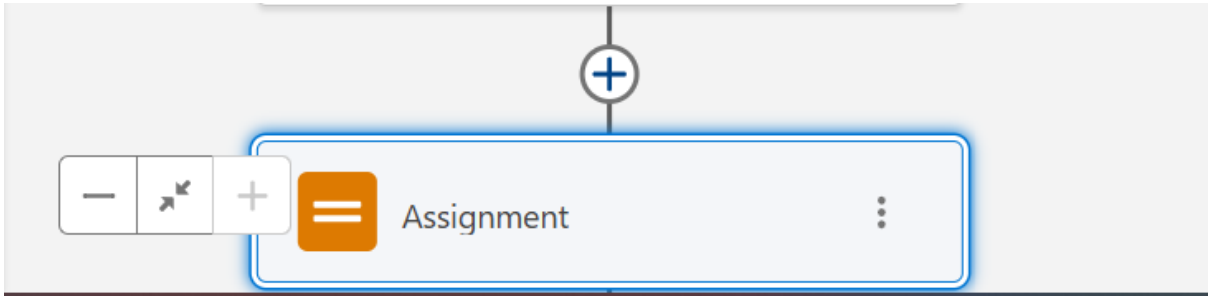







Select Elements
Auto-Layout
Run
Debug
View Tests
Save As New Version
Save
Activate
Toolbox
Manager
Search this flow...
New Resource
New Resource
Record (Single) Variables (1)
Purchase Order from Get Purchase Rec...
ELEMENTS
Get Records (1)
Get Purchase Record
Edit
Get Records
How Many Records to Store
Only the first record
All records, up to a specified limit
All records
How to Store Record Data
Automatically store all fields
Choose fields and let Salesforce do the rest
Choose fields and assign variables (advanced)
Select Purchase Order Fields to Store in Variable
Field
Id
Field
Order_Date__c
Add Field

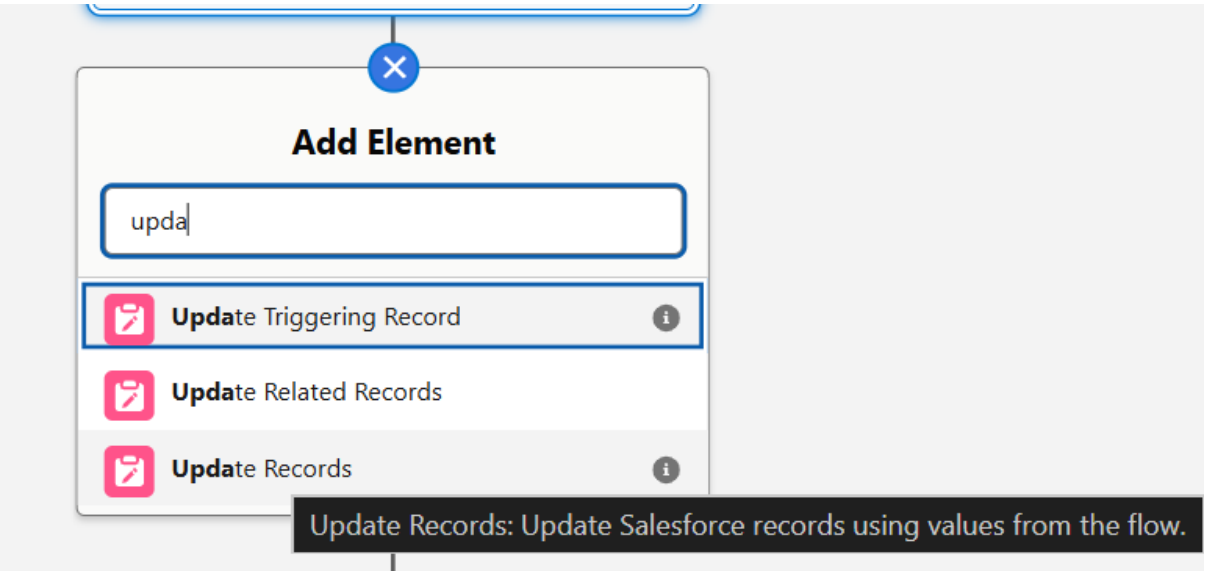

New Resource
*Resource Type
Variable
*API Name
ActualDeliveryDate
Description
*Data Type
Date
Allow multiple values (collection)
Cancel
Done
Done


Assignment

Variable | Operator | Value

ActualDeliveryDate | Add | 3

+ Add Assignment



Update Records

*Label: Updating Purchasing Order

*API Name: Updating_Purchasing_Order

Description

***How to Find Records to Update and Set Their Values**

- (•) Use the purchase order record that triggered the flow
- ( ) Update records related to the purchase order record that triggered the flow
- ( ) Use the IDs and all field values from a record or record collection
- ( ) Specify conditions to identify records, and set fields individually

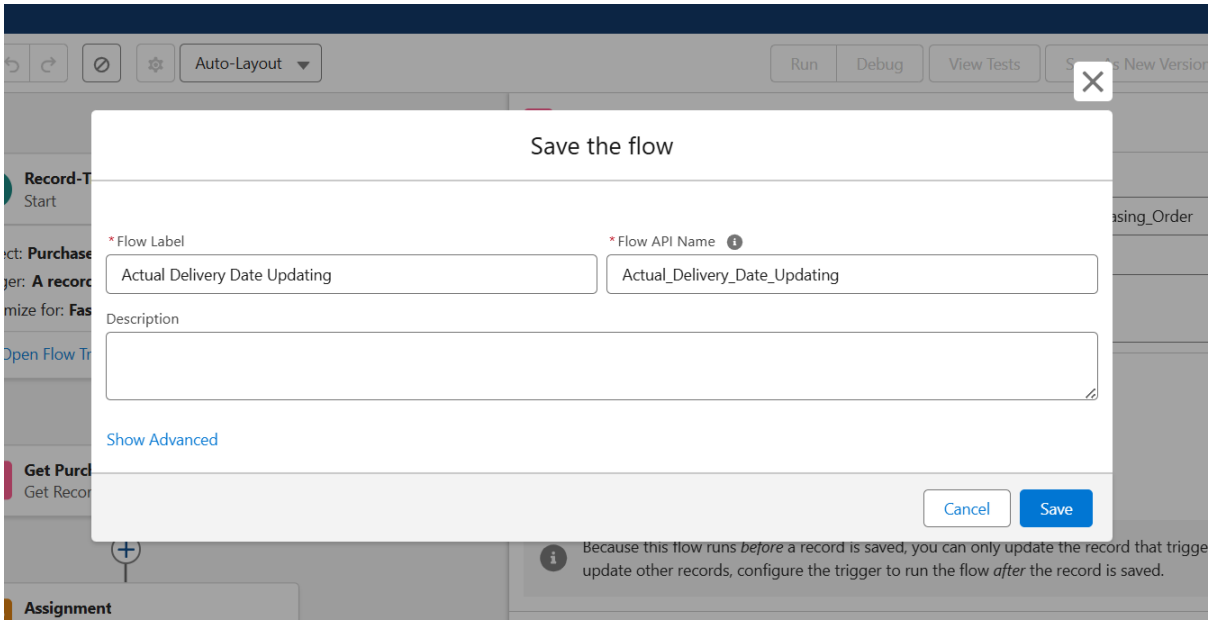
Because this flow runs *before* a record is saved, you can only update the record that triggered the flow to run. To update other records, configure the trigger to run the flow *after* the record is saved.

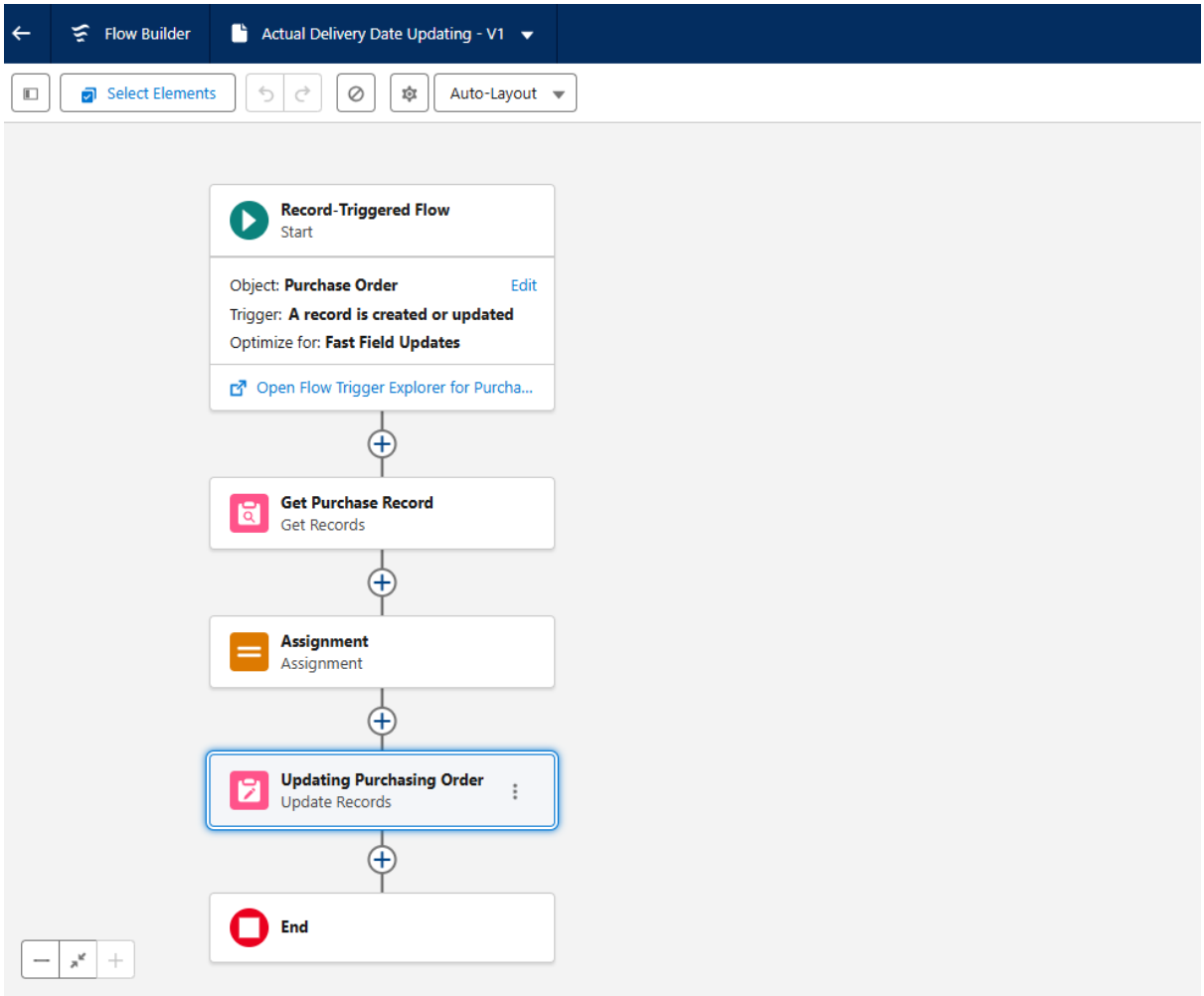
Set Filter Conditions

Condition Requirements to Update Record: None—Always Update Record

Set Field Values for the Purchase Order Record

Field: Actual Delivery Date ← Value: ActualDeliveryDate


Auto-Layout
Run
Debug
View Tests
Save the flow
*Flow Label
Actual Delivery Date Updating
*Flow API Name
Actual_Delivery_Date_Updating
Description
Show Advanced
Cancel
Save
Start
Assignment

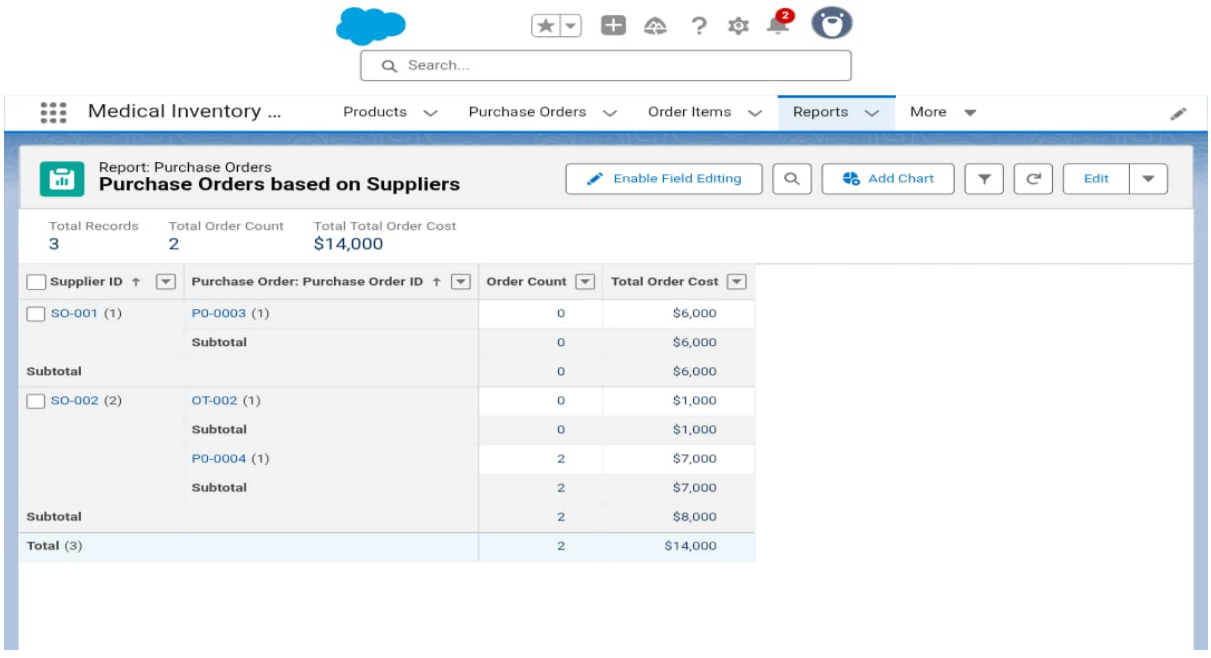

Flow Builder
Actual Delivery Date Updating - V1
Select Elements
Auto-Layout
Record-Triggered Flow
Start
Object: Purchase Order
Edit
Trigger: A record is created or updated
Optimize for: Fast Field Updates
Open Flow Trigger Explorer for Purcha...
Get Purchase Record
Get Records
Assignment
Assignment
Updating Purchasing Order
Update Records
End

➢ Create a Trigger to Calculate total amount on Order Item.

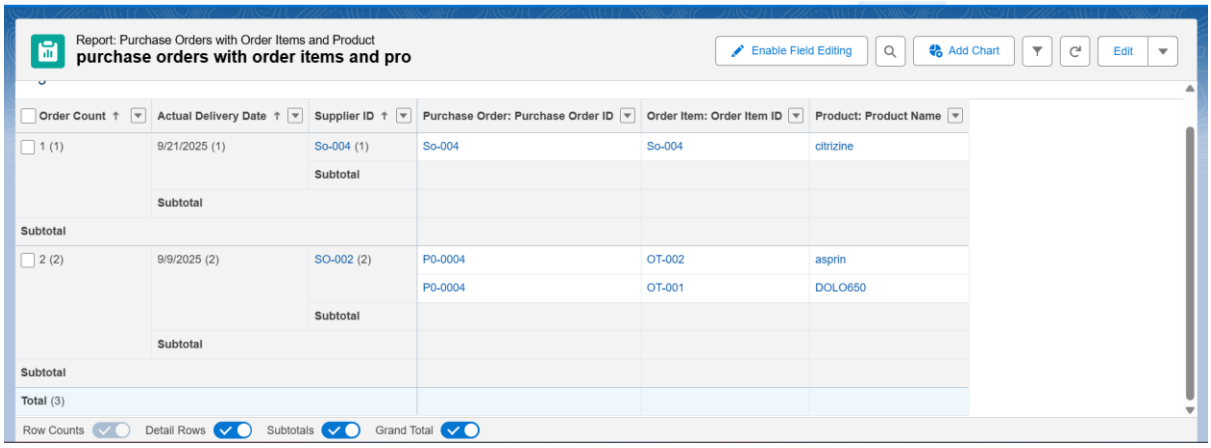
```
trigger CalculateTotalAmountTrigger on Order_Item__c (after insert, after update, after delete, after undelete) {

    // Call the handler class to handle the logic

    CalculateTotalAmountHandler.calculateTotal(Trigger.new, Trigger.old, Trigger.isInsert, Trigger.isUpdate, Trigger.isDelet

}
```

```
public class CalculateTotalAmountHandler {



    // Method to calculate the total amount for Purchase Orders based on related Order Items

    public static void calculateTotal(List<Order_Item__c> newItems, List<Order_Item__c> oldItems, Boolean isInsert, Bool
```

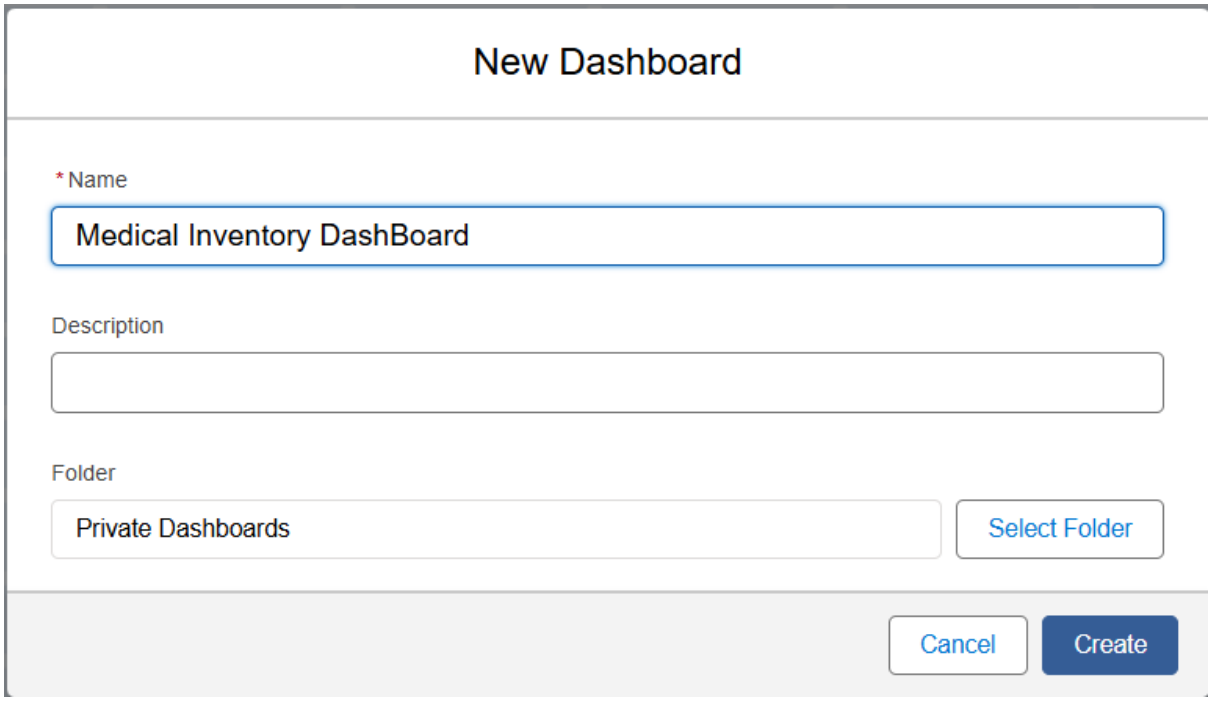
➢ Created Reports: Purchase Orders Based on Suppliers summary), Purchase Details



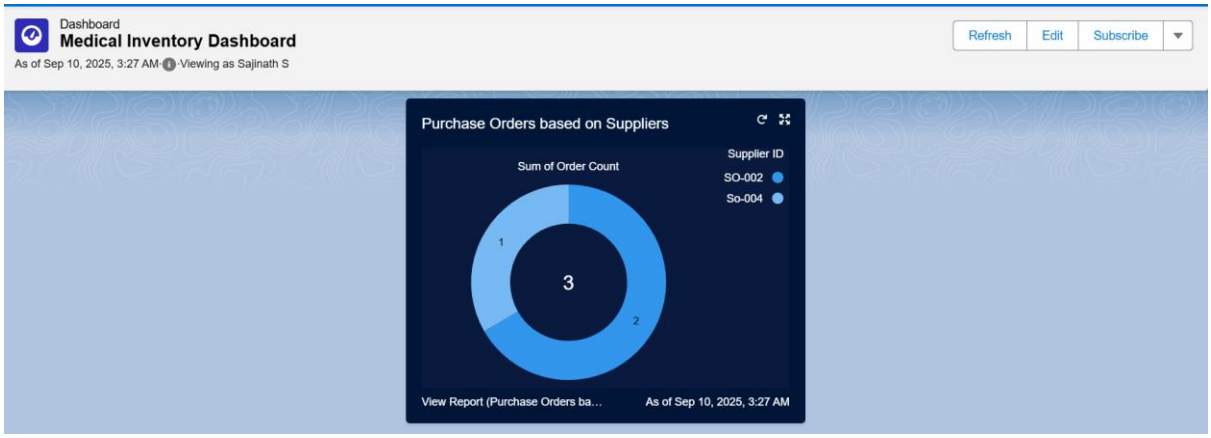
➢ Purchase Details



➢ Created Dashboard



➢ View Dashboard



➢ Advantages and Disadvantages of Medical Inventory Management

■ Advantages

> Centralized Inventory Tracking – Manage all medicines, supplies, and equipment in one place.

> Real-Time Stock Visibility – Check available stock instantly across departments or branches.

> Automation – Auto-reordering when stock is low, expiry alerts, and stock updates.

> Integration – Connects with ERP, EHR, pharmacy, billing, and POS systems.

> Compliance Support – Helps follow FDA, HIPAA, CDSCO and other medical regulations.

> Advanced Reporting – Generate audit-ready reports on usage, expiry, and stock history.

> Better Patient Care – Ensures required medicines and equipment are available on time.

> Scalability – Works for small clinics, hospitals, and large healthcare chains.

> Customization – Workflows, dashboards, and automation can be fully tailored.

> Improved Decision-Making – Data-driven insights help in better procurement planning.

▪ Disadvantages

> High Setup Cost – Salesforce licensing and customization are expensive.

> Complex Customization – Advanced medical inventory needs skilled Salesforce developers.

> Training Requirement – Staff, nurses, and pharmacists need proper training to use it.

> Internet Dependency – Being cloud-based, it requires a stable connection.

> Data Security Concerns – Sensitive healthcare data needs strict security controls.

> Ongoing Maintenance Costs – Continuous updates, integration fixes, and enhancements add expenses.

> Integration Challenges – Linking with existing hospital systems can be complex.

> User Adoption Issues – If staff don't update stock properly, data becomes inaccurate.

> Compliance Risk – Misconfigured access or incorrect reporting may cause legal issues.

> Overkill for Small Clinics – For small setups, Salesforce may be too heavy and costly.

➢ Conclusion

Medical inventory management in Salesforce provides healthcare organizations with a centralized, efficient, and automated platform to track, manage, and optimize their medical supplies. By integrating real-time inventory tracking, automated stock updates, and detailed reporting, Salesforce helps reduce manual errors, avoid stockouts or overstocking, and ensure timely availability of essential medical items.

However, successful implementation requires proper configuration, user training, and data accuracy. When used effectively, Salesforce empowers hospitals, clinics, and healthcare providers to streamline operations, improve patient care, and reduce operational costs, making it a powerful solution for modern medical inventory management.

➢ APPENDIX

Source Code: Provided in Apex Triggers and Apex Classes

- **CalculateTotalAmountTrigger**

```
trigger CalculateTotalAmountTrigger on Order_Item__c (after
insert, after update, after delete, after undelete) {
    // Call the handler class to handle the logic
    CalculateTotalAmountHandler.calculateTotal(Trigger.new,
Trigger.old, Trigger.isInsert, Trigger.isUpdate, Trigger.isDelete,
Trigger.isUndelete);
}
```

- **CalculateTotalAmountHandler**

```
 public class CalculateTotalAmountHandler {

    // Method to calculate the total amount for Purchase Orders
based on related Order Items
    public static void calculateTotal(List<Order_Item__c>
newItems, List<Order_Item__c> oldItems, Boolean isInsert,
Boolean isUpdate, Boolean isDelete, Boolean isUndelete) {

        // Collect Purchase Order IDs affected by changes in
Order_Item__c records
        Set<Id> parentIds = new Set<Id>();

        // For insert, update, and undelete scenarios
        if (isInsert || isUpdate || isUndelete) {
            for (Order_Item__c ordItem : newItems) {
                parentIds.add(ordItem.Purchase_Order_Id__);
            }
```

```
        }

        // For update and delete scenarios
        if (isUpdate || isDelete) {
            for (Order_Item__c ordItem : oldItems) {
                parentIds.add(ordItem.Purchase_Order_Id__c);
            }
        }

        // Calculate the total amounts for affected Purchase
Orders
        Map<Id, Decimal> purchaseToUpdateMap = new Map<Id,
Decimal>();

        if (!parentIds.isEmpty()) {
            // Perform an aggregate query to sum the Amount__c
for each Purchase Order
            List<AggregateResult> aggrList = [
                SELECT Purchase_Order_Id__c, SUM(Amount__c)
totalAmount
                FROM Order_Item__c
                WHERE Purchase_Order_Id__c IN :parentIds
                GROUP BY Purchase_Order_Id__c
            ];

            // Map the result to Purchase Order IDs
            for (AggregateResult aggr : aggrList) {
                Id purchaseOrderId =
(Id)aggr.get('Purchase_Order_Id__c');
                Decimal totalAmount =
(Decimal)aggr.get('totalAmount');
                purchaseToUpdateMap.put(purchaseOrderId,
totalAmount);
            }

            // Prepare Purchase Order records for update
```

```
            List<Purchase_Order__c> purchaseToUpdate = new
List<Purchase_Order__c>();
            for (Id purchaseOrderId :
purchaseToUpdateMap.keySet()) {
                Purchase_Order__c purchaseOrder = new
Purchase_Order__c(Id = purchaseOrderId,
Total_Order_cost__c =
purchaseToUpdateMap.get(purchaseOrderId));
                purchaseToUpdate.add(purchaseOrder);
            }

            // Update Purchase Orders if there are any changes
            if (!purchaseToUpdate.isEmpty()) {
                update purchaseToUpdate;
            }
        }
    }
}
```