Phase 5: Apex Programming (Developer)

PROJECT TITLE:-

Expense On a Page: An expense approval & insight system.
Industry: Finance / Corporate Expense Management.
Target User: Employees, Managers, and Finance Teams.

WORKING OF THE PHASE:-

In the whole Phase we will be working on the ***Developer Console.***

CLASSES & OBJECT:-

Making a class for the business logic for the *Expense and Expense_Line object*-

- It will be reusable class that will be used further in the project.
- The class will calculate the Total Amount of the Expense Object by adding the Amount of all the Expense_Line object those who have the same Expense Object ID.

Apex Code

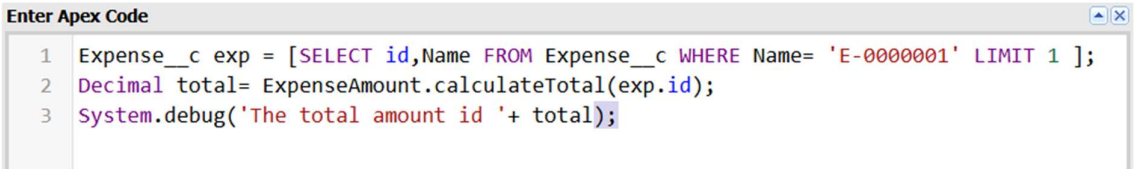
1. *Apex Class:*

```
public class ExpenseAmount {
   public static Decimal calculateTotal(Id expenseId){
     List<Expense_Line__c> lines = [
         SELECT Amount__c FROM Expense_Line__c WHERE Expense__c =
:expenseId
      ];
      Decimal total = 0;
      for (Expense_Line__c line : lines) {
         total += line.Amount__c;
      }
      return total;
   }

}
```

```
public class ExpenseAmount {
    public static Decimal calculateTotal(Id expenseId){
      List<Expense_Line__c> lines = [
            SELECT Amount__c FROM Expense_Line__c WHERE Expense__c = :expenseId
        ];
        Decimal total = 0;
        for (Expense_Line__c line : lines) {
            total += line.Amount__c;
        }
        return total;
    }

}
```

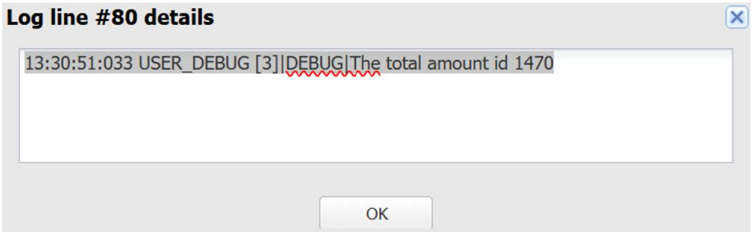
2. *Anonymous Window Code:*

```
Expense__c exp = [SELECT id,Name FROM Expense__c WHERE Name= 'E-0000001' LIMIT 1 ];
Decimal total= ExpenseAmount.calculateTotal(exp.id);
System.debug('The total amount id '+ total);
```



3. *Test Output:*

13:30:51:033 USER_DEBUG [3]|DEBUG|The total amount id 1470



<u>TRIGGERS DESIGN PATTERN:-</u>

Making a *Trigger Handle*r class so the Trigger that we make after this s*tays clean and follow all the best practices.* This class will-

- Will *update the Total Amount of the Expense object*.
- Will be the previous class we made to calculate the Total Amount: ExpenseAmount.

Apex Code

1. *Apex Class:*

```
public class UpdateTotalHandler {
   public static void updateTotals(Set<Id> expenseIds){
      List<Expense__c> expensesToUpdate = new List<Expense__c>();
      for(Id expId : expenseIds){
         Decimal total = ExpenseAmount.calculateTotal(expId);
         expensesToUpdate.add(new Expense__c(Id = expId, Total_Amount__c = total));
      }
      if(!expensesToUpdate.isEmpty()){
         update expensesToUpdate;
```

```
    }
  }
}
```

```
public class UpdateTotalHandler {
    public static void updateTotals(Set<Id> expenseIds){
        List<Expense__c> expensesToUpdate = new List<Expense__c>();
        for(Id expId : expenseIds){
            Decimal total = ExpenseAmount.calculateTotal(expId);
            expensesToUpdate.add(new Expense__c(Id = expId, Total_Amount__c = total));
        }
        if(!expensesToUpdate.isEmpty()){
            update expensesToUpdate;
        }
    }

}
```

2. *Anonymous Window Code:*

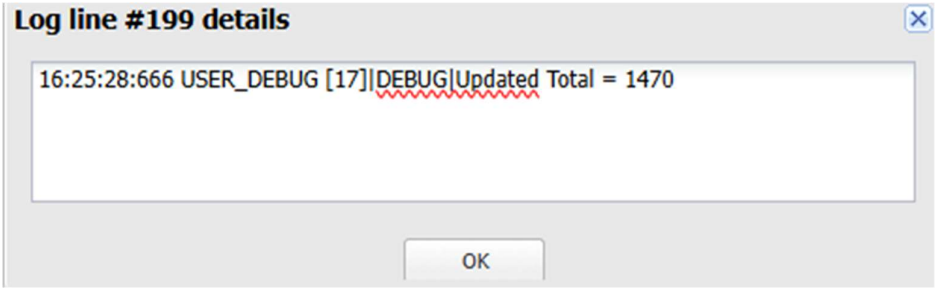
```
Expense__c exp = [SELECT Id, Name, Total_Amount__c, Submission_Date__c
            FROM Expense__c
            WHERE Name = 'E-0000001'
            LIMIT 1];

if (exp.Submission_Date__c != null && exp.Submission_Date__c > Date.today()) {
   exp.Submission_Date__c = Date.today();
   update exp;
}

Set<Id> expenseIds = new Set<Id>{ exp.Id };
UpdateTotalHandler.updateTotals(expenseIds);

Expense__c updatedExp = [SELECT Id, Name, Total_Amount__c,
Submission_Date__c
                FROM Expense__c
                WHERE Id = :exp.Id];
System.debug('Updated Total = ' + updatedExp.Total_Amount__c);
```

Enter Apex Code

```
Expense__c exp = [SELECT Id, Name, Total_Amount__c, Submission_Date__c
                  FROM Expense__c
                  WHERE Name = 'E-0000001'
                  LIMIT 1];

if (exp.Submission_Date__c != null && exp.Submission_Date__c > Date.today()) {
    exp.Submission_Date__c = Date.today();
    update exp;
}

Set<Id> expenseIds = new Set<Id>{ exp.Id };
UpdateTotalHandler.updateTotals(expenseIds);

Expense__c updatedExp = [SELECT Id, Name, Total_Amount__c, Submission_Date__c
                         FROM Expense__c
                         WHERE Id = :exp.Id];
System.debug('Updated Total = ' + updatedExp.Total_Amount__c);
```

3. *Test Output:*

16:25:28:666 USER_DEBUG [17]|DEBUG|Updated Total = 1470



APEX TRIGGER:-

Making a Tigger that *automatically update the Total Amount of the Expense Object (Parent Object) when there is a change in the records of the Expense_Line (Child Object)*.

Apex Trigger code:

```
trigger UpdateTotalAmount on Expense_Line__c (after insert, after update, after delete) {
    Set<Id> expenseIds = new Set<Id>();
   if(Trigger.isInsert || Trigger.isUpdate){
      for(Expense_Line__c el : Trigger.new){
         expenseIds.add(el.Expense__c);
      }
   }
   if(Trigger.isDelete){
      for(Expense_Line__c el : Trigger.old){
         expenseIds.add(el.Expense__c);
      }
   }
   UpdateTotalHandler.updateTotals(expenseIds);

}
```

```
trigger UpdateTotalAmount on Expense_Line__c (after insert, after update, after delete) {
     Set<Id> expenseIds = new Set<Id>();
    if(Trigger.isInsert || Trigger.isUpdate){
        for(Expense_Line__c el : Trigger.new){
            expenseIds.add(el.Expense__c);
        }
    }
    if(Trigger.isDelete){
        for(Expense_Line__c el : Trigger.old){
            expenseIds.add(el.Expense__c);
        }
    }
    UpdateTotalHandler.updateTotals(expenseIds);

}
```

SOQL & SOSL:-

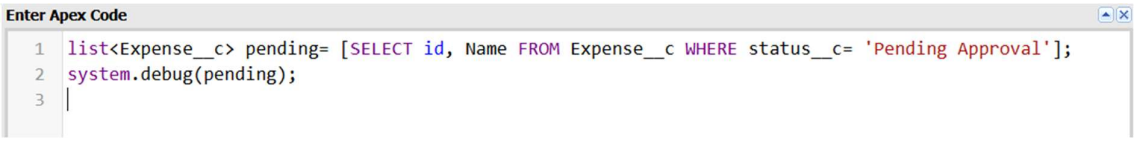
SOQL (Salesforce object Query Language):

Using SOQL to find the record the Expense Object which have their status as 'Pending Approval':-

Query:

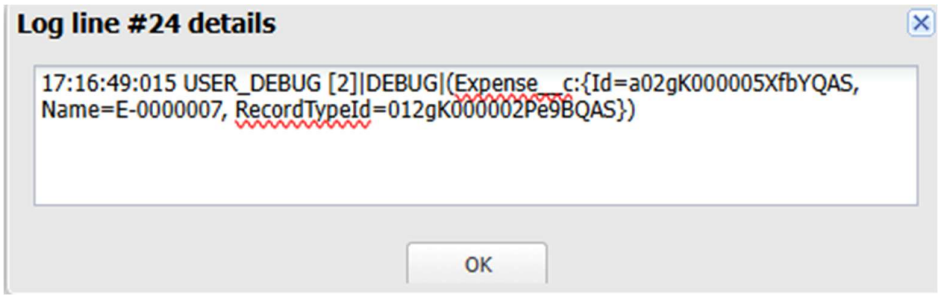
list<Expense__c> pending= [SELECT id, Name FROM Expense__c WHERE status__c= 'Pending Approval'];

system.debug(pending);



Output:

17:16:49:015 USER_DEBUG [2]|DEBUG|(Expense__c:{Id=a02gK000005XfbYQAS, Name=E-0000007, RecordTypeId=012gK000002Pe9BQAS})



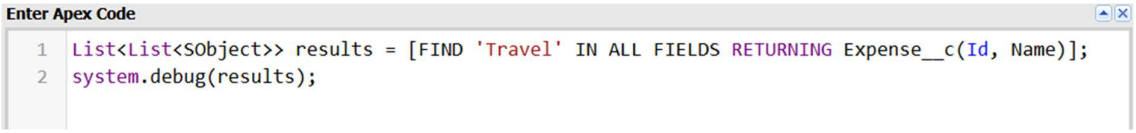
SOSL (Salesforce Object Search Language)

Using the SOSL to find travel Expense_Line Object across the whole system.

Query:

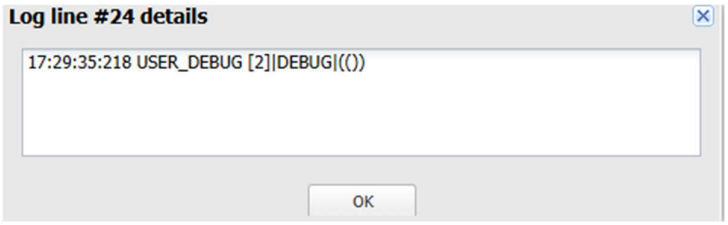
List<List<SObject>> results = [FIND 'Travel' IN ALL FIELDS RETURNING Expense__c(Id, Name)];

system.debug(results);



Output:

17:29:35:218 USER_DEBUG [2]|DEBUG|(()) {Since there is none with travel so its empty}



<u>COLLECTIONS: LISTS, SET, MAP:-</u>

1. **List:** Ordered collection that are used in previous code like
    - List<Expense_Line__c> lines = [SELECT Amount__c FROM Expense_Line__c WHERE Expense__c = :expenseId];
      Used for collecting Expense_Line object which have same Expense Object as Parent.
    - list<Expense__c> pending= [SELECT id, Name FROM Expense__c WHERE status__c= 'Pending Approval'];
      Used for collecting Expense Object with pending approval.

2. **Set:** Collection that allow no duplicate values that are used in previous code like
    - Set<Id> expenseIds = new Set<Id>();
      Using set id here so there would be no duplicate.
3. **Map:** These are key-Value pair that are used to hold value like Expense-> total.

CONTROL STATEMENT:-

1. **If:** These are used multiple times before-
   - if(Trigger.isInsert || Trigger.isUpdate)
     Used in the Apex Trigger to see if the trigger is right to fire
   - if (exp.Submission_Date__c != null && exp.Submission_Date__c > Date.today()) {
     exp.Submission_Date__c = Date.today();
     update exp; }
     Used to avoid the Submission Date Validation Rule
2. **For:** These are used multiple times before-
   - for(Id expId : expenseIds){
     Decimal total = ExpenseAmount.calculateTotal(expId);
     expensesToUpdate.add(new Expense__c(Id = expId,
     Total_Amount__c = total)); }
     Used for iterating through the expid collection object.

BATCH APEX:-

Making a *Batch apex Asynchronous process* for *recalculating the Total Amount of all the records in the Expense object* which is a large amount of data.

Apex Code:

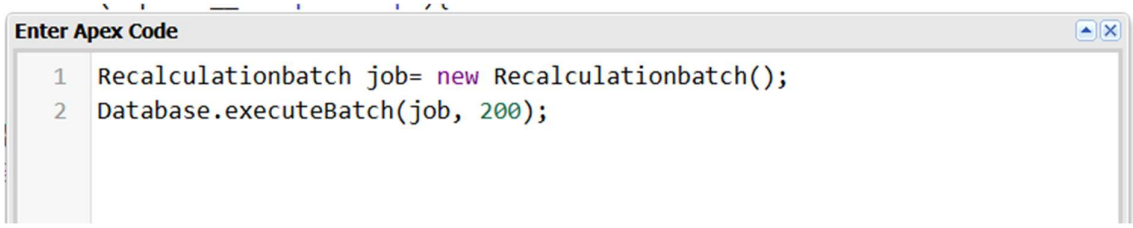
1. *Batch Apex class:*

```
global class Recalculationbatch implements Database.Batchable<Sobject> {

  global Database.QueryLocator start(Database.BatchableContext bc){
    return Database.getQueryLocator('SELECT Id FROM Expense__c');
  }
  global void execute(Database.BatchableContext bc, List<Expense__c> scope){
    for(Expense__c exp : scope){
      exp.Total_Amount__c = ExpenseAmount.calculateTotal(exp.Id);
    }
    update scope;
  }
  global void finish(Database.BatchableContext bc){
    System.debug('Recalculation complete');
  }
}
```

```
global class Recalculationbatch implements Database.Batchable<Sobject> {
    global Database.QueryLocator start(Database.BatchableContext bc){
        return Database.getQueryLocator('SELECT Id FROM Expense__c');
    }
    global void execute(Database.BatchableContext bc, List<Expense__c> scope){
        for(Expense__c exp : scope){
            exp.Total_Amount__c = ExpenseAmount.calculateTotal(exp.Id);
        }
        update scope;
    }
    global void finish(Database.BatchableContext bc){
        System.debug('Recalculation complete');
    }

}
```

2. *Anonymous Window:*

   Recalculationbatch job= new Recalculationbatch();
   Database.executeBatch(job, 200);



3. *Test Output:*

   Success

User	Application	Operation	Time	Status	Read	Size
Anisha Lamba	Unknown	Batch Apex	9/20/2025, 11:34:48 PM	Success	Unread	3.45 KB

<u>QUEUEABLE APEX:-</u>

Making a *queueable apex that will find the expense that have pending approval and will notify it to the manager.*

Apex Code:

1. *Apex Class:*

```
public class PendingNotifier implements Queueable {
public void execute(QueueableContext context) {
    List<Expense__c> pendingHighValue = [
      SELECT Id, Name, Total_Amount__c, Employee__r.ManagerId
      FROM Expense__c
      WHERE Status__c = 'Pending Approval'
      AND Total_Amount__c > 10000
```

```
    ];

    List<Task> reminders = new List<Task>();

    for (Expense__c exp : pendingHighValue) {
      if (exp.Employee__r.ManagerId != null) {
        reminders.add(new Task(
          OwnerId = exp.Employee__r.ManagerId,
          Subject = 'Approval Reminder for Expense ' + exp.Name,
          ActivityDate = System.today().addDays(3),
          WhatId = exp.Id
        ));
      }
    }

    if (!reminders.isEmpty()) {
      insert reminders;
    }

    System.debug('High value pending expenses reminders created: ' +
reminders.size());
  }
}
```

```
public class PendingNotifier implements Queueable {
public void execute(QueueableContext context) {
        List<Expense__c> pendingHighValue = [
            SELECT Id, Name, Total_Amount__c, Employee__r.ManagerId
            FROM Expense__c
            WHERE Status__c = 'Pending Approval'
            AND Total_Amount__c > 10000
        ];

        List<Task> reminders = new List<Task>();

        for (Expense__c exp : pendingHighValue) {
            if (exp.Employee__r.ManagerId != null) {
                reminders.add(new Task(
                    OwnerId = exp.Employee__r.ManagerId,
                    Subject = 'Approval Reminder for Expense ' + exp.Name,
                    ActivityDate = System.today().addDays(3),
                    WhatId = exp.Id
                ));
            }
        }

        if (!reminders.isEmpty()) {
            insert reminders;
        }

        System.debug('High value pending expenses reminders created: ' + reminders.size());
    }
}
```

2. ***Anonymous Window Code:***

   System.enqueueJob(new PendingNotifier());

Enter Apex Code

```
1  System.enqueueJob(new PendingNotifier());
```

3. ***Test Output:***

   Success

User	Application	Operation	Time	Status	Read	Size
Anisha Lamba	Unknown	QueueableHandler	9/21/2025, 12:14:...	Success		3.79 KB

<u>SCHEDULED APEX:-</u>

Making a Scheduled Apex for *sending a weekly reminder to the manager for the pending approval of the Expense Object.*

Apex Code

1. *Apex class:*

```
global class WeeklyReminder implements Schedulable {
   global void execute(SchedulableContext sc) {
      System.enqueueJob(new PendingNotifier());
   }
}
```

```
global class WeeklyReminder implements Schedulable {
    global void execute(SchedulableContext sc) {
        System.enqueueJob(new PendingNotifier());
    }
}
```

2. *Anonymous Window Code:*

```
String jobName = 'Weekly High Value Reminder';
String cronExp  = '0 0 9 ? * MON *'; // Every Monday at 9:00 AM

System.schedule(jobName, cronExp, new WeeklyReminder());
```

Enter Apex Code

```
String jobName = 'Weekly High Value Reminder';
String cronExp  = '0 0 9 ? * MON *'; // Every Monday at 9:00 AM

System.schedule(jobName, cronExp, new WeeklyReminder());
```

3. *Test Output:*

   Success

User	Application	Operation	Time	Status	Read	Size
Anisha Lamba	Unknown	/services/data/v64....	9/21/2025, 12:36:...	Success		3.31 KB

FUTURE METHODS:-

These would be used while we will be exporting the data so we will do it in upcoming phase.

EXCEPTION HANDLING:-

These are mainly used in order to prevent crashes-

Eg- try {

update expensesToUpdate;

} catch(Exception e){

System.debug('Error updating expenses: ' + e.getMessage());

}

This will prevent an crash if the Update fail .

TEST CLASSES:-

Writing the Test class to test if all the *triggers and classes are working or not.*

Apex Code:

```
@isTest
private class ExpenseTest
@isTest static void testTotalCalc(){
Expense__c exp = new Expense__c(Expense_Detail__c='Test Exp',Total_Amount__c=
10,Status__c='Draft');
insert exp;
Expense_Line__c line = new Expense_Line__c(Expense__c=exp.Id, Amount__c=100);
```

insert line;
Test.startTest();
line.Amount__c = 200;
update line;
Test.stopTest();
Expense__c updatedExp = [SELECT Total_Amount__c FROM Expense__c WHERE Id=:exp.Id];
System.assertEquals(200, updatedExp.Total_Amount__c);

}

}

```
@isTest
private class ExpenseTest {
    @isTest static void testTotalCalc(){
        Expense__c exp = new Expense__c(Expense_Detail__c='Test Exp',Total_Amount__c= 10,Status__c='Draft');
        insert exp;
        Expense_Line__c line = new Expense_Line__c(Expense__c=exp.Id, Amount__c=100);
        insert line;
        Test.startTest();
        line.Amount__c = 200;
        update line;
        Test.stopTest();
        Expense__c updatedExp = [SELECT Total_Amount__c FROM Expense__c WHERE Id=:exp.Id];
        System.assertEquals(200, updatedExp.Total_Amount__c);
    }
}
```

Test Output:

Success

User	Application	Operation	Time	Status	Read	Size
Anisha Lamba	Unknown	ApexTestHandler	9/21/2025, 1:04...	Success	Unread	1.94 KB

<u>ASYNCHRONUS PROCESSING:-</u>

- **Batch**: Handle large Expense recalculations.
- **Queueable**: Send manager notifications.
- **Future**: Integrate with external system.
- **Scheduled**: Weekly reminders.