**UNIVERSIDAD POLITÉCNICA DE SINALOA**

**PROGRAMA ACADÉMICO DE**

**INGENIERÍA EN INFORMÁTICA**

Tesina

“Generación de Base de Datos para la aplicación de Minería de Datos”

Para obtener la acreditación de las estadías profesionales y contar con los créditos necesarios para el grado de Ingeniero en Informática

**Autor: Kristian de Jesús López Montes**

**Asesor: M.C. Ramón Patricio Velázquez Cuadras**

**Asesor OR: Dr. Gilberto Martínez Luna**

Mazatlán, Sinaloa 8 de Diciembre de 2014.

Índice Temático

[Agradecimientos 7](#_Toc406757241)

[Resumen 8](#_Toc406757242)

[Abstract 8](#_Toc406757243)

[Introducción 9](#_Toc406757244)

[Capítulo I 10](#_Toc406757245)

[1.1 Antecedentes 10](#_Toc406757246)

[1.1.1 Localización 10](#_Toc406757247)

[1.1.2 Objetivos y prioridades de la institución 13](#_Toc406757248)

[1.1.3 Organigrama 14](#_Toc406757249)

[1.1.3.1 Misión 14](#_Toc406757250)

[1.1.3.2 Visión 15](#_Toc406757251)

[1.2 Planteamiento del problema 15](#_Toc406757252)

[1.2.1 Propuesta de investigación 15](#_Toc406757253)

[1.2.2.1 Objetivos Generales 15](#_Toc406757254)

[1.2.2.2 Objetivos Específicos 15](#_Toc406757255)

[1.2.3 Preguntas de investigación 15](#_Toc406757256)

[1.2.4 Hipótesis 16](#_Toc406757257)

[1.2.5 Limitaciones y supuestos 16](#_Toc406757258)

[1.2.6 Relevancia 16](#_Toc406757259)

[Capitulo II 17](#_Toc406757260)

[2.1 Marco teórico 17](#_Toc406757261)

[2.1.1 Antecedentes 17](#_Toc406757262)

[2.1.2 Herramientas 18](#_Toc406757263)

[Capitulo III 20](#_Toc406757264)

[3.1 Manejo de la información en Copa Airlines 20](#_Toc406757265)

[3.2 Automatización de Procesos 20](#_Toc406757266)

[3.2.1 Jobs (shells) 20](#_Toc406757267)

[3.3 Utilerías Teradata (bteq, tpt) 21](#_Toc406757268)

[3.3.1 Archivos de configuración 22](#_Toc406757269)

[3.4 Desarrollo 23](#_Toc406757270)

[3.4.1 Flujos ETL\_PROD 23](#_Toc406757271)

[3.4.1.1 Estructura general de un flujo ETL\_PROD 23](#_Toc406757272)

[3.4.1.2 Flujos de extracción desde fuente de datos 23](#_Toc406757273)

[3.4.1.3 Flujos de transformación y carga 24](#_Toc406757274)

[3.4.1.4 Secuencia de carga 25](#_Toc406757275)

[3.4.1.5 Descripción de flujos implementados 30](#_Toc406757276)

[3.4.1.6 Detalle interno de jobs 31](#_Toc406757277)

[3.5 Seguimiento a la ejecución de procesos 79](#_Toc406757278)

[3.5.1 Mecanismos para monitoreo de ejecución de procesos 79](#_Toc406757279)

[3.5.2 Logs de sistema operativo 79](#_Toc406757280)

[3.5.3 Bitácora ETL\_PROD 79](#_Toc406757281)

[3.5.4 Viewpoint 81](#_Toc406757282)

[3.5.5 Monitor de procesos 82](#_Toc406757283)

[3.6 Incidencias comunes y resolución 88](#_Toc406757284)

[3.6.1 Caso 1: “La tabla está siendo cargada” 88](#_Toc406757285)

[3.6.2 Caso 2: “Error en la carga” 89](#_Toc406757286)

[3.6.3 Caso 3: “Error diferencia en archivo cifras” 91](#_Toc406757287)

[Bibliografía 95](#_Toc406757288)

[Anexos 96](#_Toc406757289)

[Anexo 1 96](#_Toc406757290)

[Anexo 2 97](#_Toc406757291)

[Anexo 3 98](#_Toc406757292)

[Anexo 4 99](#_Toc406757293)

[Glosario 100](#_Toc406757294)

**Índice de Figuras**

[Figura 1. Archivo de parámetros 22](#_Toc405990330)

[Figura 2. Fases de proceso ETL\_PROD 23](#_Toc405990331)

[Figura 3. Flujo de invocaciones en el proceso de extracción 24](#_Toc405990332)

[Figura 4. Estructura general de flujos de transformación 24](#_Toc405990333)

[Figura 5. Secuencia de carga 25](#_Toc405990334)

[Figura 6. Malla de ejecución flujos 10000 30](#_Toc405990335)

[Figura 7. Diagrama de flujo del Job10001 32](#_Toc405990336)

[Figura 8. Diagrama de flujo del Job10002 33](#_Toc405990337)

[Figura 9. Diagrama de flujo del Job10003 34](#_Toc405990338)

[Figura 10. Diagrama de flujo del Job10004 35](#_Toc405990339)

[Figura 11. Diagrama de flujo del Job10005 36](#_Toc405990340)

[Figura 12. Diagrama de flujo del Job10006 37](#_Toc405990341)

[Figura 13. Diagrama de flujo del Job10007 38](#_Toc405990342)

[Figura 14. Diagrama de flujo del Job10008 39](#_Toc405990343)

[Figura 15. Diagrama de flujo del Job10009 40](#_Toc405990344)

[Figura 16. Diagrama de flujo del Job10010 41](#_Toc405990345)

[Figura 17. Diagrama de flujo del Job10011 42](#_Toc405990346)

[Figura 18. Diagrama de flujo del Job10012 43](#_Toc405990347)

[Figura 19. Diagrama de flujo del Job10013 44](#_Toc405990348)

[Figura 20. Diagrama de flujo del Job10014 45](#_Toc405990349)

[Figura 21. Diagrama de flujo del Job10015 46](#_Toc405990350)

[Figura 22. Diagrama de flujo del Job10016 47](#_Toc405990351)

[Figura 23. Diagrama de flujo del Job10017 49](#_Toc405990352)

[Figura 24. Diagrama de flujo del Job10019 51](#_Toc405990353)

[Figura 25. Diagrama de flujo del Job10020 53](#_Toc405990354)

[Figura 26. Diagrama de flujo del Job10021 55](#_Toc405990355)

[Figura 27. Diagrama de flujo del Job10022 57](#_Toc405990356)

[Figura 28. Diagrama de flujo del Job10023 59](#_Toc405990357)

[Figura 29. Diagrama de flujo del Job10024 61](#_Toc405990358)

[Figura 30. Diagrama de flujo del Job10025 62](#_Toc405990359)

[Figura 31. Diagrama de flujo del Job10026 63](#_Toc405990360)

[Figura 32. Diagrama de flujo del Job10027 64](#_Toc405990361)

[Figura 33. Diagrama de flujo del Job10028 66](#_Toc405990362)

[Figura 34. Diagrama de flujo del Job10029 69](#_Toc405990363)

[Figura 35. Diagrama de flujo del Job10030 70](#_Toc405990364)

[Figura 36. Diagrama de flujo del Job10031 72](#_Toc405990365)

[Figura 37. Diagrama de flujo del Job10032 73](#_Toc405990366)

[Figura 38. Diagrama de flujo del Job10033 74](#_Toc405990367)

[Figura 39. Diagrama de flujo del Job10034 75](#_Toc405990368)

[Figura 40. Diagrama de flujo del Job10035 76](#_Toc405990369)

[Figura 41. Diagrama de flujo del Job10036 77](#_Toc405990370)

[Figura 42. Diagrama de flujo del Job10037 78](#_Toc405990371)

[Figura 43. Bitácora ETL\_PROD 80](#_Toc405990372)

[Figura 44. Interfaz Viewpoint 82](#_Toc405990373)

[Figura 45. Monitor de procesos 83](#_Toc405990374)

[Figura 46. Desbloqueo de pestaña Área Sujeto Detalle 84](#_Toc405990375)

[Figura 47. Adición de nuevo Job 84](#_Toc405990376)

[Figura 48. Copia de fórmulas 84](#_Toc405990377)

[Figura 49. Mostrar celdas 85](#_Toc405990378)

[Figura 50. Adición de fórmulas para Monitor 85](#_Toc405990379)

[Figura 51. Mostrar pestañas escondidas 86](#_Toc405990380)

[Figura 52. Pestaña Source1 86](#_Toc405990381)

[Figura 53. Ajustes en pestaña Source1 86](#_Toc405990382)

[Figura 54. Ocultar pestaña Source1 87](#_Toc405990383)

[Figura 55. Ocultar celdas 87](#_Toc405990384)

[Figura 56. Bloqueo de pestaña Área Sujeto Detalle 87](#_Toc405990385)

[Figura 57. Mensaje de error – la tabla está siendo cargada 88](#_Toc405990386)

[Figura 58. Queries para resolución de error la tabla está siendo cargada en SQL Assistant 89](#_Toc405990387)

[Figura 59. Ejemplo de bitácora de ejecución para el Job10009 90](#_Toc405990388)

[Figura 60. Carpeta de logs, servidor ETL\_PROD 90](#_Toc405990389)

[Figura 61. Bitácora de ejecución para el Job01048 91](#_Toc405990390)

[Figura 62. Ejemplo de archivo de log para diferencia de cifras 92](#_Toc405990391)

# Agradecimientos

Para poder realizar ésta tesina de la mejor manera posible fue necesario del apoyo de muchas personas a las cuales quiero agradecer.

En primer lugar a mi madre, Rosa María Montes, quien ha sido un gran apoyo moral y económico para lograr este fin. Gracias por su paciencia.

En segundo lugar a mi esposa, Elizabeth Cervantes quien me ha apoyado incondicionalmente en este último año de la carrera.

A mis asesores de tesina, personas que admiro por su inteligencia, perseverancia y dedicación, Profesor Gilberto Martínez Luna y Ramón Patricio Velázquez Cuadras, a quienes les debo el hecho de que ésta tesina tenga los menos errores posibles.

A mis amigos Juan Cárdenas, Jose Duarte, Abel Angulo que continúan apoyándome a pesar de tantos tropiezos que eh tenido y que gracias a ellos eh seguido adelante.

Kristian de Jesús López Montes

# Resumen

En el presente trabajo se realiza el análisis y diseño de un modelo analítico de bases de datos. Utilizando principalmente Teradata Database Software el cual es un motor de base de datos relacional. En el presente documento estará orientado al procesamiento de grandes volúmenes de datos, en lo cual consiste en la compresión de datos, realizando distintos procesos. Además de cómo crear base de datos, roles, usuarios y el manejo de un usuario como una base de datos para un mejor desempeño en el Data Warehouse.

# Abstract

In the following work the analysis and design of an analytical database model is realized, mainly making use of Teradata Database Software, which is a relational database management system. This document is based on great data volumes, data compression, and the execution of different process, this includes creating data bases, roles, users, and a user management as a database, this for a better performance in the Data Warehouse.

**Palabras claves:**

Teradata, Data Warehouse, Minería de datos

# Introducción

La solución de EDW implementada en Copa está integrada por varios componentes que interactúan de forma complementaria para poder llevar datos de la fuente hacia el DWH (Data Warehouse). Cada componente tiene una función específica dentro del entorno y sigue un conjunto de reglas de operación que deben mantenerse para asegurar el correcto funcionamiento de la plataforma.

El objetivo de este documento es proporcionar una vista integral de dichos componentes y la forma en la que se relacionan para ser utilizado como guía en los procesos de operación de la plataforma.

Actualmente se está realizando una mejora en un Data Warehouse, en el cual se necesita manipular información para generar mayor tráfico de información en el menor tiempo de respuesta. Además de realizar modelos de negocio mucho más eficientes.

Se crearan distintos procesos que estarán entrelazados para un mejor manejo de la información, aun lado a esto se trabajara con archivos de tipo Shell para un manejo de datos interna.

Principalmente se utilizaran queries para la el manejo de información y el uso de una plataforma de monitoreo para la verificación de los queries y otros procesos.

# Capítulo I

## 1.1 Antecedentes

### 1.1.1 Localización

**1.1.1.1. Macro localización**

**Superficie Continental e Insular del Territorio Nacional**

**Extensión**

La extensión territorial de México, es de 1 964 375 Km² de los cuales 1 959 248 Km² son superficie continental y 5 127 Km² corresponden a superficie insular.

**Fronteras**

La República Mexicana tiene fronteras con los Estados Unidos de América, Guatemala y Belice, a lo largo de un total de 4 301 Km distribuidos de la siguiente forma:

1. Con los Estados Unidos de América, se extiende una línea fronteriza a lo largo de 3, 152 Km desde el Monumento 258 al noroeste de Tijuana hasta la desembocadura del Río Bravo en el Golfo de México. Son estados limítrofes al norte del país: Baja California, Sonora, Chihuahua, Coahuila, Nuevo León y Tamaulipas.
2. La línea fronteriza con Guatemala tiene una extensión de 956 Km.; con Belice de 193 Km. (No incluye 85 266 Km. de límite marítimo en de Chetumal). Los estados fronterizos del sur y sureste del país son: Chiapas, Tabasco, Campeche y Quintana Roo.

**Litorales**

México destaca entre los países del mundo por la extensión de sus litorales, que es de 11 122 Km., exclusivamente en su parte continental, sin incluir litorales insulares.

Los resultados fueron obtenidos por medio de diversos procesos digitales semiautomatizados que proporcionan una mayor exactitud de las dimensiones del espacio territorial mexicano. La proyección cartográfica equiárea con base en la que se hacen los cálculos de áreas, ofrece valores mejorados sustancialmente en exactitud si se comparan con los que forman el antecedente que da origen a este documento.

**Distrito Federal**

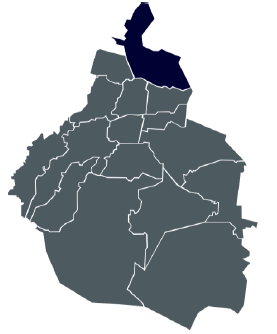
Distrito Federal



La Ciudad de México, Distrito Federal, o en su forma abreviada México, D. F., es la capital y sede de los poderes federales de los Estados Unidos Mexicanos.[9](http://es.wikipedia.org/wiki/México,_D._F.#cite_note-9) Se trata de una entidad federativa de México que no forma parte de los 31 estados mexicanos, pero pertenece a la Federación, que en conjunto conforman las 32 entidades federativas de la nación. La Ciudad de México es el núcleo urbano más grande del país, así como el principal centro político, académico, turístico, económico, financiero, empresarial y cultural.

**Gustavo A. Madero**

Delegación Gustavo A. Madero



La Delegación Gustavo A. Madero se ubica en el extremo noreste del Distrito Federal; ocupa una posición estratégica con respecto a varios municipios conurbados del Estado de México.

**1.1.1.2 Micro localización**

En las instalaciones del Centro de Investigación en Computación, Laboratorio de Bases de Datos y tecnología de Software.

**NOMBRE DE LA INSTITUCION:**

CENTRO DE INVESTIGACION EN COMPUTO (CIC)

**DOMICILIO:**

Av. Juan de Dios Bátiz s/n casi esq. Miguel Othón de Mendizábal.Unidad Profesional Adolfo López Mateos Col. Nueva Industrial Vallejo, Delegación Gustavo A. Madero C.P 07738, México D.F.

**INFORMACION SOBRE LA INSTITUCIÓN**

**GIRO:**

Investigación en computación



### 1.1.2 Objetivos y prioridades de la institución

La creación del Centro Nacional de Cálculo (CeNaC), en 1963, fue uno de los primeros esfuerzos realizados para incorporar la computación electrónica al acervo científico y tecnológico del país. Para 1965, se comenzaron a impartir estudios de postgrado, con la Maestría en Ciencias de la Computación.

El 24 de febrero de 1988 fue creado el Centro de Investigación Tecnológica en Computación (CINTEC), con el objeto de formar recursos humanos en el nivel de postgrado, en el área de la ingeniería y ciencias de la computación, para desarrollar los procesos tecnológicos de los sectores productivo y social del país. En 1989, en el CINTEC se inició la impartición de la Maestría en Ingeniería de Cómputo

El 20 de marzo de 1996, invitando científicos de reconocido prestigio en los ámbitos nacional e internacional y tomando funciones y recursos del Centro Nacional de Cálculo (CeNaC) y del Centro de Investigación Tecnológica en Computación (CINTEC), fue creado el Centro de Investigación en Computación (CIC), con el objeto de realizar investigación de vanguardia; desarrollar, implantar, aprovechar y fortalecer los sistemas de cómputo en apoyo a la docencia e investigación, actualización, especialización, y superación académica profesional, y estudios de maestría y doctorado. Éste, comenzó a laborar en las instalaciones del CeNaC.

Actualmente el Centro Nacional de Cálculo realiza sus funciones en el Edificio de la Secretaría de Administración y el Centro de Investigación Tecnológica (CINTEC) cambió su denominación a CIDETEC

El 28 de febrero de 1997 el Centro de Investigación en Computación, inaugura las instalaciones con que opera actualmente.

1.1.3 Organigrama

Organigrama



1.1.3.1 Misión

El Centro de Investigación en Computación del Instituto Politécnico Nacional, realiza investigación científica y tecnológica, así como la formación de recursos humanos en el nivel posgrado, en las áreas de Ciencias de la Computación e Ingeniería de Cómputo, para atender las necesidades planteadas por los sectores educativo, productivo y de servicios del país. Para lograrlo, emplea una estructura académica y administrativa que ofrece programas de posgrado y desarrolla proyectos de investigación científica y tecnológica con calidad, responsabilidad, ética, tolerancia y compromiso social.

1.1.3.2 Visión

Ser el Centro líder en Investigación en Ciencias de la Computación e Ingeniería de Cómputo, que desarrolle investigación científica y tecnológica comprometida con la sociedad, reconocido por el liderazgo de sus egresados en sus áreas de competencia, por imponer estándares de calidad y por contribuir al desarrollo nacional mediante las ideas generadas por su comunidad.

1.2 Planteamiento del problema

1.2.1 Propuesta de investigación

Crear y mejorar los procesos de almacenamiento y tratamientos de datos para Copa Airlines en el área de Sistemas, utilizando principalmente un Data Warehouse y otros servidores, los cuales tendrán la funcionalidad de respaldar la información, tener el área de desarrollo para poder realizar las consultas y las pruebas necesarias, y posteriormente pasar al área de producción.

1.2.2.1 Objetivos Generales

Optimizar el manejo de datos dentro del servidor de Copa Airlines, para obtener una respuesta inmediata, crear procesos para ayudar a la manipulación de información.

1.2.2.2 Objetivos Específicos

1.- Extraer la información del sistema gestor de base de datos principal, filtrarla y almacenarla de manera organizada dentro de un Data Warehouse.

2.- Implementación de Job que se encuentren entre lazados, ejecutándose de manera paralela con otros job.

3.- El uso de Basic Teradata Query y Teradata Parallel Transporter.

4.- Obtener la información en un servidor el cual tiene como finalidad respaldar la información y duplicarla en otros servidores para respaldo.

5.- Por medio de herramientas de teradata se da seguimiento a los Jobs.

1.2.3 Preguntas de investigación

¿Cómo se puede mejorar el flujo de la información de Copa Airlines?

¿Cómo automatizar procesos creados para Copa Airlines?

¿Qué herramientas serán útiles para los procesos?

¿Cuál será la finalidad de tener la información ordenada?

¿Cómo se monitorea el flujo y funcionamiento de un Job?

1.2.4 Hipótesis

Si se desarrollan consultas más eficientes para Copa Airlines, mostrando una respuesta inmediata, entonces se podrá dar mejor servicio a sus clientes e incrementar ventas. Si se realizan procesos automáticos que estén programados para ciertos momentos del día, entonces se agilizará el manejo de la información dentro de los servidores. Si se almacena la información en más de un servidor, entonces la información estará disponible en un 99.9% del tiempo. Si se cuenta con la información ordenada, entonces se podrán crear reportes más eficientes y específicos para que la empresa tenga mayor productividad.

1.2.5 Limitaciones y supuestos

Al tener la conexión al Data Warehouse se trabaja dentro de un servidor que tiene la funcionalidad de stating.

El uso de varios servidores lograra ser producente para un mejor manejo de información, además de manejar respaldos.

1.2.6 Relevancia

Optimización de procesos, almacenamiento y manipulación de información, para la generación de reportes diarios, semanales y mensuales. Para un análisis del crecimiento que ha tenido el Data Warehouse.

# Capitulo II

## 2.1 Marco teórico

### 2.1.1 Antecedentes

Un Data Warehouse es una base de datos corporativa que se caracteriza por integrar y depurar información de una o más fuentes, para posteriormente procesarla permitiendo su análisis desde muchas perspectivas y con una gran velocidad de respuesta casi inmediata. La creación de un Data Warehouse representa en la mayoría de las ocasiones el primer paso, desde el punto de vista técnico, para implantar una solución completa y fiable de Business Intelligence (BI).

Una de las principales ventajas de este tipo de bases de datos radica en las estructuras en las que se almacena la información (modelos de tablas en estrella, en copo de nieve, cubos relacionales). Este tipo de procesamiento de la información es homogénea y fiable, y permite la consulta y el tratamiento jerarquizado de la misma (siempre en un entorno diferente a los sistemas operacionales).

Las principales características de un Data Warehouse son:

**Integrado**: Los datos almacenados en el Data Warehouse deben integrarse en una estructura consistente, en caso de existir inconsistencias entre diversos sistemas de operación se eliminan. La información suele estructurarse en distintos niveles de detalle para adecuarse a las distintas necesidades.

**Temático**: Sólo los datos necesarios para el proceso de generación del conocimiento del negocio se integran desde el entorno operacional. Los datos se organizan por temas para facilitar su acceso y entendimiento por parte de los usuarios finales.

**Histórico**: El tiempo es parte implícita de la información contenida en un Data Warehouse. En los sistemas operacionales, los datos siempre reflejan el estado de la actividad del negocio en el momento presente. Por el contrario, la información almacenada en el Data Warehouse sirve, entre otras cosas, para realizar análisis de tendencias. Por lo tanto, el Data Warehouse se carga con los distintos valores que toma una variable en el tiempo para permitir comparaciones.

**No volátil**: El almacén de información de un Data Warehouse existe para ser leído, pero no modificado. La información es por tanto permanente, significando la actualización del Data Warehouse la incorporación de los últimos valores que tomaron las distintas variables contenidas en él sin ningún tipo de acción sobre lo que ya existía.

Una de las claves del éxito en la construcción de un Data Warehouse es el desarrollo de forma gradual, seleccionando a un departamento usuario como piloto y expandiendo progresivamente el almacén de datos a los demás usuarios. Por ello es importante elegir este usuario inicial o piloto, siendo importante que sea un departamento con pocos usuarios, en el que la necesidad de este tipo de sistemas es muy alta y se pueda obtener y medir resultados a corto plazo.

Principales aportaciones de un Data Warehouse:

1. Proporciona una herramienta para la toma de decisiones en cualquier área funcional, basándose en información integrada y global del negocio.
2. Facilita la aplicación de técnicas estadísticas de análisis y modelización para encontrar relaciones ocultas entre los datos del almacén; obteniendo un valor añadido para el negocio de dicha información.
3. Proporciona la capacidad de aprender de los datos del pasado y de predecir situaciones futuras en diversos escenarios.
4. Simplifica dentro de la empresa la implantación de sistemas de gestión integral de la relación con el cliente.
5. Supone una optimización tecnológica y económica en entornos de Centro de Información, estadística o de generación de informes con retornos de la inversión espectaculares.

### 2.1.2 Herramientas

La compresión de datos consiste en la aplicación de alguna técnica para reducir el volumen de los mismos, siendo el espacio ganado denominado beneficio de compresión. La aplicación de alguna técnica de compresión requiere a su vez de un tiempo de procesamiento al que podemos denominar costo de compresión.

La compresión en bases de datos estuvo limitada a técnicas que no tengan pérdidas reales de información y que además tengan un bajo costo de compresión, en otras palabras, limitadas a eliminar la información redundante.

La compresión MVC en Teradata v13 no es un algoritmo de reducción de datos sino que su función es eliminar el almacenamiento de nulos y de un conjunto limitado de valores constantes los cuales deben ser definidos por el usuario.

El funcionamiento es sencillo: para cada valor que se establezca en la definición del compress (además del nulo), Teradata establecerá un valor el cual se almacenará en el encabezado de la tabla.

Cada vez que se requiera leer o escribir alguno de estos valores, Teradata apuntará al encabezado de la tabla el cual al encontrarse en memoria es de muy rápido acceso.

En Teradata v14 selecciona automáticamente el mejor método de compresión y dinámicamente adapta el mecanismo según evolucionen los datos en el tiempo.

Esto representa un gran cambio frente a la gestión manual de la compresión que había en versiones anteriores. Los 6 métodos de compresión disponibles para el almacenamiento por columnas son length, dictionary, trim, delta on mean, null y UTF8 aunque también se puede aplicar la compresión a nivel de bloque en conjunto con alguna de estas.

# Capitulo III

## 3.1 Manejo de la información en Copa Airlines

Los flujos de trabajo implementados en Copa Airlines utilizan DataStage para la organización de los procesos bajo una estructura estándar y Control-M para calendarizar su ejecución. Dicha estructura se compondrá de diferentes fases que permiten que los datos lleguen de la fuente hacia el modelo de datos definido en el Data Warehouse (DWH) a través de la ejecución de diferentes acciones en el sistema operativo o dentro de la base de datos.

De forma general, DataStage invocará scripts implementados en el servidor ETL\_PROD que a su vez se valdrán de utilerías de Teradata para llevar los datos hacia la base de datos y realizar su movimiento a través de las diferentes fases. Dicho lo anterior, los componentes básicos de un flujo son los Jobs (shells), bteqs, tpt, y los archivos de configuración o archivos de parámetros.

## 3.2 Automatización de Procesos

### 3.2.1 Jobs (shells)

Para poder programar la ejecución recurrente de los paquetes, se crearon Jobs que pueden contener uno o más pasos que se ejecutan en un orden particular. Dichos Jobs invocan shells de UNIX que a su vez realizarán funciones de manipulación de archivos o, mediante utilerías de Teradata, la ejecución de instrucciones de base de datos.

En la solución se ha asignado una numeración particular a los shells para identificar a qué fase del proceso de carga corresponden:

* **1XXXX**[[1]](#footnote-1) – Corresponde a los flujos padre. Estos son los invocados por Control – M e incluyen las invocaciones de uno o más shells.
* **0XXXX** – Corresponden a procesos de extracción de datos de algunas de las fuentes para crear archivos que serán usados como interfaces.
* **01XXX** – Corresponden a flujos de carga a Staging de las interfaces.
* **02XXX** – Corresponden a flujos de homologación.
* **03XXX** – Corresponden a flujos de transformación y carga a imagen.
* **04XXX** – Corresponden a flujos de carga al DWH.
* **05XXX** – Corresponden a flujos de inicialización para generar los parámetros con los cuales se ejecutará el proceso durante ese día.

Los pasos para los flujos construidos son descritos a mayor detalle en la siguiente sección.

## 3.3 Utilerías Teradata (bteq, tpt)

Para realizar la carga y movimiento de datos hacia y en el Data Warehouse se utilizan dos utilerías de Teradata: tpt y bteq.

**BTEQ** (Basic Teradata Query) es una herramienta de la BD Teradata utilizada para enviar consultas de SQL en todas las plataformas. BTEQ proporciona la siguiente funcionalidad:

* Escritura y formato de informes estándar.
* Importación y exportación básica de pequeñas cantidades de datos desde y hacia la BD Teradata en todas las plataformas. Para las tablas que contienen más de varios miles de filas, se recomiendan las utilidades de carga de la BD Teradata para obtener mayor eficacia.
* Capacidad para enviar solicitudes de SQL de las siguientes formas:
  + Interactiva
  + Lote

En esta implementación los BTEQ se utilizan para realizar la carga de datos a las tablas históricas de Staging y para ejecutar el movimiento de datos entre las diferentes fases requeridas para la carga en el modelo de datos del Data Warehouse.

**TPT** (Teradata Parallel Transporter) es una utilería que permite cargar datos hacia y exportar datos de una base de datos Teradata. El tpt utiliza diferentes operadores para realizar funciones de extracción, transformación y carga de datos. Con los operadores soportados es posible:

* Funciones de extracción de datos (Operador EXPORT).
  + Obtener datos de la base de datos Teradata o de una base de datos externa.
  + Generar datos internamente.
  + Enviar datos a otros operadores a través de una interfaz.
* Funciones de carga de datos (Operador LOAD).
  + Aceptar datos de otros operadores a través de una interfaz.
  + Cargar datos a una base de datos Teradata o a una base de datos externa.
* Funciones de filtrado de datos como selección, validación, limpieza y consolidación.

En esta implementación, el tpt se utiliza para cargar las interfaces de datos fuente a Staging de Teradata.

### 3.3.1 Archivos de configuración

Dentro de la solución, se cuenta con dos tipos de archivos de configuración: los de parámetros y los de ejecución. Los primeros mantienen la lógica de ejecución para el flujo, los segundos, variables que son utilizadas en tiempo de ejecución para la operación del flujo.

Los archivos de parámetros son los que indican para cada uno de los flujos el esquema de ejecución, el archivo que se tomará para la carga, así como la tabla destino de los datos y la secuencia en la que cada paso será llevado a cabo.



Figura 1. Archivo de parámetros

Los archivos de ejecución corresponden a las plantillas con una estructura base asociada a las tablas que serán cargadas y datos variables que son llenadas en tiempo de ejecución y que dan información más específica sobre la ubicación de las tablas que serán afectadas por ese proceso. Parte de esta información variable es tomada de los archivos de parámetros.

## 3.4 Desarrollo

### 3.4.1 Flujos ETL\_PROD

### 3.4.1.1 Estructura general de un flujo ETL\_PROD

Típicamente, los flujos implantados se conformarán de los siguientes módulos que se alinean a las fases por las que pasa la información para ingresar al DWH[[2]](#footnote-2).



Figura 2. Fases de proceso ETL\_PROD

Como parte de la solución existen flujos padre (job1xxxx) que controlan la secuencia de pasos que conforman los flujos de carga. Dichos pasos incluirán a su vez las invocaciones a shells que ejecutarán acciones en el servidor ETL o en la base de datos de Teradata para cargar la información de las fuentes de datos y aplicarles las reglas de negocio definidas para que dichos datos lleguen al DWH.

En las siguientes secciones se explicará de forma más detallada cómo se maneja la extracción (de la fuente hacia Staging Teradata) y la transformación.

### 3.4.1.2 Flujos de extracción desde fuente de datos

Aunque existen algunas variantes en el esquema implementado para ciertos flujos, existe una estructura básica que cubre el comportamiento de los procesos de extracción. Los flujos de extracción dejarán la información en el área de Staging de la base de datos. El objetivo del área de Staging es mantener datos tal cual como se reciben de la fuente para poder ser manipulados de forma más eficiente explotando los recursos de la base de datos.

Como se mencionó anteriormente, existe un flujo principal o padre que tiene un proceso asociado en DataStage y que hará el llamado a los shells que componen el flujo en el orden que corresponda. Los shells a su vez incluirán llamadas a utilerías de Teradata para poder llevar los datos de los archivos fuente hacia el área de Staging. El flujo general de invocaciones en los procesos de carga a Staging es de la siguiente forma:



Figura 3. Flujo de invocaciones en el proceso de extracción

Todos los objetos mencionados en la figura residen en el servidor ETL\_PROD. Los shells se encuentran en /datastage\_data/Copa/DWH\_STG/shells y los scripts en /datastage\_data/Copa/DWH\_STG/scripts.

### 3.4.1.3 Flujos de transformación y carga

Posterior a la extracción y una vez que los datos estén en Staging de cada fuente, se realizará su transformación y carga al DWH. El proceso de transformación se compone de tres fases principales: homologación, transformación e imagen y carga a DWH.



Figura 4. Estructura general de flujos de transformación

Dado que la mayor parte de los conceptos clave que se manejan son únicos o administrados por una sola fuente, la homologación sólo se utiliza para la carga de direcciones. Para el resto de los conceptos manejados se realiza sólo la transformación y carga a imagen y posteriormente la carga a DWH.

### 3.4.1.4 Secuencia de carga

La siguiente figura presenta de forma más específica cómo se ejecuta el proceso completo de carga para una fuente a un área sujeto.



Figura 5. Secuencia de carga

A continuación se describen cada uno de los bloques que componen el proceso de carga:

1. **Job de extracción o proceso transaccional**. En este paso se generan los insumos requeridos para poder comenzar el proceso de carga de datos. Los insumos pueden ser generados a partir de procesos ejecutados en el batch de las aplicaciones fuente o con procesos propios de la solución implementada que generan archivos planos a partir de datos leídos de bases de datos fuente. Estos archivos estarán ubicados en el servidor ETL\_PROD (en el directorio /datastage\_data/Copa/DWH\_STG/input).

Dicho lo anterior, los insumos de datos pueden ser:

1. **Archivos planos**. Estos archivos planos son generados por las diferentes aplicaciones fuente identificadas para proveer datos al DWH (entre ellas: AVE, SAP, SOMS, MARC, VPL, etc.) y ubicados por sus procesos batch en el servidor ETL\_PROD[[3]](#footnote-3) para su posterior carga al DWH con los procesos implementados en esta solución.
2. **Extracciones de base de datos a archivos planos**. Para las fuentes Óptica y MDM se determinó que la extracción se haría con conexión directa hacia la base de datos de la fuente. Dado que existía un proceso base ya desarrollado para conexión a dichas fuentes que hace extracción de datos y los escribe a un archivo, se reutilizó dicho esquema para generar el insumo correspondiente para el DWH. Estos Jobs de extracción son identificados con numeración 00XXX (por ejemplo job00003 que genera el archivo de datos con la extracción de la tabla S\_ADDR\_PER de MDM) y pertenecen a la solución implementada.
3. **Extracciones directas de una base de datos hacia Staging Teradata**. Existe un caso particular (tabla MARD de ECC) que por el volumen de datos a extraer en cada ejecución se maneja diferente que el anterior. En este caso, se hace conexión directa a la base de datos de la fuente pero los datos se cargan directamente a Staging sin pasar por la generación del archivo descrito en el punto anterior. Esta extracción se realiza durante la fase de carga a Staging.

Estos procesos se ejecutarán diariamente y son prerrequisito para poder llevar a cabo el proceso de carga a Staging (paso 3), exceptuando las Extracciones directas de una base de datos hacia Staging Teradata que no representan un prerrequisito.

1. **Carga de parámetros**. En este paso se crean los archivos de configuración base que determinan el comportamiento de la ejecución para ese día, incluyendo el nombre de las interfaces que serán procesada. Los parámetros se encuentran en una tabla en Teradata (ETL\_DP\_CTL.PARAMETROS\_CARGA\_FASE) y son exportados hacia un archivo plano para ser utilizados por los scripts. La carga de parámetros se realiza una vez por ejecución durante el proceso de inicialización (job10018).

1. **Carga a Staging**. En este paso se carga la información de las fuentes de datos al área de Staging de Teradata. De forma general, en el proceso de carga a Staging se llevan a cabo cuatro pasos[[4]](#footnote-4):
2. **Validación de conteos**. Aquí se verifica que el número de registros que existen en el archivo de datos (.dat) coincida con el de cifras (.cif). Esta verificación es realizada antes de iniciar la carga de datos a Teradata. En caso de que la comparación no sea exitosa, el proceso se detiene.
   * **Carga de archivo de datos**. Utilizando la utilería tpt, el archivo de datos (.dat) es cargado al área de Staging en Teradata (ETL\_DP\_STG). En Staging existe una base de datos por cada una de las fuentes y los datos de cada archivo se cargarán en una tabla[[5]](#footnote-5) dentro de la base de datos de la fuente que le corresponda (por ejemplo, la tabla de Staging para la interfaz eilcis\_eccr\_prod de SAP RETAIL se encuentra en la base de datos ETL\_DP\_STG\_SAP\_RET y se llama CIS\_PROD). Estas tablas son temporales ya que sólo mantienen la información que se procesa en ese momento. Para la siguiente ejecución, estas tablas son limpiadas para recibir los nuevos datos.

Si existiera un error en el proceso de carga, el proceso se detiene.

1. **Carga de cifras de control y validación**. Para algunas interfaces se solicitó un archivo adicional de cifras de control (.ctl) que contiene sumarizaciones de los datos incluidos en el archivo de datos y permite evaluar la consistencia de la información que está siendo cargada. El archivo de control es cargado a Teradata (en la base de datos ETL\_DP\_CTL) utilizando la utilería tpt. Existe una tabla CTL para cada interfaz en la que aplique la validación de cifras de control.

En este paso también se verifica que la sumarización incluida en el archivo de cifras de control coincida al aplicar la misma sumarización al archivo de datos. Esta comparación se realiza utilizando una vista que se encuentra en la base de datos ETL\_DP\_CTL. Si el resultado de esta validación no es exitoso se deja registro en la bitácora de carga pero continúa la ejecución del proceso.

1. **Carga de datos al histórico**. Para facilitar el acceso a los datos en caso de que sea necesario un reproceso, se estableció como mecanismo adicional el mantener un volumen de un mes en Staging. La tabla histórica se carga utilizando un BTEQ que toma los datos que se acaban de cargar a Staging y los almacena en una estructura similar a la tabla de carga que integra adicionalmente la fecha a la que corresponden los datos que se están cargando. Se cuenta con una tabla histórica[[6]](#footnote-6) para cada interfaz que se cargue vía un archivo. Si existiera un error en el proceso de carga, el proceso se detiene.

Si los pasos mencionados se ejecutan exitosamente, la carga a Staging se marcará como finalizada y los archivos de datos ya cargados se moverán a la carpeta /datastage\_data/Copa/DWH\_STG/processed.

Los shells utilizados en las cargas a Staging son identificados con numeración 01XXX (por ejemplo, el job01064 realiza la carga a Staging de la interfaz eilfac\_vpl\_apis de VPL).

1. **Carga de datos y transformación hacia DWI**. Una vez que los datos están en Staging, inicia el proceso de transformación para convertirlos de la estructura recibida de la fuente a la definida en el modelo de datos. La transformación que se aplica en cada caso está especificada en vistas que residen en la base de datos ETL\_DP\_VWS\_TRN dentro de Teradata. Dichas vistas usarán como base los datos que existan en Staging, realizarán su procesamiento según las reglas de transformación especificadas e insertarán la versión convertida en la base de datos de Imagen (ETL\_DP\_DWI). Los shells utilizados en las cargas a imagen son identificados con numeración 03XXX (por ejemplo, el job03048 realiza la carga a imagen de FAC\_POS\_TRN).

Como se ha mencionado, existe una fase conocida como homologación que es previa a la carga a imagen. Para esta implementación, este concepto sólo se aplica al caso de las direcciones de clientes de tal forma que con atributos que son compartidos por todas las fuentes que entregan esta información, sea posible asignar un identificador único a los registros recibidos. Para la definición de estos identificadores, se toman los datos que existan en la base de datos de Staging y se ejecutan las vistas que se encuentran en la base de datos ETL\_DP\_VWS\_HOM que insertarán la clave para la combinación de datos en las tablas que se encuentran en la base de datos ETL\_DP\_HOM. Los shells utilizados en la homologación son identificados con numeración 02XXX (por ejemplo, el job02504 realiza la homologación de direcciones).

1. **Carga de datos a DWH**. El último paso en el proceso de carga es llevar los datos homologados y transformados al modelo final del DWH. Los shells utilizados en las cargas a DWH son identificados con numeración 04XXX (por ejemplo, el job04056 realiza la carga de la tabla de DWH FAC\_CTA\_MON). Los shells hacen una llamada a un BTEQ para la inserción de los datos que se encuentran en la base de datos de imagen[[7]](#footnote-7) hacia las tablas de la base de datos ETL\_DP\_DWH.

Para cada paso ejecutado se mantiene el registro del resultado de la ejecución en la bitácora de carga (ETL\_DP\_CTL.BIT\_TABLA\_CARGA). Adicionalmente se mantienen logs por cada script ejecutado en el servidor ETL\_PROD (en el directorio /datastage\_data/Copa/DWH\_STG/logs).

### 3.4.1.5 Descripción de flujos implementados

Los flujos fueron agrupados mayormente por áreas funcionales alineadas a las áreas sujeto hacia las que se está haciendo la carga. En el siguiente diagrama se muestra de manera gráfica la malla de ejecución de los Jobs principales (Jobs >=10,000)



Figura 6. Malla de ejecución flujos 10000

### 3.4.1.6 Detalle interno de jobs

En el siguiente archivo se muestra el detalle de Jobs 10000, en secuencia de ejecución.

Las siguientes secciones muestran el detalle de los Jobs.

#### Job10018

Este Job manda llamar internamente shells que a su vez ejecutan la carga de algunas tablas y generan archivos de parámetros, los cuales son necesarios para la ejecución del resto de los Jobs. A continuación se muestran los shells internos de este job:

| Nombre Shell | Proceso que ejecuta Datastage | Secuencia de ejecución de shells |
| --- | --- | --- |
| job10018.sh | CTRL00\_ETL\_DWH\_CARGA\_INICIAL | Ejecuta los shells en el siguiente orden   1. En paralelo  * job05001.sh – PARAMETROS\_CARGA\_USR * job05002.sh – PARAMETROS\_CARGA\_FASE * job05003.sh – DEPURA\_LOGS * job05004.sh – HBIT\_TABLA\_CARGA * job05005.sh – BIT\_TABLA\_CARGA  1. En paralelo  * job03503.sh – CIS\_TIPO\_TEL * job03510.sh – CIS\_SYS\_FTE |

#### Job10001

Este Job manda llamar de manera interna un Shell que genera el archivo fuente para los datos de TRAI. A continuación se muestran los shells internos de este job:

| Nombre Shell | Proceso que ejecuta Datastage | Secuencia de ejecución de shells |
| --- | --- | --- |
| job10001.sh | CTRL01\_ETL\_EXT\_TRAI | Ejecuta los shells en el siguiente orden   1. job00009.sh |

En esta ilustración se muestra la secuencia de ejecución:



Figura 7. Diagrama de flujo del Job10001

#### Job10002

Este Job manda llamar de manera interna seis shells en paralelo para generar los archivos fuente de MDM. A continuación se muestran los shells internos de este job:

| Nombre Shell | Proceso que ejecuta Datastage | Secuencia de ejecución de shells |
| --- | --- | --- |
| job10002.sh | CTRL01\_ETL\_EXT\_MDM | Ejecuta los shells en paralelo   * job00003.sh – S\_ADDR\_PER * job00004.sh – S\_CIF\_CON\_MAP * job00005.sh – S\_CIF\_EXT\_SYST * job00006.sh – S\_CONTACT * job00007.sh – S\_CON\_ADDR * job00008.sh – S\_PER\_COMM\_ADDR |

En esta ilustración se muestra la secuencia de ejecución.



Figura 8. Diagrama de flujo del Job10002

#### Job10003

Este Job manda llamar de manera interna dos shells para generar los archivos fuente de Óptica. A continuación se muestran los shells internos de este job:

| Nombre Shell | Proceso que ejecuta Datastage | Secuencia de ejecución de shells |
| --- | --- | --- |
| job10003.sh | CTRL01\_ETL\_EXT\_OPTICA | Ejecuta los shells en paralelo   * job00001.sh – FAC\_OPT\_EXP * job00002.sh – DIM\_CLIE\_FTES |

En esta ilustración se muestra la secuencia de ejecución.



Figura 9. Diagrama de flujo del Job10003

#### Job10004

Este Job manda llamar de manera interna de cinco shells que realizan la extracción de los datos de los catálogos de CP para generar los archivos de carga. A continuación se muestran los shells internos de este job:

| Nombre Shell | Proceso que ejecuta Datastage | Secuencia de ejecución de shells |
| --- | --- | --- |
| job10004.sh | CTRL01\_ETL\_EXT\_CP | Ejecuta los shells en paralelo   * job00010.sh * job00011.sh * job00012.sh * job00013.sh * job00014.sh |

En esta ilustración se muestra la secuencia de ejecución:



Figura 10. Diagrama de flujo del Job10004

#### Job10005

Este Job manda llamar de manera interna siete shells que realizan la carga a Staging de las interfaces de la fuente AVE. A continuación se muestran los shells internos de este job:

| Nombre Shell | Proceso que ejecuta Datastage | Secuencia de ejecución de shells |
| --- | --- | --- |
| job10005.sh | CTRL01\_ETL\_STG\_AVE\_PDM | Ejecuta los shells en el siguiente orden:   * job01025.sh – FAC\_TIPPAG * job01037.sh – CIS\_TIPPOSTRN * job01048.sh – FAC\_POS\_TRN * job01052.sh – FAC\_PRMCRM * job01054.sh – CIS\_TIP\_PAG * job01056.sh – FAC\_CTAMON * job01059.sh – FAC\_TRNMON |

En esta ilustración se muestra la secuencia de ejecución.



Figura 11. Diagrama de flujo del Job10005

#### Job10006

Este Job manda llamar de manera interna un shell que realiza la carga de la tabla MARD de ECC (SAP RETAIL) a Staging. A continuación se muestran los shells internos de este job:

| Nombre Shell | Proceso que ejecuta Datastage | Secuencia de ejecución de shells |
| --- | --- | --- |
| job10006.sh | CTRL01\_ETL\_STG\_ECC | Ejecuta los shells en el siguiente orden   * job01147.sh – FAC\_MARD |

En esta ilustración se muestra la secuencia de ejecución.



Figura 12. Diagrama de flujo del Job10006

#### Job10007

Este Job manda llamar de manera interna un shell que ejecuta la carga de datos del catálogo de TRAI a Staging. A continuación se muestran los shells internos de este job:

| Nombre Shell | Proceso que ejecuta Datastage | Secuencia de ejecución de shells |
| --- | --- | --- |
| job10007.sh | CTRL01\_ETL\_STG\_TRAI | Ejecuta los shells en el siguiente orden   * job01041.sh – CIS\_OLITRAI |

En esta ilustración se muestra la secuencia de ejecución.



Figura 13. Diagrama de flujo del Job10007

#### Job10008

Este Job manda llamar de manera interna seis Shells que realizan la carga a Staging de las interfaces de MDM. A continuación se muestran los shells internos de este job:

| Nombre Shell | Proceso que ejecuta Datastage | Secuencia de ejecución de shells |
| --- | --- | --- |
| job10008.sh | CTRL01\_ETL\_STG\_MDM | Ejecuta los shells en el siguiente orden   * job01029.sh – S\_ADDR\_PER\_MDM * job01030.sh – S\_CIF\_CON\_MAP\_MDM * job01031.sh – S\_CIF\_EXT\_SYST\_MDM * job01032.sh – S\_CONTACT\_MDM * job01033.sh – S\_CON\_ADDR * job01034.sh – S\_PER\_COMM\_ADDR |

En esta ilustración se muestra la secuencia de ejecución.



Figura 14. Diagrama de flujo del Job10008

#### Job10009

Este Job manda llamar de manera interna cinco shells que realizan la carga de las interfaces de PREFA a Staging. A continuación se muestran los shells internos de este job:

| Nombre Shell | Proceso que ejecuta Datastage | Secuencia de ejecución de shells |
| --- | --- | --- |
| job10009.sh | CTRL01\_ETL\_STG\_PREFA | Ejecuta los shells en el siguiente orden:   * job01015.sh – DIM\_CLIE\_FTES * job01049.sh – FAC\_EVE * job01050.sh – CIS\_EVE * job01053.sh – FAC\_EVE\_LIS * job01083.sh – FAC\_CIE\_EVE |

En esta ilustración se muestra la secuencia de ejecución:



Figura 15. Diagrama de flujo del Job10009

#### Job10010

Este Job manda llamar de manera interna dos shells que realizan la carga de las interfaces de Óptica a Staging. A continuación se muestran los shells internos de este job:

| Nombre Shell | Proceso que ejecuta Datastage | Secuencia de ejecución de shells |
| --- | --- | --- |
| job10010.sh | CTRL01\_ETL\_STG\_OPTICA | Ejecuta los shells en el siguiente orden   * job01016.sh – DIM\_CLIE\_FTES * job01017.sh – FAC\_OPT\_EXP |

En esta ilustración se muestra la secuencia de ejecución.



Figura 16. Diagrama de flujo del Job10010

#### Job10011

Este Job manda llamar de manera interna cinco shells que realizan la carga de las interfaces de SAP FI a Staging. A continuación se muestran los shells internos de este job:

| Nombre Shell | Proceso que ejecuta Datastage | Secuencia de ejecución de shells |
| --- | --- | --- |
| job10011.sh | CTRL01\_ETL\_STG\_SAP\_FI | Ejecuta los shells en el siguiente orden   * job01002.sh – FAC\_CMPRST * job01003.sh – FAC\_BAL * job01004.sh – FAC\_SDO * job01006.sh – CIS\_CTABAL * job01058.sh – CIS\_GPO\_BAL |

En esta ilustración se muestra la secuencia de ejecución.



Figura 17. Diagrama de flujo del Job10011

#### Job10012

Este Job manda llamar de manera interna ocho shells que realizan la carga de las interfaces de SAP RETAIL a Staging. A continuación se muestran los shells internos de este job:

| Nombre Shell | Proceso que ejecuta Datastage | Secuencia de ejecución de shells |
| --- | --- | --- |
| job10012.sh | CTRL01\_ETL\_STG\_SAP\_RETAIL | Ejecuta los shells en paralelo:   * job01001.sh – DIM\_JERPRO * job01008.sh – DIM\_TDA * job01009.sh – CIS\_DEVEDO * job01018.sh – FAC\_DEVPRV * job01026.sh – CIS\_PROD * job01027.sh – DIM\_PROV * job01035.sh – FAC\_INGLOG[[8]](#footnote-8) * job01038.sh – CIS\_TIPMCIA |

En esta ilustración se muestra la secuencia de ejecución.



Figura 18. Diagrama de flujo del Job10012

#### Job10013

Este Job manda llamar de manera interna dos shells que realizan la carga de las interfaces de SAP RH a Staging. A continuación se muestran los shells internos de este job:

| Nombre Shell | Proceso que ejecuta Datastage | Secuencia de ejecución de shells |
| --- | --- | --- |
| job10013.sh | CTRL01\_ETL\_STG\_SAP\_RH | Ejecuta los shells en el siguiente orden:   * job01012.sh – DIM\_VND * job01014.sh – CIS\_TRBVND * job01019.sh – CIS\_VNDCON |

En esta ilustración se muestra la secuencia de ejecución.



Figura 19. Diagrama de flujo del Job10013

#### Job10014

Este Job manda llamar de manera interna tres shells que realizan la carga de las interfaces de MARC a Staging. A continuación se muestran los shells internos de este job:

| Nombre Shell | Proceso que ejecuta Datastage | Secuencia de ejecución de shells |
| --- | --- | --- |
| job10014.sh | CTRL01\_ETL\_STG\_SAP\_SICAL | Ejecuta los shells en el siguiente orden   * job01013.sh – FAC\_ENVTUL * job01022.sh – FAC\_RECTUL * job01039.sh – CIS\_TIPCONT |

En esta ilustración se muestra la secuencia de ejecución.



Figura 20. Diagrama de flujo del Job10014

#### Job10015

Este Job manda llamar de manera interna cinco shells que realizan la carga de las interfaces de CP a Staging. A continuación se muestran los shells internos de este job:

|  |  |  |
| --- | --- | --- |
| Nombre Shell | Proceso que ejecuta Datastage | Secuencia de ejecución de shells |
| job10015.sh | CTRL01\_ETL\_STG\_CP | Ejecuta los shells en el siguiente orden   * job01042.sh – CIS\_EDOCP * job01043.sh – CIS\_CPCP * job01044.sh – CIS\_CLLCP * job01045.sh – CIS\_COLCP * job01046.sh – CIS\_MUNCP |

En esta ilustración se muestra la secuencia de ejecución.



Figura 21. Diagrama de flujo del Job10015

#### Job10016

Este Job manda llamar de manera interna siete shells para realizar la carga de las interfaces de SOMS a Staging. A continuación se muestran los shells internos de este job:

| Nombre Shell | Proceso que ejecuta Datastage | Secuencia de ejecución de shells |
| --- | --- | --- |
| job10016.sh | CTRL01\_ETL\_STG\_SOMS | Ejecuta los shells en el siguiente orden:   * job01005.sh – DIM\_CLIE\_FTES * job01010.sh – FAC\_ORDHDR * job01011.sh – FAC\_ORDDET * job01020.sh – CIS\_EDOORD * job01023.sh – FAC\_REMDET * job01024.sh – FAC\_REMHDR * job01028.sh – CIS\_EDOORDPAQ |

En esta ilustración se muestra la secuencia de ejecución.



Figura 22. Diagrama de flujo del Job10016

#### Job10017

Este Job manda llamar de manera interna 26 shells que realizan la carga de las interfaces de VPL a Staging. A continuación se muestran los shells internos de este job:

| Nombre Shell | Proceso que ejecuta Datastage | Secuencia de ejecución de shells |
| --- | --- | --- |
| job10017.sh | CTRL00\_ETL\_STG\_VPL | Ejecuta los shells en el siguiente orden:   1. En paralelo  * job01007.sh – CIS\_LOGO * job01021.sh – DIM\_CLIE\_FTES * job01036.sh – FAC\_TRNCRD * job01040.sh – FAC\_SDO\_PLAN * job01060.sh – FAC\_PAG\_REF * job01064.sh – FAC\_APIS * job01067.sh – CIS\_STA\_CTA * job01070.sh – DIM\_PLAN * job01073.sh – FAC\_SDO\_CTA * job01077.sh – CIS\_TRNCRD  1. En paralelo  * job01051.sh – FAC\_BRTL * job01055.sh – FAC\_BRPA * job01057.sh – FAC\_APIA * job01061.sh – CIS\_GPOFAC * job01062.sh – FAC\_SEG\_CRD * job01063.sh – FAC\_BRPE * job01065.sh – FAC\_BRIQ * job01069.sh – CIS\_INDVTA * job01071.sh – CIS\_TBLQ * job01075.sh – CIS\_GPOPLAN  1. En paralelo  * job01066.sh – CIS\_SEGRZN * job01068.sh – CIS\_SEGSTT * job01072.sh – CIS\_SEGPROD * job01074.sh – CIS\_GPOTRNCRD * job01081.sh – CIS\_DICCRE * job01076.sh – FAC\_BRRS |

En esta ilustración se muestra la secuencia de ejecución.



Figura 23. Diagrama de flujo del Job10017

#### Job10019

Este Job manda llamar de manera interna 23 shells que realizan la carga de los datos de MDM al modelo de clientes. A continuación se muestran los shells internos de este job:

| Nombre Shell | Proceso que ejecuta Datastage | Secuencia de ejecución de shells |
| --- | --- | --- |
| job10019.sh | CTRL00\_ETL\_DWH\_CLIENTES\_MDM | Ejecuta los shells en el siguiente orden:   1. En paralelo  * job02504.sh – SK\_DIR (homologación) * job03509.sh – DIM\_TEL (DWI) * job03541.sh – CIS\_MUN (DWI) * job03543.sh – CIS\_EDO(DWI) * job03515.sh – DIM\_EMAIL (DWI)  1. En paralelo  * job04509.sh – DIM\_TEL (DWH) * job04541.sh – CIS\_MUN (DWH) * job04543.sh – CIS\_EDO(DWH) * job04515.sh – DIM\_EMAIL (DWH)  1. En paralelo  * job03514.sh – DIM\_REL\_TEL\_CLIE (DWI) * job03516.sh – DIM\_REL\_EMAIL\_CLIE (DWI) * job03518.sh – DIM\_REL\_DIR\_CLIE (DWI)  1. En paralelo  * job04514.sh – DIM\_REL\_TEL\_CLIE (DWH) * job04516.sh – DIM\_REL\_EMAIL\_CLIE (DWH) * job04518.sh – DIM\_REL\_DIR\_CLIE (DWH)  1. En paralelo  * job03032.sh – DIM\_CLIE (DWI) * job03094.sh – DIM\_REL\_CLIE (DWI) * job03504.sh – DIM\_DIREC (DWI)  1. En paralelo  * job04032.sh – DIM\_CLIE (DWH) * job04094.sh – DIM\_REL\_CLIE (DWH) * job04504.sh – DIM\_DIREC (DWH)  1. job03095.sh – DIM\_REL\_CLIE (DWI) 2. job04095.sh – DIM\_REL\_CLIE (DWH) |

En esta ilustración se muestra la secuencia de ejecución.



Figura 24. Diagrama de flujo del Job10019

#### Job10020

Este Job manda llamar de manera interna un 23 shells que realizan la carga de los datos de PREFA al modelo de clientes. A continuación se muestran los shells internos de este job:

| Nombre Shell | Proceso que ejecuta Datastage | Secuencia de ejecución de shells |
| --- | --- | --- |
| job10020.sh | CTRL00\_ETL\_DWH\_CLIENTES\_PREFA | Ejecuta los shells en el siguiente orden   1. En paralelo  * job02504.sh – SK\_DIR (homologación) * job03522.sh – DIM\_TEL (DWI) * job03511.sh – CIS\_MUN (DWI) * job03512.sh – CIS\_EDO(DWI)  1. En paralelo  * job04509.sh – DIM\_TEL (DWH) * job04511.sh – CIS\_MUN (DWH) * job04512.sh – CIS\_EDO(DWH)  1. En paralelo  * job03526.sh – DIM\_REL\_TEL\_CLIE (DWI) * job03534.sh – DIM\_REL\_EMAIL\_CLIE (DWI) * job03538.sh – DIM\_REL\_DIR\_CLIE (DWI)  1. En paralelo  * job04526.sh – DIM\_REL\_TEL\_CLIE (DWH) * job04534.sh – DIM\_REL\_EMAIL\_CLIE (DWH) * job04538.sh – DIM\_REL\_DIR\_CLIE (DWH)  1. En paralelo  * job03032.sh – DIM\_CLIE (DWI) * job03530.sh – DIM\_EMAIL (DWI) * job03094.sh – DIM\_REL\_CLIE (DWI) * job03505.sh – DIM\_DIREC (DWI)  1. En paralelo  * job04032.sh – DIM\_CLIE (DWH) * job04530.sh – DIM\_EMAIL (DWH) * job04094.sh – DIM\_REL\_CLIE (DWH) * job04505.sh – DIM\_DIREC (DWH)  1. job03095.sh – DIM\_REL\_CLIE (DWI) 2. job04095.sh – DIM\_REL\_CLIE (DWH) |

En esta ilustración se muestra la secuencia de ejecución.



Figura 25. Diagrama de flujo del Job10020

#### Job10021

Este Job manda llamar de manera interna 23 shells que cargan los datos de ÓPTICA al modelo de clientes. A continuación se muestran los shells internos de este job:

| Nombre Shell | Proceso que ejecuta Datastage | Secuencia de ejecución de shells |
| --- | --- | --- |
| job10021.sh | CTRL00\_ETL\_DWH\_CLIENTES\_OPT | Ejecuta los shells en el siguiente orden   1. En paralelo  * job02504.sh – SK\_DIR (homologación) * job03521.sh – DIM\_TEL (DWI) * job03511.sh – CIS\_MUN (DWI) * job03512.sh – CIS\_EDO(DWI)  1. En paralelo  * job04521.sh – DIM\_TEL (DWH) * job04511.sh – CIS\_MUN (DWH) * job04512.sh – CIS\_EDO(DWH)  1. En paralelo  * job03525.sh – DIM\_REL\_TEL\_CLIE (DWI) * job03533.sh – DIM\_REL\_EMAIL\_CLIE (DWI) * job03537.sh – DIM\_REL\_DIR\_CLIE (DWI)  1. En paralelo  * job04525.sh – DIM\_REL\_TEL\_CLIE (DWH) * job04533.sh – DIM\_REL\_EMAIL\_CLIE (DWH) * job04537.sh – DIM\_REL\_DIR\_CLIE (DWH)  1. En paralelo  * job03016.sh – DIM\_CLIE (DWI) * job03094.sh – DIM\_REL\_CLIE (DWI) * job03529.sh – DIM\_EMAIL (DWI) * job03506.sh – DIM\_DIREC (DWI)  1. En paralelo  * job04016.sh – DIM\_CLIE (DWH) * job04094.sh – DIM\_REL\_CLIE (DWH) * job4529.sh – DIM\_EMAIL (DWH) * job04506.sh – DIM\_DIREC (DWH)  1. job03095.sh – DIM\_REL\_CLIE (DWI) 2. job04095.sh – DIM\_REL\_CLIE (DWH) |

En esta ilustración se muestra la secuencia de ejecución.



Figura 26. Diagrama de flujo del Job10021

#### Job10022

Este Job manda llamar de manera interna 23 shells que cargan los datos de SOMS al modelo de clientes. A continuación se muestran los shells internos de este job:

| Nombre Shell | Proceso que ejecuta Datastage | Secuencia de ejecución de shells |
| --- | --- | --- |
| job10022.sh | CTRL00\_ETL\_DWH\_CLIENTES\_SOMS | Ejecuta los shells en el siguiente orden   1. En paralelo  * job02504.sh – SK\_DIR (homologación) * job03523.sh – DIM\_TEL (DWI) * job03542.sh – CIS\_MUN (DWI) * job03512.sh – CIS\_EDO(DWI) * job03531.sh – DIM\_EMAIL (DWI)  1. En paralelo  * job04523.sh – DIM\_TEL (DWH) * job04542.sh – CIS\_MUN (DWH) * job04512.sh – CIS\_EDO(DWH) * job04531.sh – DIM\_EMAIL (DWH)  1. En paralelo  * job03527.sh – DIM\_REL\_TEL\_CLIE (DWI) * job03535.sh – DIM\_REL\_EMAIL\_CLIE (DWI) * job03539.sh – DIM\_REL\_DIR\_CLIE (DWI)  1. En paralelo  * job04527.sh – DIM\_REL\_TEL\_CLIE (DWH) * job04535.sh – DIM\_REL\_EMAIL\_CLIE (DWH) * job04539.sh – DIM\_REL\_DIR\_CLIE (DWH)  1. En paralelo  * job03005.sh – DIM\_CLIE (DWI) * job03098.sh – DIM\_REL\_CLIE (DWI) * job03507.sh – DIM\_DIREC (DWI)  1. En paralelo  * job04005.sh – DIM\_CLIE (DWH) * job04098.sh – DIM\_REL\_CLIE (DWH) * job04507.sh – DIM\_DIREC (DWH)  1. job03095.sh – DIM\_REL\_CLIE (DWI) 2. job04095.sh – DIM\_REL\_CLIE (DWH) |

En esta ilustración se muestra la secuencia de ejecución.



Figura 27. Diagrama de flujo del Job10022

#### Job10023

Este Job manda llamar de manera interna 19 shells que cargan los datos de VPL al modelo de clientes. A continuación se muestran los shells internos de este job:

| Nombre Shell | Proceso que ejecuta Datastage | Secuencia de ejecución de shells |
| --- | --- | --- |
| job10023.sh | CTRL00\_ETL\_DWH\_CLIENTES\_VPL | Ejecuta los shells en el siguiente orden:   1. En paralelo  * job02504.sh – SK\_DIR (homologación) * job03524.sh – DIM\_TEL (DWI) * job03511.sh – CIS\_MUN (DWI) * job03512.sh – CIS\_EDO(DWI)  1. En paralelo  * job04524.sh – DIM\_TEL (DWH) * job04511.sh – CIS\_MUN (DWI) * job04512.sh – CIS\_EDO(DWH)  1. En paralelo  * job03528.sh – DIM\_REL\_TEL\_CLIE (DWI) * job03536.sh – DIM\_REL\_EMAIL\_CLIE (DWI) * job03540.sh – DIM\_REL\_DIR\_CLIE (DWI)  1. En paralelo  * Job04528.sh – DIM\_REL\_TEL\_CLIE (DWH) * job04536.sh – DIM\_REL\_EMAIL\_CLIE (DWH) * job04540.sh – DIM\_REL\_DIR\_CLIE (DWH)  1. En paralelo  * job03021.sh – DIM\_CLIE (DWI) * job03532.sh – DIM\_EMAIL (DWI) * job03094.sh – DIM\_REL\_CLIE (DWI) * job03508.sh – DIM\_DIREC (DWI)  1. En paralelo  * job04021.sh – DIM\_CLIE (DWH) * job04532.sh – DIM\_EMAIL (DWH) * job04094.sh – DIM\_REL\_CLIE (DWH) * job04508.sh – DIM\_DIREC (DWH)  1. job03095.sh – DIM\_REL\_CLIE (DWI) 2. job04095.sh – DIM\_REL\_CLIE (DWH) |

En esta ilustración se muestra la secuencia de ejecución.



Figura 28. Diagrama de flujo del Job10023

#### Job10024

Este Job manda llamar de manera interna 16 shells que cargan los datos de SAP FI al modelo de Finanzas. A continuación se muestran los shells internos de este job:

| Nombre Shell | Proceso que ejecuta Datastage | Secuencia de ejecución de shells |
| --- | --- | --- |
| job10024.sh | CTRL00\_ETL\_DWH\_FINANZAS | Ejecuta los shells en el siguiente orden:   1. En paralelo  * job03001.sh – DIM\_DIR (DWI) * job03058.sh – CIS\_CTA\_GPO\_BAL (DWI)  1. En paralelo  * job04001.sh – DIM\_DIR (DWH) * job04058.sh – CIS\_CTA\_GPO\_BAL (DWH)  1. En paralelo  * job03006.sh – CIS\_CTA\_BAL (DWI) * job03502.sh – DIM\_SEC (DWI)  1. En paralelo  * job04006.sh – CIS\_CTA\_BAL (DWH) * job04502.sh – DIM\_SEC (DWH)  1. En paralelo  * job03002.sh – FAC\_CMP\_RST (DWI) * job03003.sh – FAC\_BAL (DWI) * job03004.sh – FAC\_SDO (DWI)  1. En paralelo  * job04002.sh – FAC\_CMP\_RST (DWH) * job04003.sh – FAC\_BAL (DWH) * job04004.sh – FAC\_SDO (DWH) |

En esta ilustración se muestra la secuencia de ejecución.



Figura 29. Diagrama de flujo del Job10024

#### Job10025

Este Job manda llamar de manera interna ocho shells que cargan los datos de SAP RETAIL al modelo de Inventarios. A continuación se muestran los shells internos de este job:

| Nombre Shell | Proceso que ejecuta Datastage | Secuencia de ejecución de shells |
| --- | --- | --- |
| job10025.sh | CTRL00\_ETL\_DWH\_INVENTARIO | Ejecuta los shells en el siguiente orden:1. En paralelo   * job03009.sh – CIS\_DEV\_EDO (DWI) * job03038.sh – CIS\_TIP\_MCIA (DWI) * job03027.sh – DIM\_PROV (DWI) * job03018.sh – FAC\_DEV\_PRV (DWI)  1. En paralelo  * job04009 – CIS\_DEV\_EDO (DWH) * job04038.sh – CIS\_TIP\_MCIA (DWH) * job04027.sh – DIM\_PROV (DWH) * job04018.sh – FAC\_DEV\_PRV (DWH) |

En esta ilustración se muestra la secuencia de ejecución.



Figura 30. Diagrama de flujo del Job10025

#### Job10026

Este Job manda llamar de manera interna ocho shells que cargan datos de SAP RETAIL al modelo de Localidades. A continuación se muestran los shells internos de este job:

|  |  |  |
| --- | --- | --- |
| Nombre Shell | Proceso que ejecuta Datastage | Secuencia de ejecución de shells |
| job10026.sh | CTRL00\_ETL\_DWH\_LOCALIDAD | Ejecuta los shells en el siguiente orden:   1. En paralelo  * job03511.sh – CIS\_MUN (DWI) * job03512.sh – CIS\_EDO (DWI) * job03513.sh – CIS\_PAIS (DWI)  1. En paralelo  * job04511.sh – CIS\_MUN (DWH) * job04512.sh – CIS\_EDO (DWH) * job04513.sh – CIS\_PAIS (DWH)  1. job03008.sh – DIM\_TDA(DWI) 2. job04008.sh – DIM\_TDA (DWH) |

En esta ilustración se muestra la secuencia de ejecución.



Figura 31. Diagrama de flujo del Job10026

#### Job10027

Este Job manda llamar de manera interna 10 shells que cargan datos de SAP RETAIL al modelo de Producto. A continuación se muestran los shells internos de este job:

| Nombre Shell | Proceso que ejecuta Datastage | Secuencia de ejecución de shells |
| --- | --- | --- |
| job10027.sh | CTRL00\_ETL\_DWH\_PRODUCTO | Ejecuta los shells en el siguiente orden:   1. job03001.sh – DIM\_DIR (DWI) 2. job04001.sh – DIM\_DIR (DWH) 3. job03502.sh – DIM\_SEC (DWI) 4. job04502.sh – DIM\_SEC (DWH) 5. job03501.sh – DIM\_GPO\_ART (DWI) 6. job04501.sh – DIM\_GPO\_ART (DWH) 7. job03026.sh – DIM\_PROD (DWI) 8. job04026.sh – DIM\_PROD (DWH) 9. job03517.sh – DIM\_PROD\_HIST (DWI) 10. job04517.sh – DIM\_PROD\_HIST (DWH) |

En esta ilustración se muestra la secuencia de ejecución.



Figura 32. Diagrama de flujo del Job10027

#### Job10028

Este Job manda llamar de manera interna 22 shells que cargan datos de AVE, SAP RH y Óptica al modelo de Transacciones POS. A continuación se muestran los shells internos de este job:

| Nombre Shell | Proceso que ejecuta Datastage | Secuencia de ejecución de shells |
| --- | --- | --- |
| job10028.sh | CTRL00\_ETL\_DWH\_TRANSACCIONES | Ejecuta los shells en el siguiente orden:   1. En paralelo  * job03014.sh – CIS\_TRB\_VND (DWI) * job03019.sh – CIS\_VND\_CON (DWI) * job03037.sh – CIS\_TIP\_POS\_TRN (DWI) * job03054.sh – CIS\_TIP\_PAG (DWI) * job03519.sh – CIS\_INCENTIVO (DWI)  1. En paralelo  * job04014.sh – CIS\_TRB\_VND (DWH) * job04019.sh – CIS\_VND\_CON (DWH) * job04037.sh – CIS\_TIP\_POS\_TRN (DWH) * job04054.sh – CIS\_TIP\_PAG (DWH) * job04519.sh – CIS\_INCENTIVO (DWH)  1. En paralelo  * job03012.sh – DIM\_VND (DWI) * job03017.sh – FAC\_OPT\_EXP (DWI) * job03052.sh – FAC\_PRM\_CRM (DWI)  1. En paralelo  * job04012.sh – DIM\_VND (DWH) * job04017.sh – FAC\_OPT\_EXP (DWH) * job04052.sh – FAC\_PRM\_CRM (DWH)  1. job03025.sh – FAC\_TIP\_PAG (DWI) 2. job04025.sh – FAC\_TIP\_PAG (DWH) 3. job03048.sh – FAC\_POS\_TRN (DWI) 4. job04048.sh – FAC\_POS\_TRN (DWH) 5. job03520.sh – FAC\_VTA\_PRORRATEO (DWI) 6. job04520.sh – FAC\_VTA\_PRORRATEO (DWH) |

En esta ilustración se muestra la secuencia de ejecución.



Figura 33. Diagrama de flujo del Job10028

#### Job10029

Este Job manda llamar de manera interna 50 shells que cargan datos de VPL al modelo de Crédito. A continuación se muestran los shells internos de este job:

| Nombre Shell | Proceso que ejecuta Datastage | Secuencia de ejecución de shells |
| --- | --- | --- |
| job10029.sh | CTRL00\_ETL\_DWH\_CREDITO\_VPL | Ejecuta los shells en el siguiente orden:   1. En paralelo  * job03007.sh – CIS\_LOGO (DWI) * job03061.sh – CIS\_GPO\_FAC (DWI) * job03067.sh – CIS\_STA\_CTA (DWI) * job03069.sh – CIS\_IND\_VTA (DWI) * job03071.sh – CIS\_TBLQ (DWI) * job03074.sh – CIS\_GPO\_TRN (DWI) * job03075.sh – CIS\_GPO\_PLAN (DWI)  1. En paralelo  * job04007.sh – CIS\_LOGO (DWH) * job04061.sh – CIS\_GPO\_FAC (DWH) * job04067.sh – CIS\_STA\_CTA (DWH) * job04069.sh – CIS\_IND\_VTA (DWH) * job04071.sh – CIS\_TBLQ (DWH) * job04074.sh – CIS\_GPO\_TRN (DWH) * job04075.sh – CIS\_GPO\_PLAN (DWH)  1. En paralelo  * job03051.sh – FAC\_BR\_TL (DWI) * job03055.sh – FAC\_BR\_PA (DWI) * job03057.sh – FAC\_APIA (DWI) * job03060.sh – FAC\_PAG\_REF (DWI) * job03063.sh – FAC\_BR\_PE (DWI) * job03064.sh – FAC\_APIS (DWI) * job03065.sh – FAC\_BR\_IQ (DWI) * job03076.sh – FAC\_BR\_RS (DWI) * job03081.sh – DIM\_CTA\_TRJ (DWI) * job03091.sh – DIM\_CTA (DWI) * job03096.sh – CIS\_CTA\_ANIO\_MES (DWI)  1. En paralelo  * job04051.sh – FAC\_BR\_TL (DWH) * job04055.sh – FAC\_BR\_PA (DWH) * job04057.sh – FAC\_APIA (DWH) * job04060.sh – FAC\_PAG\_REF (DWH) * job04063.sh – FAC\_BR\_PE (DWH) * job04064.sh – FAC\_APIS (DWH) * job04065.sh – FAC\_BR\_IQ (DWH) * job04076.sh – FAC\_BR\_RS (DWH) * job04081.sh – DIM\_CTA\_TRJ (DWH) * job04091.sh – DIM\_CTA (DWH) * job04096.sh – CIS\_CTA\_ANIO\_MES (DWH)  1. En paralelo  * job03036.sh – FAC\_TRN\_CRD (DWI) * job03040.sh – FAC\_SDO\_PLAN (DWI) * job03070.sh – DIM\_PLAN (DWI) * job03077.sh – DIM\_TRN\_CRD (DWI) * job03090.sh – DIM\_PLAN\_ANIO\_MES (DWI)  1. En paralelo  * job04036.sh – FAC\_TRN\_CRD (DWH) * job04040.sh – FAC\_SDO\_PLAN (DWH) * job04070.sh – DIM\_PLAN (DWH) * job04077.sh – DIM\_TRN\_CRD (DWH) * job04090.sh – DIM\_PLAN\_ANIO\_MES (DWH)  1. En paralelo  * job03073.sh – FAC\_SDO\_CTA\_MES (DWI) * job03097.sh – FAC\_SDO\_CTA (DWI)  1. En paralelo  * job04073.sh – FAC\_SDO\_CTA\_MES (DWH) * job04097.sh – FAC\_SDO\_CTA (DWH) |

En esta ilustración se muestra la secuencia de ejecución.



Figura 34. Diagrama de flujo del Job10029

#### Job10030

Este Job manda llamar de manera interna tres shells que realizan la carga a Staging de las interfaces de CYBER FINANCIAL. A continuación se muestran los shells internos de este job:

|  |  |  |
| --- | --- | --- |
| Nombre Shell | Proceso que ejecuta Datastage | Secuencia de ejecución de shells |
| job10030.sh | CTRL01\_ETL\_STG\_CYBER\_FINANCIAL | Ejecuta los siguientes shells en paralelo:   * job01078.sh – FAC\_FINCOB * job01079.sh – FAC\_BITCOB * job01080.sh – FAC\_MSTRCOB |

En esta ilustración se muestra la secuencia de ejecución.



Figura 35. Diagrama de flujo del Job10030

#### Job10031

Este Job manda llamar de manera interna 12 shells que cargan datos de AVE (MR) al modelo de PREFA. A continuación se muestran los shells internos de este job:

| Nombre Shell | Proceso que ejecuta Datastage | Secuencia de ejecución de shells |
| --- | --- | --- |
| job10031.sh | CTRL00\_ETL\_DWH\_PREFA | Ejecuta los shells en el siguiente orden:   1. En paralelo  * job03083.sh – CIS\_EVE (DWI) * job03092.sh – CIS\_TIPO\_LIS (DWI) * job03093.sh – CIS\_TIPO\_OBJ (DWI)  1. En paralelo  * job04083.sh – CIS\_EVE (DWH) * job04092.sh – CIS\_TIPO\_LIS (DWH) * job04093.sh – CIS\_TIPO\_OBJ (DWH)  1. job03049.sh – FAC\_EVE (DWI) 2. job04049.sh – FAC\_EVE (DWH) 3. En paralelo  * job03050.sh – FAC\_FIN\_EVE (DWI) * job03053.sh – FAC\_EVE\_LIS (DWI)  1. En paralelo  * job04050.sh – FAC\_FIN\_EVE (DWH) * job04053.sh – FAC\_EVE\_LIS (DWH) |

En esta ilustración se muestra la secuencia de ejecución.



Figura 36. Diagrama de flujo del Job10031

#### Job10032

Este Job manda llamar de manera interna 10 shells que cargan datos de AVE al modelo de Monedero Electrónico. A continuación se muestran los shells internos de este job:

| Nombre Shell | Proceso que ejecuta Datastage | Secuencia de ejecución de shells |
| --- | --- | --- |
| job10032.sh | CTRL00\_ETL\_DWH\_MONEDERO\_ELECTRONICO | Ejecuta los shells en el siguiente orden:   1. En paralelo  * job03502.sh – DIM\_SEC (DWI) * job03519.sh – CIS\_INCENTIVO (DWI)  1. En paralelo  * job04502.sh – DIM\_SEC (DWH) * job04519.sh – CIS\_INCENTIVO (DWH)  1. job03056.sh – FAC\_CTA\_MON (DWI) 2. job04056.sh – FAC\_CTA\_MON (DWH) 3. job03059.sh – FAC\_TRN\_MON (DWI) 4. job04059.sh – FAC\_TRN\_MON (DWH) |

En esta ilustración se muestra la secuencia de ejecución.



Figura 37. Diagrama de flujo del Job10032

#### Job10033

Este Job manda llamar de manera interna ocho shells que cargan datos de VPL al modelo de Seguros. A continuación se muestran los shells internos de este job:

| Nombre Shell | Proceso que ejecuta Datastage | Secuencia de ejecución de shells |
| --- | --- | --- |
| job10033.sh | CTRL00\_ETL\_DWH\_SEGUROS | Ejecuta los shells en el siguiente orden:   1. En paralelo  * job03066.sh – CIS\_SEG\_RZN (DWI) * job03068.sh – CIS\_SEG\_STT (DWI) * job03072.sh – CIS\_SEG\_PROD (DWI)  1. En paralelo  * job04066.sh – CIS\_SEG\_RZN (DWH) * job04068.sh – CIS\_SEG\_STT (DWH) * job04072.sh – CIS\_SEG\_PROD (DWH)  1. job03062.sh – FAC\_SEG\_CRD\_HST (DWI) 2. job04062.sh – FAC\_SEG\_CRD\_HST (DWH) |

En esta ilustración se muestra la secuencia de ejecución.



Figura 38. Diagrama de flujo del Job10033

#### Job10034

Este Job manda llamar de manera interna seis shells que cargan datos de CYBER FINANCIAL al modelo de Crédito (Cobranza). A continuación se muestran los shells internos de este job:

| Nombre Shell | Proceso que ejecuta Datastage | Secuencia de ejecución de shells |
| --- | --- | --- |
| job10034.sh | CTRL00\_ETL\_DWH\_CREDITO\_CYB\_FIN | Ejecuta los shells en el siguiente orden:   1. En paralelo  * job03078.sh – FAC\_FIN\_COB (DWI) * job03079.sh – FAC\_BIT\_COB (DWI) * job03080.sh – FAC\_MSTR\_COB (DWI)  1. En paralelo  * job04078.sh – FAC\_FIN\_COB (DWH) * job04079.sh – FAC\_BIT\_COB (DWH) * job04080.sh – FAC\_MSTR\_COB (DWH) |

En esta ilustración se muestra la secuencia de ejecución.



Figura 39. Diagrama de flujo del Job10034

#### Job10035

Este Job manda llamar de manera dos shells que cargan datos de ECC al modelo de Inventarios. A continuación se muestran los shells internos de este job:

| Nombre Shell | Proceso que ejecuta Datastage | Secuencia de ejecución de shells |
| --- | --- | --- |
| job10035.sh | CTRL00\_ETL\_DWH\_INVENTARIO\_ECC | Ejecuta los shells en el siguiente orden:   1. job03147 – FAC\_MOV\_INV (DWI) 2. job04147 – FAC\_MOV\_INV (DWH) |

En esta ilustración se muestra la secuencia de ejecución.



Figura 40. Diagrama de flujo del Job10035

#### Job10036

Este Job manda llamar de manera interna seis shells que cargan datos de MARC al modelo de Inventarios. A continuación se muestran los shells internos de este job:

|  |  |  |
| --- | --- | --- |
| Nombre Shell | Proceso que ejecuta Datastage | Secuencia de ejecución de shells |
| job10036.sh | CTRL00\_ETL\_DWH\_INVENTARIO\_SAP\_SICAL | Ejecuta los shells en el siguiente orden:   1. En paralelo  * job03013 – FAC\_ENV\_TUL (DWI) * job03022 – FAC\_REC\_TUL (DWI) * job03039 – CIS\_TIP\_CONT (DWI)  1. En paralelo  * job04013 – FAC\_ENV\_TUL (DWH) * job04022 – FAC\_REC\_TUL (DWH) * job04039 – CIS\_TIP\_CONT (DWH) |

En esta ilustración se muestra la secuencia de ejecución.



Figura 41. Diagrama de flujo del Job10036

#### Job10037

Este Job manda llamar de manera interna seis shells que cargan datos de SOMS al modelo de Inventarios. A continuación se muestran los shells internos de este job:

| Nombre Shell | Proceso que ejecuta Datastage | Secuencia de ejecución de shells |
| --- | --- | --- |
| job10037.sh | CTRL00\_ETL\_DWH\_INVENTARIO\_SOMS | Ejecuta los shells en el siguiente orden   1. En paralelo  * job03020 – CIS\_EDO\_ORD (DWI) * job03023 – FAC\_ORD\_DET (DWI) * job03024 – FAC\_ORD\_HDR (DWI)  1. En paralelo  * job04020 – CIS\_EDO\_ORD (DWH) * job04023 – FAC\_ORD\_DET (DWH) * job04024 – FAC\_ORD\_HDR (DWH) |

En esta ilustración se muestra la secuencia de ejecución.



Figura 42. Diagrama de flujo del Job10037

## 3.5 Seguimiento a la ejecución de procesos

En esta sección se presentan las herramientas que pueden ser utilizadas para monitorear la plataforma y la ejecución de los procesos además de describir cómo dichas herramientas pueden ser utilizadas para la identificación de incidencias.

### 3.5.1 Mecanismos para monitoreo de ejecución de procesos

Para dar seguimiento a la ejecución de los procesos que se describieron en la sección anterior se cuentan con diferentes mecanismos, entre ellos:

1. Logs de sistema operativo
2. Bitácora ETL\_PROD
3. ViewPoint
4. Monitor de procesos

A través de dichos mecanismos es posible verificar si algún proceso está en ejecución o finalizó ya sea correctamente o con alguna falla. Dado que el nivel de información varía entre cada uno de ellos, para algunas validaciones dichos mecanismos se complementarán entre ellos. A continuación se describe el tipo de información que puede obtenerse en cada caso.

### 3.5.2 Logs de sistema operativo

Cada objeto desarrollado que reside en el servidor ETL\_PROD genera un log al ser ejecutado. Estos se almacenan en /datastage\_data/Copa/DWH\_STG/logs. Para poder identificar los logs que corresponden a cada ejecución es necesario conocer el nombre del job del cual se requiere información y la fecha de ejecución. Típicamente el nombre del log del proceso padre (que hará las ejecuciones del resto de los shells que están incluidos en su bloque y almacenará el resultado de su ejecución individual) es como sigue: job1<XXXXX> + <fecha> + .log

Estos logs son particularmente útiles para identificar la causa de alguna falla en el proceso de extracción complementando a la información que proporcione la bitácora ETL.

### 3.5.3 Bitácora ETL\_PROD

La solución integra una tabla que mantiene el detalle de la ejecución de tareas dentro de la base de datos. Esta tabla se conoce como Bitácora ETL\_PROD y está ubicada en ETL\_DP\_CTL.BIT\_TABLA\_CARGA. La siguiente figura presenta la estructura de la bitácora:



Figura 43. Bitácora ETL\_PROD

La información que se incluye en la bitácora es la siguiente:

| Campo | Descripción |
| --- | --- |
| NU\_PROCESO | Identifica el número de Job que está siendo ejecutado. |
| NU\_SECUENCIA | Identifica el número de secuencia dentro del Job que está siendo ejecutada. En el ejemplo de la figura existen 3 secuencias. |
| NB\_PROCESO | Nombre del shell que está coordina la carga |
| NB\_TABLA | Nombre de la tabla en la que están cayendo los datos que están siendo cargados por cada secuencia. |
| NB\_FASE | Nombre de la fase que está en ejecución que corresponde a las fases de los flujos presentados en la sección 3.1 de este documento. |
| FH\_PROCESO | Fecha en la que se está haciendo el procesamiento de la información |
| FH\_INI\_CARGA | Indica la fecha y hora (tipo timestamp) en la que inició el procesamiento de la secuencia que se esté revisando |
| FH\_FIN\_CARGA | Indica la fecha y hora (tipo timestamp) en la que finalizó el procesamiento de la secuencia que se esté revisando |
| TO\_EXTRAIDOS | Número total de registros extraídos de la fuente que está siendo cargada. |
| TO\_CARGADOS | Número total de registros cargados en la base de datos. En el caso de Staging, lo ideal es que coincida con el número de TO\_EXTRAIDOS. |
| TO\_RECHAZADOS | Número total de registros que por algún motivo no fueron cargados en la base de datos. |
| TO\_BORRADOS | Número total de registros que fueron borrados durante la ejecución. Normalmente se incluye aquí la cantidad de registros que se cargaron en la ejecución previa. |
| TO\_DUPLICADOS | Número total de registros que fueron identificados como duplicados por el proceso de carga. |
| TO\_ACTUALIZADOS | Número total de registros actualizados |
| TO\_DURACION | Tiempo de duración de la secuencia que se está ejecutando |
| NB\_SCRIPT | Nombre del script que fue ejecutado durante esa secuencia. |
| RUTA\_SCRIPT\_LOG | Ruta en el servidor ETL\_PROD en la que se localizan los logs que fueron generados en la ejecución. |
| NB\_SCRIPT\_LOG | Nombre del log generado por el script ejecutado en la secuencia de carga. |
| ST\_CARGA | Estatus de la secuencia de carga, que puede ser Proceso Terminado, Procesando la Carga o distintos tipos de mensajes de error |
| NU\_CIFRAS | Número arrojado tras la validación del archivo .ctl. Lo ideal es que éste sea cero pero en caso contrario indica la diferencia entre lo que existe en el archivo de datos y lo que incluye el archivo .ctl. |

Existirá una entrada en la bitácora para cada secuencia que esté definida en el proceso. Cada secuencia incluye una llamada a un Stored Procedure (SP) que hace la escritura en esta tabla. Si existe algún error en alguna de las instrucciones ejecutadas, éste puede ser visualizado en la línea correspondiente.

Para facilitar la consulta de dicha tabla, se recomienda filtrar por número de job. Una opción es como se muestra a continuación:

**select** \* **from** ETL\_DP\_CTL.BIT\_TABLA\_CARGA

**where** nb\_proceso = 'job01013'

**order** **by** nu\_secuencia

### 3.5.4 Viewpoint

Aunque el objetivo principal de esta herramienta es realizar la administración de la base de datos, también puede ser utilizada para monitorear la ejecución de los procesos, principalmente de los recursos que están siendo consumidos por cada usuario que está corriendo procesos o consultas.

El servidor se encuentra en la red de Copa. El acceso es a través de la dirección: http://172.17.203.217/c. Se debe de contar con un usuario específico para poder acceder. La siguiente figura presenta la ventana principal de Viewpoint, con algunos módulos útiles para monitoreo. Esto puede ser configurado para integrar otros módulos.



Figura 44. Interfaz Viewpoint

Es posible visualizar detalle de algún proceso que se esté ejecutando si se selecciona el usuario particular y la sesión que está realizando dicha ejecución. Con esto se podrá ver el específico de las instrucciones que están siendo ejecutadas (SQL) y la segmentación que se está haciendo dentro del DBMS para ser procesada (Explain).

### 3.5.5 Monitor de procesos

Para poder centralizar la revisión de los procesos que están siendo ejecutados y el punto en el que se encuentran dentro del flujo completo, se creó un archivo en Excel que presenta la visión integral de la malla batch creada para esta solución.

La hoja está configurada para conectarse a la tabla de bitácora ETL\_PROD dentro de la base de datos de producción utilizando un ODBC que ya exista en la máquina en la se abra (con el usuario asociado a ese ODBC) y se actualiza cada minuto con el estatus de la ejecución. En ella se incluyen todas las fuentes de datos que fueron consideradas en esta fase mapeadas hacia las áreas sujeto en las cuales cae esta información.

Mediante un semáforo se indicará si los shells ejecutados están en proceso, finalizaron ejecución correctamente o tuvieron algún error de ejecución. En caso de que existan errores de ejecución se recomienda que se verifiquen los logs para obtener la falla particular que se presentó.

La siguiente figura muestra la interfaz del Monitor de procesos:



Figura 45. Monitor de procesos

Es importante mencionar que esta herramienta se desarrolló como soporte al seguimiento de la ejecución por parte del equipo de trabajo y, por lo mismo, no cuenta con un soporte adicional una vez entregada.

#### Integración de nuevos jobs al Monitor

Dado que es posible que se integren nuevos Jobs al ciclo de ejecución, el siguiente procedimiento describe las modificaciones que sería necesario hacer al Monitor en caso de querer que dichos jobs sean monitoreados usando este esquema.

1. Dentro del Excel de monitoreo se debe de desbloquear la pestaña Área Sujeto Detalle. La contraseña es XXXXXXXXX.



Figura 46. Desbloqueo de pestaña Área Sujeto Detalle

1. Una vez desbloqueada la pestaña, insertar una celda completa en donde se quiera agregar el nuevo Job. Poner el nombre del Job en la columna específica dependiendo su fase. Para completar el ciclo de ejecución, debe existir un script por lo menos en las fases STG, DWI y DWH. Los datos deben de registrarse en las columnas B, E, H, K, P, S.



Figura 47. Adición de nuevo Job

1. Para que aparezca el indicador de semáforo en la columna inmediata a la derecha, es posible copiar la fórmula como está en las celdas superiores o inferiores de esa misma columna cambiando los datos del número de celda en la fórmula que se copió para que queden asociados a la celda en la que se está creando el nuevo Job.



Figura 48. Copia de fórmulas

1. Seleccionar las columnas V y AE, dar clic derecho y seleccionar la opción Mostrar o Unhide. Esto desplegará un conjunto de columnas que de manera regular deben permanecer escondidas ya que contienen fórmulas esenciales para el funcionamiento del Monitor.



Figura 49. Mostrar celdas

Una vez desplegadas estas columnas, ubicar el renglón que se está adicionando, seleccionar las celdas del renglón inmediato superior desde la columna W a la AC y arrastrar las fórmulas sobre las celdas vacías (que corresponden al job que está siendo agregado).



Figura 50. Adición de fórmulas para Monitor

1. Dar clic derecho sobre alguna de las pestañas visibles y seleccionar la opción de UNHIDE y dar clic en el botón OK.



Figura 51. Mostrar pestañas escondidas.

Aparecerá un diálogo con la pestaña escondida Source1. Seleccionarla y presionar OK.



Figura 52. Pestaña Source1

1. Dentro de la pestaña Source1 seleccionar la celda X2 y arrastrarla a la altura del último registro que exista en la columna T. Esto para copiar la fórmula que aplicará al nuevo job adicionado.



Figura 53. Ajustes en pestaña Source1

1. Al finalizar, ocultar de nuevo la pestaña Source1 haciendo clic derecho en la pestaña y seleccionado Hide.



Figura 54. Ocultar pestaña Source1

Ocultar nuevamente las columnas W, X, Y, Z, AA, AB, AC, AD de la pestaña Área Sujeto Detalle. Seleccionar dichas columnas y dar clic derecho para que aparezca el menú. Ahí seleccionar Hide.



Figura 55. Ocultar celdas

1. Al finalizar, dar clic derecho sobre la pestaña Área Sujeto Detalle y seleccionar Protect Sheet. Asignar como contraseña XXXXXX.



Figura 56. Bloqueo de pestaña Área Sujeto Detalle

## 3.6 Incidencias comunes y resolución

### 3.6.1 Caso 1: “La tabla está siendo cargada”

Este error despliega un mensaje similar al siguiente:



Figura 57. Mensaje de error – la tabla está siendo cargada

Esto se refiere a que la tabla está siendo cargada o está bloqueada, por lo mismo no se puede ocupar hasta que sea liberada o haya terminado la carga exitosamente. En caso de que la tabla no cambie de estado en un periodo de 2 horas es necesario obtener la definición de la tabla bloqueada, borrarla y crearla de nuevo. Para ello, seguir estos pasos. Esto sucede principalmente cuando se está realizando la carga a Staging de las interfaces fuente y se presenta cuando alguna carga previa haya presentado una falla cuando los datos estaban siendo ya transmitidos del archivo hacia Teradata (por ejemplo, cuando un registro no cumple con el formato definido esto originará una falla en el proceso de carga que hará que la tabla receptora tenga un bloqueo).

**Paso # 1.** Utilizando la herramienta Teradata SQL Assistant, obtener la definición de la tabla que se marca como bloqueada con el siguiente código[[9]](#footnote-9):

**SHOW** **TABLE** ETL\_DP\_STG\_VPL.CIS\_DICCRE**;**

Lo cual mostrará el siguiente resultado:

**CREATE** **MULTISET** **TABLE** ETL\_DP\_STG\_VPL.CIS\_DICCRE ,**FALLBACK** ,

**NO** **BEFORE** **JOURNAL**,

**NO** **AFTER** **JOURNAL**,

**CHECKSUM** = **DEFAULT**,

**DEFAULT** MERGEBLOCKRATIO

(

CTA\_CUSTOMER DECIMAL(19,0),

CTA\_CVE DECIMAL(19,0),

CTE\_CVE DECIMAL(19,0),

CTE\_UNIC DECIMAL(19,0),

FLG\_TTLR SMALLINT,

FLG\_ACT SMALLINT,

FCH\_DESACT DATE FORMAT 'YYYY-MM-DD')

**PRIMARY** **INDEX** ( CTA\_CVE );

**Paso # 2.** Utilizando Teradata SQL Assistant, y manteniendo en pantalla el resultado del paso anterior, se debe eliminar la tabla bloqueada con la siguiente instrucción:

**DROP** **TABLE** ETL\_DP\_STG\_VPL.CIS\_DICCRE**;**

Después de volver a crear la tabla con la definición obtenida en el Paso #1, lo cual implica ejecutar el CREATE TABLE. Esto puede hacerse sombreando el texto y presionando F5.



Figura 58. Queries para resolución de error la tabla está siendo cargada en SQL Assistant

### 3.6.2 Caso 2: “Error en la carga”

En este caso la falla se debe a error de los datos que vienen en las interfaces entregadas por el sistema fuente que corresponda. Con esto nos referimos a que la carga no tuvo éxito porque el tipo de dato, longitud o el formato de dichos datos es diferente al esperado. Aunque es importante a nivel diagnóstico, este tipo de errores deben de ser reportados a un siguiente nivel de atención para que la falla sea resuelta de origen. Para realizar este diagnóstico:

**Paso #1**. Ir a la bitácora de ejecución y buscar el número de job que marcó el error. Esto se hace con el siguiente código ejecutado a través de Teradata SQL Assistant.

**SELECT** \* **FROM** ETL\_DP\_CTL.BIT\_TABLA\_CARGA

**WHERE** NB\_PROCESO = ‘job10009’;

En este caso se está revisando el job10009. El texto en rosa del query anterior debe de ser sustituido por el número de job que se quiera verificar (el nombre del job debe estar limitado por comilla simple (‘’)).

Al ejecutar el código se muestra el siguiente resultado. En el caso del ejemplo, se busca el job 10009.



Figura 59. Ejemplo de bitácora de ejecución para el Job10009

**Paso #2**. En caso de que el estatus sea “Error en la carga” buscar el log dentro de la carpeta de logs (/datastage\_data/Copa/DWH\_STG/logs) en Linux (servidor ETL\_PROD) para ver su descripción completa. El nombre de archivo de log es: job + .fecha + .log

Por ejemplo log del job10009, ejecutado el 20 de octubre de 2014, sería: Job10009.20141020.log



Figura 60. Carpeta de logs, servidor ETL\_PROD

Los tipos de error que se identifican en este caso pueden ser:

* Mal carácter, que indica que hay campos que contienen datos con caracteres especiales que no pueden ser interpretados durante el proceso de carga
* Falla en longitud del campo, que indica que alguno de los campos que se está recibiendo tiene un tamaño mayor al que está definido
* Campos faltantes o sobrantes, que indica que alguno de los registros tiene más o menos campos de los que están especificados en el layout acordado para dicha interfaz

Como se mencionó anteriormente, este tipo de error debe de ser notificado al área pertinente para que se envíe una nueva versión del archivo.

### 3.6.3 Caso 3: “Error diferencia en archivo cifras”

Esto se refiere a que el conteo de las cifras de los registros cargados (que vienen en el archivo con terminación .dat) fue diferente a las cifras de validación enviadas por los procesos transaccionales (que vienen en el archivo con terminación .cif). También se puede generar este error si el proceso no encuentra el archivo de cifras (.cif). Este error sólo se presentará durante el proceso de carga de las interfaces enviadas por las fuentes a Staging. Como en el caso anterior, este caso también es importante a nivel diagnóstico pero el error como tal debe de ser reportado a un siguiente nivel de atención para que la falla sea resuelta de origen. Para realizar este diagnóstico:

**Paso #1**. Ir a la bitácora de ejecución y buscar el número de job que marcó el error (en el ejemplo se está usando el job01048). El estatus se refleja como se muestra a continuación al verificar en la bitácora usando el Teradata SQL Assistant:



Figura 61. Bitácora de ejecución para el Job01048

**Paso #2**. Revisar el archivo de log del job que presentó la falla. Al revisar el archivo log de la ejecución se mostrará lo siguiente, lo cual indica que el conteo de cifras no cuadra con el de datos.



Figura 62. Ejemplo de archivo de log para diferencia de cifras

Como se mencionó anteriormente, este tipo de error debe de ser notificado al área pertinente para que se haga la revisión de las diferencias y se reciba una versión corregida o nueva del archivo .cif y/o .dat.

**Resultados y Discusión**

Al colaborar con esta herramienta se ha logrado llegar al resultado esperado, sin embargo aún hace falta el mejoramiento de algunos procedimientos. Los que son ejecutados de manera diaria para tener puntos de comparación y de respaldo han sido de gran importancia, ya que ha agilizado ciertos procesos que anteriormente se realizaban de manera manual.

**Conclusiones**

El trabajar con la herramienta Teradata para el manejo de un Data Warehouse es una ardua labor ya que tiene diferentes procedimientos para poder realizar ciertas operaciones con la información y poder laborar con las bases de datos, además cuenta con la facilidad de las sentencias SQL, sin embargo también se utiliza Microsoft SQL Server para poder crear los job y poder ejecutar los script ya que no tienes que preocuparte por lo que tienes que ejecutar, sino que tú indicas la hora y la fecha de la ejecución.

El tener en cuenta los registros de error te permite saber cuándo y que es lo que fallo de manera específica, la facilidad de Teradata es que todo lo almacena de manera histórica en formato \*.txt.

# Bibliografía

ABC, D. (2014). *Definición ABC*. Obtenido de http://www.definicionabc.com/tecnologia/datawarehouse.php

Coronel, C., Morris, S., & Rob, P. (2011). *Base de Datos Diseño, implementación y administración.* CENGAGE Learning.

Fernandez, C. (s.f.). *Dataprix*. Obtenido de Datawarehouse: http://www.dataprix.com/que-es-un-datawarehouse

Pagani, J. B. (24 de 10 de 2013). *Investigación IT*. Obtenido de http://investigacionit.com.ar/compresion-en-teradata/

Technopat. (13 de 11 de 2014). *Almacén de datos*. Obtenido de http://es.wikipedia.org/wiki/Almac%C3%A9n\_de\_datos

Teradata. (s.f.). *Teradata*. Obtenido de http://www.info.teradata.com/HTMLPubs/DB\_TTU\_15\_00/index.html/

# Anexos

## Anexo 1

|  |
| --- |
| LOCKING TABLE DD\_BIC\_DWH.SAK\_TRANSPORT\_EVENT\_NM FOR ACCESS  LOCKING TABLE DD\_BIC\_DWH.TRANSPORT\_EVENT FOR ACCESS  INSERT INTO DD\_RED\_METADATA.STEP\_25\_SIMM\_VLTL  (  TRAVEL\_INCIDENT\_NUM  , FLIGHT\_NUMBER\_DFN  , FLIGHT\_PRODUCT\_ID  , SCHEDULED\_LEG\_DEPART\_DTTM  , RESERVATION\_TRANSACTION\_ID  , RESERVATION\_SEQUENCE\_NUM  , BOOKED\_PARTY\_ID  , RecordLocator  , FlightNumber  , TOT\_BAGGAGE  )  SELECT TRIN.TRAVEL\_INCIDENT\_NUM,  TRIN.FLIGHT\_NUMBER\_DFN,  SSLE.FLIGHT\_PRODUCT\_ID,  SSLE.SCHEDULED\_LEG\_DEPART\_DTTM,  STP3.RESERVATION\_TRANSACTION\_ID,  STP3.RESERVATION\_SEQUENCE\_NUM,  STP3.BOOKED\_PARTY\_ID,  SKTE.RecordLocator, SKTE.FlightNumber,  TREV.TOT\_BAGGAGE  FROM DD\_RED\_METADATA.TRIN\_STEP\_1\_SSIM\_VLTL TRIN  JOIN  DD\_RED\_METADATA.SSLE\_STEP\_2\_SSIM\_VLTL SSLE  ON SSLE.FLIGHT\_NUMBER\_DFN = TRIN.FLIGHT\_NUMBER\_DFN  AND SSLE.ORIGIN\_AIRPORT\_ID = TRIN.ORIGIN\_AIRPORT\_ID  AND SSLE.DESTINATION\_AIRPORT\_ID = TRIN.DESTINATION\_AIRPORT\_ID  AND TRIN.SCHEDULED\_LEG\_DEPART\_DTTM >= SSLE.SCHEDULED\_LEG\_DEPART\_DTTM - INTERVAL '120' MINUTE  AND TRIN.SCHEDULED\_LEG\_DEPART\_DTTM <= SSLE.SCHEDULED\_LEG\_DEPART\_DTTM + INTERVAL '120' MINUTE  JOIN  DD\_RED\_METADATA.STEP\_3\_SIMM\_VLTL STP3  ON STP3.RELATED\_FLIGHT\_PRODUCT\_ID = SSLE.FLIGHT\_PRODUCT\_ID  AND SSLE.SCHEDULED\_LEG\_DEPART\_DTTM >= STP3.SCHEDULED\_SEGMENT\_DEPART\_DTTM - INTERVAL '120' MINUTE  AND SSLE.SCHEDULED\_LEG\_DEPART\_DTTM <= STP3.SCHEDULED\_SEGMENT\_DEPART\_DTTM + INTERVAL '120' MINUTE  LEFT OUTER JOIN  (  SELECT TRANSPORT\_EVENT\_NUM\_SAK,  PROCESS\_DATE, TRANS\_TRIP\_EVENT\_SUBTYPE\_CD,  RecordLocator, FlightNumber  FROM DD\_BIC\_DWH.SAK\_TRANSPORT\_EVENT\_NM  WHERE TRANS\_TRIP\_EVENT\_SUBTYPE\_CD = 'CHKIN'  AND RecordLocator <> ' '  AND RecordLocator IS NOT NULL  AND FlightNumber <> ' '  AND FlightNumber IS NOT NULL  ) SKTE  ON SKTE.RecordLocator = STP3.PNR\_NUM  AND SKTE.FlightNumber = TRIN.FLIGHT\_NUMBER\_DFN  LEFT OUTER JOIN  DD\_BIC\_DWH.TRANSPORT\_EVENT TREV  ON TREV.TRANSPORT\_EVENT\_NUM = SKTE.TRANSPORT\_EVENT\_NUM\_SAK  QUALIFY (  ROW\_NUMBER() OVER(  PARTITION BY STP3.RESERVATION\_TRANSACTION\_ID  , STP3.RESERVATION\_TRANSACTION\_ID  , STP3.RESERVATION\_SEQUENCE\_NUM  ORDER BY SKTE.PROCESS\_DATE DESC)  ) = 1; |

## Anexo 2

|  |
| --- |
| INSERT INTO DD\_MDB\_REVENUE\_MGMT.RM\_VENTAStmp\_tcn2  SELECT TK.TICKET\_NUM TICKET\_NUM,  A.TRAVEL\_COUPON\_ID COUPON\_ID,  CAST(CP.SCHEDULED\_SEGMENT\_DEPART\_DTTM AS DATE FORMAT 'YYYY-MM-DD') AS FLIGHT\_DT,  CAST( (CP.SCHEDULED\_SEGMENT\_DEPART\_DTTM (FORMAT 'HHMISS')) AS VARCHAR(10) ) FLOWN\_TIME,    CASE  WHEN TRIM(A.FARE\_BASIS\_CD) = '' THEN NULL ELSE A.FARE\_BASIS\_CD  END FAREBASIS\_CD,    CASE  WHEN TRIM(A.Booking\_Class\_Cd ) = '' THEN NULL ELSE A.Booking\_Class\_Cd  END SALES\_BOOKING\_CLASS\_CD,    CASE  WHEN TRIM(TRT.DISCOUNT\_CD ) = '' THEN NULL ELSE TRT.DISCOUNT\_CD  END TOURCODE\_CD,  A.PASSENGER\_ID PASSENGER\_ID,  A.PASSENGER\_NAME PASSENGER\_NAME,  CAST(TRT.TRANSACTION\_DTTM AS DATE) TICKET\_ISSUE\_DT,  TRP.FLIGHT\_NUMBER\_DFN FLIGHT\_NUMBER,  A.MARKETING\_FLIGHT\_NUMBER,  TRP.ORIGIN\_AIRPORT\_CD,  TRP.DESTINATION\_AIRPORT\_CD,    CASE  WHEN TRIM(A.FFP\_NUM) ='' THEN NULL ELSE A.FFP\_NUM  END FFP\_NUM,  CASE  WHEN TRIM(A.FFP\_AIRLINE\_CD)='' THEN NULL ELSE A.FFP\_AIRLINE\_CD  END FFP\_AIRLINE\_CD,  CASE  WHEN A.PNR\_NUM LIKE '%/%' THEN NULL ELSE A.PNR\_NUM  END PNR\_NUM,  CASE  WHEN A.PNR\_NUM LIKE '%/%' THEN NULL ELSE A.PNR\_CREATE\_DT  END PNR\_CREATE\_DT,  TK.TICKET\_AIRLINE\_NUM,  seat.flown\_seat\_num  FROM DD\_MDB\_REVENUE\_MGMT.TRAVEL\_PURCHASE\_ITEM\_TCN A  INNER JOIN DP\_DWH\_HOMO\_VWS.SAK\_TRAVEL\_PRODUCT TRP  ON A.TRAVEL\_PRODUCT\_ID = TRP.TRAVEL\_PRODUCT\_SAK  INNER JOIN DP\_VDWH.TRAVEL\_TRANSACTION TRT  ON A.PURCHASE\_TRANSACTION\_ID = TRT.TRAVEL\_TRANSACTION\_ID  AND TRT.TRAVEL\_TRANSACTION\_SUBTYPE\_CD = 'PURCH\_TCN'  INNER JOIN DP\_VDWH.TRAVEL\_COUPON CP  ON A.TICKET\_ID = CP.TICKET\_ID  AND A.TRAVEL\_COUPON\_ID = CP.TRAVEL\_COUPON\_ID  AND CP.TICKET\_SOURCE\_CD = 'TCN'  INNER JOIN DP\_VDWH.TICKET TK  ON A.TICKET\_ID = TK.TICKET\_ID  AND TK.TICKET\_SOURCE\_CD = 'TCN'  LEFT OUTER JOIN DP\_VDWH.TRANSPORTED\_PASSENGER seat  ON A.TICKET\_ID = seat.TICKET\_ID  AND A.TRAVEL\_COUPON\_ID = seat.TRAVEL\_COUPON\_ID  WHERE CAST( TRT.TRANSACTION\_DTTM AS DATE) >= (  SELECT Start\_Date  FROM DD\_MDB\_CUSTOMER\_MGMT.SUBJECT\_AREA\_FILTER\_DATE  WHERE Subject\_Area\_Id = 2)  AND CAST(TRT.TRANSACTION\_DTTM AS DATE) <= (  SELECT End\_Date  FROM DD\_MDB\_CUSTOMER\_MGMT.SUBJECT\_AREA\_FILTER\_DATE  WHERE Subject\_Area\_Id = 2) |

## Anexo 3

|  |
| --- |
| INSERT INTO DP\_IRROPS.IRROPS\_CHECKIN  SELECT B.SHFLIGHT\_ID,  B.RECORD\_ID,  B.FLIGHT\_NUMBER\_DFN,  B.SCHEDULED\_LEG\_DEPART\_DTTM\_TME,  B.ORIGIN\_AIRPORT\_ID,  TRIM(OLOC.AIRPORT\_IATA\_CD\_REP\_ARS) AS ORIGIN\_AIRPORT,  B.DESTINATION\_AIRPORT\_ID,  TRIM(DLOC.AIRPORT\_IATA\_CD\_REP\_ARS) AS DESTINATION\_AIRPORT,  B.COPATRANSACTION\_ID,  B.AGENTE\_CD,  B.LNIATA\_CD,  B.COPATRANSACTION\_DT,  B.FLIGHTHISTORY\_TEXTO\_DESC,  B.PASSANGER\_NAME,  B.CHECK\_IN\_METHOD\_CD,  MT.CHECK\_IN\_METHOD\_DESC,  B.DWHLOAD\_DT  FROM DP\_VDWH.PREFLIGHT\_PASANGER\_HIST B  INNER JOIN DP\_VDWH.Location OLOC  ON OLOC.LOCATION\_ID = B.ORIGIN\_AIRPORT\_ID  INNER JOIN DP\_VDWH.Location DLOC  ON DLOC.LOCATION\_ID = B.DESTINATION\_AIRPORT\_ID  INNER JOIN DP\_VDWH.CHECK\_IN\_METHOD MT  ON MT.CHECK\_IN\_METHOD\_CD = B.CHECK\_IN\_METHOD\_CD  WHERE ORIGIN\_AIRPORT\_ID IN (  SELECT ORIGIN\_AIRPORT\_ID  FROM DP\_IRROPS.IRROPS\_NUMTICKET\_T)  AND FLIGHT\_NUMBER\_DFN IN (  SELECT FLIGHT\_NUMBER\_DFN  FROM DP\_IRROPS.IRROPS\_NUMTICKET\_T)  AND DESTINATION\_AIRPORT\_ID IN (  SELECT DESTINATION\_AIRPORT\_ID  FROM DP\_IRROPS.IRROPS\_NUMTICKET\_T)  AND CAST(SCHEDULED\_LEG\_DEPART\_DTTM\_TME AS DATE) IN (  SELECT CAST(SCHEDULED\_SEGMENT\_DEPART\_DTTM AS DATE)  FROM DP\_IRROPS.IRROPS\_NUMTICKET\_T)  AND RECORD\_ID NOT IN (  SELECT RECORD\_ID  FROM DP\_IRROPS.IRROPS\_CHECKIN); |

## Anexo 4

|  |
| --- |
| INSERT INTO DP\_MDB\_QIS.RESERVATION\_SPECIAL\_RQ\_T (REQUEST\_TYPE\_CD ,REQUESTED\_TYPE\_DESC ,FFP\_NUM ,BOOKED\_PARTY\_ID ,PASANGER\_NAME ,CLIENT\_ID )  SELECT RT.REQUEST\_TYPE\_CD,  RT.REQUESTED\_TYPE\_DESC,  SPECIAL\_R.FFP\_NUM,  SPECIAL\_R.BOOKED\_PARTY\_ID,  SPECIAL\_R.PASANGER\_NAME,  SPECIAL\_R.CLIENT\_ID  FROM DP\_VDWH.REQUEST\_TYPE RT INNER JOIN (  SELECT RSR.REQUEST\_TYPE\_CD,  RSR.RESERVATION\_SEQUENCE\_NUM,  RSR.RESERVATION\_TRANSACTION\_ID,  ITEM\_RE.BOOKED\_PARTY\_ID,  ITEM\_RE.FFP\_NUM,  ITEM\_RE.PASANGER\_NAME,  ITEM\_RE.CLIENT\_ID  FROM DP\_VDWH.RESERVATION\_ITEM\_SPECIAL\_REQUE RSR INNER JOIN (  SELECT RI.RESERVATION\_TRANSACTION\_ID,  RI.RESERVATION\_SEQUENCE\_NUM,  RI.BOOKED\_PARTY\_ID,  T\_TRAN.FFP\_NUM,  SUBSTR(RI.PASANGER\_NAME ,1 ,(POSITION(('/') IN (RI.PASANGER\_NAME )))+ 3 )(NAMED PASANGER\_NAME ),  T\_TRAN.CLIENT\_ID  FROM DP\_VDWH.RESERVATION\_ITEM RI INNER JOIN (  SELECT TT.TRAVEL\_TRANSACTION\_ID,  TT.PNR\_NUM,  TRAVEL\_PUR.PASSAGER,  TT.TRAVEL\_TRANSACTION\_SUBTYPE\_CD,  TRAVEL\_PUR.FFP\_NUM,  TRAVEL\_PUR.CLIENT\_ID  FROM DP\_VDWH.TRAVEL\_TRANSACTION TT INNER JOIN DP\_MDB\_QIS.ORIGIN\_SEAT\_SSR\_T TRAVEL\_PUR ON TT.PNR\_NUM = TRAVEL\_PUR.PNR\_NUM  WHERE TT.TRAVEL\_TRANSACTION\_SUBTYPE\_CD = 'RESV') T\_TRAN ON T\_TRAN.TRAVEL\_TRANSACTION\_ID = RI.RESERVATION\_TRANSACTION\_ID  WHERE (SUBSTR( RI.PASANGER\_NAME ,1 ,(POSITION(('/') IN (RI.PASANGER\_NAME )))+ 3 ))= T\_TRAN.PASSAGER ) ITEM\_RE ON RSR.RESERVATION\_TRANSACTION\_ID = ITEM\_RE.RESERVATION\_TRANSACTION\_ID  WHERE RSR.RESERVATION\_SEQUENCE\_NUM = ITEM\_RE.RESERVATION\_SEQUENCE\_NUM ) SPECIAL\_R ON RT.REQUEST\_TYPE\_CD = SPECIAL\_R.REQUEST\_TYPE\_CD ; |

# Glosario

**Base de datos empresarial.-** Representación completa de los datos de una compañía, que proporcionan soporte para necesidades presentes y del futuro esperado.

**Base de datos de producción.-** Es la base de datos principal diseñada para dar seguimiento a las operaciones diarias de una compañía.

**Base de datos distribuida.-** Base de datos lógicamente distribuida que se guarda en dos o más sitios físicamente independientes.

**Bloqueo.-** Dispositivo empleado para garantizar el uso único de un elemento de datos para una operación de transacción particular, con lo cual se evita que otras transacciones lo utilicen. Una transacción requiere un bloqueo antes de tener acceso a los datos y ese bloqueo se libera después de la ejecución de la operación, para permitir que otras transacciones bloqueen el elemento de datos para su uso.

**Calidad de datos.-** Procesamiento completo para asegurar la precisión, validez y oportunidad de datos.

**Consulta ad hoc.-** Consulta hecha “sin pensarla”.

**Cubo de datos.-**Se refiere a la estructura multidimensional de datos que se emplea para corregir y manipular los datos en DBMS multidimensional. La ubicación de cada valor de datos del cubo de datos está basada en los ejes ‘x’, ‘y’ y ‘z’ del cubo. Los cubos de datos son estáticos (deben crearse antes de usarlos), de modo que no pueden ser creados por una consulta ad hoc.

**Data Warehouse.-** Es una colección de datos orientada a un determinado ámbito (empresa, organización, etc.), integrado, no volátil y variable en el tiempo, que ayuda a la toma de decisiones en la entidad en la que se utiliza.

**Dimensiones.-** En un diseñode esquema en estrella, se refiere a características calificadoras que dan perspectiva, adicionales a un dato determinado.

**Job.-** Es una serie de pasos específicos de operaciones secuenciales predefinidas por SQL Server Agent. Puede ejecutarse de manera repetida programando su ejecución.

**Minería de datos.-** Proceso que emplea herramientas automatizadas para analizar datos en un almacén de datos, otras fuentes y para identificar proactivamente posibles relaciones y anormalidades.

**Procedimiento.-** Serie de pasos a seguir durante la operación de una actividad o proceso.

**Replica de datos.-** Almacenamiento de fragmentos duplicados de una base de datos en múltiples sitios en un DDBMS. La réplica de los fragmentos es transparente al usuario final. Se usa para dar tolerancia a fallas y mejoramientos de desempeño.

**Réplica.-** Es el proceso de crear y administrar versiones duplica desde una base de datos. Se usa para poner copias en diferentes lugares, mejorar el tiempo de acceso y tolerancia a fallas.

1. Las X representan números secuenciales para los shells creados [↑](#footnote-ref-1)
2. Para mayor detalle en relación a la definición de las fases de transformación, revisar la estrategia ETL planteada. [↑](#footnote-ref-2)
3. Aproximadamente un 95% de los procesos implementados tienen dependencias hacia archivos generados por las diferentes aplicaciones fuente. [↑](#footnote-ref-3)
4. El número de pasos que se ejecuten depende de las secuencias que hayan sido dadas de alta en el archivo de parámetros para ese job. Si el job cuenta con tres secuencias, se ejecutarán los cuatro pasos mencionados en este bloque. Si sólo se tienen dos secuencias quiere decir que para ese archivo de datos no existe archivo de cifras de control (.ctl). [↑](#footnote-ref-4)
5. Las tablas de forma general tienen el mismo nombre que las interfaces. [↑](#footnote-ref-5)
6. Las tablas históricas se diferencian de las de carga porque su nombre comienza con H. [↑](#footnote-ref-6)
7. En el caso de direcciones de clientes se asignará el identificador creado en la fase de homologación al registro correspondiente que ha sido transformado y reside en la base de datos de imagen. Esta combinación de datos será la que se inserte en DWH. [↑](#footnote-ref-7)
8. No está en ejecución en este momento [↑](#footnote-ref-8)
9. Es importante mencionar que en Teradata las tablas se referencian como <BASE DE DATOS>.<TABLA> [↑](#footnote-ref-9)