Universidad Inca Garcilaso de la Vega

FACULTAD DE INGENIERÍA DE SISTEMAS, CÓMPUTO Y TELECOMUNICACIONES

VALIDACIÓN DE MIGRACIÓN DE DATOS SOBRE UNA EMPRESA DE RUBRO BANCARIO A TRAVÉS DE CONTROLES ELABORADOS EN LENGUAJE DE PROGRAMACIÓN PYTHON PARA VERIFICAR SU CONFIABILIDAD LIMA PERU 2023

TRABAJO DE SUFICIENCIA PROFESIONAL

Para optar el título profesional de Ingeniero de Sistemas y Cómputo

AUTOR

Maúrtua Vidal, Valeria Patricia

ASESOR

Díaz Flores, Paul Alberto

Lima - Perú
2023

VALIDACIÓN DE MIGRACIÓN DE DATOS SOBRE UNA EMPRESA DE RUBRO BANCARIO A TRAVÉS DE CONTROLES ELABORADOS EN LENGUAJE DE PROGRAMACIÓN PYTHON PARA VERIFICAR SU CONFIABILIDAD LIMA PERU 2023

INFORME DE ORIGINALIDAD

4%	4%	0%	3%
INDICE DE SIMILITUD	FUENTES DE INTERNET	PUBLICACIONES	TRABAJOS DEL ESTUDIANTE

FUENTES PRIMARIAS

1	Submitted to Universidad Inca Garcilaso de la Vega Trabajo del estudiante	1%
2	repositorio.ucss.edu.pe Fuente de Internet	1%
3	repositorio.upao.edu.pe Fuente de Internet	1%
4	repositorio.undac.edu.pe Fuente de Internet	1%
5	todociber.com.ar Fuente de Internet	1%
6	www.scielo.org.mx Fuente de Internet	1%

Excluir citas Activo

Excluir coincidencias < 50 words

DEDICATORIA

El presente proyecto va dedicado a mi familia que me ha apoyado en todo momento para poder cumplir mis objetivos personales y sobre todo profesionales, gracias por confiar en mí y por siempre empujarme a ser una mejor persona.

AGRADECIMIENTO

Le brindo un agradecimiento a mis docentes de la Universidad Inca Garcilaso de la Vega que han aportado mucho en ampliar mis conocimientos para ponerlos en práctica en el día a día; sobre todo un especial agradecimiento al docente Paul Díaz que nos brindó las pautas para realizar el presente proyecto y darle forma en cada entrega de la investigación.

RESUMEN Y PALABRAS CLAVE

Resumen: La investigación realizada tuvo como objetivo la reducción de tiempos empleados para la validación de las migraciones de un ambiente original a un ambiente final en una empresa bancaria; debido a que este ejercicio de comparación de bases era manual para analizar la completitud de universos, integridad de los datos y duplicados, con el propósito de confirmar que la migración ha sido exitosa y no existen cambios en los datos de la organización que permite la toma de decisiones. Al ser una comparación manual el tiempo empleado para esta revisión podía superar más de una semana porque las bases comparadas en algunos casos podían superar los 20 millones de registros y las 100 variables en columnas.

Para ello; se generó un script en Python que permite analizar grandes volúmenes de datos para poder reducir los tiempos de validación manteniendo los mismos criterios de revisión (completitud de universos, integridad de los datos y duplicados considerando los identificadores de las tablas) y estructurando el proceso para que sea comprendido y sencillo de actualizar a través de parámetros generales.

Palabras clave**:** Comparación, completitud, datos, identificadores, integridad.

VALIDATION OF DATA MIGRATION ON A BANKING COMPANY THROUGH CONTROLS ELABORATED IN PYTHON PROGRAMMING LANGUAGE TO VERIFY ITS RELIABILITY LIMA PERU 2023

ABSTRACT AND KEYWORDS

Abstract: The objective of the research was to reduce the time required to validate the migrations from an original environment to a final environment in a banking company, since this database comparison exercise was manual to analyze the completeness of universes, data consistency and duplicates, with the purpose of confirming that the migration has been successful and that there are no changes in the organization's data that allow decision making. Being a manual comparison, the time used for this review could exceed more than a week because the bases compared in some cases could exceed 20 million records and 100 variables in columns.

For this purpose, a Python script was generated to analyze large volumes of data in order to reduce validation times while maintaining the same review criteria (completeness of universes, consistency of data and duplicates considering the table identifiers) and structuring the process to be understood and easy to update through general parameters.

Keywords: Comparison, completeness, data, identifiers, integrity.

ÍNDICE GENERAL

INTRODUCCIÓN

El área de validación es la segunda línea de defensa o control de segunda línea de una empresa bancaria tiene la función de verificar que todas las implementaciones y migraciones que se realicen en la organización se encuentre en correctas condiciones para evitar los riesgos latentes que se producen a partir de los datos; además, por gobierno antes de cualquier uso de la información debe pasar por validación.

Esta empresa bancaria siempre se encuentra en la vanguardia para brindar un mejor servicio a sus clientes; para ello adquirió una nueva arquitectura de almacenamiento de datos en OnPremise para que las operaciones sean más rápidas. Cabe indicar; que al realizar un cambio de ambiente viene acompañado con generar una migración para que los datos que están en la herramienta del ambiente original Oracle pasen al nuevo ambiente en OnPremise. En esta migración participan:

1) Área de Data de la empresa para realizar una reingeniería; ya que los códigos deben pasar a un lenguaje de programación en Python.
2) Área de Arquitectura de datos (modelamiento) para definir las estructuras de las tablas y evitar la redundancia de variables.
3) Área de Gobierno de Datos para establecer las directivas y pautas de migración que debe cumplir Data y generar el control de primera línea para confirmar a través de una certificación que hayan realizado bien la migración.
4) Área de Validación Interna para ratificar que la migración ha sido exitosa, contemplando controles de completitud de universo para asegurarse que se mantengan los mismos registros, integridad de datos que darán la confianza que los valores de cada variable por cliente sigan siendo los mismos y duplicidad para que cada cliente sea único en las bases analizadas.

Sin embargo; el área de validación inicialmente tenía una comparación manual a través de Excel o extraían los datos de OnPremise a SAS para usar las macros de comparación y esto ocasionaba demora en la finalización de los proyectos de validación de cada tabla migrada. Por ese motivo; se generó un script en Python que mantiene los mismos controles de comparación permite analizar gran cantidad de información fácilmente y el

resultado de estas comparaciones son reportes ejecutivos que detallan las coincidencias y diferencias que se obtienen de la revisión.

CAPÍTULO 1: MARCO TEORICO DE LA INVESTIGACION

1.1 Antecedentes del estudio

1) Título: Marco de trabajo para el proceso de migración de datos de productos financieros en cooperativas de ahorro y crédito del Ecuador – Escuela Politécnica Nacional
   Autor: Guerrero Cueva, Marcos Damián
   Url: https://bibdigital.epn.edu.ec/handle/15000/21379
   Mientras la tecnología va evolucionando los sistemas legados son reemplazados con nuevos sistemas informáticos que contemplan la modernización de la infraestructura, sistema operativo y aplicaciones. No obstante, la información es fundamental y no se puede reemplazar; por ello la migración considera 3 fases claves: depurar la información que no se usa, mejorar la información proporcionada y trasladarlo del sistema legado al nuevo sistema. Cabe indicar, que las Cooperativas de Ahorro y Crédito son empresas que tienen un crecimiento veloz y tienen la necesidad de migrar a nuevos sistemas.
   El presente proyecto tiene como objetivo proponer un Marco de trabajo para migrar información financiera estableciendo normas que aseguren la información proveniente del sistema legado y la calidad de este para generar correctas operaciones. En la validación del marco de trabajo se orientó a dos empresas financieras con el propósito de estimar la dirección de la información de los clientes y la aprobación de su realización de los involucrados; para ello, se generaron encuestas a los involucrados.
   El resultado de las encuestas concluyeron que en ambas empresas la calidad de la información generó el correcto funcionamiento del sistema nuevo, pero no estaban de acuerdo con el tiempo de duración del proyecto.
   La tesis aporta a mi trabajo de investigación en establecer directrices que permitan asegurar la calidad de la información del sistema original y que la migración no solo es trasladar los datos de un lugar a otro, sino también aplicar una reestructuración del proceso, la arquitectura y de cuestionarlos si hay información que puede ser deprecada.

2) Título: Validador Migración Datos - Universidad Politécnica de Cataluña
   Autor: Daniel Siles González
   Url:
   https://upcommons.upc.edu/bitstream/handle/2117/83609/Memoria.pdf?sequence=1&isAllowed=y
   En la empresa actual donde laboro existe la necesidad de reducir tiempo, esfuerzo y costo del proyecto de validación de una migración. El propósito es la migración de una empresa a otra; para ello, se debe amoldar los sistemas de la empresa origen a la empresa destino con el objetivo que la empresa origen se elimine.
   El objetivo del proyecto es diseñar e implementar un aplicativo que genere la calidad de los datos del sistema destino a partir de la comparación del sistema origen versus el destino a través de ciertas validaciones que son definidas por los diferentes equipos de la empresa.

Se generó la reducción del costo de la validación del proyecto de migración en una sola iteración.
El proyecto contribuye en mi trabajo de investigación a buscar eficiencia en las validaciones de comparación para reducir costos en el proyecto y asegurar los datos de las empresas, haciendo uso de programas que permitan comparar datos rápidamente lo que genera que los tiempos de las validaciones sean óptimos.

3) Título: PROPUESTA DE UNA METODOLOGÍA PARA EL SEGUIMIENTO A LA IMPLEMENTACIÓN DE PROYECTOS DE MIGRACIÓN DE DATOS A PARTIR DEL MODELO "STAGE-GATE" - Universidad Pontificia Bolivariana
Autor: Laura Dinency Restrepo Girald
Url: https://repository.upb.edu.co/handle/20.500.11912/3995
Los modelos que se orientan en el seguimiento de proyectos de migración da los datos se relacionan en las fases que consideran, los entregables; así como los roles y responsabilidades de los integrantes del equipo.
El propósito del proyecto es proponer una metodología que permita darle seguimiento al desarrollo de proyectos de migración de datos, considerando una investigación exploratoria descriptiva usando el modelo Stage Gate para diseñar una metodología que contempla cinco fases. La validación de la metodología se llevó a cabo bajo un análisis de comparación cualitativa y cuantitativa. Estas comparaciones brindan niveles que permiten identificar si la calidad del dato es la correcta para realizar el pase a producción de los datos.
Los resultados del proyecto permitieron establecer mejores prácticas de comparación que permiten asegurar la migración de los datos.
La tesis aporta a mi trabajo de investigación a considerar comparaciones cuantitativas como cualitativas y niveles de aceptación para identificar si los datos corresponden a los del ambiente origen.

4) Título: Migración de datos de canales digitales a una infraestructura de Big Data en una entidad bancaria – Universidad Nacional Mayor de San Marcos
Autor: Emily Ann SEDÁN HERRERA
Url: https://renati.sunedu.gob.pe/handle/sunedu/3327273
El presente Proyecto corresponde a una migración de datos de los canales digitales de una entidad bancaria hacia Data Lake contemplando validar la calidad del dato, cuadre del ambiente Oracle versus Data Lake y mecanismos para resguardar los datos de alta criticidad.
La finalidad del proyecto es construir reglas de calidad de datos diseñadas por Gobierno, que se encargan de supervisar los datos de la empresa bancaria.
El resultado fue aplicar validaciones para garantizar la integridad de los datos de la organización.
El proyecto contribuye en mi trabajo de investigación a asegurar que se estén protegiendo los datos que identifiquen a un cliente a través de algoritmos y elaborar criterios reglas para evaluar la calidad del dato migrado.

5) Título: IMPLEMENTACIÓN DE UN SISTEMA DE BIG DATA APLICADO A LA MIGRACION DE DATOS BAJO LA DISTRIBUCION CLOUDERA CON APACHE HADOOP, EN EL BANCO INTERBANK – Universidad Tecnológica del Perú
Autor: Carlos Linares Berrocal
Url: https://repositorio.utp.edu.pe/bitstream/handle/20.500.12867/1944/Carlos%20Linares_Trabajo%20de%20Suficiencia%20Profesional_Titulo%20Profesional_2019.pdf?sequence=1&isAllowed=y
Interbank ha tenido un crecimiento en sus datos lo que representa inconveniente para la gestión y almacenamiento de datos, lo que genere un incremento en costos operativos como de infraestructura; por ese motivo, es inevitable tener una plataforma que soporte registro de tareas y procesamiento escalable y de alta disponibilidad.
El propósito del proyecto es realizar la ingesta de datos y procesamiento de datos a través de Hadoop con Apache Hive; además de evidencias los tiempos de ejecución del ambiente origen versus el ambiente destino.
Como resultado del proyecto mediante la ingesta de datos Apache Sqoop a Hadoop, se pudo centralizar todos los datos provenientes de la fuente de datos del ambiente origen depositados al ambiente destino (Data Lake), lo que permitirá no solo trabajar con información estructurada si no también semiestructurada y no estructurada; con el fin de potenciar la toma de decisiones en la compañía.
La tesis aporta a mi trabajo de investigación a evaluar los tiempos de ejecución del ambiente origen y destino; ya que el generar una migración a una nueva plataforma debe brindar reducción en el procesamiento de los datos.

1.2 Marco conceptual

Migración de Datos
Definición:
"La migración de datos comprende en identificar el ambiente origen y el ambiente destino donde serán alojados los datos. Sobre ello, se genera una comparación mediante el cual se determina qué modificaciones sufrirán los elementos de los esquemas conceptuales; y en base a ello, detectar entre qué elementos debe aplicarse el traspaso de información" (Carsí, Ramos, Silva, Pérez y Anaya, 2002)
Para Saborío y Chinchilla (2014), se concluye que todo proceso de migración debe estar relacionado a un proyecto existente y debe considerarse la depuración de datos con el objetivo de garantizar pureza en los datos y mantener los que sean necesarios para a organización.
Ambas definiciones aportan a la investigación, ya que es importante identificar cual es el ambiente original y cuál será el ambiente destino de la solución para poder realizar la transferencia de información; además esta trasferencia no es exacta, porque podrían existir datos que ya no sean requeridos por la organización o se identifiquen variables duplicadas en más de una tabla.

Validación de Datos
Definición:
La validación de datos tiene como objetivo comparar esquemas conceptuales que forman parte de la migración; y como resultado, brinda el detalle de la información eliminada, la nueva información y las actualizaciones que han sufrido los datos. (Carsí, Ramos, Silva, Pérez y Anaya, 2002, pag. 17)
El aporte en mi investigación es poder tener en cuenta que el generar una validación puede brindarse resultados para identificar si la data ha sido modificada, si no se ha considerado alguna variable o cliente en la migración y si hay nuevos datos que están siendo obtenidos en el esquema destino.

Empresa del rubro Bancario
Definición:
Según (Chipana, 2023) el rubro bancario es una entidad donde solicitas dinero a un plazo con un interés asociado en base a lo que la empresa considere. Con esto, las empresas bancarias, venden este dinero a un precio e interés mayor, así podrán generar sus ganancias correspondientes y devolviendo el monto que anteriormente se había solicitado.
Esta definición aporta en mi investigación debido a que si los datos no se migran completa y correctamente puede generar que se brinde dinero a personas que no cumplan con sus cuotas de pago; lo que impacta directamente a la empresa bancaria.

Controles de calidad de datos
Definición:
El control de calidad es el conjunto de procedimientos que ayudan a cerciorarse que las auditorías se realizan de acuerdo a las normas establecidas, asegurando un servicio profesional con eficiencia y eficacia (Calderon y Rios, 2018)
El tener controles de calidad de datos asegura la data que viaja a los diferentes usos de la organización.
Exactitud
Definición:
"Hace referencia a que los datos poseen atributos representados de manera correcta (puede ser sintáctica o semántica)." (Pasini, Torres, Esponda y Pesado, 2021)
Completitud
Definición:
Los datos poseen valores para todos los atributos esperados. (Pasini, Torres, Esponda y Pesado, 2021)

Tabla 1. Definición de métrica para la Completitud

Tipo	**Inherente**
Nombre de la medida de calidad	Completitud de los datos en un archivo
Función de medición	Valor = A/B
Variables	A = número de datos requeridos para el contexto particular en el archivo B= número de datos en el contexto particular de uso previsto

(Pasini, Torres, Esponda y Pesado, 2021)

Consistencia
Definición:
Los datos no son contradictorios y son consistentes unos con otros. (Pasini, Torres, Esponda y Pesado, 2021)

Tabla 2. Criterios de decisión para las características Exactitud, Consistencia, Completitud y Comprensibilidad.

Características	Nivel	Rango de valores
Exactitud	Inaceptable	Si Valor >= 0 y Valor < 0,3
Consistencia	Mínimamente aceptable	Si Valor >= 0,3 y Valor < 0,7
Completitud	Rango objetivo	Si Valor >= 0,7 y Valor < 0,9
Comprensibilidad	Excede los requerimientos	Si Valor >= 0,9

(Pasini, Torres, Esponda y Pesado, 2021)

Credibilidad
Definición:
Los datos poseen atributos ciertos y creíbles (incluye concepto de autenticidad). (Pasini, Torres, Esponda y Pesado, 2021)
Actualidad
Definición:
Los datos poseen atributos que son actualizados adecuadamente. (Pasini, Torres, Esponda y Pesado, 2021)

Lenguaje de Programación
Python
Definición:
Para (Masip, 2018), Python se trata de un lenguaje de programación de muy alto nivel, con una sintaxis muy clara y una apuesta firme por la legibilidad del código. Sin duda es un lenguaje de programación muy versátil, fuertemente tipado, imperativo y orientado a objetos, aunque contiene también características que lo convierten en un lenguaje de paradigma funcional.
El uso de Python proporciona el análisis de grandes volúmenes de datos y generar estructuras de código claras y óptimas.

Librerías
Definición:
Según (Barón, 2018) permiten simplificar la utilización de serie de instrucciones, es fácilmente portable y es llamado de manera directa y abreviada.
El uso de librerías en Python ayuda a esquematizar y reducir líneas de código; ya que estas librerías contienen lógicas que para usarlas solo es necesario invocarlas.

Confiabilidad
Definición:
“Buscar que la investigación sea estable, segura, congruente, igual a sí misma en diferentes tiempos y previsible para el futuro. Reducir la amenaza que representa la confiabilidad interna mediante … la confirmación de que lo visto o registrado por el investigador coincide o es consistente con lo que ven o dicen los sujetos del grupo estudiado …” (Borjas, 2020)

Generar una correcta validación de la migración de los datos de la organización da como resultado que los usuarios que deben consumir la data tengan seguridad y confianza en el contenido ya que ellos son los que van a tomar decisiones en base a la información migrada.

CAPITULO II: PLANTEAMIENTO DEL PROBLEMA

2.1 Descripción de la realidad problemática

El avance de la tecnología conlleva el uso de nuevas herramientas, forzando a las empresas a estar en la vanguardia y brindar las mejores soluciones a sus clientes. Por ese motivo; las empresas siempre buscan plataformas que sean sostenibles y les permitan procesar grandes volúmenes de datos para obtener rápidamente la información y ser más eficientes.

Existen empresas que tienen mayor presión de ir a la par con la tecnología, como lo son las de rubro bancario; debido a que tienen la necesidad de agilizar sus procesos para ofrecer la mejor experiencia a los clientes.

Por ese motivo; la empresa de rubro bancario optó por migrar a un ambiente OnPremise los datos la de la organización con el propósito de tener una plataforma estable, óptima para consultas, con conectores a los aplicativos de la empresa con el objetivo de enriquecer diariamente las bases de datos de la empresa y establecer una mejor arquitectura de datos.

Cabe indicar que la migración de un ambiente a otro no es una tarea sencilla; ya que tiene como principal objetivo que los datos de la entidad bancaria sigan siendo los mismos; sin embargo, el propósito de la migración no es mantener la misma estructura que el ambiente original, debido a que se ha realizado una reestructuración total de los procesos, scripts y tablas para tener un esquema más ordenado.

Es pertinente resaltar que la migración también puede motivar alguna mejora en la lógica de una variable o la automatización del script (beneficioso para los tiempos de ejecución de los diferentes procesos); sin embargo, esto ocasiona un riesgo latente porque la información podría no ser la misma y se dude de su confiabilidad. Esto produce que los usuarios internos de la empresa de rubro bancario desconfíen en los datos que se están migrando que probablemente podría afectar los diferentes usos de la organización (por ejemplo: toma de decisiones, evaluación crediticia, campañas, entre otras); porque los datos comportamentales, financieros, geográficos y económicos son los que brindan una mejor predicción a nivel del negocio. Además; en la empresa de rubro bancario existen normativas a nivel de gobierno, que se orienta a que exista una validación antes del uso en la gestión de algún proceso productivo.

2.2 Formulación del problema general y específicos

2.2.1 Problema General

¿Cómo la Validación de la migración de datos sobre una empresa de rubro bancario a través de controles elaborados en lenguaje de programación Python verifica su confiabilidad?

2.2.2 Problemas Específicos

a) ¿Cómo se realiza la validación de datos en una migración de una empresa de rubro bancario?
b) ¿Cómo los controles elaborados en lenguaje de programación Python verifican la confiabilidad de los datos migrados en una empresa del rubro bancario?
c) ¿Cómo la validación de migración de datos asegura a una empresa de rubro bancario que la data es consistente para el consumo de los usuarios internos?

2.3 Objetivo general y específicos

2.3.1 Objetivo General

Validar la correcta migración de los datos de una empresa bancaria para confirmar su confiabilidad a través de controles elaborados en lenguaje de programación Python.

2.3.2 Objetivos Específicos

a) Realizar la validación de datos para el uso de la información en una empresa del rubro bancario.
b) Establecer los controles elaborados en lenguaje de programación Python para verificar la confiabilidad de los datos migrados en una empresa del rubro bancario.
c) Asegurar a una empresa de rubro bancario que la data es consistente para el consumo de los usuarios internos.

CAPITULO III: JUSTIFICACION Y DELIMITACION DE LA INVESTIGACION

3.1 Justificación e importancia del estudio

3.1.1 Justificación

Los datos de las organizaciones principalmente las del rubro bancario ayudan a tomar correctas decisiones; éstos datos pueden relacionarse a información comportamental de los clientes que permiten predecir si en caso de brindarles un crédito cumplirán con el pago del préstamo asignado. Por ese motivo; la migración de las tablas analíticas de una empresa bancaria de un ambiente a otro, podría generar un alto impacto en la organización debido a que el alcance del proyecto considera abarcar una reestructuración total de los programas (scripts) que calculan los datos, cambio de lenguaje de programación (Oracle a Python), renombre de las variables y delimitación de la migración de variables que podrían afectar algún proceso que las requiera insumir.

Justificación Tecnológica:

El uso de Python ayudará a procesar grandes volúmenes de datos, ya que la empresa procesa información para más de 23 millones de personas; y realizar una comparación manual no sería eficiente.

Justificación Económica:

Reducción de recursos para generar la validación de datos que han sido migrados.

Impedir que se asignen créditos a clientes que no tienen un buen comportamiento en el sistema financiero.

Justificación Social:

Asegurar que los datos que serán evaluados por los usuarios internos de la empresa bancaria sigan manteniendo el mismo valor y conserve la integridad de la información.

3.1.2 Importancia del Estudio

El generar la validación de los datos que se están migrando en la empresa bancaria permitirá asegurar la confiabilidad de la información y cumplirá con el gobierno que tiene estipulado la empresa, ya que ante el uso de cualquier información debe existir una validación para asegurar la correcta programación de las lógicas de las variables, brindar

una correcta calidad de la información y tener un flujo de ejecución de los procesos más eficientes.

3.2 Delimitación del estudio

3.2.1 Delimitación Espacial

El trabajo de investigación se desarrolla en una empresa de rubro bancario ubicada en Perú, departamento de Lima.

3.2.2 Delimitación Temporal

El proyecto tiene una duración de 4 años; considerando como año de inicio del 2021.

3.2.3 Delimitación Social

El grupo de objeto de estudio son los colaboradores internos de la empresa bancaria que usarán la información.

3.2.4 Delimitación Conceptual

La validación de la migración se orientará en el uso de la herramienta Python y de librerías propias de la herramienta.

CAPITULO IV: FORMULACION DEL DISEÑO

4.1 Propuesta de solución

4.1.1 Metodología de la solución

Se consideró la metodología de scrum para el desarrollo de este proyecto ya que nos permite aplicar un marco ágil de trabajo detallando los avances de forma diaria y mencionando si hay limitantes o dependencias para destrabarlo y fluya el término del script de comparación para enchufarlo en el proceso de validación lo más pronto posible.

4.1.2 Identificación de requerimientos

4.1.2.1 Logísticos

Validador:

REQL001 - PC o laptop con capacidad de 8GB de RAM como mínimo.

REQL002 - Instalación de la aplicación Anaconda (Python) o uso del lenguaje de programación Python a través del Datalake.

REQL003 - Acceso a las tablas del ambiente origen y ambiente destino.

Área que generó la migración:

REQL004 - Documento de alcance: Describe la tabla y variables del ambiente origen; así como, la tabla y variables del ambiente destino, para identificar las equivalencias.

REQL005 - Documento de certificación: Contiene los controles aplicados por el área que generó la migración para confirmar la correcta migración de los datos (control de primera línea); controles: cantidad de registros, promedios, sumatorias, duplicados por llave.

4.1.2.2 Funcionales

REQF001 – Completitud de universos: El script debe permitir identificar la correcta migración de todos registros en base a la llave primaria de la tabla.

REQF002 – Exactitud de la información: Identificar la coincidencia de los valores por cada variable y detallar el porcentaje y cantidad de coincidencias y diferencias de cada una de ellas.

REQF003 – Duplicidad de la información: Considerando la llave primaria de la taba identificar si existen registros que se repiten, debido a que la información debe ser única.

4.1.2.3 No Funcionales

REQNF001 - Facilidad de uso: El script de comparación debe ser fácil de usar y esto se realizará mediante parámetros generales que se encuentren en la parte inicial del programa.

REQNF002 - Rendimiento: La ejecución del proceso de comparación no debe demorar mucho tiempo ya que debe ser eficiente para brindar los resultados en un tiempo reducido.

4.1.3 Desarrollo de la solución o implementación

4.1.3.1 Análisis

Se identificó el punto de dolor de la validación el cual se realizaba a través de comparaciones manuales en Excel o a través del programa SAS para hacer uso de las macros que contiene esta herramienta, pero esto requería el consumo inicial de las bases a comparar y de igual forma los tiempos no eran óptimos.

Por ende; se plantearon soluciones para el desarrollo de un script con la finalidad de reducir los tiempos de validación que inicialmente tomaba entre 1 a 3 semanas bajo una comparación en herramientas externas a la plataforma onpremise. Las soluciones estaban orientadas en 2 lenguajes de programación: R y Python; sin embargo, teniendo en cuenta que Python es una herramienta más comercial y con librerías que permiten optimizar código se decidió considerar una programación del script de comparación en Python a través de Pyspark.

Posterior a este análisis se preparó el ambiente de trabajo de la solución para iniciar con la codificación del script de comparación.

4.1.3.2 Diseño

La propuesta de solución de los resultados abarcó una generación de reportes que permite dar visibilidad de las coincidencias y diferencias del proceso:

1) **Resultados de Control de Integridad:**
   El reporte de los resultados detalla las tablas comparadas de Oracle y Datalake Onpremise asociado al mes que fue comparado. Esta comparación se realiza bajo las llaves identificadoras de las tablas y se considera una diferencia relativa con una tolerancia del 1% entre valor y valor.

Además, el reporte muestra la leyenda de los colores que se reflejan en el porcentaje de coincidencias de las variables:
Verde → Mayor igual al 99% hasta el 100% de coincidencia
Amarillo → Mayor igual al 95% y menor al 99%
Rojo → Mayor igual a 0% y menor al 95%
Finalmente; se muestra una tabla que lista las variables comparadas, la cantidad de coincidencias por variable, la cantidad de diferencias por variable, el porcentaje de coincidencia por variable, el porcentaje de diferencias por variable y el tipo de diferencia usada en la comparación: relativa o absoluta

Resultados de Control de Integridad

Universos Comparados **Diferencia Relativa: 0.01**
Tabla A: Tabla de Oracle where codmes =202305
Tabla B: Tabla de Datalake Onpremise where codmes=202305

Llaves de Universos
Tabla A: Identificador de Oracle
Tabla B: Identificador de Datalake Onpremise

Leyenda
Verde: [1,0.99["Cumple los Criterios de Integridad"
Amarillo: [0.99,0.95["Existen algunos criterios de Integridad"
Rojo: [0.95,0["Existen Problemas de Calidad de los Datos"

Variable HIVE	# Coincidencias	# Diferencias	% Coincidencias	% Diferencias	Tipo
CODCLAVEUNICOCLI	2305794.0	988.0	99.9572%	0.0428%	Absoluta
CODINTERNOCOMPUTACIONAL	2306782.0	0.0	100.0%	0.0%	Absoluta
CODMODELORIESGO	2306782.0	0.0	100.0%	0.0%	Absoluta
CTDDIAATRASOMAXBCP	2306782.0	0.0	100.0%	0.0%	Relativa
CTDDIAATRASOMAXSBS	2306746.0	36.0	99.9984%	0.0016%	Relativa
CTDENTIDADFINANCIERAREPORTANTE	2304009.0	2773.0	99.8798%	0.1202%	Relativa
CTDENTIDADFINANCIERAREPORTANTEDEUDA	2306637.0	145.0	99.9937%	0.0063%	Relativa

Fuente: Elaboración propia

2) Resultados de Control de Completitud de Universos:
El reporte que brinda los resultados de la completitud de universos detalla inicialmente las tablas comparadas, el mes comparado, los identificadores usados para la comparación.
Adicionalmente; describe el porcentaje y cantidad de coincidencias en universos por cada tipo de resultado que corresponde a:

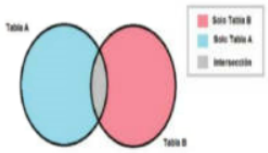
a. Intersección: identificador que se encuentra en tabla A y también en tabla B
b. Solo tabla A: Registros que se encuentran únicamente en la tabla A y no en la tabla B.
c. Solo tabla B: Registros que se encuentran únicamente en la tabla B y no en la tabla A.

Por ultimo; refleja la cantidad de registros que se encuentra en la tabla A y en la tabla B.

Resultados de Control Completitud de Universos

Universos Comparados
Tabla A: Tabla de Oracle where codmes =202305
Tabla B: Tabla de Datalake Onpremise where codmes=202305

Llaves de Universos
Llaves de Tabla A: Identificador de Oracle
Llaves de Tabla B: Identificador de Datalake Onpremise



Resultado	% Resultado	# Obs. en Resultado
Interseccion	99.951 %	2306782.0
Solo A	0.0001 %	3.0
Solo B	0.0489 %	1128.0

Numero de Observaciones en Tabla A: 2306785.0
Numero de Observaciones en Tabla B: 2307910.0

Fuente: Elaboración propia

3) Control de Duplicidad de Registros Universo:

El resultado del reporte de análisis de duplicidad de registros considera la descripción de la tabla de Datalake Onpremise analizado, la llave identificadora utilizada para la revisión y el periodo analizado.

Por otro lado; se mencionan las siguientes características de la tabla revisada:

a. Registros únicos: No considera la cantidad de registros duplicados de la base; sólo los únicos.
b. Número de registros duplicados únicos: Considera la cantidad de registros duplicados únicos de la base; es decir, si una llave tiene 4 duplicados, solo considera en el conteo a 1 porque se repite 3 veces ese registro.
c. Número de registros de la Base de datos: Total de registros de la tabla.
d. Número de registros duplicados en la Base de datos: Cantidad de registros que se encuentran duplicados (registros únicos + número de registros duplicados únicos)
e. Número de registros adicionales: Considera la cantidad de registros duplicados adicionales de la base; es decir, si una llave tiene 4 duplicados, solo considera en el conteo a 3 porque se repite 3 veces ese registro.
f. Porcentaje dual: Porcentaje de registros duplicados en la base para el mes revisado.

Control de Duplicidad de Registros Universo

Universo
Tabla A: Tabla Datalake Onpremise where codmes =202305

Llaves Primarias (PK)
Tabla A: Identificador de Datalake Onpremise

Tabla en Analisis: Tabla Datalake Onpremise where codmes =202305

Registros Unicos	Nro Registros Dup Unicos	Nro Registros bbdd	Nro Registros dup bbdd	Nro Reg Adicionales	Prc Dup
2306785.0	0.0	2306785.0	0.0	0.0	0.0 %

Fuente: Elaboración propia

4.1.3.3 Desarrollo

Una vez identificado la problemática y el alcance del proyecto se inició con la codificación del script de comparación teniendo en cuenta que el procedimiento de actualización del script para cada validación debe ser lo más óptimo posible.

En primer lugar, se importaron las librerías necesarias que nos ayudarán para usar las funciones que por default existen en la herramienta:

```
from pyspark.sql import *

import pandas as pd

import os

import numpy as np

import time

import math

import itertools

import subprocess

import json

import codecs

import pytz
```

```
from pyspark.sql.functions import trim

from datetime import datetime

from pyspark.sql.types import StringType

from pyspark.sql import functions as f

from datetime import datetime

from pyspark.sql import SparkSession

from pyspark.sql.utils import AnalysisException

from pyspark.sql.window import *

import fpdf

from fpdf import FPDF

from pyspark.sql.functions import col,concat,desc,from_unixtime,to_date,unix_timestamp,current_date, sum, lit, when, current_timestamp,col,when,lit,rand

import sys

import json

from openpyxl import Workbook

from openpyxl.utils.dataframe import dataframe_to_rows
```

Se identificaron los parámetros generales del proceso de comparación para evitar que en las funciones se generen cambios. Estos parámetros son los siguientes:

```
# Define the variables to save the results

    tableName = "tabla_onpremise"

    #Schema for save

    dbName = 'esquema'

    location = 'ruta_trabajo'
```

#Save All or Save only the differences

flg_save = False

#Activar este flgs si se quiere guardar todas las comparaciones esto requiere mas espacio en disco

flg_save_all_comparison = False

#Export Results of Diferences to Excel

flg_export_diff = False

#define variables for comparison

tablaOracle = "esquema.tabla_oracle"

tablaHUE = "esquema.tabla_onpremise"

fields_to_compare_ora=['variable1_Oracle','variable2_Oracle','variable3_Oracle']
fields_to_compare_dl=['variable1_Datalake','variable2_Datalake','variable3_Datalake]

clavesHUE=['identificador_onpremise']

clavesOracle=['identificador_oracle']

dif_rel=0.01

lst_rango_mes= ['añomes_1','añomes_2','añomes_3']

#Para activar el flg de importacion de oracle debe expresar la query en el metodo generate_sqlquery_ora()

flg_import_from_oracle = True

 # #Configuracion de tabla a analizar

oracle_table = "esquema_oracle.table_oracle"

hive_table = "esquema_onpremise.tabla_onpremise"

#Configuracion de rango de meses a analizar

flg_universo = False

flg_duplicates = True

#Ejecucion

spark = create_spark()
ingesta_ora(spark,lst_rango_mes,tableName,dbName,location,generate_sqlquery_ingesta_ora())
ingesta_dl(spark,lst_rango_mes,tableName,dbName,location,generate_sqlquery_ingesta_dl())
compare_universo(spark,generate_sqlquery_universo_ora(),generate_sqlquery_universo_dl())

validacion_unicidad_dl(spark,hive_table, lst_rango_mes, clavesHUE)#ok

comparisons(spark,tablaOracle, tablaHUE, lst_rango_mes, fields_to_compare_ora,fields_to_compare_dl,clavesHUE,clavesOracle,dif_rel,flg_save, flg_save_all_comparison,flg_export_diff,flg_import_from_oracle,generate_sqlquery_ingesta_ora(),generate_sqlquery_dl(),tableName,dbName,location)

Una vez definido los parámetros se procede a generar la extracción de la información de los ambientes a comparar, en este caso Oracle y Datalake Onpremise. Para ello, se genera una conexión a Oracle que nos permite obtener la información directamente del esquema y nombre de la tabla del ambiente origen y luego se procede a extraer la información del esquema y tabla del ambiente destino en este caso Datalake Onpremise:

#Ingesta de datos de Oracle

def ingesta_ora(spark,lst_rango_mes,tableName,dbName,location,sqlQuery):

contador = 0

for mes in lst_rango_mes:

#leer oracle

df = credencials_ora(spark,sqlQuery.format(mes))

 # writeToHivePartitioned(df, <tabla a crear en lake>,'codmes') solo el nombre sin el esquema

```
    #df.show(5)
    if contador == 0:
        writeToHivePartitioned(spark,df, tableName+'_ORA',"CODMES",True, True,dbName,location)
    else:
        writeToHivePartitioned(spark,df, tableName+'_ORA',"CODMES",False, False,dbName,location)
    contador = contador +1
#Ingesta de datos de Data Lake
def ingesta_dl(spark,lst_rango_mes,tableName,dbName,location,sqlQuery):
  contador = 0
  for mes in lst_rango_mes:
    df = spark.sql(sqlQuery.format(mes))
    # Imprimir sqlQuery
    #print(sqlQuery)
    df.show(5)
    if contador == 0:
        writeToHivePartitioned(spark,df, tableName+'_DL',"CODMES",True, True,dbName,location)
    else:
        writeToHivePartitioned(spark,df, tableName+'_DL',"CODMES",False, False,dbName,location)
    contador = contador +1
```

Ya teniendo las tablas de Oracle y Datalake Onpremise se generan las funciones que nos permitirán dar el resultado de las comparaciones de:

1) **Completitud de universos:**

def compare_universo(spark, oracle_query, hive_query):

 # Ejecutar la consulta en Oracle utilizando PySpark

resultado_oracle = credencials_ora(spark, oracle_query)

resultado_pandas_oracle = resultado_oracle.toPandas()

 # Ejecutar la consulta en Hue utilizando PySpark

resultado_datalake = spark.sql(hive_query)

resultado_pandas_datalake = resultado_datalake.toPandas()

 # Obtener los datos de Oracle y Datalake en forma de listas de tuplas

oracle_data = resultado_pandas_oracle[['CODMES', 'CANTIDAD']].to_records(index=False).tolist()

datalake_data = resultado_pandas_datalake[['CODMES', 'CANTIDAD']].to_records(index=False).tolist()

df_oracle = pd.DataFrame(oracle_data, columns=['CODMES', 'CANTIDAD_ORACLE'])

df_datalake = pd.DataFrame(datalake_data, columns=['CODMES', 'CANTIDAD_DATALAKE'])

 # Unir los DataFrames en uno solo por el campo CODMES

df_result = pd.merge(df_oracle, df_datalake, on='CODMES', how='inner')

 # Guardar los resultados en un archivo Excel

df_result.to_excel("resultado_universo_ora_dl.xlsx", index=False)

2) **Duplicidad de registros en base al identificador de la tabla:**

def validacion_unicidad_dl(spark,table_hue, codmes_list, key_list):

 # Crear un DataFrame para almacenar los resultados

result_df = pd.DataFrame(columns=["CODMES", "Cantidad de Registros Duplicados"])

```
# Realizar la consulta en Hive
df = spark.table(table_hue)
# Recorrer cada codmes para validar la unicidad
for codmes in codmes_list:
    # Filtrar los registros para el codmes actual
    codmes_df = df.filter(df["CODMES"] == codmes)
    # Realizar la agrupacion por todas las llaves
    group_cols = [df[col] for col in key_list]
    grouped_df = codmes_df.groupBy(*group_cols).count().filter("count > 1")
    # Contar la cantidad de registros duplicados para el codmes actual
    duplicates_count = grouped_df.count()
    # Agregar los resultados al DataFrame
    result_df = result_df.append({"CODMES": codmes, "Cantidad de Registros Duplicados": duplicates_count}, ignore_index=True)
# Exportar los resultados a un archivo de Excel
output_file = f"resultados_unicidad.xlsx"
result_df.to_excel(output_file, sheet_name="Resultados", index=False)
print(f"Archivo de Excel generado: {output_file}")
```

3) Integridad de los datos:

```
def comparisons(spark,tablaOracle, tablaHUE, lst_rango_mes, fields_to_compare_ora,fields_to_compare_dl,clavesHUE,clavesOracle,dif_rel,flg_save,flg_save_all_comparison,flg_export_diff,flg_import_from_oracle,sqlQuery_from_oracle,sqlQuery_from_hue,tableName,dbName,location):
    query_case_when = ""
    query_sum = ""
```

query_save = ""

contador=0

 # Validar los datos ingresados

flg_validacion = validar_datos(tablaOracle, tablaHUE, lst_rango_mes, fields_to_compare_ora,fields_to_compare_dl,clavesHUE,clavesOracle,dif_rel)

if not flg_validacion:

return

 # Loop through the months in lst_rango_mes

for codmes_filter in lst_rango_mes:

print(codmes_filter)

#Crear carpeta

create_folder_if_not_exists(codmes_filter)

query_case_when,query_sum,query_save = makeScriptComparison(spark,create_table_oracle(spark,sqlQuery_from_oracle, codmes_filter,flg_import_from_oracle), create_table_dl(spark,sqlQuery_from_hue,codmes_filter), codmes_filter, fields_to_compare_ora,fields_to_compare_dl,clavesHUE,clavesOracle,dif_rel,query_case_when,query_sum,query_save,flg_save,flg_save_all_comparison,flg_export_diff,contador,tableName+"_DIFF",dbName,location)

contador = contador +1

Función validar_datos: Validación de datos

def validar_datos(tablaOracle, tablaHUE, lst_rango_mes, fields_to_compare_ora,fields_to_compare_dl,clavesHUE,clavesOracle,dif_rel):

flg_validacion = False

if len(tablaOracle) == 0 or len(tablaHUE) == 0 or dif_rel < 0:

print("Debe ingresar el nombre de la tabla Oracle, la tabla HUE y el valor de la diferencia relativa mayor o igual a cero")

flg_validacion = False

if len(fields_to_compare_ora) == len(fields_to_compare_dl) and len(fields_to_compare_ora)> 0 and len(fields_to_compare_dl)> 0:

flg_validacion = True

else:

print("La cantidad de campos a comparar en Oracle y HUE deben ser la misma y mayor que cero")

flg_validacion = False

if len(clavesHUE) == 0 or len(clavesOracle) == 0:

print("Debe ingresar al menos una clave para realizar el join")

flg_validacion = False

if len(fields_to_compare_ora) == len(fields_to_compare_dl):

flg_validacion = True

else:

print("La cantidad de campos a comparar en Oracle y HUE debe ser la misma")

flg_validacion = False

if len(lst_rango_mes) == 0:

print("Debe ingresar al menos un mes para realizar la comparacion")

flg_validacion = False

else:

flg_validacion = True

if len(clavesHUE) == len(clavesOracle) and len(clavesHUE) > 0 and len(clavesOracle) > 0:

```
        flg_validacion = True
    else:
        print("La cantidad de claves en Oracle y HUE deben ser la misma longitud y mayor que cero")
        flg_validacion = False
    return flg_validacion
```

Función makeScriptComparison: Script de comparación

```
def makeScriptComparison(spark,data_ora, data_dl, codmes_filter, fields_to_compare_ora,fields_to_compare_dl,clavesHUE,clavesOracle,dif_rel,query_case_when,query_sum,query_save,flg_save,flg_save_all_comparison,flg_export_diff,contador,tableName,dbName,location):
    #print("Esquema DL")
    #data_dl.printSchema()
    if query_case_when == "":
        # Prepare the select statement with case-when conditions for field comparisons
        select_statement = "SELECT\n"
        if flg_save or flg_export_diff:
            # Add the fields from tablaHUE and tablaOracle to the SELECT statement
            for key_dl in clavesHUE:
                select_statement += f"B.{key_dl} as {key_dl}_Hue,\n"
            # Add the fields from tablaHUE and tablaOracle to the SELECT statement
            for field_ora, field_dl in zip(fields_to_compare_ora, fields_to_compare_dl):
```

```
        select_statement += f"A.{field_ora} as {field_ora}_Oracle, B.{field_dl} as {field_dl}_Hue,\n"

    # Loop through both fields_to_compare_ora and fields_to_compare_dl together

    for field_ora, field_dl in zip(fields_to_compare_ora, fields_to_compare_dl):

        tipo1 = data_dl.schema[field_dl].dataType.simpleString()

        tipo2 = data_ora.schema[field_ora].dataType.simpleString()

        tipo_cast = validar_tipo_dato(tipo1, tipo2)

        if is_numeric(str(tipo_cast)):

            select_statement += f"CASE WHEN (B.{field_ora} != 0 AND ABS(1 - (A.{field_dl} / B.{field_ora})) <= {dif_rel}) OR (CAST(A.{field_dl} AS {tipo_cast}) = CAST(B.{field_ora} AS {tipo_cast})) OR (A.{field_dl} IS NULL AND B.{field_ora} IS NULL) THEN 1 ELSE 0 END AS FLG_{field_dl},\n"

        else:

            select_statement += f"CASE WHEN (CAST(A.{field_dl} AS {tipo_cast}) = CAST(B.{field_ora} AS {tipo_cast})) OR (A.{field_dl} IS NULL AND B.{field_ora} IS NULL) THEN 1 ELSE 0 END AS FLG_{field_dl},\n"

    # Finalize the select statement

    select_statement = select_statement.rstrip(",\n")

    select_statement += f"\nFROM data_dl A\nFULL OUTER JOIN data_ora B\n"

    # Build the join condition for the keys (clavesHUE and clavesOracle)

    join_condition = "ON "

    for idx, clave_dl in enumerate(clavesHUE):

        clave_ora = clavesOracle[idx]

        if idx > 0:

            join_condition += " AND "
```

```
            join_condition += f"(A.{clave_dl} = B.{clave_ora})"
        select_statement += join_condition
        # Execute the query and create a temporary view for further analysis
        print(select_statement)
        query_case_when = select_statement
    dfcons = spark.sql(query_case_when)
    dfcons.createOrReplaceTempView("flags")
    print("Esquema FLAGS")
    #Save the results in HUE
    if flg_save:
        if query_save == "":
            # Generate the query to select records with FLG_{field} equal to zero for each field in HUE
            query_save = "SELECT '{}' as CODMES,*\nFROM flags"
            #Filter Low Pass for save only differences
            if flg_save_all_comparison == False:
                query_save += "\nWHERE "
                for idx, field_dl in enumerate(fields_to_compare_dl):
                    if idx > 0:
                        query_save += "OR "
                    query_save += f"FLG_{field_dl} = 0\n"""
            print(query_save)
        # Execute the query to get the records with FLG_{field} equal to zero for each field in HUE
        dfcons = spark.sql(query_save.format(codmes_filter))
```

```
    if contador == 0:
        writeToHivePartitioned(spark,dfcons, tableName,"CODMES",True, True,dbName,location)
    else:
        writeToHivePartitioned(spark,dfcons, tableName,"CODMES",False, False,dbName,location)
  # Uso de la funcion export_queries_to_excel
  if flg_export_diff:
    # Crear un archivo de Excel
    archivo_excel = codmes_filter+"/resultado_diferencias.xlsx"
    libro_excel = Workbook()
    select_statement = "SELECT\n"
    # Add the fields from tablaHUE and tablaOracle to the SELECT statement
    for key_dl in clavesHUE:
        select_statement += f"{key_dl}_Hue,\n"
    temp_select = select_statement
    # Add the fields from tablaHUE and tablaOracle to the SELECT statement
    for field_ora, field_dl in zip(fields_to_compare_ora, fields_to_compare_dl):
        temp_select += f"{field_ora}_Oracle, {field_dl}_Hue FROM flags WHERE FLG_{field_dl} = 0"
        resultado_pandas = spark.sql(temp_select).toPandas()
        # Crear una nueva hoja en el archivo de Excel y guardar el resultado
        hoja = libro_excel.create_sheet(title=field_dl)
        for row in dataframe_to_rows(resultado_pandas, index=False, header=True):
```

```
            try:
                hoja.append(row)
            except Exception as e:
                # Si ocurre un IllegalCharacterError, guardar la informacion en un archivo de log
                mensaje_error = f"Se produjo un error en la hoja '{field_dl}': {str(e)}. guardando en hoja en texto."
                print(mensaje_error)
                row_str = [str(value) for value in row]
                with open(codmes_filter+f"/{field_dl}_error_log.txt", "w") as log_file:
                    log_file.write(row_str)
        # Reiniciar la variable select_statement
        temp_select = select_statement
    # Guardar el archivo de Excel
    libro_excel.save(archivo_excel)
if query_sum == "":
    # Prepare the final select statement to get summary results
    final_select = "SELECT\n"
    for field in fields_to_compare_dl:
        final_select += f"SUM(FLG_{field}) as {field},\n"
    final_select += f"count(*) as totales\nFROM flags"
    # Execute the final query to get the summary results
    print(final_select)
    query_sum = final_select
```

```
final = spark.sql(query_sum)
 # Convert the Spark DataFrame to a pandas DataFrame
final = final.toPandas()
# Mostrar el DataFrame
#print(salida)
# Obtener el valor de la variable "totales"
total_comparados = final.loc[0, "totales"]
#print("Valor de la variable totales:", total_comparados)
# Inicializar una lista para almacenar los resultados de cada campo
results = []
# Recorrer cada columna y calcular las estadisticas
for col_name in final.columns:
    # Excluir la columna "totales" de las estadisticas
    if col_name != 'totales':
        # Contar el numero de coincidencias y diferencias para el campo actual
        coincidencias = final.loc[0, col_name]
        diferencias = total_comparados - coincidencias
        # Calcular los porcentajes de coincidencias y diferencias
        pct_coincidencias = (coincidencias / total_comparados) * 100
        pct_diferencias = 100 - pct_coincidencias
        diferencias = total_comparados - coincidencias
        # Agregar los resultados a la lista
        results.append([col_name, col_name, total_comparados, coincidencias, diferencias, pct_coincidencias, pct_diferencias])
```

results = pd.DataFrame(results, columns=['Variable HUE','Variable Oracle', 'Conteo', '# Coincidencias', '# Diferencias', '% Coincidencias', '% Diferencias'])

results['Tipo']='Relativa'

results = results.round(4)

#Generar PDF
PDF_Integridad(results,tableName,tableName,clavesHUE,clavesOracle,dif_rel, codmes_filter)

return query_case_when,query_sum,query_save

4.1.3.4 Pruebas

Con la finalidad de testear el script de comparación se realizaron pruebas considerando bases con cantidad de registros pequeños; estos resultados se compararon con una comparación manual que se realizó en Excel.

Para las bases con gran volumen de registros y columnas se descargó las tablas a comparar en la herramienta de SAS y se usó la macro de comparación; en donde, se llegó a la conclusión que el script estaba generando los resultados adecuados en las coincidencias y diferencias por cada variable.

4.1.3.5 Implementación

Posterior a las pruebas aplicadas al script de comparación se generaron las coordinaciones para desplegar el código a todas las próximas tablas que pasarán por una validación; así mismo, se gestionó la publicación del script en las herramientas estandarizadas de la empresa bancaria para mantener una única versión vigente. Así mismo, se elaboró una documentación para reflejar el flujo de la ejecución del script de comparación y los controles que a la fecha se realizan.

4.1.3.6 Cronograma del desarrollo del trabajo

Fase	Actividades	Semana 1	Semana 2	Semana 3	Semana 4	Semana 5	Semana 6
Análisis	*Detectar la posible solución de la problemática*						
	Preparar el entorno de trabajo						
Diseño	*Definir el diseño del reporte de resultados de la comparación*						
Desarrollo	*Codificar la comparación de la integridad de los datos*						
	Codificar la comparación de completitud de universos						
	Codificar la comparación de duplicidad de la información						
Pruebas	Generar pruebas para verificar la correcta comparación						
Implementación	Pase a producción del código comparativo						

Fuente: Elaboración propia

Todo el proyecto tiene un estimado de 1 mes y medio (6 semanas)

CAPITULO V: PRUEBA DE DISEÑO

5.1 Aplicación de la propuesta de solución

El resultado de la comparación beneficia significativamente a la empresa de rubro bancario, ya que previo a la creación del script de comparación el tiempo empleado de validación era entre 1 a 3 semanas dependiendo la cantidad de registros de la tabla y la cantidad de campos a comparar ya que era una validación no directa a través de uso de Excel y de herramientas externas como SAS. Sin embargo; posterior a la implementación del script de comparación el proceso de comparación para validar la migración de Oracle a Datalake toma de 1 a 2 días nuevamente dependiendo la cantidad de registros de la tabla y la cantidad de campos a comparar. Pero es un tiempo bastante óptimo ya que el script está automatizado y parametrizado lo que permite generar modificaciones únicamente sobre las tablas y variables que se van a comparar; por ende, agiliza la validación y los usuarios internos de la empresa pueden hacer uso de la información la cual ya es confiable porque pasó con una revisión de validación interna.

CONCLUSIONES

La Validación de la migración de datos sobre una empresa de rubro bancario a través de controles elaborados en lenguaje de programación Python verificó su confiabilidad.

La validación de datos en una migración se realiza a través de la completitud de universo para confirmar que los registros sigan siendo los mismos, mediante la integridad de los datos para verificar que nos existan cambios en los valores asignados a cada registro y bajo un análisis de duplicidad para que cada registro sea único en base a la llave identificadora o primaria.

La validación habilita el uso de los datos en la organización y esto brinda la seguridad que la data es consistente para el consumo de los usuarios internos de la empresa.

RECOMENDACIONES

Se recomienda que ante cualquier inclusión de algún control para mejorar la validación se mantenga la estructura de uso de parámetros generales y se busque siempre la reducción de tiempos en la revisión. Además; reflejar en la documentación elaborada los cambios realizados en el script de comparación para siempre mantener lo vigente para la consulta de alguna auditoría o integrante del equipo que requiera comprender el proceso.

REFERENCIAS BIBLIOGRAFICAS

(Carsí, Ramos, Silva, Pérez y Anaya, 2002)
J. A. Carsí, I. Ramos, J. Silva, J. Pérez, V. Anaya (2002) Un Generador Automático de Planes de Migración de Datos

Saborío y Chinchilla (2014)
José Iván Saborío-Acuña, Ricardo Chinchilla-Arley (2014) Metodología para la migración de datos bibliográficos entre programas de software de automatización: de CEPAL a MARC

(Chipana, 2023)
Gherard Anthony Chipana Quiñones (2023) Implementación de un Sistema de Automatización de Pruebas con
Karate Framework para Mejorar el Proceso de Testing en una Empresa del Rubro Bancario, Lima, 2023

(Calderon y Rios, 2018)
Calderon Luna Russell, Ríos Espinoza Anghelo Jefferson (2018) Auditoria gubernamental y la auditoria de cumplimiento en la fiscalización y control de calidad en la gestión de las entidades públicas del distrito de Yanacancha periodo 2018

(Pasini, Torres, Esponda y Pesado, 2021)
Pasini Ariel Cristian, Torres Juan Ignacio, Esponda Silvia, Pesado Patricia Mabel (2021) Calidad de datos aplicada a la base de datos abierta de casos registrados de COVID-19

(Masip, 2018)
David Masip Rodó (2018) El lenguaje Python

(Barón, 2018)
Barón Alfaro Johan Andrey (2018) Desarrollo de librería en Python para el control de instrucciones sobre el módulo de comunicaciones SIM5320A

(Borjas, 2020)
Jorge Edgardo Borjas García (2020) Validez y confiabilidad en la recolección y análisis de datos bajo un enfoque cualitativo