**Documentation for Custom Potential Duplicate Component**

In the custom potential duplicate component, we can select multiple records at a time and merge the fields for master records, and can delete the duplicate records.

So, here is my Step-by-Step implementation which I have followed to achieve this output.

**Step-1:** At first I gave matching rules for the Sobject and assigned it to duplicate rules.

**Step-2:** Then I used the apex class to get the duplicate records from the salesforce dynamically.

**Step-3:** I am using the wrapper class to get the fields for every Soject in salesforce.

Here Is my Apex class code:

**Wrapper Class:**

public class DuplicateRecordCountWrapper {

@AuraEnabled

public List<String> fieldsNames { get; set; }

@AuraEnabled

public List<sObject> duplicateRecords { get; set; }

}

**Apex Class:**

public class PotentialDuplicateComponentController {

@AuraEnabled(cacheable=true)

public static DuplicateRecordCountWrapper searchPotentialDuplicates(String objectApiName, Id recordId) {

DuplicateRecordCountWrapper duplicateReccc= new DuplicateRecordCountWrapper();

String SobjectApiName = objectApiName;

Map<String, Schema.SObjectType> schemaMap = Schema.getGlobalDescribe();

Map<String, Schema.SObjectField> fieldMap = schemaMap.get(SobjectApiName).getDescribe().fields.getMap();

List<String>apiNames = new list<String>();

Set <String>duplicatefiled = new Set<String>();

for(String apiName : fieldMap.keyset())

{

apiNames.add(apiName);

}

list<Id> idss = new List<id>();

string allFieldString = string.join(apiNames,',');

system.debug('alll field values' +allFieldString);

String query = 'SELECT ' + allFieldString + ' FROM ' + SobjectApiName + ' WHERE Id =\'' + recordId +'\'';

List<sObject> sObjectListForDupCheck = Database.query(query);

System.debug('98' +sObjectListForDupCheck);

Datacloud.FindDuplicatesResult[] findDupeResults = Datacloud.FindDuplicates.findDuplicates(sObjectListForDupCheck);

system.debug('7777 '+findDupeResults);

List<sObject> potentialDuplicates = new List<sObject>();

List<sObject> potentialDuplicat = new List<sObject>();

List<sObject> potentialDuplicatt = new List<sObject>();

for (Datacloud.FindDuplicatesResult findDupeResult : findDupeResults) {

system.debug('1 '+findDupeResults);

for (Datacloud.DuplicateResult dupeResult : findDupeResult.getDuplicateResults()) {

system.debug('2 '+findDupeResults);

for (Datacloud.MatchResult matchResult : dupeResult.getMatchResults()) {

system.debug('3 '+findDupeResults);

for (Datacloud.MatchRecord matchRecord : matchResult.getMatchRecords()) {

for(Datacloud.FieldDiff ass :matchRecord.getFieldDiffs()){

system.debug('6 '+ass.getName());

duplicatefiled.add(ass.getName());

potentialDuplicat.add((sObject)matchRecord.getRecord());

}

}

}

}

}

List<String> duplicatefiledd = new list<String> ();

duplicatefiledd.addall(duplicatefiled);

duplicateReccc.fieldsNames=duplicatefiledd;

System.debug('duplicateReccc.fieldsNames' +duplicateReccc.fieldsNames.size());

//if(potentialDuplicatt.isEmpty()){

idss.add(recordId);

for(sObject aa :potentialDuplicat){

idss.add(aa.id); //containing all duplicate and original data ID

}

system.debug('8 '+duplicatefiled);

string allFields = string.join(duplicateReccc.fieldsNames,',');

system.debug('9 '+allFields);

List<String> fields = new List<String>(Schema.getGlobalDescribe().get(SobjectApiName).getDescribe().fields.getMap().keySet());

string querys = 'SELECT ' + String.join(fields, ',') + ' FROM ' + SobjectApiName + ' WHERE Id IN :idss';

System.debug('fields1' +fields.size());

System.debug('fields2' +fields);

List<sObject> sObjectListForDupChecks = Database.query(querys);

duplicateReccc.duplicateRecords= sObjectListForDupChecks;

System.debug('25'+duplicateReccc);

system.debug('377'+duplicateReccc.duplicateRecords);

return duplicateReccc;

}

@AuraEnabled(cacheable=true)

public static List<String> getSObjectFields(String objectApiName) {

Map<String, Schema.SObjectType> sobjects = Schema.getGlobalDescribe();

Schema.SObjectType objectType = sobjects.get(objectApiName);

List<String> fieldNames = new List<String>();

if (objectType != null) {

Schema.DescribeSObjectResult describeResult = objectType.getDescribe();//gets a Schema.DescribeSObjectResult object for the objectType SObject, which provides metadata information about the SObject.

for (Schema.SObjectField field : describeResult.fields.getMap().values()) {//loops through all the fields on the objectType SObject.

String fieldName = field.getDescribe().getName();//gets the API name of the current field and assigns it to a new String

if (field.getDescribe().getType() == Schema.DisplayType.COMBOBOX ||

field.getDescribe().getType() == Schema.DisplayType.MULTIPICKLIST) {// checks if the current field is either a picklist or a multi-select picklist

// For picklist and multi-select picklist fields, retrieve the picklist values

Schema.DescribeFieldResult fieldResult = field.getDescribe();

List<Schema.PicklistEntry> picklistValues = fieldResult.getPicklistValues();

if (picklistValues != null && !picklistValues.isEmpty()) {

for (Schema.PicklistEntry picklistValue : picklistValues) {

fieldNames.add(fieldName + '.' + picklistValue.getValue());

}

}

} else {

// For all other fields, add the field name to the list

fieldNames.add(fieldName);

}

}

}

return fieldNames;

}

@AuraEnabled

public static void updateMasterRecord(String masterrecId, List<String> fiel,List<String> valu,List<String> recordId) {

System.debug('masterrecId: ' + masterrecId);

System.debug('myObjs: ' + fiel);

System.debug('valu: ' + valu);

Id masterrecordId = masterrecId;

Schema.SObjectType sobjectType = masterrecordId.getSObjectType();

String objectName = sobjectType.getDescribe().getName();

SObject ac = Schema.getGlobalDescribe().get(objectName).newSObject();

Map<String,String> mmap = new Map<String,String>();

for(Integer i = 0;i<=fiel.size();i++){

for(Integer j = 0 ; j<valu.size();j++){

if(i==j){

ac.put(fiel[i], valu[j]);

}

}

}

ac.put('Id', masterrecordId);

System.debug('myObjssst: ' + ac);

List<sObject> dss = new List<sObject>();

dss.add(ac);

System.debug('myObjsww: ' + dss);

Database.update(dss);

//delete duplicates

List<Id> duplicateIds = new List<Id>();

for(String idStr : recordId){

if(idStr != masterrecId){

duplicateIds.add(Id.valueOf(idStr));

System.debug('recordId' +recordId);

}

}

deleteDuplicates(duplicateIds);

}

@AuraEnabled

public static void deleteDuplicates(List<Id> duplicateIds) {

if(duplicateIds.isEmpty()) return;

Schema.SObjectType sobjectType = duplicateIds[0].getSObjectType();

String objectName = sobjectType.getDescribe().getName();

List<sObject> deleteRecord = new List<sObject>();

for(Id i : duplicateIds){

SObject ac = Schema.getGlobalDescribe().get(objectName).newSObject();

ac.put('Id', i);

deleteRecord.add(ac);

}

Database.delete(deleteRecord);

}

}

**Step-4:** Now Create an LWC Component and Import all the methods and set the variables accordingly.

**Here is LWC Java Script code:**

import { LightningElement, api, wire, track } from 'lwc';

import searchPotentialDuplicates from '@salesforce/apex/PotentialDuplicateComponentController.searchPotentialDuplicates';

import updateMasterRecord from '@salesforce/apex/PotentialDuplicateComponentController.updateMasterRecord';

import sobjectfields from '@salesforce/apex/PotentialDuplicateComponentController.getSObjectFields';

import { getObjectInfo } from '@salesforce/apex';

import { ShowToastEvent } from 'lightning/platformShowToastEvent';

const columnss = [

{ label: 'API Name', fieldName: 'fieldName' }

];

export default class PotentialDuplicates extends LightningElement {

@api recordId;

@api objectApiName;

@track potentialDuplicates;

@track count;

@track isModalOpen = false;

@track isSecondModalOpen=false;

@track selectedRowIds = [];

@track heading;

@track sobjectName;

@track DuplicateRecordCountWrapper;

@track fieldNames = [];

@track duplicateFieldsnames;

@track record;

@track fieldss = [];

@track fieldInfo = [];

@track columnss;

@track dattaa = [];

@track dataValue = [];

@track abc = [];

@track listofrecords = [];

@track selectedRowIdss;

@api listOfRecords;

selectedField = '';

@track masterRecordId;

@track selectedmaster;

@track listofrecords = [];

@track selectedvalues= [];

@track selectedmaster;

@track myObjs= [];

@track isThirdModel = false;

@track getMapdata =[];

@wire(searchPotentialDuplicates, { objectApiName: '$objectApiName', recordId: '$recordId' })

wiredResult({ error, data }) {

if (data) {

console.log('datat' + data);

this.DuplicateRecordCountWrapper = data;

console.log('lkjh' + this.DuplicateRecordCountWrapper.duplicateRecords);

console.log({ data });

this.potentialDuplicates = this.DuplicateRecordCountWrapper.duplicateRecords;

this.duplicateFieldsnames = this.DuplicateRecordCountWrapper.fieldsNames;

this.count = this.potentialDuplicates.length - 1;

this.error = undefined;

// set selected row as default

const currentRecord = this.potentialDuplicates.find(record => record.Id === this.recordId);

if (currentRecord) {

this.selectedRowIds = [currentRecord.Id];

}

} else if (error) {

this.error = error;

this.potentialDuplicates = undefined;

}

}

get columns() {

if (this.potentialDuplicates && this.potentialDuplicates.length > 0) {

const fieldNames = Object.keys(this.potentialDuplicates[0])

.filter(fieldName => fieldName != 'Id' && this.duplicateFieldsnames.includes(fieldName));

console.log('FIELDS' + fieldNames);

return [

...fieldNames.map(fieldName => {

return {

// if(duplicateFieldsnames.includes(fieldName)){

label: fieldName,

fieldName: fieldName

//}

};

})

];

}

return [];

}

openModal() {

this.isModalOpen = true;

console.log('ffff' + this.potentialDuplicates);

console.log('fhjkl' + columns);

}

closeModal() {

this.isModalOpen = false;

}

closeSecModal() {

this.isSecondModalOpen = false;

this.isThirdModel = false;

}

@track isNotSelectedRow = true;

handleRowSelection(event) {

const selectedRows = event.detail.selectedRows;

this.selectedRowIds = selectedRows.map(row => row.Id);

console.log('67'+this.selectedRowIds);

const recordIds = selectedRows.map(row => row.Id);

this.selectedRowIdss= recordIds;

if(this.selectedRowIdss.length != 1 || this.selectedRowIdss.length != 0){

this.isNotSelectedRow = false;

}

else{

this.isNotSelectedRow = true;

}

console.log('68'+this.selectedRowIdss);

console.log("Selected Record Ids: ", recordIds);

}

handleNext() {

this.selectedDuplicates = this.potentialDuplicates.filter(record => this.selectedRowIds.includes(record.Id));

console.log('106 ' + this.selectedRowIds);

console.log('107 ' + this.selectedDuplicates);

this.isModalOpen = false; // hide the first modal

this.isSecondModalOpen = true; // show the second modal

console.log(this.objectApiName + ' == This -- ' + this.recordId);

this.displayData();

this.selectedRowIdss = this.selectedRowIds;

console.log('isSecondModalOpen' +this.isSecondModalOpen);

console.log(' isModalOpen' + this.isModalOpen);

}

handlePrevious() {

this.isSecondModalOpen = false;

this.isModalOpen = true;

this.listofrecords=[];

}

connectedCallback() {

this.sobjectName = this.objectApiName.charAt(0).toUpperCase() + this.objectApiName.slice(1);

this.heading = `Potential Duplicate ${this.sobjectName} Records`;

}

@wire(sobjectfields, { objectApiName: '$objectApiName' })

wiredFields({ error, data }) {

if (data) {

this.fieldss = data;

console.log('Fields:', data);

console.log('this.fieldss' +JSON.stringify(this.fieldss));

} else if (error) {

console.error(error);

}

}

displayData() {

var keyItem = this.fieldss;

// for (let fieldName in this.selectedDuplicates[1]) {

// keyItem.push(fieldName);

// console.log('keyItem' +keyItem);

// console.log('this.selectedDuplicates[0]' +JSON.stringify(this.selectedDuplicates[1]));

// }

for(var item in keyItem){

var fieldvalues = [];

var value = keyItem[item];

console.log('value' +value);

for(var entry in this.selectedDuplicates){

console.log('entry' +entry);

var fieldvalue1 = this.selectedDuplicates[entry][value];

if(!fieldvalue1){

fieldvalue1 = 'NA';

}

fieldvalues.push(fieldvalue1);

// When fields value and name are null or empty string then the code is failing so we are checking null and empty and assigning value it to NA

}

console.log('fieldvalues' +fieldvalues);

this.listofrecords.push({value, fieldvalues});

}

console.log('selectedDuplicates' +this.selectedDuplicates);

console.log('listofcount' + this.listofrecords.count);

console.log('listofrecords' + this.listofrecords);

console.log('listofrecords' + JSON.stringify(this.listofrecords));

//this.isSecondModalOpen = true;

}

masterrecId(event) {

this.selectedmaster = event.target.value;

this.masterRecordId = this.selectedmaster;

console.log('selectedmaster: ' + this.selectedmaster);

console.log('selected master: ' + JSON.stringify(this.masterrecord));

console.log('selectedMasterRecord: ' + JSON.stringify(this.selectedMasterRecord));

}

@track selectedvalues;

@track sfield=[];

@track svalue=[];

handleRadioChange(event){

console.log('this is values' );

this.selectedvalues = event.target.value;

console.log('fieldff' +event.target.name);

console.log('this.selectedvalues' +event.target.value);

this.sfield.push(event.target.name );

this.svalue.push(event.target.value);

var myObj = {

fieldName : event.target.name,

fieldValue : event.target.value

};

console.log('uij '+this.ss);

console.log('myobj' +myObj.fieldName + myObj.fieldValue);

this.myObjs.push(myObj);

//this.getMapdata.push(event.target.name,event.target.value);

console.log('myobjvalues' +JSON.stringify(this.myObjs));

}

handlenaxt1(){

console.log('myobjvalue' +JSON.stringify(this.myObjs));

console.log('ioio '+JSON.stringify(this.myObjs).fieldName);

this.isThirdModel = true;

this.isSecondModalOpen = false;

this.getMapdata.push(this.selectedmaster);

console.log('73hyfo '+ JSON.stringify(this.getMapdata));

console.log('yyy '+ this.objectApiName);

//this.ss.push(myObjs.fieldName, myObjs.fieldValue);

console.log('ioia '+this.ss);

}

handlemerge() {

console.log('merge is called ');

updateMasterRecord({masterrecId :this.selectedmaster, fiel:this.sfield,valu:this.svalue,recordId:this.selectedRowIds})

.then(result => {

console.log(result);

const evt = new ShowToastEvent({ title:"Success", message: "Successful", variant: 'success' }); this.dispatchEvent(evt);

// Handle success

this.isThirdModel = false;

})

.catch(error => {

console.log(error);

const evt = new ShowToastEvent({ title: "Error", message: error, variant: 'error'}); this.dispatchEvent(evt);

// Handle error

});

}

}

**Here is LWC HTML Code:**

<template>

<lightning-card>

<div class="container">

<div class="merge-icon">

<lightning-icon icon-name="standard:merge" alternative-text="Merge"></lightning-icon>

</div>

<div class="content">

<h2 id="modal-heading-50" class="slds-text-heading\_medium slds-hyphenate">We found {count} potential duplicates of this {objectApiName}</h2>

</div>

</div>

<lightning-button variant="base" label="View Duplicates" href="#" target="\_blank" onclick={openModal}></lightning-button>

<template if:true={isModalOpen}>

<div class="tt">

<section role="dialog" aria-labelledby="modal-heading-01" aria-modal="true"

aria-describedby="modal-content-id-1" class="slds-modal slds-fade-in-open">

<div class="slds-modal\_\_container">

<header class="slds-modal\_\_header">

<button class="slds-button slds-button\_icon slds-modal\_\_close slds-button\_icon-inverse"

title="Close" onclick={closeModal}>

<lightning-icon icon-name="utility:close" alternative-text="close" variant="inverse"

size="small"></lightning-icon>

<span class="slds-assistive-text">Close</span>

</button>

<h2 id="modal-heading-01" class="slds-text-heading\_medium slds-hyphenate">Potential

Duplicate Records</h2>

</header>

<div class="slds-modal\_\_content slds-p-around\_medium" id="modal-content-id-1">

<h2 id="demo" class="demo">View Duplicates</h2>

<h2 id="modal-heading-0" class="slds-text-heading\_medium slds-hyphenate">

{objectApiName}({count})</h2><br />

<lightning-datatable columns={columns} data={potentialDuplicates} key-field="Id"

onrowselection={handleRowSelection}

selected-rows={selectedRowIds}></lightning-datatable>

</div>

<footer class="slds-modal\_\_footer">

<lightning-progress-indicator current-step=1 type="base" variant="base">

<lightning-progress-step label="VIEW DUPLICATES" value="1"></lightning-progress-step>

<lightning-progress-step label="COMPARE RECORDS" value="2"></lightning-progress-step>

<lightning-progress-step label="CONFIRM MERGING" value="3"></lightning-progress-step>

<lightning-progress-step if:true={showIndicator} class="slds-hide" label="Step 2" value="3">

</lightning-progress-step>

</lightning-progress-indicator>

<button class="slds-button slds-button\_neutral" onclick={handleNext}

disabled={isNotSelectedRow}>Next</button>

</footer>

</div>

</section>

<div class="slds-backdrop slds-backdrop\_open"></div>

</div>

</template>

<!-- Second Modal -->

<template if:true={isSecondModalOpen}>

<section role="dialog" aria-labelledby="modal-heading-02" aria-modal="true"

aria-describedby="modal-content-id-2" class="slds-modal slds-fade-in-open">

<div class="slds-modal\_\_container">

<header class="slds-modal\_\_header">

<button class="slds-button slds-button\_icon slds-modal\_\_close slds-button\_icon-inverse"

title="Close" onclick={closeSecModal}>

<lightning-icon icon-name="utility:close" alternative-text="close" variant="inverse"

size="small"></lightning-icon>

<span class="slds-assistive-text">Close</span>

</button>

<h2 id="modal-heading-02" class="slds-text-heading\_medium slds-hyphenate">Potential

Duplicate Records</h2>

</header>

<div class="slds-modal\_\_content slds-p-around\_medium" id="modal-content-id-2">

<div class="xys abc tt-width: 100%; overflow-x: auto;">

Compare {objectApiName}

</div>

<div class="xys tt-width: 100%; overflow-x: auto;">

When you merge, the master record is updated with the values you choose, and relationships to other items are shifted to the master record.

</div></br>

<div style="width: 100%; overflow-x: auto;">

<table class="slds-table slds-table\_cell-buffer slds-table\_bordered">

<thead>

<tr>

<th class="slds-cell-wrap slds-p-left\_medium" scope="col">Principal Records</th>

<template for:each={selectedRowIdss} for:item="item12" for:index="index">

<th key={item12} scope="col">

<!-- <label class="slds-radio"> -->

<!-- class="slds-cell-wrap" -->

<input type="radio" name="fun" id={item12} value={item12} data-value=" USE AS PRINCIPAL" onclick={masterrecId}/>

<!-- <span class="slds-radio--faux"></span> -->

USE AS PRINCIPAL

<!-- </label> -->

</th>

</template>

</tr>

</thead>

<tbody>

<template for:each={listofrecords} for:item="item2" if:true={item2}>

<tr key={item2.fieldName}>

<td class="slds-cell-wrap slds-p-left\_medium">{item2.value}</td>

<template for:each={item2.fieldvalues} for:item="entryitem" for:index="index2">

<td key={entryitem.Id} class="slds-cell-wrap">

<div class="slds-form-element">

<div class="slds-form-element\_\_control">

<label class="slds-radio">

<input type="radio" name={item2.value} id={item2.entryitem} value={entryitem} data-value={entryitem} onchange={handleRadioChange}/>

<span class="slds-radio--faux"></span>

{entryitem}

</label>

</div>

</div>

</td>

</template>

</tr>

</template>

</tbody>

</table>

</div>

</div>

<footer>

<div class="slds-modal\_\_footer slds-modal\_\_footer\_directional">

<button class="slds-button slds-button\_neutral" onclick={handlePrevious}>Previous</button>

<lightning-progress-indicator current-step=2 type="base" variant="base">

<lightning-progress-step label="VIEW DUPLICATES" value="1"></lightning-progress-step>

<lightning-progress-step label="COMPARE RECORDS" value="2"></lightning-progress-step>

<lightning-progress-step label="CONFIRM MERGING" value="3"></lightning-progress-step>

<lightning-progress-step if:true={showIndicator} class="slds-hide" label="Step 2" value="3">

</lightning-progress-step>

</lightning-progress-indicator>

<button class="slds-button slds-button\_brand" onclick={handlenaxt1}>Next</button>

</div>

</footer>

</div>

</section>

<div class="slds-backdrop slds-backdrop\_open"></div>

</template>

<!-- Third model -->

<template if:true={isThirdModel}>

<section role="dialog" aria-labelledby="modal-heading-02" aria-modal="true"

aria-describedby="modal-content-id-2" class="slds-modal slds-fade-in-open">

<div class="slds-modal\_\_container">

<header class="slds-modal\_\_header">

<button class="slds-button slds-button\_icon slds-modal\_\_close slds-button\_icon-inverse"

title="Close" onclick={closeSecModal}>

<lightning-icon icon-name="utility:close" alternative-text="close" variant="inverse"

size="small"></lightning-icon>

<span class="slds-assistive-text">Close</span>

</button>

<h2 id="modal-heading-03" class="gg-slds-text-heading\_medium slds-hyphenate"><b>Potential

Duplicate Records</b></h2>

</header>

<div class="slds-modal\_\_content slds-p-around\_medium" id="modal-content-id-3">

<div class="xys tt-width: 100%; overflow-x: auto;">

Conform to Merge

</div>

<div class="xys tt-width: 100%; overflow-x: auto;">

We’re ready to merge these records

</div>

<div class="xys tt-width: 100%; overflow-x: auto;">

You're about to merge these {objectApiName}. You can't undo merging.

</div>

</div>

<footer class="slds-modal\_\_footer">

<lightning-progress-indicator current-step=3 type="base" variant="base">

<lightning-progress-step label="VIEW DUPLICATES" value="1"></lightning-progress-step>

<lightning-progress-step label="COMPARE RECORDS" value="2"></lightning-progress-step>

<lightning-progress-step label="CONFIRM MERGING" value="3"></lightning-progress-step>

<lightning-progress-step if:true={showIndicator} class="slds-hide" label="Step 2" value="3">

</lightning-progress-step>

</lightning-progress-indicator>

<button class="slds-button slds-button\_neutral" onclick={handlemerge}>Merge</button>

</footer>

</div>

</section>

<div class="slds-backdrop slds-backdrop\_open"></div>

</template>

</lightning-card>

</template>

**Here is LWC CSS code:**

.demo {

text-align: center;

font-size: 25px;

}

.td {

text-align: center;

}

.tt{

text-align: center;

}

.gg{

font-weight: bold;

font-size: 24px;

}

.slds-modal\_\_container{

width: 90% !important;

max-width: 90% !important;

max-height: 80vh !important;

height: 80vh !important;

}

.slds-table--header-fixed {

position: relative;

}

.abc{

line-height:30px;

}

.slds-text-heading\_medium.slds-hyphenate {

font-weight: bold;

font-size: small;

}

.demo1 {

text-align: center;

font-size: 25px;

}

.fre-slds-text-heading\_medium {

font-weight: bold;

font-size: 24px;

}

.oty-slds-text-heading\_medium {

font-weight: bold;

font-size: 24px;

}

.slds-modal\_\_content table {

overflow-x: auto;

}

.xys{

text-align: center ;

}

.container {

display: flex;

align-items: center;

}

.merge-icon {

margin-right: 10px;

}

.content {

flex-grow: 1;

}

.abc{

font-size: medium;

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**SCREENSHOTS:**

Screenshot-1: Displaying the count of duplicate records.



Screenshot-2: Displaying the duplicate records in data table along with master records with default checked.



Screenshot-3: Comparing the records and selecting the values of the field for merging.



Screenshot-4: Merge modal where there is no going back to the previous page.

