Contents

□ DOCUMENTO EJECUTIVO COMPLETO

CompareTables - Motor de Comparación de Tablas Spark

□ ÍNDICE

□ RESUMEN EJECUTIVO

¿Qué es CompareTables?

CompareTables es un **motor de comparación de tablas empresarial** desarrollado en Apache Spark que permite:

- **Comparar automáticamente** dos conjuntos de datos (referencia vs. nuevo)
- **Detectar diferencias** a nivel de fila y columna
- **Identificar duplicados** exactos y con variaciones
- **Generar métricas de calidad** de datos
- **Producir reportes ejecutivos** en formato tabular y Excel

Valor de Negocio

Beneficio	Descripción
Auditoría de Datos	Verificar integridad en migraciones de sistemas
Control de Calidad	Detectar inconsistencias y duplicados
Compliance	Documentar diferencias para auditorías
Eficiencia Operativa	Automatizar comparaciones manuales
Toma de Decisiones	Métricas claras para evaluar calidad de datos

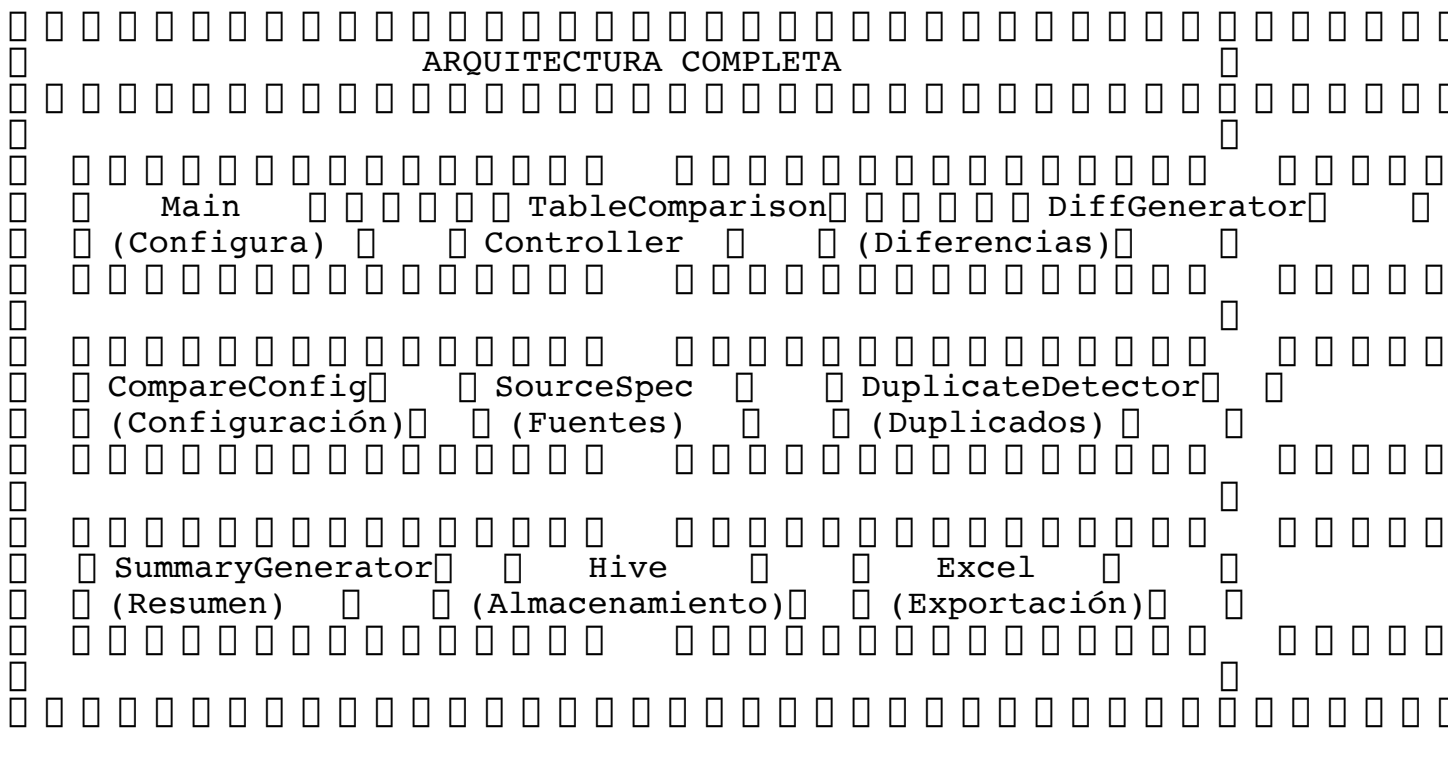
Casos de Uso Principales

1. **Migración de Sistemas**: Comparar datos antes y después de una migración
2. **Reconciliación**: Verificar consistencia entre sistemas operativos
3. **Control de Calidad**: Detectar anomalías en procesos ETL
4. **Auditoría**: Documentar diferencias para cumplimiento regulatorio

---

□ ARQUITECTURA DEL SISTEMA

Componentes Principales



Flujo de Datos

1. **Configuración**: Main.scala establece parámetros y fuentes de datos
2. **Control**: TableComparisonController orquesta todo el proceso
3. **Comparación**: DiffGenerator detecta diferencias entre datasets
4. **Duplicados**: DuplicateDetector identifica filas duplicadas
5. **Resumen**: SummaryGenerator consolida métricas y KPIs
6. **Salida**: Resultados en Hive + Excel para análisis

□ FUNCIONAMIENTO DETALLADO

Fase 1: Preparación de Datos

```
// 1. Carga de fuentes (Hive, CSV, Parquet)
val rawRef = loadSource(session, refSource, partitionSpec)
val rawNew = loadSource(session, newSource, partitionSpec)

// 2. Filtrado de columnas
val colsToCompare = rawRef.columns.toSeq
  .filterNot(ignoreCols.contains)        // Excluye columnas ignoradas
  .filterNot(partitionKeys.contains)     // Excluye claves de partición
  .filterNot(compositeKeyCols.contains)  // Excluye claves compuestas

// 3. Reparticionamiento para rendimiento
val refDf = rawRef
  .select(neededCols.map(col): _*)
  .repartition(100, compositeKeyCols.map(col): _*)
  .persist(StorageLevel.MEMORY_AND_DISK)
```

¿Qué hace? - Carga datos desde múltiples fuentes (Hive, archivos) - Aplica filtros de partición automáticamente - Optimiza rendimiento con reparticionamiento inteligente - Mantiene datos en memoria para operaciones repetitivas

Fase 2: Generación de Diferencias

```
// 1. Normalización de claves
val norm: Column => Column = c =>
  when(trim(c.cast(StringType)) === "", lit(null)).otherwise(c)

// 2. Agregación por claves compuestas
val aggs: Seq[Column] = compareCols.map { c =>
  val dt = refBase.schema(c).dataType
  val canon = canonicalize(col(c), dt)
  config.aggOverrides.get(c) match {
    case Some(MaxAgg) => max(canon.cast(dt)).as(c)
    case Some(MinAgg) => min(canon.cast(dt)).as(c)
    case None => max(canon).as(c)
  }
}

// 3. Full Outer Join con política de NULLs
val joinCond = compositeKeyCols.map { k =>
  val l = col(s"ref.$k"); val r = col(s"new.$k")
  if (config.nullKeyMatches) l <=> r else (l.isNotNull && r.isNotNull &&
}.reduce(_ && _)
```

¿Qué hace? - Normaliza claves vacías a NULL para joins robustos - Agrupa filas duplicadas por claves compuestas - Aplica estrategias de agregación configurables - Realiza join completo para

detectar todos los casos

Fase 3: Detección de Duplicados

```
// 1. Unión y etiquetado de origen
val withSrc = refDf.withColumn("_src", lit("ref"))
  .unionByName(newDf.withColumn("_src", lit("new")))

// 2. Cálculo de hash por fila
val hashCol = sha2(
  concat_ws("§", base.columns.filter(_ != "_src").map { c =>
    coalesce(col(c).cast(StringType), lit("__NULL__"))
  }: _*),
  256
)

// 3. Agregación y análisis
val aggExprs = Seq(
  count(lit(1)).as("occurrences"),
  (count(lit(1)) - countDistinct("_row_hash")).as("exact_dup"),
  greatest(lit(0), countDistinct("_row_hash") - lit(1)).as("var_dup")
)
```

¿Qué hace? - Combina ambos datasets con etiquetas de origen - Calcula hash único por fila para identificación de duplicados - Distingue entre duplicados exactos y con variaciones - Proporciona contexto detallado de las variaciones

Fase 4: Generación de Resumen

```
// 1. Análisis de presencia
val idsRef = refDf.select(cid.as("cid")).distinct()
val idsNew = newDf.select(cid.as("cid")).distinct()
val idsBoth = idsRef.intersect(idsNew)
val idsOnlyR = idsRef.except(idsNew)
val idsOnlyN = idsNew.except(idsRef)

// 2. Cálculo de métricas de calidad
val qualityIds = idsExact.except(dupIdsAny)
val qualityOk = qualityIds.count()

// 3. Construcción de filas de resumen
val rows = Seq(
  row("KPIS", "IDs Uniques", refTableName, nRefIds, 0, ""),
  row("MATCH", "1:1 (exact matches)", "BOTH", idsExact.count(), nBothIds
  row("GAP", "1:0 (only in reference)", refTableName, idsOnlyR.count(),
)
```

¿Qué hace? - Calcula métricas de cobertura y calidad - Identifica patrones de datos (gaps, duplicados, variaciones) - Proporciona ejemplos para facilitar investigación - Genera KPIs ejecutivos

para toma de decisiones

□ TABLAS DE RESULTADO

1. Tabla DIFFERENCES (Diferencias)

Propósito: Detectar diferencias a nivel de columna entre datasets

Estructura:

```
CREATE TABLE result_differences (
  id STRING,            -- Clave compuesta del registro
  `column` STRING,      -- Nombre de la columna comparada
  value_ref STRING,     -- Valor en dataset de referencia
  value_new STRING,     -- Valor en dataset nuevo
  results STRING        -- Resultado de la comparación
)
PARTITIONED BY (initiative STRING, data_date_part STRING)
```

Códigos de Resultado: | Código | Significado | Ejemplo | |——–|————-|———| | MATCH | Valores idénticos | `country: US = US` | | NO_MATCH | Valores diferentes | `country: FR ≠ BR` | | ONLY_IN_REF | Solo existe en referencia | `country: MX vs -` | | ONLY_IN_NEW | Solo existe en nuevo | `country: - vs DE` |

Ejemplo de Datos:

```
| id | column   | value_ref | value_new | results     |
|----|----------|-----------|-----------|-------------|
| 1  | country  | US        | US        | MATCH       |
| 4  | country  | FR        | BR        | NO_MATCH    |
| 3  | country  | MX        | -         | ONLY_IN_REF |
| 6  | country  | -         | DE        | ONLY_IN_NEW |
| 2  | amount   | 1.00000000000000001 | 1.00000000000000001 | MATCH |
```

2. Tabla DUPLICATES (Duplicados)

Propósito: Identificar filas duplicadas y sus variaciones

Estructura:

```
CREATE TABLE result_duplicates (
  origin STRING,           -- Origen: "ref" o "new"
  id STRING,               -- Clave compuesta del registro
  exact_duplicates STRING, -- Número de duplicados exactos
  dups_w_variations STRING, -- Número con variaciones
  occurrences STRING,      -- Total de ocurrencias
  variations STRING        -- Detalle de las variaciones
)
PARTITIONED BY (initiative STRING, data_date_part STRING)
```

Interpretación de Campos: - **exact_duplicates**: Filas idénticas (mismo hash) - **dups_w_variations**: Filas con misma clave pero valores diferentes - **occurrences**: Total de filas para esa clave - **variations**: Descripción de las diferencias encontradas

Ejemplo de Datos:

```
| origin           | id  | exact_duplicates | dups_w_variations | occurrence
|----------------------|-----|------------------|-------------
------|-------------|------------------------------|
| default.ref_customers| 1   | 1                | 0                 | 2         | -
|
| default.ref_customers| 4   | 0                | 1                 | 2         | countr
| default.new_customers| 4   | 2                | 1                 | 4         | amount
| default.new_customers| 6   | 2                | 1                 | 3         | amount
```

3. Tabla SUMMARY (Resumen)

Propósito: Consolidar métricas ejecutivas y KPIs de calidad

Estructura:

```sql
CREATE TABLE result_summary (
  bloque STRING,          -- Categoría de métrica
  metrica STRING,         -- Nombre específico de la métrica
  universo STRING,        -- Contexto de la medición
  numerador STRING,       -- Valor medido
  denominador STRING,     -- Base de cálculo
  pct STRING,            -- Porcentaje calculado
  ejemplos STRING        -- IDs de ejemplo para investigación
)
PARTITIONED BY (initiative STRING, data_date_part STRING)
```

Ejemplo de Datos:

```
| bloque  | metrica              | universo | numerador | denominador | pct
|---------|----------------------|----------|-----------|-
------------|--------|----------|
| KPIS    | IDs Uniques          | REF      | 10        | -
| -       | -        |
| KPIS    | IDs Uniques          | NEW      | 8         | -
| -       | -        |
| KPIS    | Total REF            | ROWS     | 13        | -
| -       | -        |
| KPIS    | Total NEW            | ROWS     | 16        | -
| -       | -        |
| KPIS   | Total (NEW-REF)       | ROWS    | 3       | 13       | 23.1%  | -
|
| KPIS   | Quality global        | REF     | 5       | 10       | 50.0%  | -
|
| MATCH   | 1:1 (exact matches)     | BOTH    | 2       | 7        | 28.6%  | 1,
| NO MATCH| 1:1 (match not identical)| BOTH    | 5       | 7        | 71.4%  |
```

| GAP | 1:0 (only in reference) | REF | 3 | 10 | 30.0% | 10
| GAP | 0:1 (only in new) | NEW | 1 | 8 | 12.5% | 6
| DUPS | duplicates (both) | BOTH | 2 | 7 | 28.6% | 4,N
| DUPS | duplicates (ref) | REF | 1 | 10 | 10.0% | 1
| DUPS | duplicates (new) | NEW | 2 | 8 | 25.0% | 4,6

□ INTERPRETACIÓN DE MÉTRICAS

Bloque KPIS (Indicadores Clave de Rendimiento)

IDs Únicos

- **REF**: Número de identificadores únicos en el dataset de referencia
- **NEW**: Número de identificadores únicos en el dataset nuevo
- **Interpretación**: Muestra la cobertura de datos en cada sistema

Total de Filas

- **REF**: Número total de filas en el dataset de referencia
- **NEW**: Número total de filas en el dataset nuevo
- **Total (NEW-REF)**: Diferencia neta de filas
- **Interpretación**: Indica crecimiento o reducción del volumen de datos

Quality Global

- **Cálculo**: IDs exactos sin duplicados / Total IDs únicos en REF
- **Interpretación**: Porcentaje de registros perfectos en el sistema de referencia
- **Objetivo**: >90% para sistemas de producción

Bloque MATCH (Coincidencias)

1:1 (exact matches)

- **Cálculo**: IDs con valores idénticos en ambos datasets
- **Denominador**: Total de IDs presentes en ambos datasets
- **Interpretación**: Registros que migraron correctamente sin cambios

1:1 (match not identical)

- **Cálculo**: IDs presentes en ambos datasets pero con valores diferentes
- **Denominador**: Total de IDs presentes en ambos datasets
- **Interpretación**: Registros que cambiaron durante la migración

Bloque GAP (Brechas)

1:0 (only in reference)

- **Cálculo**: IDs que existen solo en el dataset de referencia

- **Denominador**: Total de IDs únicos en el dataset de referencia
- **Interpretación**: Registros perdidos durante la migración

0:1 (only in new)

- **Cálculo**: IDs que existen solo en el dataset nuevo
- **Denominador**: Total de IDs únicos en el dataset nuevo
- **Interpretación**: Registros nuevos o duplicados creados

Bloque DUPS (Duplicados)

duplicates (both)

- **Cálculo**: IDs duplicados en ambos datasets
- **Denominador**: Total de IDs presentes en ambos datasets
- **Interpretación**: Problemas de duplicación persistentes

duplicates (ref/new)

- **Cálculo**: IDs duplicados solo en un dataset específico
- **Denominador**: Total de IDs únicos en ese dataset
- **Interpretación**: Problemas de duplicación específicos de cada sistema

---

□ CASOS DE USO PRÁCTICOS

Caso 1: Migración de Sistema CRM

Escenario: Migración de sistema CRM legacy a nuevo sistema cloud

Configuración:

```
val config = CompareConfig(
  refTable = "legacy.crm_customers",
  newTable = "cloud.crm_customers",
  compositeKeyCols = Seq("customer_id", "email"),
  ignoreCols = Seq("last_modified", "system_version"),
  initiativeName = "CRM_Migration_2025",
  checkDuplicates = true
)
```

Interpretación de Resultados: - **Quality Global < 80%**: Problemas críticos en migración - **GAP 1:0 > 10%**: Pérdida significativa de clientes - **DUPS > 20%**: Problemas de deduplicación en sistema legacy

Caso 2: Reconciliación Financiera

Escenario: Verificar consistencia entre sistema contable y ERP

Configuración:

```
val config = CompareConfig(
  refTable = "accounting.transactions",
  newTable = "erp.transactions",
  compositeKeyCols = Seq("transaction_id"),
  ignoreCols = Seq("created_at", "updated_at"),
  initiativeName = "Financial_Reconciliation_Q1",
  checkDuplicates = false  // No relevante para transacciones
)
```

Interpretación de Resultados: - **NO_MATCH > 5%**: Inconsistencias en montos o fechas - **ONLY_IN_REF > 0%**: Transacciones no sincronizadas - **ONLY_IN_NEW > 0%**: Transacciones duplicadas o erróneas

Caso 3: Control de Calidad ETL

Escenario: Verificar integridad de datos después de proceso ETL

Configuración:

```
val config = CompareConfig(
  refTable = "raw.customer_data",
  newTable = "processed.customer_data",
  compositeKeyCols = Seq("customer_id"),
  ignoreCols = Seq("etl_timestamp", "processing_status"),
  initiativeName = "ETL_Quality_Check_Daily",
  checkDuplicates = true
)
```

Interpretación de Resultados: - **Quality Global > 95%**: Proceso ETL funcionando correctamente - **DUPS < 5%**: Deduplicación efectiva - **GAP < 2%**: Pérdida mínima de datos

---

□ CONFIGURACIÓN Y PERSONALIZACIÓN

Parámetros de Configuración

Fuentes de Datos

```
// Hive Tables
val refSource = HiveTable("database.table_name")

// File Sources
val refSource = FileSource(
  path = "/path/to/data",
  format = "parquet",  // "csv" | "parquet"
  options = Map("header" -> "true", "delimiter" -> ","),
  schema = Some(customSchema)
)
```

Estrategias de Agregación

```
val config = CompareConfig(
  // ... otros parámetros ...
  aggOverrides = Map(
    "amount" -> MaxAgg,           // Usar valor máximo
    "status" -> FirstNonNullAgg,  // Usar primer valor no nulo
    "date" -> MinAgg              // Usar fecha más antigua
  )
)
```

Manejo de Claves NULL

```
val config = CompareConfig(
  // ... otros parámetros ...
  nullKeyMatches = true,  // true: NULL = NULL, false: NULL ≠ NULL
  includeDupInQuality = false  // Incluir duplicados en cálculo de calid
)
```

Configuración de Rendimiento

```
// En TableComparisonController
session.conf.set("spark.sql.shuffle.partitions", "100")
session.conf.set("spark.sql.sources.partitionOverwriteMode", "dynamic")
session.conf.set("hive.exec.dynamic.partition", "true")
session.conf.set("hive.exec.dynamic.partition.mode", "nonstrict")

// Reparticionamiento inteligente
.repartition(100, compositeKeyCols.map(col): _*)
.persist(StorageLevel.MEMORY_AND_DISK)
```

□ RENDIMIENTO Y ESCALABILIDAD

Optimizaciones Implementadas

1. Filtrado de Columnas Constantes

```
def isConstantColumn(df: DataFrame, colName: String): Boolean =
  df.select(col(colName)).distinct().limit(2).count() <= 1

// Excluye columnas que no aportan valor a la comparación
val consts = commonCols.filter { c =>
  baseCols.contains(c) &&
  isConstantColumn(nRef, c) &&
  isConstantColumn(nNew, c)
}
```

2. Agregación Inteligente

```
// Agrupa por claves antes de comparar
val refAgg = refBase
  .groupBy(compositeKeyCols.map(col): _*)
  .agg(aggs.head, aggs.tail: _*)
  .withColumn("_present", lit(1))
```

3. Persistencia Selectiva

```
// Mantiene en memoria solo los datos necesarios
val refDf = rawRef
  .select(neededCols.map(col): _*)
  .repartition(100, compositeKeyCols.map(col): _*)
  .persist(StorageLevel.MEMORY_AND_DISK)

// Libera memoria después de uso
refDf.unpersist()
```

Escalabilidad

Particionamiento

- **Hive**: Particionado por `initiative` y `data_date_part`
- **Spark**: Reparticionamiento inteligente por claves compuestas
- **Storage**: Parquet para compresión y consultas eficientes

Configuración de Clusters

```
// Ajustar según recursos disponibles
session.conf.set("spark.sql.shuffle.partitions", "200")  // Para cluster
session.conf.set("spark.executor.memory", "8g")          // Memoria por
session.conf.set("spark.executor.cores", "4")            // Cores por ex
```

□ MANTENIMIENTO Y OPERACIONES

Monitoreo Diario

Métricas a Verificar

1. **Tiempo de Ejecución**: Debe ser consistente
2. **Calidad de Datos**: Quality Global > 90%
3. **Volumen de Diferencias**: NO_MATCH < 10%
4. **Duplicados**: DUPS < 15%

Alertas Recomendadas

```
// Ejemplo de configuración de alertas
if (qualityGlobal < 0.9) {
  sendAlert("Data quality below threshold: " + qualityGlobal)
}
if (noMatchPercentage > 0.1) {
  sendAlert("High number of mismatches: " + noMatchPercentage)
}
```

Mantenimiento Preventivo

Limpieza de Datos

```
// Limpiar tablas de resultados antiguas
def cleanOldResults(spark: SparkSession, daysToKeep: Int): Unit = {
  val cutoffDate = LocalDate.now().minusDays(daysToKeep)
  spark.sql(s"DELETE FROM result_summary WHERE data_date_part < '$cutoff
}
```

Optimización de Particiones

```
-- Consolidar particiones pequeñas
ALTER TABLE result_differences PARTITION (initiative='Swift', data_date_
CONCATENATE;
```

Backup y Recuperación

Estrategia de Backup

```
# Backup diario de tablas de resultados
hive -e "EXPORT TABLE result_summary TO '/backup/result_summary_$(date +
hive -e "EXPORT TABLE result_differences TO '/backup/result_differences_
hive -e "EXPORT TABLE result_duplicates TO '/backup/result_duplicates_$(
```

Recuperación

```
-- Restaurar desde backup
IMPORT TABLE result_summary FROM '/backup/result_summary_20250811';
IMPORT TABLE result_differences FROM '/backup/result_differences_2025081
IMPORT TABLE result_duplicates FROM '/backup/result_duplicates_20250811'
```

□ ANEXOS TÉCNICOS

A. Estructura de Datos de Ejemplo

Dataset de Referencia (REF)

```
val ref = Seq(
  Row(1, "US", new BigDecimal("100.40"), "active"),
```

```
  Row(1, "US", new BigDecimal("100.40"), "active"),       // Duplicado ex
  Row(2, "ES ", new BigDecimal("1.0000000000000000001"), "expired"),
  Row(3, "MX", new BigDecimal("150.00"), "active"),
  Row(4, "FR", new BigDecimal("200.00"), "new"),
  Row(4, "BR", new BigDecimal("201.00"), "new"),          // Variación
  Row(5, "FR", new BigDecimal("300.00"), "active"),
  Row(5, "FR", new BigDecimal("300.50"), "active"),       // Variación
  Row(7, "PT", new BigDecimal("300.50"), "active"),
  Row(8, "BR", new BigDecimal("100.50"), "pending"),
  Row(10, "GR", new BigDecimal("60.00"), "new"),
  Row(null, "GR", new BigDecimal("61.00"), "new"),        // Clave NULL
  Row(null, "GR", new BigDecimal("60.00"), "new")         // Clave NULL d
)
```

Dataset Nuevo (NEW)

```
val nw = Seq(
  Row(1, "US", new BigDecimal("100.40"), "active"),
  Row(2, "ES", new BigDecimal("1.0000000000000000001"), "expired"),
  Row(4, "BR", new BigDecimal("201.00"), "new"),
  Row(4, "BR", new BigDecimal("200.00"), "new"),          // Duplicado co
  Row(4, "BR", new BigDecimal("200.00"), "new"),          // Duplicado ex
  Row(4, "BR", new BigDecimal("200.00"), "new"),          // Duplicado ex
  Row(6, "DE", new BigDecimal("400.00"), "new"),          // Nuevo ID
  Row(6, "DE", new BigDecimal("400.00"), "new"),          // Duplicado ex
  Row(6, "DE", new BigDecimal("400.10"), "new"),          // Variación
  Row(7, "", new BigDecimal("300.50"), "active"),         // Valor vacío
  Row(8, "BR", null, "pending"),                          // Valor NULL
  Row(null, "GR", new BigDecimal("60.00"), "new"),        // Clave NULL
  Row(null, "GR", new BigDecimal("60.00"), "new"),        // Clave NULL d
  Row(null, "GR", new BigDecimal("60.00"), "new"),        // Clave NULL d
  Row(null, "GR", new BigDecimal("61.00"), "new")         // Clave NULL c
)
```

B. Configuración de Spark

Configuración Local

```
val spark = SparkSession.builder()
  .appName("CompareTablesMain")
  .master("local[*]")
  .enableHiveSupport()
  .config("spark.sql.warehouse.dir", "./spark-warehouse")
  .config("hive.metastore.warehouse.dir", "./spark-warehouse")
  .getOrCreate()
```

Configuración de Producción

```scala
val spark = SparkSession.builder()
  .appName("CompareTablesProduction")
  .config("spark.sql.adaptive.enabled", "true")
  .config("spark.sql.adaptive.coalescePartitions.enabled", "true")
  .config("spark.sql.adaptive.skewJoin.enabled", "true")
  .config("spark.sql.adaptive.localShuffleReader.enabled", "true")
  .enableHiveSupport()
  .getOrCreate()
```

C. Comandos de Operación

Ejecución Manual

```bash
# Compilar
sbt compile

# Ejecutar
sbt run

# Ejecutar tests
sbt test

# Limpiar
sbt clean
```

Ejecución Programada

```bash
# Cron job diario
0 2 * * * cd /path/to/CompareTablesSparkScala && ./run_compare.sh

# Con logging
0 2 * * * cd /path/to/CompareTablesSparkScala && ./run_compare.sh > /var
```

□ RESUMEN EJECUTIVO FINAL

¿Qué Hace CompareTables?

CompareTables es un **motor de comparación de datos empresarial** que:

1. **Compara automáticamente** dos conjuntos de datos (referencia vs. nuevo)
2. **Detecta diferencias** a nivel de fila y columna con precisión
3. **Identifica duplicados** exactos y con variaciones
4. **Genera métricas de calidad** para evaluación ejecutiva
5. **Produce reportes** en formato tabular (Hive) y presentación (Excel)

Valor para la Organización

- **Auditoría**: Verificación automática de integridad en migraciones

- **Compliance**: Documentación de diferencias para auditorías regulatorias
- **Eficiencia**: Eliminación de comparaciones manuales propensas a errores
- **Calidad**: Métricas objetivas para evaluar calidad de datos
- **Decisiones**: Información clara para toma de decisiones ejecutivas

Casos de Uso Principales

1. **Migración de Sistemas**: Verificar integridad en transiciones
2. **Reconciliación**: Validar consistencia entre sistemas operativos
3. **Control de Calidad**: Monitorear procesos ETL y transformaciones
4. **Auditoría**: Documentar diferencias para cumplimiento

Métricas Clave a Monitorear

- **Quality Global**: >90% para sistemas de producción
- **Diferencias (NO_MATCH)**: <10% para migraciones exitosas
- **Duplicados**: <15% para datos limpios
- **Gaps**: <5% para cobertura completa

Recomendaciones de Implementación

1. **Fase 1**: Implementar comparaciones básicas para casos críticos
2. **Fase 2**: Agregar detección de duplicados para control de calidad
3. **Fase 3**: Implementar monitoreo automático y alertas
4. **Fase 4**: Integrar con sistemas de BI para dashboards ejecutivos

---

Documento preparado para presentación ejecutiva Fecha: Agosto 2025 **Versión**: 1.0 **Autor**: Equipo de Desarrollo CompareTables