Internal Tools - Table Comparison Engine

Version: 1.0.5-SNAPSHOT | **Stack:** Scala 2.12.17 + Spark 3.5.0 ---

🚀 Guía de Uso - Motor de Comparación de Tablas Spark

Motor distribuido para comparar tablas Spark a nivel de fila, columna y clave compuesta. Genera 3 tablas de salida con análisis exhaustivo: diferencias, duplicados y métricas de calidad.

> ⚡ **TL;DR (30 segundos):** Motor Spark para comparar tablas grandes con análisis exhaustivo. > - ✅ **Detecta diferencias** columna por columna entre REF y NEW > - ✅ **Identifica duplicados** y variaciones por clave compuesta > - ✅ **Calcula métricas de calidad** automáticas (Global Quality) > - ✅ **Genera 3 tablas SQL** listas para análisis: differences, duplicates, summary > - 🚀 **Copy-paste ready**: Ver ejemplo básico

---

📋 Índice

🎯 Parte I: Guía de Uso

1. Inicio Rápido
2. Configuración Completa
3. Entendiendo los Resultados
4. Casos de Uso Comunes
5. Diagnóstico y Troubleshooting

⚙️ Parte II: Referencia Técnica

1. Arquitectura del Motor
2. Schema y Semántica de Salidas
3. Optimizaciones y Performance
4. Limitaciones y Buenas Prácticas

PARTE I: GUÍA DE USO

---

1. Inicio Rápido

1.1 Ejemplo Básico (Comparación Simple)

Compara dos tablas del mismo día con claves compuestas:

```bash
spark-submit \ --class com.santander.cib.adhc.internal_aml_tools.Main \ --master yarn --deploy-mode cluster \ --driver-memory 4g --executor-memory 8g \ cib-adhc-internaltools-1.0.5-SNAPSHOT.jar \ refTable=default.payments_ref \ newTable=default.payments_new \ compositeKeyCols=transaction_id,customer_id \ partitionSpec="geo=ES/data_date_part=2025-11-19/" \ ignoreCols=ingestion_ts,audit_user \ initiativeName=PaymentsMigration \ tablePrefix=default.comparison_ \ outputBucket=s3a://my-bucket/comparisons \ executionDate=2025-11-19 \ checkDuplicates=true
```

¿Qué hace esto?

- Compara payments_ref vs payments_new usando transaction_id + customer_id como clave
- Filtra por España y fecha 2025-11-19
- Ignora columnas técnicas (ingestion_ts , audit_user)
- Genera 3 tablas: comparison_differences , comparison_duplicates , comparison_summary
- Detecta duplicados en ambos lados

1.2 Verificar Resultados

```sql
-- 1. Ver métricas generales SELECT block, metric, numerator, denominator, pct FROM default.comparison_summary WHERE initiative = 'PaymentsMigration' ORDER BY block, metric;

-- 2. Ver diferencias (solo no coincidentes) SELECT id, column, value_ref, value_new FROM default.comparison_differences WHERE results = 'NO_MATCH' LIMIT 100;

-- 3. Ver duplicados problemáticos SELECT origin, id, occurrences, dupes_w_variations, variations FROM default.comparison_duplicates WHERE dupes_w_variations > 0 ORDER BY CAST(occurrences AS INT) DESC;
```

---

2. Configuración Completa

2.1 Parámetros Obligatorios

Parámetro	Descripción	Ejemplo
refTable	Tabla de referencia (histórica)	default.payments_ref
newTable	Tabla nueva (candidata)	default.payments_new
compositeKeyCols	Columnas clave, separadas por comas	transaction_id,customer_id
initiativeName	Etiqueta para identificar la comparación	PaymentsMigration
tablePrefix	Prefijo para tablas resultado	default.comparison_
outputBucket	Ruta S3 base para outputs	s3a://bucket/comparisons
executionDate	Fecha de ejecución (ISO)	2025-11-19

2.2 Parámetros Opcionales Básicos

Parámetro	Default	Descripción	Ejemplo
partitionSpec	-	Filtro de particiones para ambas tablas	geo=ES/data_date_part=2025-11-19/
ignoreCols	-	Columnas a excluir de la comparación (CSV)	ingestion_ts,audit_user,version
checkDuplicates	false	Activar análisis de duplicados	true
includeEqualsInDiff	false	Incluir coincidencias (MATCH) en tabla differences	false
priorityCol	-	Columna para resolver duplicados (mantiene valor más alto)	update_timestamp , version

---

2.3 Filtrado de Particiones (partitionSpec)

Sintaxis: columna1=valores/columna2=valores/columna3=valores

Formatos soportados:

Formato	Ejemplo	Significado
Valor único	geo=ES	Solo España
Wildcard	geo=*	Todos los geos (sin filtro)
Lista corchetes	geo=[ES,PT,FR]	España, Portugal o Francia
Lista pipe	geo=(ES\	PT\ FR) España, Portugal o Francia
IN corchetes	geo=IN[ES,PT]	España o Portugal
IN paréntesis	geo=IN(ES,PT)	España o Portugal

Ejemplos:

` bash

Una fecha, un geo

partitionSpec="geo=ES/data_date_part=2025-11-19/"

Múltiples geos, una fecha

partitionSpec="geo=[ES,PT,FR]/data_date_part=2025-11-19/"

Todos los geos, una fecha (resuelve automáticamente)

partitionSpec="geo=*/data_date_part=2025-11-19/"

Múltiples fechas (lista explícita)

partitionSpec="geo=ES/data_date_part=[2025-11-18,2025-11-19,2025-11-20]/"

Tres niveles de partición

partitionSpec="geo=ES/data_date_part=2025-11-19/process_name=Guarantees/" `

2.4 Parámetros Avanzados

2.4.1 Ventanas Temporales

Compara diferentes rangos de fechas en cada tabla manteniendo el mismo executionDate de salida:

Parámetro	Descripción	Ejemplo
refWindowDays	Ventana temporal REF (start..end)	-2..+2 (5 días: -2,-1,0,+1,+2)
newWindowDays	Ventana temporal NEW (start..end)	0..+1 (2 días: 0,+1)

` bash

Ejemplo: REF lee 7 días atrás, NEW lee hoy + 1 día

executionDate=2025-11-19 partitionSpec="geo=ES/data_date_part=2025-11-19/" refWindowDays=-7..0 newWindowDays=0..+1

REF lee: 2025-11-12 hasta 2025-11-19

NEW lee: 2025-11-19 hasta 2025-11-20

`

2.4.2 Overrides por Lado

Especifica particiones **completamente diferentes** para cada tabla:

Parámetro	Descripción	Ejemplo	
refPartitionSpec	Override completo para REF	geo=ES/data_date_part=[2025-11-18,2025-11-19]	
newPartitionSpec	Override completo para NEW	geo=(PT\	ES)/data_date_part=IN(2025-11-19)

` bash

Ejemplo: Comparar 1 día REF vs 19 meses NEW

refPartitionSpec="data_date_part=2025-11-05/process_group=gar_group" newPartitionSpec="data_date_part=[2024-05-01,2024-06-01,...,2025-11-01]/process_name=(Cash|Guarantees)" `

Precedencia: refPartitionSpec > refWindowDays > partitionSpec

2.4.3 Filtros SQL Personalizados (Nuevo ✨)

Filtra filas **después** del filtrado de particiones usando expresiones SQL de Spark:

Parámetro	Descripción	Ejemplo
refFilter	Expresión SQL para filtrar REF	geo IN ('ES','FR') AND time LIKE '06:%'
newFilter	Expresión SQL para filtrar NEW	amount >= 1000 AND status = 'ACTIVE'

Operadores soportados:

Operador	Ejemplo
IN , NOT IN	geo IN ('ES','FR') , geo NOT IN ('BR')
= , != , <> , > , >= , < , <=	status = 'ACTIVE' , amount >= 1000
LIKE	time LIKE '06:%' , message LIKE '%URGENT%'
IS NULL , IS NOT NULL	rejected IS NULL
AND , OR , NOT	geo = 'ES' AND amount > 1000
BETWEEN	amount BETWEEN 1000 AND 50000
Paréntesis	(geo = 'ES' OR geo = 'FR') AND amount > 1000

Ejemplos:

` bash

Filtros simples

refFilter="geo IN ('ES','FR','PT')" refFilter="time LIKE '06:%'" refFilter="amount >= 1000"

Filtros combinados

refFilter="geo IN ('ES','FR') AND time LIKE '06:%'" refFilter="amount BETWEEN 1000 AND 50000 AND status = 'ACTIVE'" refFilter="(geo = 'ES' OR geo = 'FR') AND (message_type IN ('MT103','MT202') OR validation_type = 'AUTO')"

Filtros con NULL

refFilter="rejected IS NULL OR rejected = ''"

Case-insensitive

refFilter="UPPER(status) = 'ACTIVE'" `

💡 **Tips de Performance:**

- **Filtros por columnas particionadas** → usa partitionSpec en su lugar (más rápido)
- **LIKE 'pattern%'** → OK, pero evita LIKE '%pattern%' (scan completo)
- **Combina estratégicamente:** Filtro grueso con particiones + filtro fino con SQL

` bash partitionSpec="data_date_part=2025-10-01/" # Filtro grueso por día refFilter="time LIKE '06:%'" # Filtro fino por hora `

2.5 Orden de Aplicación de Filtros

`

1. Carga inicial de tablas

↓

1. Filtrado de particiones

(partitionSpec / refPartitionSpec / newPartitionSpec / refWindowDays / newWindowDays) ↓

1. Filtros SQL personalizados

(refFilter / newFilter) ↓

1. Exclusión de columnas

(ignoreCols) ↓

1. Comparación

(solo filas y columnas que pasaron todos los filtros) `

---

3. Entendiendo los Resultados

El motor genera **3 tablas de salida** con información complementaria:

Tabla	Prefijo	Contenido
Differences	{tablePrefix}differences	Diferencias columna por columna (NO_MATCH, ONLY_IN_*)
Duplicates	{tablePrefix}duplicates	Análisis de duplicados por clave (ocurrencias, variaciones)
Summary	{tablePrefix}summary	KPIs y métricas agregadas (calidad global, gaps, etc.)

3.1 Tabla differences - Vista Rápida

¿Qué muestra? Diferencias detalladas a nivel de columna para cada clave.

` sql -- Ver solo diferencias (excluir coincidencias) SELECT id, column, value_ref, value_new, results FROM comparison_differences WHERE results NOT IN ('MATCH', 'EXACT_MATCH') ORDER BY id, column; `

Etiquetas de resultados:

Tag	Significado
EXACT_MATCH	**Todas** las columnas idénticas (1 fila por clave)
MATCH	Columna específica coincide
NO_MATCH	Columna específica difiere
ONLY_IN_REF	Clave o columna solo existe en REF
ONLY_IN_NEW	Clave o columna solo existe en NEW

Ejemplos:

```
id="123", column="amount", value_ref="100.50", value_new="100.51", results="NO_MATCH"
id="456", column="*", value_ref="-", value_new="-", results="EXACT_MATCH" id="789",
column="country", value_ref="ES", value_new="-", results="ONLY_IN_REF"
```

3.2 Tabla duplicates - Análisis de Duplicados

¿Qué muestra? Análisis exhaustivo de unicidad por clave compuesta, detectando duplicados exactos y con variaciones en cada tabla (REF y NEW).

3.2.1 Schema Completo

Columna	Tipo	Descripción	
origin	String	"ref" o "new" (tabla de origen)	
id	String	Clave compuesta (NULL-safe: valores NULL → "NULL")	
category	String	"both", "only_ref", o "only_new" (coherente con summary.DUPS)	
exact_duplicates	String	Número de **copias exactas** (mismo hash SHA256)	
dupes_w_variations	String	Número de **grupos con variaciones** (hashes distintos)	
occurrences	String	**Total de filas** con esta clave	
variations	String	**Detalle de variaciones**: "campo: [val1,val2] \	campo2: [x,y]"

3.2.2 Interpretación de Métricas

Fórmulas:
```
exact_duplicates = occurrences - count(distinct _row_hash)
dupes_w_variations = max(0, count(distinct _row_hash) - 1)
```

Escenarios típicos:

Caso	occurrences	exact_dup	dupes_w_var	variations	Interpretación
A	1	-	-	-	✅ **No duplicado** (no aparece en tabla)
B	3	2	0	-	3 filas idénticas (copias exactas)
C	3	1	1	amount: [100,200]	2 filas iguales + 1 con amount diferente
D	5	0	4	status: [A,B,C,D,E]	5 filas todas distintas (máxima variación)

Ejemplo real: ` origin="ref", id="TXN_123_CUST_456", category="both" exact_duplicates="2", dupes_w_variations="1", occurrences="4" variations="amount: [100.00,100.50] | status: [ACTIVE,PENDING]"

→ Interpretación: • 4 filas con esta clave en REF • 2 copias exactas (mismo hash) • 2 grupos con variaciones distintas • Varían los campos: amount (2 valores) y status (2 valores) `

3.2.3 Columna `category` - Coherencia con Summary

La columna `category` categoriza cada ID duplicado según su presencia en REF/NEW:

Category	Significado	Ejemplo
both	ID duplicado en **ambas** tablas	ID aparece 2+ veces en REF **Y** 2+ veces en NEW
only_ref	ID duplicado **solo en REF**	ID aparece 2+ veces en REF pero 0 o 1 vez en NEW
only_new	ID duplicado **solo en NEW**	ID aparece 2+ veces en NEW pero 0 o 1 vez en REF

Coherencia con `summary.DUPS`: ` sql -- Ambos reportan las mismas categorías con los mismos criterios SELECT category, COUNT(DISTINCT id) FROM duplicates GROUP BY category; -- ↓ coincide con ↓ SELECT metric, numerator FROM summary WHERE block = 'DUPS'; `

3.2.4 Queries Útiles

` sql -- 1. Ver solo duplicados problemáticos (con variaciones) SELECT origin, id, category, occurrences, dupes_w_variations, variations FROM comparison_duplicates WHERE dupes_w_variations > 0 ORDER BY CAST(occurrences AS INT) DESC;

-- 2. Duplicados solo por copias exactas (sin variaciones) SELECT origin, id, category, exact_duplicates, occurrences FROM comparison_duplicates WHERE exact_duplicates > 0 AND dupes_w_variations = 0 ORDER BY CAST(exact_duplicates AS INT) DESC;

-- 3. Duplicados problemáticos en ambos lados (categoría "both") SELECT origin, id,

```
occurrences, variations FROM comparison_duplicates WHERE category = 'both' AND
dupes_w_variations > 0 ORDER BY origin, CAST(occurrences AS INT) DESC;

-- 4. Top 10 IDs con más ocurrencias SELECT origin, id, occurrences, exact_duplicates,
variations FROM comparison_duplicates ORDER BY CAST(occurrences AS INT) DESC LIMIT 10;

-- 5. Análisis de campos que más varían SELECT origin, COUNT(DISTINCT id) as
affected_ids, SUM(CASE WHEN variations LIKE '%amount:%' THEN 1 ELSE 0 END) as vary_amount,
SUM(CASE WHEN variations LIKE '%status:%' THEN 1 ELSE 0 END) as vary_status FROM
comparison_duplicates WHERE dupes_w_variations > 0 GROUP BY origin;
```

3.2.5 Parámetro `priorityCol` - Resolución Inteligente de Duplicados

¿Qué hace? Filtra duplicados **antes** del análisis, manteniendo solo la fila con **mayor prioridad** dentro de cada grupo (clave + origen).

¿Cuándo usarlo?

Escenario	¿Usar priorityCol?	Columna recomendada
Tabla snapshot (1 fila por ID, datos estáticos)	❌ No necesario	-
Tabla histórica con versiones (CDC, SCD Type 2)	✅ **Sí**	version , update_timestamp , effective_date
Tabla con retries/reprocessing (mismo ID, múltiples intentos)	✅ **Sí**	processing_timestamp , retry_count
Tabla transaccional (cada fila es única por diseño)	❌ No necesario	-
Tabla con múltiples updates del mismo registro	✅ **Sí**	last_modified_date , sequence_number

Cómo funciona:

```
scala // Pseudocódigo interno Window.partitionBy(origin, key1, key2, ...) // Agrupa por
origen + clave .orderBy(priorityCol DESC NULLS LAST) // Ordena: valores altos primero,
NULL al final → Selecciona row_number() = 1 (fila con valor MÁS ALTO)
```

Criterios de ordenación:

- ✅ **Valores altos tienen prioridad:** 1000 > 100 > 10
- ✅ **Timestamps más recientes primero:** 2025-11-21 > 2025-11-20

- ✅ **NULLs al final (menor prioridad)**: Se descartan si existen valores no-NULL

Ejemplo - Tabla con múltiples updates:

` sql -- ANTES de priorityCol (datos crudos) REF table:
┌────┬──────────────────────┬────────┬────────┐ | id | update_timestamp | status | amount | ├────┼──────────────────────┼────────┼────────┤ | 123 | 2025-11-21 10:00:00 | A | 100 | ← Update v1 | 123 | 2025-11-21 10:05:00 | A | 100 | ← Update v2 (sin cambios) | 123 | 2025-11-21 10:10:00 | I | 200 | ← Update v3 (cambió) ✓ MÁS RECIENTE
└────┴──────────────────────┴────────┴────────┘

SIN priorityCol: → Detecta duplicado: 3 filas con id=123 → exact_duplicates="1" (2 filas iguales) → dupes_w_variations="2" (3 hashes distintos) → occurrences="3" → variations="status: [A,I] | amount: [100,200]"

CON priorityCol="update_timestamp": → Filtro previo: solo mantiene fila con 10:10:00 (timestamp más alto) → Resultado: 1 sola fila por id=123 → NO se reporta como duplicado (occurrences=1 → no entra en tabla) `

Uso en ejecución:

` bash

Ejemplo 1: Tabla histórica con timestamps

spark-submit --class com.santander.cib.adhc.internal_aml_tools.Main \ cib-adhc-internaltools-1.0.5-SNAPSHOT.jar \ refTable=default.transactions_history \ newTable=default.transactions_current \ compositeKeyCols=transaction_id \ partitionSpec="data_date_part=2025-11-21/" \ priorityCol=update_timestamp \ checkDuplicates=true \ ...

Ejemplo 2: Tabla con versionado numérico

priorityCol=version_number

Ejemplo 3: Tabla CDC con secuencia

priorityCol=sequence_id

Ejemplo 4: Tabla con flag de prioridad explícito

```
priorityCol=priority_flag # (valores: 1=alta, 0=baja)
```

Validaciones automáticas:

- ✅ Si priorityCol no existe en el schema → Se ignora (sin error)
- ✅ Si priorityCol es NULL/vacío → Se ignora
- ✅ Si la columna existe → Se aplica correctamente

Impacto en métricas:

Métrica	Sin priorityCol	Con priorityCol
Filas procesadas	Todas las filas	Solo filas con máxima prioridad
Duplicados detectados	Incluye versiones intermedias	Solo duplicados "reales"
Global Quality	Penalizado por versiones	Refleja calidad real
Performance	Más I/O y procesamiento	Menor volumen, más rápido

💡 **Recomendación:**

- Si tu tabla tiene campos como update_timestamp, version, last_modified_date → **Usa priorityCol**
- Si cada fila es única por diseño → No uses priorityCol (añade overhead innecesario)

3.3 Tabla summary - Vista Rápida

¿Qué muestra? KPIs de alto nivel: tamaños, intersección, gaps, calidad global.

```
sql -- Ver resumen completo SELECT block, metric, universe, numerator, denominator, pct, samples FROM comparison_summary ORDER BY block, metric;
```

Bloques principales:

Block	Métricas
KPIS	Unique IDs (REF/NEW), Total rows, Total diff, **Global quality**
EXACT MATCH	1:1 con todas las columnas idénticas
PARTIAL MATCH	1:1 con al menos una columna diferente
GAP	1:0 (solo en REF), 0:1 (solo en NEW)
DUPS	Duplicados en ambos lados, solo REF, solo NEW

Métrica clave: Global Quality

```
Global quality = (claves con EXACT_MATCH sin duplicados) / (total claves REF) * 100

¿Puedo reemplazar la tabla? → Global quality > 95% es buen indicador
```

---

4. Casos de Uso Comunes

4.1 Comparación Simple (Mismo Día, Misma Estructura)

```
bash spark-submit --class com.santander.cib.adhc.internal_aml_tools.Main \ cib-adhc-internaltools-1.0.5-SNAPSHOT.jar \ refTable=default.payments_old \ newTable=default.payments_new \ compositeKeyCols=txn_id,customer_id \ partitionSpec="geo=ES/data_date_part=2025-11-19/" \ ignoreCols=load_ts,audit_user \ initiativeName=PaymentsComparison \ tablePrefix=default.cmp_ \ outputBucket=s3a://bucket/comparisons \ executionDate=2025-11-19 \ checkDuplicates=true
```

4.2 Comparación con Ventanas Temporales Diferentes

```
bash

# REF: 7 días históricos, NEW: Hoy + 1 día futuro

spark-submit --class com.santander.cib.adhc.internal_aml_tools.Main \ cib-adhc-internaltools-1.0.5-SNAPSHOT.jar \ refTable=default.transactions_ref \ newTable=default.transactions_new \ compositeKeyCols=id \ partitionSpec="geo=*/data_date_part=2025-11-19/" \ refWindowDays=-7..0 \ newWindowDays=0..+1 \ initiativeName=WindowComparison \ tablePrefix=default.cmp_ \ outputBucket=s3a://bucket/comparisons \ executionDate=2025-11-19
```

4.3 Comparación con Particiones Completamente Diferentes

```bash
# REF: 1 día + 1 proceso, NEW: 19 meses + múltiples procesos
spark-submit --class com.santander.cib.adhc.internal_aml_tools.Main \
  cib-adhc-internaltools-1.0.5-SNAPSHOT.jar \
  refTable=default.old_payments \
  newTable=default.new_payments \
  compositeKeyCols=payment_id \
  refPartitionSpec="data_date_part=2025-11-05/process_group=guarantees" \
  newPartitionSpec="data_date_part=[2024-05-01,2024-06-01,2024-07-01,2024-08-01,2024-09-01,2024-10-01,2024-11-01,2024-12-01,2025-01-01,2025-02-01,2025-03-01,2025-04-01,2025-05-01,2025-06-01,2025-07-01,2025-08-01,2025-09-01,2025-10-01,2025-11-01]/process_name=(Cash|Guarantees|Swift)" \
  initiativeName=HistoricalMigration \
  tablePrefix=default.cmp_ \
  outputBucket=s3a://bucket/comparisons \
  executionDate=2025-11-19
```

4.4 Comparación con Filtros SQL Personalizados

```bash
# Solo transacciones de hora 06:xx en España y Francia para REF
# Excluyendo Brasil en NEW
spark-submit --class com.santander.cib.adhc.internal_aml_tools.Main \
  cib-adhc-internaltools-1.0.5-SNAPSHOT.jar \
  refTable=default.swift_ref \
  newTable=default.swift_new \
  compositeKeyCols=uetr,message_type \
  partitionSpec="data_date_part=2025-10-01/" \
  refFilter="geo IN ('ES','FR') AND time LIKE '06:%'" \
  newFilter="geo NOT IN ('BR') AND amount >= 1000" \
  ignoreCols=session_sequence,data_timestamp_part \
  initiativeName=SwiftFiltered \
  tablePrefix=default.cmp_ \
  outputBucket=s3a://bucket/comparisons \
  executionDate=2025-10-01 \
  checkDuplicates=true
```

4.5 Comparación con Resolución Automática de Duplicados (priorityCol)

```bash

# Tabla histórica con múltiples versiones/updates del mismo registro

# priorityCol mantiene solo la fila con timestamp más alto por cada ID

spark-submit --class com.santander.cib.adhc.internal_aml_tools.Main \ cib-adhc-internaltools-1.0.5-SNAPSHOT.jar \ refTable=default.transactions_history \ newTable=default.transactions_current \ compositeKeyCols=transaction_id \ partitionSpec="data_date_part=2025-11-21/" \ priorityCol=update_timestamp \ checkDuplicates=true \ initiativeName=HistoryComparison \ tablePrefix=default.cmp_ \ outputBucket=s3a://bucket/comparisons \ executionDate=2025-11-21

# ✅ Beneficio: Solo detecta duplicados "reales", no versiones intermedias

# ✅ Resultado: Global Quality más preciso (no penalizado por updates)

`

**Casos ideales para priorityCol :**

- Tablas CDC (Change Data Capture) → priorityCol=op_timestamp
- Tablas versionadas → priorityCol=version_number
- Tablas con reprocessing → priorityCol=processing_timestamp
- Tablas SCD Type 2 → priorityCol=effective_date
```

4.6 Comparación de Tablas con Esquemas Diferentes

` bash

REF tiene columna 'process_group', NEW tiene 'process_name'

El motor compara automáticamente solo columnas comunes

```
spark-submit --class com.santander.cib.adhc.internal_aml_tools.Main \ cib-adhc-
internaltools-1.0.5-SNAPSHOT.jar \ refTable=default.legacy_table \
newTable=default.migrated_table \ compositeKeyCols=id,geo \
partitionSpec="data_date_part=2025-11-19/" \ ignoreCols=process_group,process_name \
initiativeName=SchemaMismatch \ tablePrefix=default.cmp_ \
outputBucket=s3a://bucket/comparisons \ executionDate=2025-11-19 `
```

4.6 Configuración para Postman (Actualizada Nov 2025)

Endpoint típico: POST https://your-spark-cluster/api/jobs

Body (JSON): ` json { "class": "com.santander.cib.adhc.internal_aml_tools.Main", "appResource": "s3a://artifacts/cib-adhc-internaltools-1.0.5-SNAPSHOT.jar", "sparkProperties": { "spark.master": "yarn", "spark.submit.deployMode": "cluster", "spark.driver.memory": "4g", "spark.executor.memory": "8g", "spark.executor.cores": "4", "spark.dynamicAllocation.enabled": "true", "spark.sql.adaptive.enabled": "true", "spark.sql.hive.convertMetastoreParquet": "true" }, "arguments": ["refTable=scib_cm_cmplnc_trans_messages_swift_s3.swift_transactions", "newTable=scib_bu_cmplnc_trans_messages_swift_s3.swift_transactions_new", "compositeKeyCols=geo,uetr,type,in_out,message_type,validation_type,rejected", "partitionSpec=data_date_part=2025-11-20/", "ignoreCols=session_sequence,data_timestamp_part,ingestion_ts", "initiativeName=Swift_Nov2025", "tablePrefix=scib_bu_cmplnc_trans_messages_swift_s3.results_", "outputBucket=s3a://scib-pre-bu-cmplnc-trans-messages/internal-tools", "executionDate=2025-11-20", "checkDuplicates=true", "includeEqualsInDiff=false", "priorityCol=update_timestamp",

"refFilter=geo IN ('ES','FR') AND time LIKE '06:%'", "newFilter=geo NOT IN ('BR')"] } `

Headers necesarios: ` Content-Type: application/json Authorization: Bearer X-Requested-By: postman `

4.7 Configuración para Airflow DAG (Actualizada Nov 2025)

Variable Airflow - table_comparison_config :

` json { "spark_submit_config": { "class": "com.santander.cib.adhc.internal_aml_tools.Main", "jar_path": "s3a://scib-pre-bu-artifacts/cib-adhc-internaltools-1.0.5-SNAPSHOT.jar", "spark_conf": { "spark.master": "yarn", "spark.submit.deployMode": "cluster", "spark.driver.memory": "4g", "spark.executor.memory": "8g", "spark.executor.cores": "4", "spark.num.executors": "10", "spark.dynamicAllocation.enabled": "true", "spark.sql.adaptive.enabled": "true", "spark.sql.adaptive.coalescePartitions.enabled": "true", "spark.sql.hive.convertMetastoreParquet": "true" } }, "comparison_params": { "refTable": "{{ var.value.ref_table }}", "newTable": "{{ var.value.new_table }}", "compositeKeyCols": "transaction_id,customer_id,geo", "partitionSpec": "data_date_part={{ ds }}/", "ignoreCols": "ingestion_ts,audit_user,version,load_date", "initiativeName": "{{ var.value.initiative_name }}", "tablePrefix": "{{ var.value.schema }}.comparison_", "outputBucket": "s3a://{{ var.value.bucket }}/comparisons", "executionDate": "{{ ds }}", "checkDuplicates": "true", "includeEqualsInDiff": "false", "priorityCol": "update_timestamp" }, "advanced_params": { "refWindowDays": "-7..0", "newWindowDays": "0..+1", "refFilter": "amount >= 1000 AND status = 'ACTIVE'", "newFilter": "geo NOT IN ('BR','MX')" } } `

Ejemplo DAG Python:

` python from airflow import DAG from airflow.providers.apache.spark.operators.spark_submit import SparkSubmitOperator from airflow.models import Variable from datetime import datetime, timedelta

Cargar configuración

config = Variable.get("table_comparison_config", deserialize_json=True)

default_args = { 'owner': 'data-quality', 'depends_on_past': False, 'start_date': datetime(2025, 11, 1), 'email_on_failure': True, 'email': ['data-quality-team@santander.com'], 'retries': 2, 'retry_delay': timedelta(minutes=5) }

dag = DAG('table_comparison_daily', default_args=default_args, description='Comparación diaria de tablas', schedule_interval='0 6 *', # 6 AM diario catchup=False, tags=['data-quality', 'table-comparison'])

Construir argumentos

```
spark_conf = config['spark_submit_config']['spark_conf'] params =
config['comparison_params'] advanced = config.get('advanced_params', {})

arguments = [ f"refTable={params['refTable']}", f"newTable={params['newTable']}",
f"compositeKeyCols={params['compositeKeyCols']}", f"partitionSpec=
{params['partitionSpec']}", f"ignoreCols={params['ignoreCols']}", f"initiativeName=
{params['initiativeName']}", f"tablePrefix={params['tablePrefix']}", f"outputBucket=
{params['outputBucket']}", f"executionDate={params['executionDate']}", f"checkDuplicates=
{params['checkDuplicates']}", f"includeEqualsInDiff={params['includeEqualsInDiff']}" ]
```

Añadir parámetros avanzados si existen

```
if 'refWindowDays' in advanced: arguments.append(f"refWindowDays=
{advanced['refWindowDays']}") if 'newWindowDays' in advanced:
arguments.append(f"newWindowDays={advanced['newWindowDays']}") if 'refFilter' in advanced:
arguments.append(f"refFilter={advanced['refFilter']}") if 'newFilter' in advanced:
arguments.append(f"newFilter={advanced['newFilter']}")

table_comparison_task = SparkSubmitOperator( task_id='run_table_comparison',
application=config['spark_submit_config']['jar_path'],
java_class=config['spark_submit_config']['class'], conf=spark_conf,
application_args=arguments, dag=dag )
```

Tarea de validación post-comparación

```
from airflow.providers.amazon.aws.operators.athena import AthenaOperator

validate_results = AthenaOperator( task_id='validate_comparison_results', query=""" SELECT
metric, numerator, denominator, pct FROM {{ var.value.schema }}.comparison_summary WHERE
initiative = '{{ var.value.initiative_name }}' AND data_date_part = '{{ ds }}' AND metric
= 'Global quality' AND CAST(REPLACE(pct, '%', '') AS DOUBLE) < 95.0 """, database='{{
var.value.database }}', output_location='s3://{{ var.value.bucket }}/athena-results/',
dag=dag )
```

```
table_comparison_task >> validate_results `
```

Variables Airflow necesarias:

```
` json { "ref_table": "scib_cm_schema.payments_ref", "new_table":
"scib_bu_schema.payments_new", "initiative_name": "PaymentsMigration_Nov2025", "schema":
"scib_bu_schema", "bucket": "scib-pre-bu-data-quality", "database": "data_quality_db" }
`
```

---

5. Diagnóstico y Troubleshooting

5.1 Guía de Diagnóstico Rápido

Síntoma	Tabla a Revisar	Query	Interpretación
Global quality bajo	summary	WHERE metric='Global quality'	Revisar % PARTIAL MATCH y duplicados
Muchas diferencias	differences	WHERE results='NO_MATCH' GROUP BY column	Identificar columnas problemáticas
GAP alto (1:0 o 0:1)	summary	WHERE block='GAP'	Verificar partitionSpec y filtros
Duplicados	duplicates	WHERE dupes_w_variations > 0	Revisar calidad datos upstream
Nulls inesperados	differences	WHERE value_new='-' OR value_ref='-'	Verificar mappings ETL

5.2 Queries de Diagnóstico

```sql
-- 1. Top 10 columnas con más diferencias
SELECT column, COUNT(*) as mismatch_count
FROM comparison_differences
WHERE results = 'NO_MATCH'
GROUP BY column
ORDER BY mismatch_count DESC
LIMIT 10;

-- 2. Claves con duplicados en ambos lados (problema sistémico)
SELECT d1.id, d1.occurrences as ref_occ, d2.occurrences as new_occ
FROM comparison_duplicates d1
JOIN comparison_duplicates d2 ON d1.id = d2.id
WHERE d1.origin = 'ref' AND d2.origin = 'new';

-- 3. Claves con PARTIAL MATCH (al menos una columna difiere)
SELECT DISTINCT id
FROM comparison_differences
WHERE results = 'NO_MATCH';

-- 4. Verificar si hay claves con valores vacíos
SELECT * FROM comparison_differences
WHERE id = 'NULL'
LIMIT 100;

-- 5. Resumen de resultados por etiqueta
SELECT results, COUNT(*) as cnt
FROM comparison_differences
GROUP BY results
ORDER BY cnt DESC;
```

5.3 Problemas Comunes y Soluciones

Problema	Causa Probable	Solución
Error: "Task not serializable"	Tabla usa HiveTableScan	Recrear tabla con USING parquet
Error: "UNRESOLVED_COLUMN"	Columnas diferentes entre REF/NEW	Verificar schema, añadir a ignoreCols
Muchos id="NULL"	Claves vacías concentradas	Rellenar keys en ingesta o filtrar
Diferencias en espacios	"ES " vs "ES"	Normalizar con TRIM() antes de comparar
Performance lento	LIKE '%pattern%' en filtros	Usar LIKE 'pattern%' o partitionSpec
Duplicados altos	Cargas repetidas upstream	Deduplicar antes de comparar

5.4 Interpretación de Logs

` [SCHEMA] REF cols=120 | NEW cols=118 → REF tiene 2 columnas más que NEW

[SCHEMA] ✓ Columns only in REF (2): old_field1, deprecated_field2 → Estas columnas no se compararán (marcarán ONLY_IN_REF en differences)

[SCHEMA] Type/nullability differences (5):

- amount: type-mismatch | type decimal(10,2) vs decimal(12,4)

→ Comparación numérica seguirá funcionando, pero revisar precisión

[FILTER] ✓ Applied on 'ref': geo IN ('ES','FR') [FILTER] Rows: 1000000 → 150000 (15.00%) → Filtro SQL redujo filas a 15% (esperado)

[PARTITIONS] ✓ Filtered input files: REF=20 files, NEW=57 files → Particiones cargadas correctamente

[INFO] Excluding constant columns with SAME value on both sides: audit_version,load_date
→ Columnas constantes excluidas automáticamente (reduce ruido) `

PARTE II: REFERENCIA TÉCNICA

---

6. Arquitectura del Motor

6.1 Flujo de Ejecución

```
Main.scala → TableComparatorApp → TableComparisonController
├─ 1. Configuración Inicial
│  ├─ Parseo argumentos KV
│  ├─ Habilitación DataSource readers (anti-serialization)
│  └─ Validación tablas destino
├─ 2. Carga de Datos
│  ├─ PartitionPruning (wildcards + overrides + ventanas)
│  ├─ SchemaChecker (validación compatibilidad)
│  └─ Aplicación filtros SQL personalizados
├─ 3. Preparación
│  ├─ Normalización keys vacías → NULL
│  ├─ Exclusión columnas constantes idénticas
│  ├─ Selección columnas comparables (comunes a ambos lados)
│  └─ Reparticionamiento inteligente
├─ 4. Comparación (DiffGenerator)
│  ├─ Agregación por key (MAX/MIN/FIRST según tipo)
│  ├─ FULL OUTER JOIN null-safe
│  └─ Etiquetado: EXACT_MATCH/MATCH/NO_MATCH/ONLY_IN_*
├─ 5. Análisis Duplicados (DuplicateDetector)
│  ├─ Hash SHA256 null-safe por fila
│  ├─ Agrupación por origin + keys
│  └─ Conteo: exact_dups, variations, occurrences
├─ 6. Generación Métricas (SummaryGenerator)
│  ├─ KPIs: tamaños, intersección, gaps
│  └─ Global quality: (exact_match sin dups) / total_ref
└─ 7. Escritura Resultados
   ├─ differences (mode=Overwrite, particionado)
   ├─ duplicates (mode=Overwrite, particionado)
   └─ summary (mode=Overwrite, particionado)
```

6.2 Componentes Clave

Componente	Responsabilidad	Ubicación
Main.scala	Entry point, routing, SparkSession	internal_aml_tools/Main.scala
TableComparatorApp	Parsing args KV, construcción config	app/table_comparator/TableComparatorApp.scala
TableComparisonController	Orquestador principal	app/table_comparator/TableComparisonController.scala
PartitionPruning	Resolución wildcards, filtrado particiones	app/table_comparator/PartitionPruning.scala
SchemaChecker	Validación y logging de esquemas	app/table_comparator/SchemaChecker.scala
DiffGenerator	Lógica comparación columna por columna	app/table_comparator/DiffGenerator.scala
DuplicateDetector	Detección y análisis duplicados	app/table_comparator/DuplicateDetector.scala
SummaryGenerator	Cálculo métricas agregadas	app/table_comparator/SummaryGenerator.scala

---

7. Schema y Semántica de Salidas

7.1 Tabla result_differences

Schema:

```
id STRING -- Composite key (NULL-safe, "_" separated) column STRING -- Column name being compared value_ref STRING -- Value in reference table (formatted) value_new STRING -- Value in new table (formatted) results STRING -- Comparison result tag initiative STRING -- Label from initiativeName param data_date_part STRING -- Execution date (ISO)
```

Etiquetas de Resultados:

Tag	Significado	Cuándo aparece
EXACT_MATCH	**Todas** las columnas idénticas	Registro existe en ambos lados y todos los valores coinciden
MATCH	Columna específica idéntica	Registro en ambos lados, valor columna coincide
NO_MATCH	Columna específica difiere	Registro en ambos lados, valor columna difiere
ONLY_IN_REF	Solo en tabla referencia	Registro o columna solo existe en REF
ONLY_IN_NEW	Solo en tabla nueva	Registro o columna solo existe en NEW

Cómo se Calcula (Vista Funcional)

Paso 1: Normalización de Keys Vacías ` scala // DiffGenerator.scala línea 115 // Keys vacías → NULL (permite null-safe equality en joins) normalizeKeysToNull(df, compositeKeyCols) `

- Valores vacíos en columnas clave → **NULL**
- Todas las filas sin key → agrupadas bajo id="NULL"
- Afecta intersección y denominadores en summary

Paso 2: Exclusión Automática de Columnas Constantes ` scala // DiffGenerator.scala líneas 190-215 // Calcula countDistinct en una sola agregación por lado constantStats(df, candidateCols) // Excluye columnas con <=1 valor distinto Y mismo valor en ambos lados `

- Si columna tiene **mismo valor constante** en REF y NEW → **excluida automáticamente**
- Evita ruido en tablas anchas
- **Log:** [INFO] Excluding constant columns with SAME value on both sides: col1,col2

Paso 3: Política de Prioridad (Opcional) ` scala // DiffGenerator.scala líneas 148-155 // Si priorityCol definido: top-1 por key ordenado desc_nulls_last preOrderByPriority(df, keys, config) `

- Si priorityCol configurado → **solo se mantiene fila con mayor prioridad** por key

- Estabiliza resultado ante duplicados operacionales

- Duplicados siguen visibles en result_duplicates

Paso 4: Valor Representativo por Key

Cuando una key aparece múltiples veces, se elige **un valor por columna** para comparar:

Tipo de Dato	Estrategia Default	Override Disponible
Numéricos/Decimal	MAX	"max" , "min" , "first_non_null"
Date/Timestamp	MAX	"max" , "min" , "first_non_null"
Boolean	MAX	"max" , "min" , "first_non_null"
String	MAX (orden natural)	"max" , "min" , "first_non_null"
Map	MAX (después de ordenar entries y to_json)	-
Array	MAX (to_json, orden importa)	-
Struct	MAX (to_json)	-
Binary	MAX (base64 encoding)	-

` scala // DiffGenerator.scala líneas 220-235 // Ejemplo: config.aggOverrides = Map("amount" -> "min", "status" -> "first") `

Paso 5: Política de Nulls en Keys ` scala // CompareConfig.scala nullKeyMatches: Boolean = true // Default

// DiffGenerator.scala línea 253 // Join condition: if (nullKeyMatches) left <=> right // NULL == NULL else (left.isNotNull && right.isNotNull && left === right) // NULL != NULL `

Paso 6: Formato Fiel de Valores ` scala // DiffGenerator.scala líneas 23-30 formatValue(column, dataType) `

- **Decimals:** preservan escala para display (1.000000000000000001)

- **Comparación numérica** (no textual): 1.0 == 1.00
- **Nulls/vacíos**: mostrados como "-" para legibilidad

Reglas de Comparación por Tipo (Implementación Real)

Tipo	Canonicalización	Comparación	Observaciones
Numeric/Decimal	Sin cambio	Value equality	1.0 == 1.00 ; 100.50 ≠ 100.49
String	when(isNull, null).otherwise(cast)	Case-sensitive	Espacios cuentan: "ESSP" ≠ "ES"
Date/Timestamp	Sin cambio	Exact equality	-
Boolean	Sin cambio	Exact equality	-
Map	array_sort(map_entries) → to_json	JSON string	**Orden keys NO importa**
Array	to_json	JSON string	**Orden SÍ importa**
Struct	to_json	JSON string	Field-by-field via JSON
Binary	encode(base64)	String equality	Encoded as base64

` scala // DiffGenerator.scala líneas 32-48 (canonicalize function) `

Ejemplos de Comportamiento:

` sql -- Caso 1: Whitespace en strings id=2, column=country value_ref="ES " (con espacio) value_new="ES" results=NO_MATCH

-- Caso 2: Solo en un lado id=3 presente solo en REF → Genera N filas (una por cada columna comparada): id=3, column=amount, value_ref=150.00, value_new="-", results=ONLY_IN_REF id=3, column=country, value_ref=MX, value_new="-", results=ONLY_IN_REF ...

-- Caso 3: Keys vacías agregadas id="NULL" (varias filas con key vacía en ambos lados) → Agregación puede resultar en MATCH si valor representativo coincide → Variaciones internas visibles en result_duplicates `

Cómo Leer Efectivamente:

1. **Filtrar diferencias reales:**

```sql SELECT * FROM result_differences WHERE results NOT IN ('MATCH', 'EXACT_MATCH') ORDER BY id, column ```

1. **Investigar ONLY_IN_* masivos:**
    - Verificar partitionSpec (filtrado correcto)
    - Revisar keys vacías concentradas en id="NULL"

1. **Key con MATCH pero sospecha de variaciones:**

```sql SELECT * FROM result_duplicates WHERE id = '' ```

EXACT_MATCH vs MATCH: ```scala // DiffGenerator.scala líneas 64-71 // Si TODAS las columnas coinciden → 1 fila EXACT_MATCH // En lugar de N filas MATCH (una por columna) ```

- Reduce volumen: 100 columnas idénticas → **1 fila** en vez de 100
- Facilita identificación de registros perfectos

7.2 Tabla result_duplicates

Mide la **calidad de unicidad** de cada identificador en ambos universos.

Schema: ```origin STRING -- "ref" | "new" id STRING -- Composite key (NULL-safe) exact_duplicates STRING -- Count of rows with identical hash dupes_w_variations STRING -- Count of distinct hashes - 1 (max 0) occurrences STRING -- Total rows for this key variations STRING -- "col: [v1,v2] | col2: [v3,v4]" initiative STRING -- Label from initiativeName data_date_part STRING -- Execution date (ISO) ```

Cómo se Genera (Implementación Real)

```scala // DuplicateDetector.scala
```

```
// 1. Une ambas tablas con columna _src ("ref" | "new") unionWithOrigin(refDf, newDf)

// 2. Aplica priorityCol si configurado (top-1 por _src+keys) applyPriorityIf(config, df, keys)

// 3. Calcula hash SHA256 null-safe por fila withRowHash(df) // Excluye columna _src del hash // Hash = sha2(concat_ws("||", col1_or_"__NULL__", col2_or_"__NULL__", ...))

// 4. Agrupa por (origin + keys) groupBy(_src, compositeKeys) .agg( count(*) as occurrences, count(*) - countDistinct(_row_hash) as exact_duplicates, greatest(0, countDistinct(_row_hash) - 1) as dupes_w_variations, array_sort(collect_set(col)) as col_set // Por cada col ) .filter(occurrences > 1) // Solo keys duplicadas

// 5. Formatea variations: "col: [v1,v2] | col2: [v3,v4]" // Excluye token "__NULL__" del output `
```

Interpretación por Origen:

origin	Significado	Acción sugerida
ref	Duplicados **solo** en tabla histórica	Revisar procesos upstream REF
new	Duplicados **solo** en tabla candidata	Revisar procesos upstream NEW
Ambos	Mismo ID duplicado en REF **y** NEW	Problema sistémico, corregir en ambos flujos

Diagnóstico Rápido:

- **exact_duplicates** **alto** → Copias exactas (reprocesos, cargas duplicadas)
- Acción: Deduplicar antes de comparar
- **dupes_w_variations** **alto** → Key reescrita con valores diferentes
- Acción: Definir reglas consolidación, usar priorityCol

Ejemplo Real (Extracto):

origin	id	exact_dup	dupes_w_var	occ	variations	
ref	5	0	1	2	amount: [300.00,300.50]	
ref	NULL	0	1	2	amount: [60.00,61.00]	
new	NULL	2	1	4	amount: [60.00,61.00]	
new	6	1	1	3	amount: [400.00,400.10]	
ref	4	0	1	2	country: [BR,FR] \	amount: [200.00,201.00]
new	4	2	1	4	amount: [200.00,201.00]	

Interpretación:

- **exact_dup > 0**: Hay *x* filas con hash idéntico (copias exactas)
- **dupes_w_var > 0**: Existen al menos 2 hashes diferentes para este ID (alguna columna cambia)

Casos Comunes:

Situación	exact_dup	var_dup	Ejemplo
2 filas idénticas	1	0	amount `todo igual`
2 idénticas + 1 variación	1	1	amount `400.00 vs 400.10`
2 filas diferentes	0	1	300.00 vs 300.50
Fila única (no dup)	0	0	Sin duplicados
3 idénticas + 1 diferente	2	1	Mix de copias y variación

7.3 Tabla result_summary

Panel de KPIs a nivel de key construido desde las 3 salidas. Responde en segundos: tamaños, intersección, gaps, duplicados y **calidad global**.

Schema: ` block STRING -- Familia de métrica (KPIS, EXACT MATCH, PARTIAL MATCH, GAP, DUPS) metric STRING -- Descripción legible de lo que se cuenta universe STRING -- Scope de cálculo (REF, NEW, BOTH, ROWS) numerator STRING -- Cantidad principal denominator STRING -- Referencia para % (si aplica, sino "-") pct STRING -- Porcentaje formateado con 4 decimales samples STRING -- IDs de muestra para inspección rápida (ordenados) initiative STRING -- Label from initiativeName data_date_part STRING -- Execution date (ISO) `

Cómo se Calcula Cada Bloque:

` scala // SummaryGenerator.scala líneas 40-120

// 1. KPIS Block val idsRef = refDf.select(buildCid(keys)).distinct() // Unique IDs REF val idsNew = newDf.select(buildCid(keys)).distinct() // Unique IDs NEW val idsBoth = idsRef.intersect(idsNew) // Intersection

val totalRowsRef = refDf.count() // Total rows (con duplicados) val totalRowsNew = newDf.count() val totalDiff = totalRowsNew - totalRowsRef val totalDiffPct = totalDiff / totalRowsRef * 100

// 2. EXACT MATCH / PARTIAL MATCH (universe = BOTH) val diffAgg = diffDf.groupBy("id") .agg(max(when(results === "no_match", 1).otherwise(0)) as has_nm, max(when(results.isin("only_in_ref","only_in_new"), 1)) as has_only) .withColumn("has_diff", greatest(has_nm, has_only))

val idsVariations = diffAgg.filter(has_diff === 1).intersect(idsBoth) val idsExact = idsBoth.except(idsVariations)

// 3. GAP Block val idsOnlyRef = idsRef.except(idsNew) // 1:0 val idsOnlyNew = idsNew.except(idsRef) // 0:1

// 4. DUPS Block val dupRef = refDf.groupBy(cid).count().filter(count > 1) val dupNew = newDf.groupBy(cid).count().filter(count > 1) val dupBoth = dupRef.intersect(dupNew) val dupOnlyRef = dupRef.except(dupNew) val dupOnlyNew = dupNew.except(dupRef)

// 5. Global Quality val anyDup = dupRef.union(dupNew).distinct() val qualityOk = idsExact.except(anyDup).count() val qualityPct = qualityOk / nRefIds * 100 `

Bloques y Métricas:

Block	Metric	Universe	Numerator	Denominator	Fórmula %
KPIS	Unique IDs	REF/NEW	distinct(keys)	-	-
KPIS	Total rows	ROWS	count(*)	-	-
KPIS	Total diff(new-ref)	ROWS	NEW - REF	REF	(NEW-REF)/REF*100
KPIS	Global quality	REF	exact_match sin dups	Unique IDs REF	OK/REF*100
EXACT MATCH	1:1 (all columns)	BOTH	idsExact	idsBoth	exact/both*100
PARTIAL MATCH	1:1 (match & no_match cols)	BOTH	idsVariations	idsBoth	var/both*100
GAP	1:0 (only in ref)	REF	idsOnlyRef	nRefIds	only/ref*100
GAP	0:1 (only in new)	NEW	idsOnlyNew	nNewIds	only/new*100
DUPS	duplicates (both)	BOTH	dupBoth	idsBoth	dup/both*100
DUPS	duplicates (only in ref)	REF	dupOnlyRef	nRefIds	dup/ref*100
DUPS	duplicates (only in new)	NEW	dupOnlyNew	nNewIds	dup/new*100

Notas sobre Denominadores:

- universe=REF/NEW → denominador - (excepto Total diff y Global quality)
- universe=BOTH → denominador = **keys en intersección**
- universe=ROWS → conteo de filas físicas
- Porcentaje formato: "XX.XXXX%" (4 decimales), "-" si denominador=0

Tips de Lectura:

1. **% alto PARTIAL MATCH en BOTH** → revisar normalizaciones (espacios, mayúsculas), reglas de agregación

2. **Key con variations en duplicates** → ese ID **NO suma** al numerador de Global quality
3. **Drill-down rápido:**

` sql SELECT * FROM result_differences WHERE results NOT IN ('MATCH', 'EXACT_MATCH') ORDER BY id, column `

---

8. Optimizaciones y Performance

8.1 Gestión de Serialización (Crítico en PRE)

Problema: Tablas creadas con Hive SerDe causan Task not serializable en entornos distribuidos.

Solución implementada: ` scala // TableComparisonController.scala // 1. Forzar DataSource readers spark.conf.set("spark.sql.hive.convertMetastoreParquet", "true") spark.conf.set("spark.sql.hive.convertMetastoreOrc", "true")

// 2. Limpiar caché metastore spark.catalog.clearCache()

// 3. Validar plan físico assertFileSource(df, label) // Falla si detecta HiveTableScanExec `

Si ves este error: ` Exception: Task not serializable Physical plan contains: HiveTableScanExec `

Solución: ` sql -- Recrear tabla como DataSource Parquet DROP TABLE IF EXISTS default.result_differences; CREATE TABLE default.result_differences (id STRING, column STRING, value_ref STRING, value_new STRING, results STRING, initiative STRING, data_date_part STRING) USING parquet PARTITIONED BY (initiative, data_date_part) LOCATION 's3a://bucket/path/differences'; `

8.2 Reparticionamiento Inteligente

` scala // PrepUtils.scala def pickTargetPartitions(spark: SparkSession): Int = { val base = spark.sparkContext.defaultParallelism base * 2 // 2x parallelism para mejor utilización }

// Reparticiona por composite keys antes de comparar df.repartition(nParts, compositeKeyCols.map(col): _*) `

Caché estratégico: ` scala refDf.persist(StorageLevel.MEMORY_AND_DISK) newDf.persist(StorageLevel.MEMORY_AND_DISK)

`// Liberación explícita df.unpersist(blocking = true)`

Coalesce en salidas: `scala out.coalesce(1) // 1 archivo por partición .write.mode(SaveMode.Overwrite) .insertInto(tableName)`

8.3 Políticas de Null Handling

Keys vacías → NULL: `scala // DiffGenerator.scala when(trim(col(key).cast(StringType)) === "", lit(null)) .otherwise(col(key))`

Null-safe equality en joins: `scala // config.nullKeyMatches = true (default) left_key <=> right_key // NULL == NULL`

Display de nulls: `scala when(col.isNull || trim(col) === "", lit("-")) .otherwise(col.cast(StringType))`

Hash null-safe: `scala // DuplicateDetector.scala coalesce(col(c).cast(StringType), lit("__NULL__"))`

8.4 Exclusión Automática de Columnas Constantes

`scala // DiffGenerator.scala // Calcula countDistinct en AMBOS lados constantStats(refDf, candidateCols) constantStats(newDf, candidateCols)

// Excluye si: // - countDistinct(col) <= 1 en REF // - countDistinct(col) <= 1 en NEW // - Valor representativo idéntico`

Log ejemplo: `[INFO] Excluding constant columns with SAME value on both sides: audit_version,load_date`

Rationale: Evita ruido en tablas anchas (100+ columnas con valores fijos).

---

9. Limitaciones y Buenas Prácticas

9.1 Limitaciones Conocidas

Limitación	Workaround
aggOverrides no configurable vía KV	Configurar en código Scala
priorityCol no via KV	Configurar en código Scala
exportExcelPath no vía KV	Configurar en código Scala
Max 256 fechas en resolución wildcards	Dividir en múltiples ejecuciones
Schema mismatch no bloquea ejecución	Revisar logs antes de interpretar
Columnas partition en comparación	Agregar a ignoreCols

9.2 Buenas Prácticas

1. **Validar particiones antes de ejecutar:** ` sql SHOW PARTITIONS default.ref_table; -- Confirmar que existen las particiones esperadas `

2. **Usar checkDuplicates=true en primera ejecución:**

- Identifica problemas de calidad upstream
- Puedes desactivar después si no es necesario

3. **Monitorear Global quality en CI:** ` sql SELECT numerator, denominator, pct FROM result_summary WHERE metric = 'Global quality' AND pct < '95.0000%'; -- Alerta si < 95% `

4. **Filtrar coincidencias para análisis:** ` sql SELECT * FROM result_differences WHERE results NOT IN ('MATCH', 'EXACT_MATCH') `

5. **Recrear tablas destino como DataSource:** ` sql DROP TABLE IF EXISTS default.result_differences; CREATE TABLE ... USING parquet ... `

6. **Combinar filtros estratégicamente:** ` bash

Filtro grueso: particiones

partitionSpec="data_date_part=2025-10-01/"

Filtro fino: SQL

refFilter="time LIKE '06:%' AND amount >= 1000" `

7. Usar ventanas temporales para comparaciones históricas: ` bash refWindowDays=-7..0 # 7 días históricos newWindowDays=0..+1 # Hoy + mañana `

9.3 Ejemplo Completo de Producción

` bash #!/bin/bash

compare_tables.sh

SPARK_HOME=/opt/spark JAR=s3a://artifacts/cib-adhc-internaltools-1.0.5-SNAPSHOT.jar

$SPARK_HOME/bin/spark-submit \ --master yarn --deploy-mode cluster \ --conf spark.sql.adaptive.enabled=true \ --conf spark.sql.adaptive.coalescePartitions.enabled=true \ --conf spark.dynamicAllocation.enabled=true \ --conf spark.sql.hive.convertMetastoreParquet=true \ --driver-memory 4g \ --executor-memory 8g \ --executor-cores 4 \ --num-executors 10 \ --class com.santander.cib.adhc.internal_aml_tools.Main \ $JAR \ refTable=default.aml_transactions_ref \ newTable=default.aml_transactions_new \ compositeKeyCols=transaction_id,customer_id \ partitionSpec="geo=ES|PT/data_date_part=2025-11-19/" \ ignoreCols=ingestion_ts,audit_user,version \ initiativeName=AML_Q4_Migration \ tablePrefix=default.aml_cmp_ \ outputBucket=s3a://scib-pre-bu-aml/comparisons \ executionDate=2025-11-19 \ checkDuplicates=true \ includeEqualsInDiff=false `

Verificación post-ejecución: ` sql -- 1. Ver resumen SELECT block, metric, numerator, denominator, pct FROM default.aml_cmp_summary WHERE initiative = 'AML_Q4_Migration' ORDER BY block, metric;

-- 2. Columnas problemáticas SELECT column, COUNT(*) as cnt FROM default.aml_cmp_differences WHERE results = 'NO_MATCH' GROUP BY column ORDER BY cnt DESC LIMIT 20;

-- 3. Duplicados críticos SELECT * FROM default.aml_cmp_duplicates WHERE dupes_w_variations > 0 ORDER BY CAST(occurrences AS INT) DESC LIMIT 50; `

9.4 Referencias del Código Fuente

Componente	Archivo	Responsabilidad
Entry point	Main.scala	Routing y SparkSession
Parser args	TableComparatorApp.scala	Construcción config
Orquestador	TableComparisonController.scala	Flujo principal
Particiones	PartitionPruning.scala	Wildcards y filtrado
Schemas	SchemaChecker.scala	Validación
Comparación	DiffGenerator.scala	Lógica columna por columna
Duplicados	DuplicateDetector.scala	Análisis y hash
Métricas	SummaryGenerator.scala	KPIs agregados

Stack tecnológico:

- Scala: 2.12.17
- Spark: 3.5.0
- Maven: Build management
- Log4j: 2.17.1

9.5 Preguntas Frecuentes (FAQ)

P: ¿Puedo comparar tablas con esquemas diferentes? R: ✅ Sí. El motor compara automáticamente solo las columnas comunes. Las columnas únicas aparecen como `ONLY_IN_REF` o `ONLY_IN_NEW` en la tabla differences.

P: ¿Cómo manejo duplicados en las claves? R: Activa `checkDuplicates=true` para detectarlos. Usa `priorityCol` para desempate automático: ` bash priorityCol=update_timestamp # Mantiene fila con timestamp más alto priorityCol=version_number # Mantiene versión más reciente ` Ver sección 3.2.5 para detalles completos.

P: ¿Qué significa "Global Quality < 95%"? R: Menos del 95% de las claves tienen coincidencia exacta sin duplicados. Investiga con: ` sql SELECT * FROM differences WHERE results = 'NO_MATCH' SELECT * FROM duplicates WHERE dupes_w_variations > 0 `

P: ¿Por qué veo `id="NULL"` en los resultados? R: Claves vacías se normalizan a NULL y se agrupan. Solución: rellenar keys en la ingesta o añadir filtro: ` bash refFilter="key_column IS NOT NULL" `

P: ¿Cómo filtro solo transacciones de una hora específica? R: Usa filtros SQL personalizados: `bash refFilter="time LIKE '06:%'" # Solo hora 06:00-06:59`

P: ¿El motor bloquea la ejecución si los esquemas no coinciden? R: ❌ No. Solo genera logs de advertencia y compara columnas comunes. Revisa [SCHEMA] logs antes de interpretar resultados.

P: ¿Cómo optimizo comparaciones de tablas muy grandes (TB)? R:

- Usa partitionSpec para filtrar particiones (más rápido que SQL)
- Activa spark.sql.adaptive.enabled=true
- Aumenta executor-memory y num-executors
- Considera dividir en múltiples ejecuciones por rango de fechas

P: ¿Por qué columnas constantes no aparecen en differences? R: Se excluyen automáticamente si tienen el mismo valor en REF y NEW. Log: `[INFO] Excluding constant columns with SAME value on both sides: audit_version`

P: ¿Puedo exportar el summary a Excel? R: Sí, pero requiere configuración en código Scala (no disponible vía KV args): `scala val config = CompareConfig(..., exportExcelPath = Some("s3a://bucket/summary.xlsx"))`

P: Error "Task not serializable" ¿qué hago? R: La tabla usa HiveTableScan. Recréala como DataSource Parquet: `sql DROP TABLE IF EXISTS your_table; CREATE TABLE your_table (...) USING parquet LOCATION 's3a://...';`

9.6 Tarjeta de Referencia Rápida

¿Quiero...?	Parámetro o Query
Filtrar por fecha específica	partitionSpec="data_date_part=2025-11-19/"
Filtrar por múltiples geos	partitionSpec="geo=[ES,PT,FR]/..."
Todos los geos disponibles	partitionSpec="geo=*/..." `(resuelve automático)`
Filtrar por SQL (valores)	refFilter="amount >= 1000 AND status='ACTIVE'"
Comparar ventanas temporales	refWindowDays=-7..0 newWindowDays=0..+1
Ver solo diferencias	SELECT * FROM differences WHERE results='NO_MATCH'
Ver calidad global	SELECT * FROM summary WHERE metric='Global quality'
Top columnas problemáticas	SELECT column, COUNT(*) FROM differences WHERE results='NO_MATCH' GROUP BY column
Duplicados críticos	SELECT * FROM duplicates WHERE dupes_w_variations > 0
Duplicados por categoría	SELECT * FROM duplicates WHERE category='both'
Excluir columnas	ignoreCols=ingestion_ts,audit_user,version
Detectar duplicados	checkDuplicates=true
Resolver duplicados automático	priorityCol=update_timestamp `(mantiene más reciente)`
No incluir coincidencias	includeEqualsInDiff=false `(default)`

Comandos útiles: ` bash

Ver particiones disponibles antes de ejecutar

```
SHOW PARTITIONS default.your_table;
```

Verificar schema de tablas

```
DESCRIBE default.your_table;
```

Contar resultados por tipo

```
SELECT results, COUNT(*) FROM differences GROUP BY results;
```

Verificar calidad por iniciativa

```
SELECT initiative, metric, pct FROM summary WHERE metric='Global quality';
```

📚 **Última actualización:** 2025-11-21 📦 **Versión documento:** 3.2 (con análisis detallado de duplicados y priorityCol)