Architecture Système de Contrôle de Données  
Salesforce - FundInfo

# 1. Vue d'ensemble

Ce document présente l'architecture complète d'un système de contrôle de données robuste, flexible et user-friendly pour Salesforce. Le système permet de définir, exécuter et corriger des contrôles de qualité de données de manière automatisée via des batchs et à la demande.

# 2. Architecture Globale

## 2.1 Composants Principaux

* Le système est composé de 5 couches principales :
* Couche Métadonnées : Définition des contrôles via Custom Metadata
* Couche Exécution : Moteur d'exécution des contrôles (Batch + Queueable)
* Couche Stockage : Enregistrement des résultats et violations
* Couche Interface : LWC pour visualisation et correction
* Couche Notification : Alertes et rapports

# 3. Définition des Contrôles (Custom Metadata)

Les contrôles sont définis via Custom Metadata Type pour permettre une configuration flexible sans déploiement de code.

## 3.1 Data\_Quality\_Rule\_\_mdt

* Champs principaux :
* **Rule\_Code\_\_c** (Text(20)): Code unique (ex: OFST001410)
* **Rule\_Name\_\_c** (Text(255)): Nom du contrôle
* **Object\_API\_Name\_\_c** (Text(80)): Objet cible (Share\_Class\_\_c)
* **Priority\_\_c** (Picklist): P1, P2, P3
* **Rule\_Type\_\_c** (Picklist): Presence, Format, Coherence, Range, Cross\_Object
* **Description\_\_c** (Long Text): Description détaillée
* **Field\_API\_Name\_\_c** (Text(255)): Champ(s) à contrôler (séparés par virgules)
* **Validation\_Formula\_\_c** (Long Text): Formule Salesforce pour le contrôle
* **Error\_Message\_\_c** (Text(255)): Message d'erreur
* **Scope\_Filter\_\_c** (Long Text): SOQL WHERE clause pour le scope
* **Active\_\_c** (Checkbox): Actif/Inactif
* **Batch\_Size\_\_c** (Number): Taille des lots (default: 200)
* **Notification\_Emails\_\_c** (Text(255)): Emails séparés par des virgules
* **Auto\_Fix\_Class\_\_c** (Text(255)): Classe Apex pour correction auto (optionnel)
* **Business\_Owner\_\_c** (Text(100)): Propriétaire métier
* **Manco\_Scope\_\_c** (Multi-Picklist): NIMI, Ossiam, DNCA, etc.

## 3.2 Exemple de Configuration

Rule\_Code\_\_c: OFST001410  
Rule\_Name\_\_c: LEI Of Custodian Bank - Contrôle de présence  
Object\_API\_Name\_\_c: Share\_Class\_\_c  
Priority\_\_c: P1  
Rule\_Type\_\_c: Presence  
Field\_API\_Name\_\_c: Share\_class\_\_r.Product\_\_r.LEI\_Of\_Custodian\_Bank\_\_c  
Validation\_Formula\_\_c: ISBLANK(Share\_class\_\_r.Product\_\_r.LEI\_Of\_Custodian\_Bank\_\_c)  
Error\_Message\_\_c: Le champ LEI Of Custodian Bank est obligatoire  
Scope\_Filter\_\_c: Share\_class\_\_r.Product\_\_r.Manco\_\_c IN ('Ossiam') AND   
 Share\_class\_\_r.Product\_\_r.Mandate\_Open\_Dedicated\_\_c = 'Open-End Fund'  
Active\_\_c: true  
Batch\_Size\_\_c: 200

# 4. Objets Personnalisés de Stockage

## 4.1 Data\_Quality\_Execution\_\_c

Enregistre chaque exécution de contrôle

* **Name** (Auto-Number): DQE-{0000}
* **Execution\_Date\_\_c** (DateTime): Date/heure de lancement
* **Status\_\_c** (Picklist): Running, Completed, Failed, Cancelled
* **Execution\_Type\_\_c** (Picklist): Batch, Manual, Scheduled
* **Total\_Records\_Processed\_\_c** (Number): Nombre total d'enregistrements
* **Total\_Violations\_\_c** (Number): Nombre total de violations
* **Total\_Rules\_Executed\_\_c** (Number): Nombre de règles exécutées
* **Started\_By\_\_c** (Lookup(User)): Utilisateur ayant lancé
* **Completed\_Date\_\_c** (DateTime): Date/heure de fin
* **Duration\_Seconds\_\_c** (Number): Durée en secondes
* **Error\_Message\_\_c** (Long Text): Message d'erreur si échec
* **Scope\_Filter\_\_c** (Long Text): Filtre appliqué
* **Rule\_Codes\_\_c** (Long Text): Codes des règles exécutées

## 4.2 Data\_Quality\_Violation\_\_c

Enregistre chaque violation détectée

* **Name** (Auto-Number): DQV-{000000}
* **Execution\_\_c** (Master-Detail(Data\_Quality\_Execution\_\_c)): Lien vers l'exécution
* **Rule\_Code\_\_c** (Text(20)): Code de la règle
* **Rule\_Name\_\_c** (Text(255)): Nom de la règle
* **Record\_Id\_\_c** (Text(18)): ID de l'enregistrement en erreur
* **Record\_Link\_\_c** (Formula(Text)): Lien cliquable vers l'enregistrement
* **Object\_Type\_\_c** (Text(80)): Type d'objet
* **Priority\_\_c** (Text(2)): P1, P2, P3
* **Field\_Name\_\_c** (Text(255)): Champ(s) en erreur
* **Current\_Value\_\_c** (Long Text): Valeur actuelle
* **Expected\_Value\_\_c** (Long Text): Valeur attendue (si applicable)
* **Error\_Message\_\_c** (Text(255)): Message d'erreur
* **Status\_\_c** (Picklist): Open, In\_Progress, Fixed, Ignored, False\_Positive
* **Assigned\_To\_\_c** (Lookup(User)): Assigné à
* **Resolution\_Notes\_\_c** (Long Text): Notes de résolution
* **Fixed\_Date\_\_c** (DateTime): Date de correction
* **Fixed\_By\_\_c** (Lookup(User)): Corrigé par
* **Detection\_Date\_\_c** (DateTime): Date de détection
* **Age\_Days\_\_c** (Formula(Number)): TODAY() - DATEVALUE(Detection\_Date\_\_c)

# 5. Moteur d'Exécution (Apex)

## 5.1 DataQualityBatch (Batch Apex)

Classe principale pour l'exécution en batch des contrôles.

global class DataQualityBatch implements Database.Batchable<sObject>, Database.Stateful {  
   
 // Variables d'état  
 private List<Data\_Quality\_Rule\_\_mdt> rules;  
 private Data\_Quality\_Execution\_\_c execution;  
 private Integer totalViolations = 0;  
 private Integer totalRecordsProcessed = 0;  
 private String scopeFilter;  
   
 /\*\*  
 \* Constructeur - accepte une liste de codes de règles ou exécute toutes les règles actives  
 \*/  
 global DataQualityBatch(List<String> ruleCodes, String additionalFilter) {  
 this.scopeFilter = additionalFilter;  
   
 // Récupérer les règles  
 String query = 'SELECT Id, Rule\_Code\_\_c, Rule\_Name\_\_c, Object\_API\_Name\_\_c, ' +  
 'Field\_API\_Name\_\_c, Validation\_Formula\_\_c, Error\_Message\_\_c, ' +  
 'Scope\_Filter\_\_c, Priority\_\_c, Rule\_Type\_\_c, Batch\_Size\_\_c, ' +  
 'Notification\_Emails\_\_c FROM Data\_Quality\_Rule\_\_mdt ' +  
 'WHERE Active\_\_c = true';  
   
 if (ruleCodes != null && !ruleCodes.isEmpty()) {  
 query += ' AND Rule\_Code\_\_c IN :ruleCodes';  
 }  
   
 query += ' ORDER BY Priority\_\_c, Rule\_Code\_\_c';  
 this.rules = Database.query(query);  
   
 // Créer l'enregistrement d'exécution  
 this.execution = new Data\_Quality\_Execution\_\_c(  
 Execution\_Date\_\_c = System.now(),  
 Status\_\_c = 'Running',  
 Execution\_Type\_\_c = 'Batch',  
 Started\_By\_\_c = UserInfo.getUserId(),  
 Total\_Rules\_Executed\_\_c = this.rules.size(),  
 Scope\_Filter\_\_c = this.scopeFilter  
 );  
 insert this.execution;  
 }  
   
 /\*\*  
 \* Start - construit la requête SOQL dynamique  
 \*/  
 global Database.QueryLocator start(Database.BatchableContext BC) {  
 // Pour simplifier, on traite les Share\_Class\_\_c  
 // Dans une vraie implémentation, il faudrait gérer différents objets  
 String query = 'SELECT Id, Name, ' +  
 'Share\_class\_\_r.Product\_\_r.LEI\_Of\_Custodian\_Bank\_\_c, ' +  
 'Share\_class\_\_r.Product\_\_r.Domicile\_Of\_Custodian\_\_c, ' +  
 'Share\_class\_\_r.Product\_\_r.Manco\_\_c, ' +  
 'Share\_class\_\_r.Product\_\_r.Inception\_Date\_\_c, ' +  
 'Share\_class\_\_r.Shareclass\_Inception\_Date\_\_c, ' +  
 'Share\_class\_\_r.Closing\_Shareclass\_Date\_\_c, ' +  
 'Share\_class\_\_r.Management\_Fees\_\_c, ' +  
 'Share\_class\_\_r.Management\_Fees\_Ex\_Ante\_\_c, ' +  
 'Share\_class\_\_r.Maximum\_Sales\_Charges\_\_c ' +  
 'FROM Share\_Class\_\_c';  
   
 if (String.isNotBlank(this.scopeFilter)) {  
 query += ' WHERE ' + this.scopeFilter;  
 }  
   
 return Database.getQueryLocator(query);  
 }  
   
 /\*\*  
 \* Execute - évalue chaque règle pour chaque enregistrement  
 \*/  
 global void execute(Database.BatchableContext BC, List<sObject> scope) {  
 List<Data\_Quality\_Violation\_\_c> violations = new List<Data\_Quality\_Violation\_\_c>();  
   
 for (sObject record : scope) {  
 for (Data\_Quality\_Rule\_\_mdt rule : this.rules) {  
 // Vérifier le scope filter de la règle  
 if (String.isNotBlank(rule.Scope\_Filter\_\_c)) {  
 if (!evaluateScopeFilter(record, rule.Scope\_Filter\_\_c)) {  
 continue;  
 }  
 }  
   
 // Évaluer la règle  
 Boolean hasViolation = evaluateRule(record, rule);  
   
 if (hasViolation) {  
 violations.add(createViolation(record, rule));  
 totalViolations++;  
 }  
 }  
 totalRecordsProcessed++;  
 }  
   
 if (!violations.isEmpty()) {  
 insert violations;  
 }  
 }  
   
 /\*\*  
 \* Finish - met à jour l'exécution et envoie les notifications  
 \*/  
 global void finish(Database.BatchableContext BC) {  
 execution.Status\_\_c = 'Completed';  
 execution.Completed\_Date\_\_c = System.now();  
 execution.Total\_Records\_Processed\_\_c = totalRecordsProcessed;  
 execution.Total\_Violations\_\_c = totalViolations;  
 execution.Duration\_Seconds\_\_c =   
 (execution.Completed\_Date\_\_c.getTime() -   
 execution.Execution\_Date\_\_c.getTime()) / 1000;  
 update execution;  
   
 // Envoyer notifications  
 sendNotifications();  
 }  
   
 /\*\*  
 \* Évalue une règle sur un enregistrement  
 \*/  
 private Boolean evaluateRule(sObject record, Data\_Quality\_Rule\_\_mdt rule) {  
 try {  
 // Utiliser la formule de validation  
 // Note: dans une vraie implémentation, il faudrait un parser de formules  
 // ou utiliser Metadata API pour évaluer les formules  
   
 String formula = rule.Validation\_Formula\_\_c;  
 String fieldValue = getFieldValue(record, rule.Field\_API\_Name\_\_c);  
   
 // Exemples de logique d'évaluation  
 if (rule.Rule\_Type\_\_c == 'Presence') {  
 return String.isBlank(fieldValue);  
 } else if (rule.Rule\_Type\_\_c == 'Range') {  
 // Parser la formule pour extraire min/max  
 return !isInRange(fieldValue, formula);  
 }  
   
 return false;  
 } catch (Exception e) {  
 System.debug('Error evaluating rule: ' + e.getMessage());  
 return false;  
 }  
 }  
   
 /\*\*  
 \* Évalue le scope filter  
 \*/  
 private Boolean evaluateScopeFilter(sObject record, String scopeFilter) {  
 // Implémentation simplifiée  
 // Dans la réalité, parser le WHERE clause et évaluer  
 return true;  
 }  
   
 /\*\*  
 \* Crée un enregistrement de violation  
 \*/  
 private Data\_Quality\_Violation\_\_c createViolation(  
 sObject record,   
 Data\_Quality\_Rule\_\_mdt rule  
 ) {  
 String recordId = (String)record.get('Id');  
 String currentValue = getFieldValue(record, rule.Field\_API\_Name\_\_c);  
   
 return new Data\_Quality\_Violation\_\_c(  
 Execution\_\_c = execution.Id,  
 Rule\_Code\_\_c = rule.Rule\_Code\_\_c,  
 Rule\_Name\_\_c = rule.Rule\_Name\_\_c,  
 Record\_Id\_\_c = recordId,  
 Object\_Type\_\_c = rule.Object\_API\_Name\_\_c,  
 Priority\_\_c = rule.Priority\_\_c,  
 Field\_Name\_\_c = rule.Field\_API\_Name\_\_c,  
 Current\_Value\_\_c = currentValue,  
 Error\_Message\_\_c = rule.Error\_Message\_\_c,  
 Status\_\_c = 'Open',  
 Detection\_Date\_\_c = System.now()  
 );  
 }  
   
 /\*\*  
 \* Récupère la valeur d'un champ (avec support des relations)  
 \*/  
 private String getFieldValue(sObject record, String fieldPath) {  
 try {  
 if (!fieldPath.contains('.')) {  
 Object value = record.get(fieldPath);  
 return value != null ? String.valueOf(value) : null;  
 }  
   
 // Gérer les relations  
 List<String> parts = fieldPath.split('\.');  
 sObject currentObj = record;  
   
 for (Integer i = 0; i < parts.size() - 1; i++) {  
 currentObj = currentObj.getSObject(parts[i]);  
 if (currentObj == null) return null;  
 }  
   
 Object value = currentObj.get(parts[parts.size() - 1]);  
 return value != null ? String.valueOf(value) : null;  
 } catch (Exception e) {  
 return null;  
 }  
 }  
   
 /\*\*  
 \* Vérifie si une valeur est dans une plage  
 \*/  
 private Boolean isInRange(String value, String formula) {  
 // Implémentation simplifiée  
 return true;  
 }  
   
 /\*\*  
 \* Envoie les notifications  
 \*/  
 private void sendNotifications() {  
 // Implémenter l'envoi d'emails aux propriétaires  
 if (totalViolations > 0) {  
 // Logique de notification  
 }  
 }  
}

## 5.2 DataQualityExecutor (Contrôleur Apex)

Classe pour lancer les contrôles à la demande.

public with sharing class DataQualityExecutor {  
   
 /\*\*  
 \* Lance un batch de contrôle avec des filtres optionnels  
 \*/  
 @AuraEnabled  
 public static String executeBatch(  
 List<String> ruleCodes,   
 String scopeFilter,  
 Integer batchSize  
 ) {  
 try {  
 if (batchSize == null || batchSize <= 0) {  
 batchSize = 200;  
 }  
   
 DataQualityBatch batch = new DataQualityBatch(ruleCodes, scopeFilter);  
 Id batchId = Database.executeBatch(batch, batchSize);  
   
 return batchId;  
 } catch (Exception e) {  
 throw new AuraHandledException('Erreur lors du lancement: ' + e.getMessage());  
 }  
 }  
   
 /\*\*  
 \* Récupère toutes les règles actives  
 \*/  
 @AuraEnabled(cacheable=true)  
 public static List<Data\_Quality\_Rule\_\_mdt> getActiveRules(String objectType) {  
 String query = 'SELECT Id, Rule\_Code\_\_c, Rule\_Name\_\_c, Object\_API\_Name\_\_c, ' +  
 'Priority\_\_c, Rule\_Type\_\_c, Description\_\_c, Active\_\_c ' +  
 'FROM Data\_Quality\_Rule\_\_mdt WHERE Active\_\_c = true';  
   
 if (String.isNotBlank(objectType)) {  
 query += ' AND Object\_API\_Name\_\_c = :objectType';  
 }  
   
 query += ' ORDER BY Priority\_\_c, Rule\_Code\_\_c';  
   
 return Database.query(query);  
 }  
   
 /\*\*  
 \* Récupère les exécutions récentes  
 \*/  
 @AuraEnabled(cacheable=true)  
 public static List<Data\_Quality\_Execution\_\_c> getRecentExecutions(Integer limitCount) {  
 if (limitCount == null || limitCount <= 0) {  
 limitCount = 50;  
 }  
   
 return [  
 SELECT Id, Name, Execution\_Date\_\_c, Status\_\_c, Execution\_Type\_\_c,  
 Total\_Records\_Processed\_\_c, Total\_Violations\_\_c,   
 Total\_Rules\_Executed\_\_c, Started\_By\_\_r.Name,  
 Completed\_Date\_\_c, Duration\_Seconds\_\_c  
 FROM Data\_Quality\_Execution\_\_c  
 ORDER BY Execution\_Date\_\_c DESC  
 LIMIT :limitCount  
 ];  
 }  
   
 /\*\*  
 \* Récupère les violations d'une exécution  
 \*/  
 @AuraEnabled(cacheable=true)  
 public static List<Data\_Quality\_Violation\_\_c> getViolations(  
 Id executionId,  
 String status,  
 String priority  
 ) {  
 String query = 'SELECT Id, Name, Rule\_Code\_\_c, Rule\_Name\_\_c, ' +  
 'Record\_Id\_\_c, Object\_Type\_\_c, Priority\_\_c, ' +  
 'Field\_Name\_\_c, Current\_Value\_\_c, Expected\_Value\_\_c, ' +  
 'Error\_Message\_\_c, Status\_\_c, Assigned\_To\_\_r.Name, ' +  
 'Detection\_Date\_\_c, Age\_Days\_\_c ' +  
 'FROM Data\_Quality\_Violation\_\_c ';  
   
 List<String> conditions = new List<String>();  
   
 if (executionId != null) {  
 conditions.add('Execution\_\_c = :executionId');  
 }  
 if (String.isNotBlank(status)) {  
 conditions.add('Status\_\_c = :status');  
 }  
 if (String.isNotBlank(priority)) {  
 conditions.add('Priority\_\_c = :priority');  
 }  
   
 if (!conditions.isEmpty()) {  
 query += ' WHERE ' + String.join(conditions, ' AND ');  
 }  
   
 query += ' ORDER BY Priority\_\_c, Detection\_Date\_\_c DESC LIMIT 2000';  
   
 return Database.query(query);  
 }  
   
 /\*\*  
 \* Met à jour le statut d'une violation  
 \*/  
 @AuraEnabled  
 public static void updateViolationStatus(  
 Id violationId,  
 String newStatus,  
 String resolutionNotes  
 ) {  
 try {  
 Data\_Quality\_Violation\_\_c violation = [  
 SELECT Id, Status\_\_c   
 FROM Data\_Quality\_Violation\_\_c   
 WHERE Id = :violationId  
 LIMIT 1  
 ];  
   
 violation.Status\_\_c = newStatus;  
   
 if (String.isNotBlank(resolutionNotes)) {  
 violation.Resolution\_Notes\_\_c = resolutionNotes;  
 }  
   
 if (newStatus == 'Fixed') {  
 violation.Fixed\_Date\_\_c = System.now();  
 violation.Fixed\_By\_\_c = UserInfo.getUserId();  
 }  
   
 update violation;  
 } catch (Exception e) {  
 throw new AuraHandledException('Erreur: ' + e.getMessage());  
 }  
 }  
   
 /\*\*  
 \* Fixe automatiquement une violation si possible  
 \*/  
 @AuraEnabled  
 public static Boolean autoFixViolation(Id violationId) {  
 try {  
 Data\_Quality\_Violation\_\_c violation = [  
 SELECT Id, Record\_Id\_\_c, Field\_Name\_\_c, Expected\_Value\_\_c,  
 Execution\_\_r.Rule\_Codes\_\_c  
 FROM Data\_Quality\_Violation\_\_c   
 WHERE Id = :violationId  
 LIMIT 1  
 ];  
   
 // Récupérer la règle pour voir s'il y a une classe d'auto-fix  
 Data\_Quality\_Rule\_\_mdt rule = [  
 SELECT Auto\_Fix\_Class\_\_c  
 FROM Data\_Quality\_Rule\_\_mdt  
 WHERE Rule\_Code\_\_c = :violation.Rule\_Code\_\_c  
 LIMIT 1  
 ];  
   
 if (String.isNotBlank(rule.Auto\_Fix\_Class\_\_c)) {  
 // Instancier et exécuter la classe d'auto-fix  
 Type autoFixType = Type.forName(rule.Auto\_Fix\_Class\_\_c);  
 DataQualityAutoFix autoFixer = (DataQualityAutoFix)autoFixType.newInstance();  
   
 Boolean fixed = autoFixer.fix(violation);  
   
 if (fixed) {  
 violation.Status\_\_c = 'Fixed';  
 violation.Fixed\_Date\_\_c = System.now();  
 violation.Fixed\_By\_\_c = UserInfo.getUserId();  
 update violation;  
 }  
   
 return fixed;  
 }  
   
 return false;  
 } catch (Exception e) {  
 throw new AuraHandledException('Erreur auto-fix: ' + e.getMessage());  
 }  
 }  
}

## 5.3 Interface pour Auto-Fix

/\*\*  
 \* Interface pour les classes d'auto-correction  
 \*/  
public interface DataQualityAutoFix {  
 Boolean fix(Data\_Quality\_Violation\_\_c violation);  
}  
  
/\*\*  
 \* Exemple d'implémentation pour un auto-fix  
 \*/  
public class LEICustodianBankAutoFix implements DataQualityAutoFix {  
   
 public Boolean fix(Data\_Quality\_Violation\_\_c violation) {  
 try {  
 // Récupérer l'enregistrement  
 Share\_Class\_\_c sc = [  
 SELECT Id, Share\_class\_\_r.Product\_\_r.LEI\_Of\_Custodian\_Bank\_\_c,  
 Share\_class\_\_r.Product\_\_r.Custodian\_Bank\_Name\_\_c  
 FROM Share\_Class\_\_c  
 WHERE Id = :violation.Record\_Id\_\_c  
 LIMIT 1  
 ];  
   
 // Logique de correction automatique  
 // Par exemple, récupérer le LEI depuis une source externe  
 String lei = lookupLEIFromExternalSource(  
 sc.Share\_class\_\_r.Product\_\_r.Custodian\_Bank\_Name\_\_c  
 );  
   
 if (String.isNotBlank(lei)) {  
 // Mettre à jour le produit  
 Product\_\_c prod = new Product\_\_c(  
 Id = sc.Share\_class\_\_r.Product\_\_c,  
 LEI\_Of\_Custodian\_Bank\_\_c = lei  
 );  
 update prod;  
 return true;  
 }  
   
 return false;  
 } catch (Exception e) {  
 System.debug('Error in auto-fix: ' + e.getMessage());  
 return false;  
 }  
 }  
   
 private String lookupLEIFromExternalSource(String bankName) {  
 // Callout vers une API externe  
 return null;  
 }  
}

# 6. Interface Utilisateur (Lightning Web Component)

## 6.1 dataQualityDashboard (LWC)

Composant principal pour visualiser et gérer les contrôles.

// dataQualityDashboard.js  
import { LightningElement, track, wire } from 'lwc';  
import { ShowToastEvent } from 'lightning/platformShowToastEvent';  
import { refreshApex } from '@salesforce/apex';  
  
import getActiveRules from '@salesforce/apex/DataQualityExecutor.getActiveRules';  
import getRecentExecutions from '@salesforce/apex/DataQualityExecutor.getRecentExecutions';  
import getViolations from '@salesforce/apex/DataQualityExecutor.getViolations';  
import executeBatch from '@salesforce/apex/DataQualityExecutor.executeBatch';  
import updateViolationStatus from '@salesforce/apex/DataQualityExecutor.updateViolationStatus';  
import autoFixViolation from '@salesforce/apex/DataQualityExecutor.autoFixViolation';  
  
export default class DataQualityDashboard extends LightningElement {  
 @track activeTab = 'rules';  
 @track selectedRules = [];  
 @track selectedExecution;  
 @track showViolationModal = false;  
 @track selectedViolation;  
   
 // Colonnes pour les tables  
 ruleColumns = [  
 { label: 'Code', fieldName: 'Rule\_Code\_\_c', type: 'text' },  
 { label: 'Nom', fieldName: 'Rule\_Name\_\_c', type: 'text' },  
 { label: 'Priorité', fieldName: 'Priority\_\_c', type: 'text' },  
 { label: 'Type', fieldName: 'Rule\_Type\_\_c', type: 'text' },  
 { label: 'Objet', fieldName: 'Object\_API\_Name\_\_c', type: 'text' },  
 { label: 'Actif', fieldName: 'Active\_\_c', type: 'boolean' }  
 ];  
   
 executionColumns = [  
 { label: 'Nom', fieldName: 'Name', type: 'text' },  
 { label: 'Date', fieldName: 'Execution\_Date\_\_c', type: 'date' },  
 { label: 'Statut', fieldName: 'Status\_\_c', type: 'text' },  
 { label: 'Type', fieldName: 'Execution\_Type\_\_c', type: 'text' },  
 { label: 'Enregistrements', fieldName: 'Total\_Records\_Processed\_\_c', type: 'number' },  
 { label: 'Violations', fieldName: 'Total\_Violations\_\_c', type: 'number' },  
 { label: 'Durée (s)', fieldName: 'Duration\_Seconds\_\_c', type: 'number' }  
 ];  
   
 violationColumns = [  
 { label: 'Nom', fieldName: 'Name', type: 'text' },  
 { label: 'Règle', fieldName: 'Rule\_Code\_\_c', type: 'text' },  
 { label: 'Priorité', fieldName: 'Priority\_\_c', type: 'text' },  
 { label: 'Champ', fieldName: 'Field\_Name\_\_c', type: 'text' },  
 { label: 'Valeur Actuelle', fieldName: 'Current\_Value\_\_c', type: 'text' },  
 { label: 'Statut', fieldName: 'Status\_\_c', type: 'text' },  
 { label: 'Âge (jours)', fieldName: 'Age\_Days\_\_c', type: 'number' },  
 {   
 type: 'action',   
 typeAttributes: {   
 rowActions: [  
 { label: 'Voir', name: 'view' },  
 { label: 'Corriger', name: 'fix' },  
 { label: 'Auto-Fix', name: 'autofix' },  
 { label: 'Ignorer', name: 'ignore' }  
 ]   
 }   
 }  
 ];  
   
 @wire(getActiveRules, { objectType: '' })  
 rules;  
   
 @wire(getRecentExecutions, { limitCount: 50 })  
 executions;  
   
 @track violations = [];  
   
 handleTabChange(event) {  
 this.activeTab = event.target.value;  
 }  
   
 handleRuleSelection(event) {  
 this.selectedRules = event.detail.selectedRows.map(row => row.Rule\_Code\_\_c);  
 }  
   
 handleExecutionSelection(event) {  
 const selectedRow = event.detail.selectedRows[0];  
 if (selectedRow) {  
 this.selectedExecution = selectedRow.Id;  
 this.loadViolations();  
 }  
 }  
   
 async loadViolations() {  
 try {  
 this.violations = await getViolations({  
 executionId: this.selectedExecution,  
 status: null,  
 priority: null  
 });  
 } catch (error) {  
 this.showToast('Erreur', error.body.message, 'error');  
 }  
 }  
   
 async handleExecuteBatch() {  
 try {  
 const batchId = await executeBatch({  
 ruleCodes: this.selectedRules.length > 0 ? this.selectedRules : null,  
 scopeFilter: null,  
 batchSize: 200  
 });  
   
 this.showToast(  
 'Succès',   
 'Batch lancé avec succès. ID: ' + batchId,   
 'success'  
 );  
   
 // Rafraîchir les exécutions  
 return refreshApex(this.executions);  
 } catch (error) {  
 this.showToast('Erreur', error.body.message, 'error');  
 }  
 }  
   
 handleViolationAction(event) {  
 const actionName = event.detail.action.name;  
 const row = event.detail.row;  
   
 switch (actionName) {  
 case 'view':  
 this.viewViolation(row);  
 break;  
 case 'fix':  
 this.fixViolation(row);  
 break;  
 case 'autofix':  
 this.autoFixViolation(row);  
 break;  
 case 'ignore':  
 this.ignoreViolation(row);  
 break;  
 }  
 }  
   
 viewViolation(violation) {  
 this.selectedViolation = violation;  
 this.showViolationModal = true;  
 }  
   
 async fixViolation(violation) {  
 // Ouvrir l'enregistrement dans une modal d'édition  
 // Utiliser NavigationMixin pour naviguer vers l'enregistrement  
 window.open('/' + violation.Record\_Id\_\_c, '\_blank');  
 }  
   
 async autoFixViolation(violation) {  
 try {  
 const fixed = await autoFixViolation({ violationId: violation.Id });  
   
 if (fixed) {  
 this.showToast('Succès', 'Violation corrigée automatiquement', 'success');  
 this.loadViolations();  
 } else {  
 this.showToast('Info', 'Impossible de corriger automatiquement', 'info');  
 }  
 } catch (error) {  
 this.showToast('Erreur', error.body.message, 'error');  
 }  
 }  
   
 async ignoreViolation(violation) {  
 try {  
 await updateViolationStatus({  
 violationId: violation.Id,  
 newStatus: 'Ignored',  
 resolutionNotes: 'Ignoré par l\'utilisateur'  
 });  
   
 this.showToast('Succès', 'Violation ignorée', 'success');  
 this.loadViolations();  
 } catch (error) {  
 this.showToast('Erreur', error.body.message, 'error');  
 }  
 }  
   
 closeModal() {  
 this.showViolationModal = false;  
 this.selectedViolation = null;  
 }  
   
 showToast(title, message, variant) {  
 const event = new ShowToastEvent({  
 title: title,  
 message: message,  
 variant: variant  
 });  
 this.dispatchEvent(event);  
 }  
}

## 6.2 Template HTML

<!-- dataQualityDashboard.html -->  
<template>  
 <lightning-card title="Tableau de Bord - Contrôle Qualité des Données"   
 icon-name="standard:record">  
   
 <!-- Onglets -->  
 <lightning-tabset active-tab-value={activeTab} onselect={handleTabChange}>  
   
 <!-- Onglet Règles -->  
 <lightning-tab label="Règles de Contrôle" value="rules">  
 <div class="slds-m-around\_medium">  
 <lightning-button   
 label="Exécuter les Contrôles Sélectionnés"  
 variant="brand"  
 onclick={handleExecuteBatch}  
 disabled={selectedRules.length === 0}  
 class="slds-m-bottom\_small">  
 </lightning-button>  
   
 <lightning-datatable  
 key-field="Id"  
 data={rules.data}  
 columns={ruleColumns}  
 onrowselection={handleRuleSelection}>  
 </lightning-datatable>  
 </div>  
 </lightning-tab>  
   
 <!-- Onglet Exécutions -->  
 <lightning-tab label="Historique d'Exécution" value="executions">  
 <div class="slds-m-around\_medium">  
 <lightning-datatable  
 key-field="Id"  
 data={executions.data}  
 columns={executionColumns}  
 onrowselection={handleExecutionSelection}>  
 </lightning-datatable>  
 </div>  
 </lightning-tab>  
   
 <!-- Onglet Violations -->  
 <lightning-tab label="Violations Détectées" value="violations">  
 <div class="slds-m-around\_medium">  
 <lightning-datatable  
 key-field="Id"  
 data={violations}  
 columns={violationColumns}  
 onrowaction={handleViolationAction}>  
 </lightning-datatable>  
 </div>  
 </lightning-tab>  
   
 </lightning-tabset>  
   
 <!-- Modal de détail de violation -->  
 <template if:true={showViolationModal}>  
 <section role="dialog" class="slds-modal slds-fade-in-open">  
 <div class="slds-modal\_\_container">  
 <header class="slds-modal\_\_header">  
 <h2 class="slds-text-heading\_medium">Détails de la Violation</h2>  
 </header>  
 <div class="slds-modal\_\_content">  
 <!-- Contenu du modal -->  
 </div>  
 <footer class="slds-modal\_\_footer">  
 <lightning-button   
 label="Fermer"   
 onclick={closeModal}>  
 </lightning-button>  
 </footer>  
 </div>  
 </section>  
 <div class="slds-backdrop slds-backdrop\_open"></div>  
 </template>  
   
 </lightning-card>  
</template>

# 7. Planification et Automatisation

## 7.1 Classe Schedulable

global class DataQualityScheduler implements Schedulable {  
   
 private List<String> ruleCodes;  
 private String scopeFilter;  
 private Integer batchSize;  
   
 global DataQualityScheduler() {  
 this(null, null, 200);  
 }  
   
 global DataQualityScheduler(List<String> ruleCodes, String scopeFilter, Integer batchSize) {  
 this.ruleCodes = ruleCodes;  
 this.scopeFilter = scopeFilter;  
 this.batchSize = batchSize;  
 }  
   
 global void execute(SchedulableContext sc) {  
 DataQualityBatch batch = new DataQualityBatch(this.ruleCodes, this.scopeFilter);  
 Database.executeBatch(batch, this.batchSize);  
 }  
}  
  
// Configuration dans Anonymous Apex ou via l'interface  
// Exécution quotidienne à 2h du matin  
String cronExp = '0 0 2 \* \* ?';  
System.schedule('Data Quality Daily Check', cronExp, new DataQualityScheduler());  
  
// Exécution hebdomadaire le dimanche à 3h du matin  
String cronExp = '0 0 3 ? \* SUN';  
System.schedule('Data Quality Weekly Check', cronExp, new DataQualityScheduler());

## 7.2 Flow pour Exécution à la Demande

Créer un Screen Flow pour permettre aux utilisateurs de lancer des contrôles :

* Screen 1 : Sélection des règles
* Screen 2 : Options de filtrage (optionnel)
* Action Apex : Appel de DataQualityExecutor.executeBatch()
* Screen 3 : Confirmation et ID du batch

# 8. Système de Notifications

## 8.1 Email Templates

Créer des Email Templates pour les notifications :

* Data Quality Execution Summary (après chaque exécution)
* High Priority Violations Alert (violations P1)
* Daily Digest (résumé quotidien)

## 8.2 Classe de Notification

public class DataQualityNotifier {  
   
 public static void sendExecutionSummary(Data\_Quality\_Execution\_\_c execution) {  
 List<Messaging.SingleEmailMessage> emails = new List<Messaging.SingleEmailMessage>();  
   
 // Récupérer les violations P1  
 List<Data\_Quality\_Violation\_\_c> p1Violations = [  
 SELECT Id, Rule\_Code\_\_c, Rule\_Name\_\_c, Priority\_\_c  
 FROM Data\_Quality\_Violation\_\_c  
 WHERE Execution\_\_c = :execution.Id   
 AND Priority\_\_c = 'P1'  
 ];  
   
 // Template email  
 EmailTemplate template = [  
 SELECT Id, DeveloperName   
 FROM EmailTemplate   
 WHERE DeveloperName = 'Data\_Quality\_Execution\_Summary'   
 LIMIT 1  
 ];  
   
 // Récupérer les destinataires depuis les règles  
 Set<String> recipientEmails = new Set<String>();  
 for (Data\_Quality\_Rule\_\_mdt rule : [  
 SELECT Notification\_Emails\_\_c   
 FROM Data\_Quality\_Rule\_\_mdt   
 WHERE Active\_\_c = true   
 AND Notification\_Emails\_\_c != null  
 ]) {  
 if (String.isNotBlank(rule.Notification\_Emails\_\_c)) {  
 recipientEmails.addAll(rule.Notification\_Emails\_\_c.split(','));  
 }  
 }  
   
 for (String email : recipientEmails) {  
 Messaging.SingleEmailMessage mail = new Messaging.SingleEmailMessage();  
 mail.setToAddresses(new List<String>{ email.trim() });  
 mail.setTemplateId(template.Id);  
 mail.setTargetObjectId(execution.Started\_By\_\_c);  
 mail.setWhatId(execution.Id);  
 mail.setSaveAsActivity(false);  
 emails.add(mail);  
 }  
   
 if (!emails.isEmpty()) {  
 Messaging.sendEmail(emails);  
 }  
 }  
   
 public static void sendDailyDigest() {  
 // Récupérer les violations ouvertes des 24 dernières heures  
 List<Data\_Quality\_Violation\_\_c> recentViolations = [  
 SELECT Id, Rule\_Code\_\_c, Priority\_\_c, Status\_\_c  
 FROM Data\_Quality\_Violation\_\_c  
 WHERE Detection\_Date\_\_c = LAST\_N\_DAYS:1  
 AND Status\_\_c = 'Open'  
 ORDER BY Priority\_\_c, Detection\_Date\_\_c DESC  
 ];  
   
 // Statistiques  
 Map<String, Integer> statsByPriority = new Map<String, Integer>();  
 statsByPriority.put('P1', 0);  
 statsByPriority.put('P2', 0);  
 statsByPriority.put('P3', 0);  
   
 for (Data\_Quality\_Violation\_\_c v : recentViolations) {  
 Integer count = statsByPriority.get(v.Priority\_\_c);  
 statsByPriority.put(v.Priority\_\_c, count + 1);  
 }  
   
 // Envoyer email digest  
 // ...  
 }  
}

# 9. Rapports et Dashboards

## 9.1 Rapports à Créer

* Violations Ouvertes par Priorité
* Tendance des Violations (7/30/90 jours)
* Top 10 Règles avec le Plus de Violations
* Temps Moyen de Résolution par Priorité
* Violations par Objet/Champ
* Historique d'Exécution (succès/échecs)

## 9.2 Dashboard Exécutif

Créer un dashboard avec :

* Gauge : % de données conformes
* Graphique : Violations par priorité
* Graphique : Tendance mensuelle
* Tableau : Top violations non résolues
* Métrique : Temps moyen de résolution

# 10. Tests Unitaires

@isTest  
private class DataQualityBatch\_Test {  
   
 @testSetup  
 static void setup() {  
 // Créer des données de test  
 Account acc = new Account(Name = 'Test Account');  
 insert acc;  
   
 Product\_\_c prod = new Product\_\_c(  
 Name = 'Test Product',  
 Manco\_\_c = 'Ossiam',  
 Mandate\_Open\_Dedicated\_\_c = 'Open-End Fund'  
 );  
 insert prod;  
   
 Share\_Class\_\_c sc = new Share\_Class\_\_c(  
 Name = 'Test Share Class',  
 Product\_\_c = prod.Id,  
 Management\_Fees\_\_c = 0.25,  
 Management\_Fees\_Ex\_Ante\_\_c = 0.30 // Violation intentionnelle  
 );  
 insert sc;  
 }  
   
 @isTest  
 static void testBatchExecution() {  
 Test.startTest();  
   
 DataQualityBatch batch = new DataQualityBatch(null, null);  
 Id batchId = Database.executeBatch(batch, 200);  
   
 Test.stopTest();  
   
 // Vérifier qu'une exécution a été créée  
 List<Data\_Quality\_Execution\_\_c> executions = [  
 SELECT Id, Status\_\_c, Total\_Violations\_\_c   
 FROM Data\_Quality\_Execution\_\_c  
 ];  
   
 System.assertEquals(1, executions.size(), 'Une exécution devrait être créée');  
 System.assertEquals('Completed', executions[0].Status\_\_c);  
   
 // Vérifier qu'au moins une violation a été détectée  
 List<Data\_Quality\_Violation\_\_c> violations = [  
 SELECT Id FROM Data\_Quality\_Violation\_\_c  
 ];  
   
 System.assert(violations.size() > 0, 'Au moins une violation devrait être détectée');  
 }  
   
 @isTest  
 static void testViolationUpdate() {  
 // Créer une exécution et une violation  
 Data\_Quality\_Execution\_\_c exec = new Data\_Quality\_Execution\_\_c(  
 Status\_\_c = 'Completed',  
 Execution\_Date\_\_c = System.now()  
 );  
 insert exec;  
   
 Data\_Quality\_Violation\_\_c violation = new Data\_Quality\_Violation\_\_c(  
 Execution\_\_c = exec.Id,  
 Rule\_Code\_\_c = 'TEST001',  
 Status\_\_c = 'Open',  
 Record\_Id\_\_c = '001000000000000'  
 );  
 insert violation;  
   
 Test.startTest();  
   
 DataQualityExecutor.updateViolationStatus(  
 violation.Id,   
 'Fixed',   
 'Corrigé manuellement'  
 );  
   
 Test.stopTest();  
   
 violation = [  
 SELECT Status\_\_c, Fixed\_Date\_\_c, Fixed\_By\_\_c   
 FROM Data\_Quality\_Violation\_\_c   
 WHERE Id = :violation.Id  
 ];  
   
 System.assertEquals('Fixed', violation.Status\_\_c);  
 System.assertNotEquals(null, violation.Fixed\_Date\_\_c);  
 System.assertEquals(UserInfo.getUserId(), violation.Fixed\_By\_\_c);  
 }  
}

# 11. Guide de Déploiement

## 11.1 Ordre de Déploiement

1. 1. Custom Metadata Type : Data\_Quality\_Rule\_\_mdt
2. 2. Custom Objects : Data\_Quality\_Execution\_\_c et Data\_Quality\_Violation\_\_c
3. 3. Apex Classes : DataQualityAutoFix, DataQualityBatch, DataQualityExecutor, DataQualityScheduler
4. 4. Test Classes
5. 5. Lightning Web Components
6. 6. Email Templates
7. 7. Custom Metadata Records (règles de contrôle)
8. 8. Reports et Dashboards
9. 9. Permission Sets et Profils
10. 10. Scheduled Jobs (via Anonymous Apex)

## 11.2 Configuration Post-Déploiement

* Importer les règles de contrôle dans Custom Metadata
* Configurer les Email Templates
* Créer les Permission Sets
* Planifier les jobs récurrents
* Créer les rapports et dashboards
* Former les utilisateurs

# 12. Avantages de cette Architecture

* **Flexibilité :** Les règles sont configurées via Custom Metadata sans nécessiter de déploiement de code
* **Scalabilité :** Le système batch peut traiter de gros volumes de données
* **Maintenabilité :** Architecture modulaire et séparation des responsabilités
* **User-Friendly :** Interface LWC intuitive pour visualiser et corriger les violations
* **Traçabilité :** Historique complet des exécutions et violations
* **Automatisation :** Exécution planifiée et possibilité d'auto-correction
* **Notifications :** Alertes automatiques pour les violations critiques
* **Reporting :** Tableaux de bord et rapports pour le suivi de la qualité
* **Extensibilité :** Facile d'ajouter de nouvelles règles ou de nouveaux types de contrôles

# 13. Évolutions Futures Possibles

* Intelligence Artificielle pour détection proactive des anomalies
* Intégration avec des outils externes de data quality
* Moteur de règles plus avancé (expressions complexes, ML)
* Dashboard temps réel avec Einstein Analytics
* API REST pour exécution externe
* Système de workflow d'approbation pour les corrections
* Versioning des règles de contrôle
* Support multi-objets simultané

# 14. Conclusion

Cette architecture fournit une base solide pour un système de contrôle de qualité de données robuste, flexible et évolutif dans Salesforce. Elle permet de :

* Définir des contrôles de manière déclarative via Custom Metadata
* Exécuter des contrôles en batch pour de gros volumes
* Lancer des contrôles à la demande via une interface intuitive
* Visualiser et corriger les violations efficacement
* Automatiser les corrections lorsque possible
* Suivre et reporter sur la qualité des données

Le système peut être adapté et étendu selon les besoins spécifiques de votre organisation, en ajoutant de nouvelles règles, de nouveaux types de contrôles, ou en intégrant avec d'autres systèmes existants.