UNIVERSIDAD NACIONAL DE LA PLATA

FACULTAD DE INFORMÁTICA

TESINA DE LICENCIATURA

Programa de Apoyo al Egreso de Profesionales en Actividad

TÍTULO: Proceso de Migración de una Base de Datos Operativa para el SENASA

AUTOR: Malek Camilo

DIRECTOR ACADÉMICO: Mag. Rodolfo Bertone

DIRECTOR PROFESIONAL: Lic. Guillermo Capelli

CARRERA: Licenciatura en Informática

Resumen

El presente trabajo de tesis tiene como principal objetivo exponer los procedimientos y criterios para lograr la migración de una base de datos, entendiendo esta como un almacén de datos, relacionados y estructurados, controlados por sistemas de gestión. Esta exposición es producto de la tarea realizada para el Servicio Nacional de Sanidad y Calidad Agroalimentaria (Senasa).

Palabras Clave

Base de datos, Migración, Scripts, Bash, Performance, Encoding, Tipos de datos, Índices de texto, PostgreSQL, Oracle.

Conclusiones

Se alcanzaron los objetivos esperados, se logró la migración de datos y una performance de la base de datos acorde a los requerimientos del organismo.
A partir de la nueva definición obtenida bajo un motor como Oracle, se estudiaron diferentes contextos para evaluar el rendimiento final del motor en cuanto a la utilización de la base de datos.

Trabajos Realizados

Análisis y cambio del encoding del juego de caracteres, pasando de norma ISO 8859-1 a UTF-8.
Análisis de los tipos de datos de origen y su equivalencia en el motor de base de datos destino.
Recreación de toda la estructura y transferencia de la información mediante scripts desarrollados en bash desde la base de datos origen hacia la base de datos destino.
Implementación de índices de texto de Oracle para aumentar el rendimiento en ciertas aplicaciones.

Trabajos Futuros

Esta migración se realizó entre los años 2010 y 2011, con las herramientas disponibles en aquel momento, por ello, como trabajo futuro se podrían adaptar los scripts a un lenguaje de scripting más moderno y flexible aprovechando técnicas de concurrencia o paralelismo, disminuyendo los tiempos de migración.
Asimismo, se podría extender la compatibilidad con otros motores de base de datos utilizando la misma técnica que se adoptó en este documento.

Fecha de la presentación: Febrero 2022

Proceso de Migración de una Base de Datos Operativa para el Senasa

Tesina de Licenciatura en Informática

Malek Camilo

Director Académico: Mag. Rodolfo Bertone
Director Profesional: Lic. Guillermo Capelli

Facultad de Informática
Universidad Nacional de La PLata

Agradecimientos

A mi hijo, Nicolás, por ser el motor principal que me impulsó a terminar mi carrera y fuente de motivación para ser cada día un poquito mejor.

A mi compañera de vida, Lucía Terruzzi, por tenerme paciencia.

A Graciela Curihuil, por estar siempre presente.

A Felicitas Ahumada y a Juan Marra, por darme la oportunidad de trabajar en este interesante proyecto.

Al Senasa, y por sobre todo a Guillermo Capelli y a Héctor Bilbao, por confiar en mi para esta apasionante tarea.

A los desarrolladores de aplicaciones del Senasa, por su estrecha y amena colaboración.

A Pedro Baravalle y a Damián Ienco, por sus aportes metodológicos.

Finalmente, a la Universidad Nacional de La Plata, por brindarme una educación de excelencia, la cual, al tiempo que pasan los años, más la valoro.

Índice general

Índice de códigos

Índice de figuras

Resumen

El presente trabajo de tesis tiene como principal objetivo exponer los procedimientos y criterios para lograr la migración de una base de datos, entendiendo esta como un almacén de datos, relacionados y estructurados, controlados por sistemas de gestión. Esta exposición es producto de la tarea realizada para el Servicio Nacional de Sanidad y Calidad Agroalimentaria (Senasa).

Inicialmente se llevó a cabo un proceso de planificación y análisis del trabajo a desarrollar, de manera de acercarnos lo máximo posible al éxito de la operación y reducir los costos acarreados por errores de datos.

Para asegurar el volumen de datos se realizaron conteos de registros utilizando las vistas de catálogo y las utilidades ofrecidas por las herramientas de los diferentes motores de base de datos.

Se acordó con el Área de Infraestructura del Senasa la regla general de cómo se mapearía cada tipo de dato. Luego se trabajó con los casos puntuales de cada aplicación.

Antes de iniciar la migración de la base de datos, se desactivaron los disparadores (*triggers*) o restricciones (*constraints*) que podían generar un error en el momento en que el DBMS (*DataBase Management System*) ejecute el proceso de escritura de datos.

Se realizó un análisis del conjunto de caracteres a utilizar, tanto para la creación de la base de datos, como para las herramientas que se utilizarían en la migración.

Para realizar la migración de un motor de base de datos a otro se redactó un procedimiento con las instrucciones a seguir por cada aplicación a migrar, contando con dos iteraciones de las mismas.

La primera iteración se realizó en un entorno de pruebas para asegurar la consistencia de datos y estructuras. Una vez validada la corrección de la primera iteración, se realizó la migración al entorno productivo.

Paralelamente, se siguió un protocolo de migración agrupando las aplicaciones que compartían integridad referencial, de esta forma, se migraba atómicamente dicho grupo de aplicaciones.

A partir del estudio del problema y su nueva implantación en un motor de base de datos diferente, se plantearon los siguientes cambios o estudios:

- Cambio del *encoding* del juego de caracteres, pasando de norma *ISO 8859-1* a *UTF-8* por temas relacionados con la internacionalización de caracteres a futuro.

- Recreación de toda la estructura debido al cambio de motor de base de datos, realizando *scripts* en *bash*, y consultando el catálogo de la base de datos de origen.

- Carga de la información a archivos mediante *scripts* desde el origen, para luego realizar la recuperación de la información hacia la base de datos destino. Para llevar a cabo esta tarea, se utilizaron herramientas como: *psql* y *COPY* del lado de PostgreSQL y *sqlldr* del lado de Oracle.
- Análisis de herramientas de Oracle para aumentar el rendimiento en ciertas aplicaciones, como por ejemplo, la utilización de *Oracle Text*.
- Por último, se analizaron los tipos de datos de origen y se buscó su equivalencia en el motor de base de datos destino.

Introducción

Primeramente, se presenta la motivación de esta tesina (Sección 1.1). Luego se enuncian los objetivos y la metodología a emplear (Sección 1.2). A continuación los resultados obtenidos (Sección 1.3). Finalmente, se describe la organización del documento (Sección 1.4).

1.1. Motivación

Todas las aplicaciones web del Senasa -alrededor de 42-, tenían implantadas su base de datos con el motor de PostgreSQL desde los tiempos de su definición, hacia el año 2006.[1]

A partir de la utilización y evolución natural de los sistemas, la *performance* obtenida del motor de base de datos se fue degradando. Además, la escasa o casi nula documentación asociada a la base de datos existente hacía muy complejo el trabajo de puesta a punto.

Ante la necesidad de lanzar una nueva aplicación web de gran alcance llamada SIGSA (Sistema de Gestión Sanitaria), quedó en evidencia que los resultados esperados no se iban a lograr si se continuaba con el uso del motor de base de datos existente. Para justificar lo señalado, se realizaron pruebas iniciales sin obtener los resultados esperados.

Es en este escenario que se presentó como alternativa la migración de datos del motor de base de datos PostgreSQL a un motor más potente: Oracle.[2]

1.2. Objetivos y metodología

El objetivo principal de la migración era lograr un correcto traspaso de la información contenida en un motor de base de datos -en este caso PostgreSQL-, a otro motor, Oracle, sin pérdida de información y de forma transparente, logrando así poca o casi nula intervención en el desarrollo de las aplicaciones. La metodología implementada para dicha migración consta de *scripts* desarrollados en *bash*, utilizando distintas herramientas provistas por ambos motores de base de datos.

1.3. Resultados obtenidos

Se alcanzaron los objetivos esperados, se logró la migración de datos y una *performance* de la base de datos acorde a los requerimientos del organismo.

Además, a partir de la nueva definición obtenida bajo un motor como Oracle, se estudiaron diferentes contextos para evaluar el rendimiento final del motor en cuanto a la utilización de la base de datos.

1.4. Organización del documento

El resto del documento se organiza de la siguiente forma:

- En el capítulo 2 se expone el análisis realizado del nuevo conjunto de caracteres a utilizar y las pruebas realizadas avalando dicha elección.
- En el capítulo 3 se realiza una comparación de los tipos de datos soportados por ambos motores de base de datos y se traza una equivalencia.
- En el capítulo 4 se explican los *scripts* desarrollados para la migración y se ejemplifica la ejecución de los mismos.
- En el capítulo 5 se describe como se agruparon las aplicaciones para su migración a la fase de producción.
- En el capítulo 6 se detallan las planillas de control utilizadas como guías.
- En el capítulo 7 -ya con la migración finalizada-, se describe la utilización de un índice especial de Oracle llamado "índice de texto"que aporta *performance* a una aplicación en particular.
- Por último, en el capítulo 8, se exponen las conclusiones obtenidas del trabajo.

Conjunto de caracteres

El cambio de juego de caracteres de la base de datos origen fue planteado al Senasa básicamente para lograr una compatibilidad con diferentes idiomas en el futuro cercano.

Sabiendo que las aplicaciones que utilizan la base de datos del organismo registran importaciones y exportaciones de materia prima de origen animal y vegetal, desde distintos países, ya sean estos del Mercosur o de otras regiones del mundo, se creyó conveniente dejar los principios asentados para dicha internacionalización.

Además, y como se explica a lo largo del documento, el juego de caracteres no se puede cambiar una vez creada la base de datos, por lo que fue un tema a tratar desde el inicio del proyecto y de resolución definitiva.

2.1. Análisis del conjunto de caracteres

En este apartado se describe el informe técnico que se realizó -antes de realizar la migración-, sobre el conjunto de caracteres a utilizar.

El conjunto de caracteres se especifica cuando se crea la base de datos y esto determina qué caracteres podrán ser representados en la misma. Dicha elección influye en cómo se crean los objetos en la base de datos y en cómo se desarrollan las aplicaciones que procesan la información. Esta elección repercute en la interoperabilidad de los recursos del sistema operativo y el rendimiento de la base de datos.

Un grupo de caracteres (por ejemplo: caracteres alfabéticos, ideográficos, símbolos, signos de puntuación, caracteres de control, etc.) pueden ser codificados como un conjunto de códigos de caracteres.

Oracle soporta la mayoría de los conjuntos de caracteres usados en la industria de la tecnología, ya sean estos nacionales, internacionales o propietarios.

Se analizó y evaluó qué conjunto de caracteres se debía utilizar en la base de datos con el fin de garantizar la consistencia de la información existente al momento de la migración a la tecnología Oracle y en la futura operación de las aplicaciones.

En este caso en particular se decidió utilizar *UTF-8 (8-bit Unicode Transformation Format*) como conjunto de caracteres de internacionalización.[3]

Entre sus características podemos citar [4]:

- La forma de codificación *UTF-8* mantiene la transparencia para todos los puntos del código *ASCII*, gracias a ello cualquier mensaje *ASCII* se representa sin cambios.

- Es una forma de codificación de ancho variable, que utiliza unidades de código, representados de 1 a 4 *bytes*, de 8 *bits*, en la que los *bits* altos de cada unidad de código indican la parte de la secuencia de la unidad de código a la que cada *byte* pertenece.
- Es razonablemente compacto en términos de la cantidad de *bytes* utilizados. Comparado con *UTF-16*, es mucho más pequeño para la sintaxis *ASCII* y los idiomas occidentales, pero significativamente más grande para los sistemas de escritura asiáticos como el hindi, tailandés, chino, japonés y coreano.
- Incluye sincronización automática. Esto significa que el primer *byte* de una secuencia de unidad de código *UTF-8* indica el número de *bytes* a seguir en una secuencia *multibyte*, permitiendo un análisis directo muy eficiente. Es muy beneficioso para encontrar el comienzo de un carácter cuando se comienza desde una ubicación arbitraria en un flujo de *bytes*. Los programas necesitan buscar como máximo cuatro *bytes* hacia atrás y, por lo general, mucho menos. Incluso, es una tarea sencilla reconocer un *byte* inicial, porque los *bytes* iniciales están restringidos a un rango fijo de valores.
- Al igual que con las otras formas de codificación, no hay superposición de valores de *bytes*, por lo tanto no es posible confundirlos entre sí.

2.2. Pruebas realizadas

Las pruebas realizadas, antes de implementar *UTF-8*, fueron las siguientes:

- Se crearon dos bases de datos de prueba con diferentes conjuntos de caracteres, una de ellas con *WE8MSWIN1252* y otra con *AL32UTF8*.
- Se realizaron diferentes pruebas insertando caracteres españoles y símbolos desde una aplicación cliente y otra desde el mismo servidor que alojaban a las bases de datos.
- Se verificó que los datos fueran almacenados y mostrados correctamente.

Codificación caracteres en *WIN1252*:

CHARACTERSET	TYPES_USED_IN
AL16UTF16	NCHAR
AL16UTF16	NCLOB
AL16UTF16	NVARCHAR2
WE8MSWIN1252	CHAR
WE8MSWIN1252	CLOB
WE8MSWIN1252	LONG
WE8MSWIN1252	VARCHAR2

Codificación caracteres en *UTF-8*:

CHARACTERSET	TYPES_USED_IN
AL16UTF16	NCHAR
AL16UTF16	NCLOB
AL16UTF16	NVARCHAR2
AL32UTF8	CHAR
AL32UTF8	CLOB
AL32UTF8	LONG
AL32UTF8	VARCHAR2

Pruebas de almacenamiento y lectura en *UTF-8* con los clientes *sqldeveloper* y *sqlplus*:

PRUEBA_CHAR	PRUEBA_VARCHAR
^‘{}[]+~*$%°|!&	^‘{}[]+~*$%°|!&
áéíóúÁñÉÍÓÚÑ	áéíóúÁñÉÍÓÚÑ

De acuerdo a las pruebas y a la investigación realizada, se concluyó que:

1- Las bases de datos usan el *CHARACTERSET = AL16UTF8* como *NATIONAL CHARACTERSET*, lo cual es correcto y recomendado por Oracle.

> *The national character set, also called NCHAR character set, is the character set used to store and process data of data types NVARCHAR2, NCHAR, and NCLOB. Unless installation requirements of any of your applications specify otherwise, accept the default value of AL16UTF16 as the national character set.*[5]

2- Oracle aconseja el uso del conjunto de caracteres *AL32UTF8* para la base de datos, esto es indispensable para cualquier tecnología multilenguaje. Se deja un fragmento de la documentación de Oracle avalando dicha elección:

> *Selecting a Character Set*
>
> *Oracle recommends AL32UTF8 as the database character set. AL32UTF8 is Oracle's name for the UTF-8 encoding of the Unicode standard. The Unicode standard is the universal character set that supports most of the currently spoken languages of the world.*
> *The use of the Unicode standard is indispensable for any multilingual technology, including database processing.*
> *After a database is created and accumulates production data, changing the database character set is a time consuming and complex project. Therefore, it is very important to select the right character set at installation time. Even if the database does not currently store multilingual data but is expected to store multilingual data within a few years, the choice of AL32UTF8 for the database character set is usually the only good decision.*
> *...*
> *Oracle Client libraries automatically perform the necessary character set conversion between the database character set and the character sets used by non-Windows client applications.*
> *You may also choose to use any other character set from the presented list of character sets. You can use this option to select a particular character set required by an application vendor, or choose a particular character set that is the common character set used by all clients*

connecting to this database.
As AL32UTF8 is a multibyte character set, database operations on character data may be slightly slower when compared to single-byte database character sets, such as WE8MSWIN1252.
Storage space requirements for text in most languages that use characters outside of the ASCII repertoire are higher in AL32UTF8 compared to legacy character sets supporting the language.
Note that the increase in storage space concerns only character data and only data that is not in English.
The universality and flexibility of Unicode usually outweighs these additional costs. [5]

3- Además, se podía seleccionar un conjunto de caracteres específico si era requerido por una aplicación específica.

Equivalencias de tipos de datos

Uno de los inconvenientes presentados al momento de migrar los datos de un producto de base de datos a otro era la compatibilidad de los tipos de datos a representar.

Por lo tanto se confeccionaron tablas de compatibilidad que se sometieron a pruebas y revisiones constantes con el personal técnico del Senasa hasta lograr una versión final y estable.

Algunos tipos de datos tenían una equivalencia directa, otros no existían y otros directamente no se implementaron.

Estos son los tipos de datos soportados por PostgreSQL:

- Los tipos *smallint (2 bytes)*, *integer (4 bytes)* y *bigint (8 bytes)* almacenan números enteros, es decir, números sin componentes fraccionarios, de varios rangos. Los intentos de almacenar valores fuera del rango permitido darían como resultado un error.

  Los nombres del tipo *int2*, *int4* e *int8* son extensiones que también son utilizadas por algunos otros sistemas de bases de datos *SQL*.

- El tipo *numeric (variable bytes)* puede almacenar números de hasta 1000 dígitos de precisión y realizar cálculos con exactitud.

- Los tipos de datos *real (4 bytes)* y *double precision (8 bytes)* son tipos numéricos inexactos de precisión variable.

  En la práctica, estos tipos suelen ser implementaciones del estándar IEEE 754 para aritmética binaria de coma flotante (precisión simple y doble, respectivamente), en la medida en que el procesador subyacente, el sistema operativo y el compilador lo admitan.

- Los tipos *character varying(n)* y *character(n)*, donde n es un número entero positivo, pueden almacenar cadenas de hasta n caracteres de longitud. Intentar almacenar una cadena más larga en una columna de estos tipos resultará un error, a menos que los caracteres sobrantes sean todos espacios, en cuyo caso la cadena se truncará a la longitud máxima. Si la cadena a almacenar es más corta que la longitud declarada, los valores de tipo *character* se rellenarán con espacios; los valores de *character varying* simplemente almacenarán la cadena más corta.

  PostgreSQL proporciona el tipo *text*, que almacena cadenas de cualquier longitud. Aunque el tipo de texto no está en el estándar *SQL*, varios sistemas de administración de bases de datos *SQL* también lo tienen.

- PostgreSQL proporciona el tipo *boolean* de SQL estándar, utiliza 1 *byte* de almacenamiento, y puede tener uno de solo dos estados: *true* o *false*.

  Un tercer estado, *unknown*, está representado por el valor nulo de SQL.

- Para los tipos de datos relacionados con fecha y hora, PostgreSQL nos ofrece:

  - *timestamp[(p)][without time zone]*: representa fecha y hora sin *time zone*, ocupa 8 *bytes*.
  - *timestamp[(p)][with time zone]*: representa fecha y hora con *time zone*, ocupa 8 *bytes*.
  - *time [(p)] [without time zone]*: representa solo hora sin *time zone*, ocupa 8 *bytes*.
  - *time [(p)] [with time zone]*: representa solo hora con *time zone*, ocupa 12 *bytes*.
  - *date*: representa solo fecha, ocupa 4 *bytes*.

    Los tipos de datos *time* y *timestamp* aceptan un valor de precisión opcional p que especifica el número de dígitos fraccionarios retenidos en el campo "segundos". De forma predeterminada, no hay un límite explícito en la precisión.

- El tipo de datos *bytea* permite el almacenamiento de cadenas binarias.

  Una cadena binaria es una secuencia de octetos (o *bytes*). Las cadenas binarias se distinguen de las cadenas de caracteres de dos maneras: en primer lugar, las cadenas binarias permiten específicamente almacenar octetos de valor cero y otros octetos "no imprimibles"(generalmente, octetos fuera del rango de 32 a 126).

  Las cadenas de caracteres no permiten cero octetos y tampoco permiten ningún otro valor de octeto y secuencias de valores de octeto que no sean válidos según la codificación del juego de caracteres seleccionado de la base de datos.

  En segundo lugar, las operaciones en cadenas binarias procesan los *bytes* reales, mientras que el procesamiento de cadenas de caracteres depende de la configuración regional.

  En resumen, las cadenas binarias son apropiadas para almacenar datos que el programador considera "*bytes* sin formato", mientras que las cadenas de caracteres son apropiadas para almacenar texto.

Tipos de datos soportados por Oracle:

- *Varchar2(s [byte — char])*, cadena de caracteres de longitud variable que tiene como tamaño máximo el valor de *s* expresado en *byte* o *char*.

  El tamaño máximo es de 4000 *bytes* o caracteres, y la mínima es de 1 *byte* o un carácter.

- *Number[(p [, s])]*, número con *p* precisión (parte entera) y escala *s* (parte decimal).

  La precisión *p* puede variar de 1 a 38 y la escala puede variar desde -84 hasta 127.

  Tanto la precisión y la escala se encuentran en dígitos decimales. Un valor numérico requiere 1 a 22 *bytes*.

- *Float[(p)]*, es un subtipo del tipo de datos *number* con precisión *p*.

  Un valor de coma flotante se representa internamente como un *number*.

  La precisión *p* puede variar desde 1 hasta 126 dígitos binarios. Un valor flotante requiere 1 a 22 *bytes*.

- El tipo de dato *date* es un intervalo de fechas válidas desde el 1 de enero de 4712 antes de Cristo hasta el 31 de diciembre de 9999.

  El formato por defecto se determina explícitamente por el parámetro NLS_DATE_FORMAT o implícitamente por el parámetro.

  El tamaño es de 7 *bytes*. Este tipo de datos contiene los campos de fecha y hora. No tiene fracciones de segundo o de una zona horaria.

- *Timestamp [(fractional_seconds_precision)]*, este tipo de dato representa año, mes y día como valores de la fecha, así como la hora, minutos y segundos como valores de tiempo, donde fractional_seconds_precision es el número de dígitos en la parte fraccionaria del segundo del campo *datetime*.

  Los valores aceptados de fractional_seconds_precision son del 0 al 9. El valor por defecto es 6.

  El formato por defecto se determina explícitamente por el parámetro NLS_TIMESTAMP_FORMAT o implícitamente por el parámetro NLS_TERRITORY.

  El tamaño es de 7 o 11 *bytes*, dependiendo de la precisión. Contiene las fracciones de segundo, pero no tiene una zona horaria.

- *Char [(s [byte — char])]*, este tipo de datos representa una cadena de caracteres de longitud fija de *s bytes* de tamaño o de caracteres.

  El tamaño máximo es de 2000 *bytes* o caracteres, el tamaño predeterminado y mínimo es de 1 *byte*.

- El tipo de dato *clob* es un objeto de tipo *lob* que contiene caracteres de un *byte* o *multibyte*.

  Son compatibles tanto de ancho fijo y conjuntos de ancho variable de caracteres, con el caracter de base de datos establecida.

  El tamaño máximo es (4 *gigabytes* - 1) * (tamaño del bloque de la base de datos).

- El tipo de dato *blob* es un objeto de tipo *lob* binario.

  El tamaño máximo es (4 *gigabytes* - 1) * (tamaño del bloque de la base de datos).

Tabla resultante luego de las iteraciones mencionadas.

PostgreSQL	**Oracle**
BOOLEAN	NUMBER(1,0)
CHARACTER(n)	CHAR(n)
CHARACTER VARYING(n)	VARCHAR2(n)
BIGINT	NUMBER(p,s)
SMALLINT	NUMBER(p,s)
INTEGER	NUMBER(p)
NUMERIC	NUMBER(p,s)
REAL	FLOAT(63)
DOUBLE PRECISION	FLOAT(126)
DATE	DATE
TIME WITHOUT TIME ZONE	TIMESTAMP
TIMESTAMP WITH TIME ZONE	TIMESTAMP WITH TIME ZONE
TIMESTAMP WITHOUT TIME ZONE	TIMESTAMP
TEXT	CLOB
BYTEA	BLOB
OID	BLOB

table_schema	table_name	column_name	data_type	udt_name	character_n
adt	actividad	nombre	character varying	varchar	255
adt	actividad	id_arancel	bigint	int8	
adt	actividad	id_coordinacion	integer	int4	
adt	actividad	id	integer	int4	
adt	aplicacion_por_producto	momento_aplicacion	character varying	varchar	255
adt	aplicacion_por_producto	id_producto	bigint	int8	
adt	aplicacion_por_producto	dosis	character varying	varchar	255
adt	aplicacion_por_producto	id_plaga	bigint	int8	
adt	aplicacion_por_producto	id_aplicacion	integer	int4	
adt	aplicacion_por_producto	id_cultivo	bigint	int8	
adt	aprobacion	id	integer	int4	
adt	aprobacion	fecha_asignacion	date	date	
adt	aprobacion	id_tramite	integer	int4	
adt	aprobacion	fecha_ult_mod	date	date	
adt	aprobacion	id_usuario_aprobacion	integer	int4	
adt	aprobacion	fecha_creacion	date	date	
adt	aprobacion	id_area_aprobacion	integer	int4	
adt	aprobacion	fecha_aprobacion	date	date	
adt	aprobacion	id_estado_aprobacion	integer	int4	
adt	aprobacion	observacion	character varying	varchar	255
adt	aprobacion	observacion_cliente	character varying	varchar	255
adt	aprobacion	usuario_creacion	character varying	varchar	50
adt	aprobacion	usuario_ult_mod	character varying	varchar	50
adt	arancel	id	bigint	int8	
adt	arancel	descripcion	character varying	varchar	255
adt	arancel	importe	double precision	float8	
adt	area	id	bigint	int8	
adt	area	nombre	character varying	varchar	50

Figura 3.1: Planilla de tipo de datos de PostgreSQL

De la tabla anterior se desprende la siguiente tabla la cuál representa las equivalencias de columnas de la tabla *information_schema.columns* que fue utilizada en el *script create_tables.sql*.

Los tipos de datos, *time without timezone, timestamp with time zone y oid* no se encuentra en dicha tabla.

El tipo *oid (Object identifiers)* fue eliminado de la migración porque este es un tipo de dato usado internamente por PostgreSQL. Los tipo *time without timezone* y *timestamp with time zone* fueron reemplazados en el origen por el tipo *timestamp without timezone* que es equivalente a *timestamp*.

PostgreSQL data_type	**PostgreSQL udt_name**
boolean	bool
character varying	varchar
character	bpchar
text	text
smallint	int2
integer	int4
bigint	int8
numeric	numeric
real	float4
double precision	float8
date	date
bytea	bytea
timestamp without time zone	timestamp

Migración

En este capítulo se exponen los *scripts* que se utilizaron para realizar la migración de datos.

Cabe aclarar que antes de optar por los *scripts* en *bash* que en las sucesivas secciones se detallan, se probaron distintas herramientas para realizar dicha migración.

Entre ellas se encuentra *Pentaho*[6], que proveía una edición comunitaria que permitía realizar extracción, transformación y carga de datos desde distintos motores de base de datos.

El inconveniente de esta herramienta -como así también en otras que fueron probadas-, es que se tenía muy poco control sobre las distintas etapas de la migración, no era sencillo *tracear* los errores de carga de datos y al ser productos cerrados. no tenían la flexibilidad necesaria para los requerimientos que iban surgiendo.

De todas maneras, se aprovechó *Pentaho* para realizar la migración de los objetos binarios (*blob*) puesto que era bastante complicado realizarlo vía *script* por el formato de almacenamiento que utiliza PostgreSQL para este tipo de datos.

Con estas dificultades se decidió, entonces, comenzar a desarrollar los *scripts* de manera tal que pudiesen solventar las faltas de estas herramientas.

De los *scripts* de migración se excluyeron las funciones y los *triggers* almacenados por no tener una forma trivial de automatizarlos. Por ello, dichos objetos se migraron de forma manual y con colaboración de los agentes desarrolladores.

De todas maneras no fue una tarea complicada puesto que la gran mayoría de las aplicaciones contaba con poco desarrollo de funciones y *triggers* en el motor de base de datos.

4.1. Creación de objetos

Una vez analizado y definido el conjunto de caracteres a utilizar en la nueva base de datos Oracle, se procedió a crear los esquemas de base de datos para cada aplicación. Esto suponía la creación del usuario y su *tablespace*.

Con los esquemas ya definidos se crearon los objetos correspondientes desde la base de datos origen a la de destino mediante los siguientes *scripts*, los cuales se detallan a continuación:

El *script create_obj.sh* es el encargado de conectarse a la base de datos origen (PostgreSQL) y ejecutar distintas consultas *sql* sobre el catálogo de dicha base para obtener los objetos a crear, escribiendo en archivos las distintas salidas. Se debe parametrizar el *script* con el nombre del esquema.

Código 4.1 create_obj.sh

```
1  #!/bin/sh
2  export PGCLIENTENCODING='UTF8'
3  export PGPASSWORD='******'
4
5  CONF=sql.conf
6  source $CONF
7
8  rm cs.sql
9  rm ct.sql
10 rm cc.sql
11 psql -h $PHOST -p $PPORT -U postgres -d $PDB -A -t -F ' ' -v sch="'$1'
       " -f create_sequences.sql -o cs.sql
12 psql -h $PHOST -p $PPORT -U postgres -d $PDB -A -t -F ' ' -v sch="'$1'
       " -f create_constraints.sql -o cc.sql
13 psql -h $PHOST -p $PPORT -U postgres -d $PDB -A -t -F ' ' -v tab=
       table_name -v sch="'$1'" -f tablas_pg.sql -o tablas_pg.txt
14 for i in `cat tablas_pg.txt` ; do
15   psql -h $PHOST -p $PPORT -U postgres -d $PDB -A -t -F ' ' -v tab="'
       $i'" -v sch="'$1'" -f create_tables.sql >>ct.sql
16 done;
```

En la linea 5 del *script create_obj.sh* se realiza el *seteo* del archivo de configuración, que se muestra a continuación, y posee los datos de la base de datos origen a conectarse.

Este archivo de configuración además es utilizado por los *scripts load_data.sh*, *generate_seq.sh* y *create_seq.sh* redactados en este capítulo.

Código 4.2 sql.conf

```
1 PHOST="PostgreSQl Host"
2 PDB="PostgreSQL DB"
3 PPORT="PostgreSQL Puerto"
```

En la linea 11 del *script create_obj.sh* se encuentra el comando que genera las sentencias de creación de las secuencias del esquema pasado como parámetro utilizando el *script create_sequences.sql*. Este *script* devuelve, en un *string*, todas las secuencias de la tabla *sequences* del esquema *information_schema* que correspondan con el esquema dado.

Código 4.3 create_sequences.sql

```
1 SELECT 'CREATE SEQUENCE '||sequence_schema||'.'||sequence_name
2 ||' INCREMENT BY '||COALESCE(increment,1)||' MINVALUE '||COALESCE(
      minimum_value,1)||' NOMAXVALUE START WITH 1 NOCACHE;' AS
      secuencias
3 FROM information_schema.sequences
4 WHERE sequence_schema=:sch;
```

En la linea 12 del *script create_obj.sh* se encuentra el comando que genera las sentencias de creación de las restricciones del esquema pasado como parámetro utilizando el *script create_constraints.sql*. Este *script* devuelve, en un *string*, todas las restricciones del *join* resultante de las tablas *constraints, key_colums_usage, referential_constraints y constraint_column_usage* del esquema *information_schema* que correspondan con el esquema dado.

Las restricciones contempladas son de clave primaria, clave foránea y clave única. Oracle por defecto crea índices tanto para las claves primarias como para las claves únicas.

Código 4.4 create_constraints.sql

```
1  SELECT 'ALTER TABLE '||tc.table_name||' ADD '||
2  CASE tc.constraint_type
3    WHEN 'FOREIGN KEY' THEN tc.constraint_type ||' ('||kcu.column_name||
       ')'||' REFERENCES '||ccu.table_name||' ('||ccu.column_name||');'
4    WHEN 'PRIMARY KEY' THEN tc.constraint_type||' ('||kcu.column_name||'
       );'
5    WHEN 'UNIQUE' THEN tc.constraint_type||' ('||kcu.column_name||');'
6  END
7  FROM information_schema.table_constraints tc
8    LEFT JOIN information_schema.key_column_usage kcu
9    ON tc.constraint_catalog = kcu.constraint_catalog
10   AND tc.constraint_schema = kcu.constraint_schema
11   AND tc.constraint_name = kcu.constraint_name
12   LEFT JOIN information_schema.referential_constraints rc
13   ON tc.constraint_catalog = rc.constraint_catalog
14   AND tc.constraint_schema = rc.constraint_schema
15   AND tc.constraint_name = rc.constraint_name
16   LEFT JOIN information_schema.constraint_column_usage ccu
17   ON rc.unique_constraint_catalog = ccu.constraint_catalog
18   AND rc.unique_constraint_schema = ccu.constraint_schema
19   AND rc.unique_constraint_name = ccu.constraint_name
20 WHERE tc.table_schema = :sch and tc.constraint_type <> 'CHECK';
```

En la linea 13 del *script create_obj.sh* se ejecuta la consulta del *script tablas_pg.sql* que genera un archivo *tablas_pg.txt*.

Código 4.5 tablas_pg.sql

```
1 SELECT :tab
2 FROM information_schema.tables
3 WHERE lower(table_schema)=:sch AND lower(table_type)='base table'
```

En la linea 14 se itera sobre el archivo generado anteriormente y se ejecuta en cada iteración el *script create_tables.sql* que devuelve, en un *string*, la sentencia correspondiente para la creación de la misma para el esquema dado.

Código 4.6 create_tables.sql

```
SELECT 'CREATE TABLE '||:sch||'.'||:tab||' (';
  SELECT
  CASE WHEN ordinal_position = 1 THEN ' ' ELSE ',' END|| column_name||
    ' '||
  CASE WHEN lower(udt_name)='bool' THEN 'NUMBER(1,0)'
    WHEN lower(udt_name)='varchar' THEN 'VARCHAR2('||
    character_maximum_length||' char)'
    WHEN lower(udt_name)='bpchar' THEN 'CHAR('||
    character_maximum_length||' char)'
    WHEN lower(udt_name)='text' THEN 'CLOB'
    WHEN lower(udt_name)='int2' THEN 'NUMBER'
    WHEN lower(udt_name)='int4' THEN 'NUMBER'
    WHEN lower(udt_name)='int8' THEN 'NUMBER'
    WHEN lower(udt_name)='numeric' THEN 'NUMBER'
    WHEN lower(udt_name)='timestamp' THEN upper(udt_name)
    WHEN lower(udt_name)='bytea' THEN 'BLOB'
    WHEN lower(udt_name)='float8' THEN 'FLOAT(126)'
    WHEN lower(udt_name)='float4' THEN 'FLOAT(63)'
      ELSE upper(udt_name) END||
    CASE WHEN is_nullable = 'NO' THEN ' NOT NULL' ELSE '' END
FROM information_schema.columns
WHERE table_schema=:sch and table_name=:tab;
SELECT ');';
```

4.2. Migración de datos

A continuación se describe la secuencia de pasos que se siguieron para realizar la migración desde un entorno de base de datos a otro:

1. Chequear que las instancias de Oracle estén funcionando correctamente y chequear conectividad.
2. Coordinar las pruebas de las aplicaciones con los programadores cuando se realice la migración.
3. Bloquear la escritura sobre los esquemas a migrar en el ambiente de producción.
4. Realizar el volcado de los datos del servidor de producción.
5. Generar los archivos de control para realizar la carga con el *SQL*Loader*.
6. Deshabilitar las restricciones de las tablas de Oracle en los esquemas a migrar.
7. Realizar la carga de datos con *SQL*Loader*.
8. Chequear los *logs* del *SQL*Loader* para corregir los datos que no se han podido cargar. Realizar las modificaciones necesarias para la carga completa de datos.
9. Activar las restricciones de las tablas de Oracle en los esquemas a migrar.
10. Una vez realizada la carga de datos se asignarán los valores correspondiente a las secuencias.
11. Probar las aplicaciones.

Para las aplicaciones que utilicen tablas externas a su esquema, se dejará una tarea periódica de replicación de datos.

Se replicará la carga de datos en los ambientes restantes mediante las aplicaciones de Oracle *exp* e *imp*.

Los pasos 4, 5, 6, y 9 se realizan automáticamente mediante el *script load_data.sh*.

Antes de ejecutar la carga de datos mediante el *script* anteriormente mencionado se debe ejecutar el *script generate_cf.sh* que genera de manera dinámica archivos de control para ser utilizados posteriormente por el *sql*ldr*. Como se podrá observar en el *script* se realiza un *join* de dos archivos de tablas creados de ambas bases de datos. Esto es así porque las

tablas podían diferir en su nombre, ya que Oracle solo acepta longitudes de nombre para los objetos de 30 caracteres como máximo.

Código 4.7 generate_cf.sh

```
1  #!/bin/sh
2  export PGCLIENTENCODING='UTF8'
3  export PGPASSWORD='******'
4  export NLS_LANG='AMERICAN_AMERICA.UTF8'
5  export NLS_TIMESTAMP_FORMAT='RRRR-MM-DD HH24:MI:SS.FF3'
6  export NLS_NUMERIC_CHARACTERS='.,'
7  export NLS_DATE_FORMAT=RRRR-MM-DD
8  LOG=/tmp/$1.log
9  CONF=sqlldr.conf
10 source $CONF
11
12 if [ -d $1 ]; then
13   cd $1
14 else
15   mkdir $1
16   cd $1
17 fi
18 echo 'Creando archivo de tablas para Oracle'
19 sqlplus -s $1/$2@$ODB @../tablas.sql $1
20 echo 'Creando archivo de tablas para Postres'
21 psql -h $PHOST -p $PPORT -U postgres -d $PDB -A -t -F ' ' -v tab=
       table_name -v sch="'$3'" -f ../tablas_pg.sql -o tablas_pg.txt
22 echo 'Creando los control files'
23 join tablas_pg.txt tablas.txt| sed -e 's/ /:/g' >tablas_join.txt
24 inicio=`date +%s`
25 for i in `cat tablas_join.txt` ; do
26   pg=`echo $i|cut -f2 -d ':'|cut -f2 -d'"'`
27   pg_dc=`echo $i|cut -f2 -d ':'`
28   ora=`echo $i|cut -f3 -d ':'`
29   sqlplus -s $1/$2@$ODB @../control_files.sql $ora
30   echo '---Tabla: '$pg' archivo de control generado'
31 done;
32 fin=`date +%s`
33 let total=$fin-$inicio
34 echo "=========================="
35 echo "TIEMPO: $total SEGUNDOS"
36 echo "=========================="
```

En la linea 9 del *script generate_cf.sh* tenemos el archivo de configuración *sqlldr.conf* para poder configurar la base de datos de origen y la base de datos de destino.

Este archivo de configuración se utilizará en los *scripts load_data.sh*, *generate_seq.sh* y *create_seq.sh* que se describen en este mismo capítulo.

Código 4.8 sqlldr.conf

```
1 ODB="Oracle DB"
2 PHOST="PostgreSQl Host"
3 PDB="PostgreSQL DB"
4 PPORT="PostgreSQL Puerto"
```

En la linea 19 del *script generate_cf.sh* se genera el archivo *tablas.sql*, aquí se vuelcan las tablas de la base de datos Oracle para el esquema pasado como parámetro al *script*.

Código 4.9 tablas.sql

```
1 set verify off echo off ver off feed off pages 0 linesize 200
2 spool tablas.txt
3 SELECT row_number() over (order by table_name),lower(table_name)
4 FROM user_tables;
5 spool off
6 exit
```

En la linea 21 del *script generate_cf.sh* se genera el archivo *tablas_pg.sql* en donde se vuelcan las tablas de la base de datos de PostgreSQL para el esquema pasado como parámetro al *script*.

Código 4.10 tablas_pg.sql

```
1 SELECT row_number()OVER (ORDER BY table_name),'"'||:tab||'"'
2 FROM information_schema.tables
3 WHERE lower(table_schema)=:sch AND lower(table_type)='base table'
```

Realizado el *join* de ambas tablas en la linea 23 del *script generate_cf.sh*, en la linea 25 se itera sobre este archivo y se generan -en la linea 29-, los archivos de control que utilizará *sqlldr* mediante el *script control_files.sql*.

Código 4.11 control_files.sql

```
1  set verify off echo off ver off feed off pages 0 linesize 120
2  spool &1..ctl
3
4  SELECT 'LOAD DATA'||chr(10)||
5    'INFILE '''||lower(table_name)||'.dat'' "str ''|\n''"'||chr(10)||
6    'REPLACE'||chr(10)||
7    'INTO TABLE '||table_name||chr(10)||
8    'FIELDS TERMINATED BY ''¬'''||chr(10)||
9    'TRAILING NULLCOLS'||chr(10)||'('
10 FROM user_tables
11 WHERE table_name = upper ('&1');
```

```
12
13 SELECT decode (column_id,1,' ',',')||
14   rpad (column_name,35,' ')||
15   decode (data_type,
16     'VARCHAR2','CHAR(512000) "TRIM(:'||column_name||')"',
17     'FLOAT','DECIMAL EXTERNAL NULLIF('||column_name||'=BLANKS)',
18     'NUMBER',
19   decode (data_precision,0,'INTEGER EXTERNAL NULLIF ('||column_name||'
       =BLANKS)',
20   decode (data_scale, 0,'INTEGER EXTERNAL NULLIF ('||column_name||'=
       BLANKS)',
21     'DECIMAL EXTERNAL NULLIF ('||column_name||'=BLANKS)')),
22     'CHAR','CHAR "TRIM(:'||column_name||')"',
23     'DATE','DATE NULLIF ('||column_name||'=BLANKS)',
24     'CLOB','CHAR(512000) NULLIF ('||column_name||'=BLANKS)',
25     'BLOB','FILLER CHAR(1)',
26     'TIMESTAMP(6)','TIMESTAMP "RRRR-MM-DD HH24:MI:SS.FF6" NULLIF ('||
       column_name||'=BLANKS)',null)
27 FROM user_tab_columns
28 WHERE table_name = upper ('&1')
29 ORDER BY column_id;
30 SELECT ')'
31 FROM dual;
32 spool off
33 exit
```

Finalizada la ejecución del *script generate_cf.sh* se debe ejecutar el *script load_data.sh* que es el encargado de realizar la descarga de información de las tablas del esquema dado y luego realizar la carga mediante esos archivos a la base de datos Oracle.

Para para realizar la descarga desde PostgreSQL se utiliza el comando *psql*, parametrizado de tal manera que los archivos se formateen de una manera en particular.

La carga de datos en Oracle se efectúa con el comando *sqlldr*, activando la opción correspondiente a la carga directa. Esto quiere decir que los registros se graban directamente en los archivos de datos sin pasar por el registro de transacción, optimizando la *performance*.

Código 4.12 load_data.sh

```
1 #!/bin/bash
2 ### Modo de uso:
3 ### ./load_data.sh user_oracle pass_oracle esquema_postgres
4
5 export PGCLIENTENCODING='UTF8'
6 export PGPASSWORD='******'
7 export NLS_LANG='AMERICAN_AMERICA.UTF8'
8 export NLS_TIMESTAMP_FORMAT='YYYY-MM-DD HH24:MI:SS.FF3'
9 export NLS_TIMESTAMP_TZ_FORMAT='YYYY-MM-DD HH24:MI:SS.FF6 TZR';
10 export NLS_NUMERIC_CHARACTERS='.,'
11 export NLS_DATE_FORMAT=YYYY-MM-DD
12 LOG=/tmp/$1.log
13 CONF=sqlldr.conf
14 source $CONF
```

```
15
16 if [ -d $1 ]; then
17         cd $1
18 else
19         echo "=============================================================="
20         echo "== EJECUTE PRIMERO EL SCRIPT generate_cf.sh =="
21         echo "=============================================================="
22 fi
23 inicio=`date +%s`
24 sqlplus -s $1/$2@$ODB @../g_disable_constraints.sql >/dev/null
25 sqlplus -s $1/$2@$ODB @disable_constraints.sql >disable_constraints.
       log
26 for i in `cat tablas_join.txt` ; do
27   pg=`echo $i|cut -f2 -d ':'|cut -f2 -d'"'`
28   pg_dc=`echo $i|cut -f2 -d ':'`
29   ora=`echo $i|cut -f3 -d ':'`
30   col=`psql -h $PHOST -p $PPORT -U postgres -d $PDB -A -t -R ',' -v
       sch="'$3'" -v pg="'$pg'" -f ../columns_pg.sql`
31   psql -h $PHOST -p $PPORT -U postgres -d $PDB -A -v col="$col" -v sch
       ="$3" -v pg="$pg_dc" -F '¬' -R $'|\n' -f ../copy.sql -o dump.tmp
32   sed '$d' dump.tmp > $ora.dat
33   echo '---Tabla: '$pg' volcada'
34   sqlldr userid=$1/$2@$ODB control=$ora.ctl log=$ora.log DIRECT=TRUE
       readsize=10000000 bindsize=50000 errors=999 skip=1>> $LOG
35   echo '------Tabla: '$pg' cargada'
36 done;
37 rm dump.tmp
38 sqlplus -s $1/$2@$ODB @../g_enable_constraints.sql >/dev/null
39 sqlplus -s $1/$2@$ODB @enable_constraints.sql >enable_constraints.log
40 fin=`date +%s`
41 let total=$fin-$inicio
42 echo "=========================="
43 echo "TIEMPO: $total SEGUNDOS"
44 echo "=========================="
```

En la linea 24 del *script load_data.sh* se ejecuta la generación de sentencias para desactivar las restricciones mediante el *script g_disable_constraints.sql*.

Código 4.13 g_disable_constraints.sql

```
1  set verify off echo off ver off feed off pages 0 linesize 200
2  spool disable_constraints.sql
3  SELECT 'ALTER TABLE ' ||substr(c.table_name,1,35)|| ' DISABLE
       CONSTRAINT ' ||constraint_name|| ' CASCADE;'
4  FROM  user_constraints c INNER JOIN user_tables u ON c.table_name=u.
       table_name;
5
6  SELECT 'ALTER TRIGGER '||table_owner||'.'||trigger_name||' DISABLE;'
7  FROM user_triggers;
8
9  SELECT 'exit'
10 FROM dual;
11 spool off
12 exit
```

Como se ha mencionado, ciertas columnas necesitan de un mapeo espe-

cial, ya que no tienen una equivalencia directa entre ambas base de datos. Por ello, en la línea 30 del *script load_data.sh* se realiza un mapeo de las columnas utilizando el *script columns_pg.sql*.

Notemos qué, para los tipos de datos *boolean* y *bytea* el mapeo es explícito.

Código 4.14 columns_pg.sql

```
1 SELECT CASE WHEN lower(data_type)='boolean' THEN 'CAST('||column_name
      ||' AS INT)'
2   WHEN lower(data_type)='bytea' THEN 'null is not null '||column_name
3   ELSE '"'||column_name||'"' END
4 FROM information_schema.columns
5 WHERE table_schema=:sch AND table_name=:pg
```

En la linea 31 se realiza la descarga de datos utiliznado el siguiente *script copy.sql*, que devuelve todos los registros del esquema y tabla pasadas como parámetro.

Código 4.15 copy.sql

```
1 SELECT :col
2 FROM :sch.:pg
```

En la linea 38 del *script load_data.sh* se ejecuta la generación de sentencias para activar las restricciones mediante el *script g_enable_constraints.sql*.

Código 4.16 g_enable_constraints.sql

```
1 set verify off echo off ver off feed off pages 0 linesize 200
2 spool enable_constraints.sql
3 SELECT 'ALTER TABLE ' ||substr(c.table_name,1,35)|| ' ENABLE
      CONSTRAINT ' ||constraint_name|| ';'
4 FROM  user_constraints c INNER JOIN user_tables u ON c.table_name=u.
      table_name
5 ORDER BY constraint_type;
6
7 SELECT 'ALTER TRIGGER '||table_owner||'.'||trigger_name||' ENABLE;'
8 FROM user_triggers;
9
10 SELECT 'exit'
11 FROM dual;
12 spool off
13 exit
```

Cargados los registros en las tablas en la base de datos de destino, se

debían *resetear* las secuencias al valor correspondiente. Por lo tanto, se utilizó el siguiente *script* para dicho cometido:

Código 4.17 generate_seq.sh

```
1  #!/bin/sh
2  #LOG=/tmp/$1.log
3  export PGPASSWORD='******'
4  CONF=sqlldr.conf
5  source $CONF
6
7  if [ -d $1 ]; then
8    cd $1
9  else
10   mkdir $1
11   cd $1
12 fi
13 inicio=`date +%s`
14 sqlplus -s $1/$2@$ODB @../seq.sql $1
15 psql -h $PHOST -p $PPORT -U postgres -d $PDB -A -t -F ' ' -v seq=
       sequence_name -v sch="'$3'" -f ../seq_pg.sql -o seq_pg.txt
16 join seq_pg.txt seq.txt| sed -e 's/ /:/g' >seq_join.txt
17 for i in `cat seq_join.txt` ; do
18   echo $i
19 done;
20 fin=`date +%s`
21 let total=$fin-$inicio
22 echo "=========================="
23 echo "TIEMPO: $total SEGUNDOS"
24 echo "=========================="
```

En la linea 14 del *script generate_seq.sh* se obtienen las secuencias creadas con anterioridad en la base de datos Oracle mediante la ejecución del *script seq.sql*.

Código 4.18 seq.sql

```
1 set verify off echo off ver off feed off pages 0 linesize 200
2 spool seq.txt
3 SELECT row_number() over (order by sequence_name),lower(sequence_name)
4 FROM user_sequences;
5 spool off
6 exit
```

Análogamente, en la linea 15, se obtienen las secuencias de la base de datos origen ejecutando el *script seq_pg.sql*.

Código 4.19 seq_pg.sql

```
1 SELECT row_number()OVER (ORDER BY sequence_name),'" '||:seq||'" '
```

```
2 FROM information_schema.sequences
3 WHERE lower(sequence_schema)=:sch
```

Generadas las secuencias (se ejecutó el *script generate_seq.sh*), se realiza un *join* de los archivos obtenidos en *seq.sql* y *seq_pg.sql*, ya que los nombres de las secuencias -como se ha comentado más arriba-, pueden variar de una base de datos a otra.

Se procede a recrear las secuencias con los valores de inicialización mediante el *script create_seq.sh*.

Código 4.20 create_seq.sh

```
1 #!/bin/sh
2 export PGPASSWORD='******'
3 CONF=sqlldr.conf
4 source $CONF
5
6 if [ -d $1 ]; then
7   cd $1
8 else
9   echo "=============================================================="
10  echo "== EJECUTE PRIMERO EL SCRIPT generate_seq.sh =="
11  echo "=============================================================="
12 fi
13 inicio=`date +%s`
14 echo --Init seq > ini_seq.sql
15 for i in `cat seq_join.txt` ; do
16   pg=`echo $i|cut -f2 -d ':'|cut -f2 -d'"'"'`
17   pg_dc=`echo $i|cut -f2 -d ':'`
18   ora=`echo $i|cut -f3 -d ':'`
19   currval=`psql -h $PHOST -p $PPORT -U postgres -d $PDB -A -t -v sch="
       '$3'" -v pg="'$pg_dc'" -f ../seq_curr_pg.sql`
20   echo "DROP SEQUENCE $ora;" >> ini_seq.sql
21   echo "CREATE SEQUENCE $ora START WITH $currval INCREMENT BY 1
       NOMAXVALUE NOMINVALUE NOCACHE;" >> ini_seq.sql
22   echo "GRANT SELECT ON $ora TO $1_app;" >> ini_seq.sql
23 done;
24 echo "EXIT" >> ini_seq.sql
25 sqlplus -s $1/$2@$ODB @ini_seq.sql
26 fin=`date +%s`
27 let total=$fin-$inicio
28 echo "============================"
29 echo "TIEMPO: $total SEGUNDOS"
30 echo "============================"
```

En la linea 15 se comienza a iterar sobre todas las secuencias obtenidas en el archivo textitseq_join.txt.

En la linea 19 del *script create_seq.sh*, se obtiene el valor actual de la secuencia pasada por parámetro en la base de datos origen, ejecutando el *script seq_curr_pg.sql*

Código 4.21 seq_curr_pg.sql

```
1 SELECT NEXTVAL(:sch||'.'||:pg)
```

Finalmente, en la linea 25 del *script* *create_seq.sh* se ejecuta el *script* *ini_seq.sql* que contiene las sentencias de recreación de las secuencias.

4.3. Herramientas utilizadas en los *scripts*

A continuación se describen las herramientas utilizadas en los *scripts* desarrollados para la carga y descarga de datos, desde una base de datos a otra.

En una primera instancia se utilizó la herramienta *COPY* para realizar la descarga de datos en PostgreSQL obteniendo que el servidor escribiera directamente en archivos -sin pasar por un cliente-, los datos resultantes de las tablas y luego transferir dichos archivos al servidor destino. Recapitulando, se decidió utilizar la herramienta *psql* generando una conexión remota a la base de datos origen, para luego escribir los archivos en el servidor que tenía instalada la base de datos Oracle, obteniendo tiempos similares y centralizando la ejecución en un solo lugar.

Para la carga de datos se utilizó la herramienta *sqlldr* de Oracle, la cual se describe a continuación.

*SQL*Loader* carga datos de archivos externos en tablas de una base de datos Oracle. Tiene un potente motor de análisis de datos que limita poco el formato de los datos en el archivo de datos. Puede usar *SQL*Loader* para hacer lo siguiente:

- Cargar datos a través de una red. Esto significa que puede ejecutar el cliente *SQL*Loader* en un sistema diferente del que ejecuta el servidor *SQL*Loader*.
- Se pueden cargar datos de varios archivos de datos durante la misma sesión de carga.
- Permite cargar datos en varias tablas durante la misma sesión de carga.
- Permite especificar el conjunto de caracteres de los datos.
- Permite cargar datos de forma selectiva (puede cargar registros en función de los valores de los registros).
- Se pueden manipular los datos antes de cargarlos, utilizando funciones SQL.
- Permite generar valores de clave secuenciales únicos en columnas especificadas.

- Se puede utilizar el sistema de archivos del sistema operativo para acceder a los archivos de datos.
- Se pueden cargar datos desde disco, cinta o canalización con nombre.
- Genera informes de errores sofisticados, que ayudan en gran medida a la resolución de problemas.
- Carga datos relacionales de objetos arbitrariamente complejos.
- Utiliza archivos de datos secundarios para cargar *LOB* y colecciones.
- Permite la carga directa o convencional. Si bien la carga convencional es muy flexible, la carga directa proporciona un rendimiento de carga superior.

4.4. Ejemplo de ejecución de los *scripts*

En esta sección se mostrará un ejemplo de ejecución de todos los *scripts* para la aplicación *stttp* en el mismo orden descripto en las secciones anteriores.

Ejecutamos el siguiente *script* parametrizando el esquema de ejemplo:

Código 4.22 Ejecución create_obj.sh

```
$./create_obj.sh stttp
```

Archivo generado en la linea 11 del *script create_obj.sh*.

Código 4.23 Salida del archivo cs.sql

```
$ head cs.sql
CREATE SEQUENCE stttp.seq_ejercicio INCREMENT BY 1 MINVALUE 1
NOMAXVALUE START WITH 1 NOCACHE;
CREATE SEQUENCE stttp.seq_rango_ejercicio INCREMENT BY 1 MINVALUE 1
NOMAXVALUE START WITH 1 NOCACHE;
CREATE SEQUENCE stttp.seq_arancel INCREMENT BY 1 MINVALUE 1 NOMAXVALUE
 START WITH 1 NOCACHE;
CREATE SEQUENCE stttp.seq_area INCREMENT BY 1 MINVALUE 1 NOMAXVALUE
START WITH 1 NOCACHE;
CREATE SEQUENCE stttp.seq_boleta_de_deposito INCREMENT BY 1 MINVALUE 1
 NOMAXVALUE START WITH 1 NOCACHE;
CREATE SEQUENCE stttp.seq_cambio INCREMENT BY 1 MINVALUE 1 NOMAXVALUE
START WITH 1 NOCACHE;
CREATE SEQUENCE stttp.seq_cargo INCREMENT BY 1 MINVALUE 1 NOMAXVALUE
START WITH 1 NOCACHE;
CREATE SEQUENCE stttp.seq_deposito INCREMENT BY 1 MINVALUE 1
NOMAXVALUE START WITH 1 NOCACHE;
```

```
CREATE SEQUENCE stttp.seq_deposito_item_cobrable INCREMENT BY 1
MINVALUE 1 NOMAXVALUE START WITH 1 NOCACHE;
CREATE SEQUENCE stttp.seq_dominio INCREMENT BY 1 MINVALUE 1 NOMAXVALUE
 START WITH 1 NOCACHE;
```

Archivo generado en la linea 12 del *script create_obj.sh*.

Código 4.24 Salida del archivo cc.sql

```
$ head cc.sql
ALTER TABLE stttp.ejercicio_contable ADD PRIMARY KEY (id);
ALTER TABLE stttp.rango_ejercicio ADD PRIMARY KEY (id);
ALTER TABLE stttp.tipo_rendicion ADD PRIMARY KEY (id);
ALTER TABLE stttp.cuve ADD PRIMARY KEY (id);
ALTER TABLE stttp.secuencias ADD PRIMARY KEY (id);
ALTER TABLE stttp.variables_globales ADD PRIMARY KEY (id);
ALTER TABLE stttp.movimiento ADD PRIMARY KEY (id);
ALTER TABLE stttp.nro_de_rendicion ADD PRIMARY KEY (id);
ALTER TABLE stttp.cargo_cambio ADD UNIQUE (id_cambio);
ALTER TABLE stttp.cargo_cambio ADD UNIQUE (id_cargo);
```

Archivo generado en la linea 13 del *script create_obj.sh*.

Código 4.25 Salida del archivo ct.sql

```
$ head -19 ct.sql
CREATE TABLE stttp.ejercicio_contable (
 id NUMBER NOT NULL
,fecha_fin TIMESTAMP
,fecha_inicio TIMESTAMP
,version NUMBER
,id_cambio NUMBER
);
CREATE TABLE stttp.rango_ejercicio (
 id NUMBER NOT NULL
,fecha_desde TIMESTAMP
,fecha_hasta TIMESTAMP
,id_ejercicio NUMBER
,version NUMBER
);
CREATE TABLE stttp.tipo_rendicion (
 id NUMBER NOT NULL
,nombre VARCHAR2(50 char)
,version NUMBER
);
```

Generados los archivos de creación de objetos, se ejecuta manualmente, conectándonos con el usuario que responde a la aplicación dada.

Código 4.26 Ejecución en Oracle de los archivos generados con el *script create_obj.sh*

```
$ sqlplus /nolog

SQL> connect stttp@ORACLEDB
Connected

SQL> @cs.sql

Sequence created.
Sequence created.
Sequence created.

...

SQL> @ct.sql
Table created.
Table created.
Table created.

...

SQL> @cc.sql
Table altered.
Table altered.
Table altered.

...
```

Creadas las tablas con sus restricciones y las secuencias, se prepara el ambiente para la migración de datos.

Comenzamos creando dinámicamente los archivos de control a utilizar para la aplicación *stttp*.

Código 4.27 Ejecución del *script generate_cf.sh* para el esquema stttp.

```
$ ./generate_cf.sh stttp stttp stttp
Creando archivo de tablas para Oracle
Creando archivo de tablas para Postres
Creando los control files
---Tabla: arancel archivo de control generado
---Tabla: area archivo de control generado
---Tabla: cambio archivo de control generado
---Tabla: cargo archivo de control generado
---Tabla: cargo_cambio archivo de control generado
---Tabla: cuve archivo de control generado
---Tabla: deposito archivo de control generado
---Tabla: deposito_item_cobrable archivo de control generado
---Tabla: dominio archivo de control generado
---Tabla: egreso archivo de control generado
---Tabla: ejercicio_contable archivo de control generado
---Tabla: especie archivo de control generado
---Tabla: estado archivo de control generado
---Tabla: externo_firma_representante archivo de control generado
---Tabla: firma_representante archivo de control generado
---Tabla: global_oficina_regional archivo de control generado
```

```
---Tabla: informacion_de_rendiciones archivo de control generado
---Tabla: interno_puesto_autorizado archivo de control generado
---Tabla: item_cobrable archivo de control generado
---Tabla: movimiento archivo de control generado
---Tabla: nro_comprobante archivo de control generado
---Tabla: nro_comprobante_generado archivo de control generado
---Tabla: nro_de_cargo archivo de control generado
---Tabla: nro_de_rendicion archivo de control generado
---Tabla: numeracion_comprobantes archivo de control generado
---Tabla: oficina_frontera archivo de control generado
---Tabla: oficina_regional archivo de control generado
---Tabla: oficina_tipo_transito archivo de control generado
---Tabla: permiso archivo de control generado
---Tabla: permiso_cambio archivo de control generado
---Tabla: pfi archivo de control generado
---Tabla: producto archivo de control generado
---Tabla: productotransportado archivo de control generado
---Tabla: puesto_inspector_area archivo de control generado
---Tabla: rango_ejercicio archivo de control generado
---Tabla: rendicion archivo de control generado
---Tabla: rendicion_cambio archivo de control generado
---Tabla: restriccion_dia archivo de control generado
---Tabla: secuencias archivo de control generado
---Tabla: solicitud archivo de control generado
---Tabla: solicitud_cambio archivo de control generado
---Tabla: tipo_arancel archivo de control generado
---Tabla: tipo_de_conservante archivo de control generado
---Tabla: tipo_rendicion archivo de control generado
---Tabla: tipo_transito archivo de control generado
---Tabla: tipo_usuario archivo de control generado
---Tabla: tipo_usuario_area archivo de control generado
---Tabla: usuario archivo de control generado
---Tabla: valor_arancel archivo de control generado
---Tabla: variables_globales archivo de control generado
...
========================
TIEMPO: 360 SEGUNDOS
========================
```

Como vimos anteriormente en en la linea 23 del *script generate_cf.sh* se genera un archivo con el *join* de las tablas de PostgreSQL (a la izquiera) y Oracle (a la derecha) el cual se itera para luego generar los *control_files*.

Este es un ejemplo del archivo generado para la aplicación *stttp*:

Código 4.28 Fragmento del archivo *tablas_join.txt* para el esquema stttp

```
$ head tablas_join.txt
1:"arancel":arancel:
2:"area":area:
3:"cambio":cambio:
4:"cargo":cargo:
5:"cargo_cambio":cargo_cambio:
6:"cuve":cuve:
7:"deposito":deposito:
8:"deposito_item_cobrable":deposito_item_cobrable:
...
```

Ejecutado el *script* anterior, estos son los archivos de control generados para *stttp*.

Código 4.29 Lista de los archivos de control generados para el esquema stttp

```
$ ls -Gg *.ctl
-rw-r--r-- 1 2663 ene 31 11:31 arancel.ctl
-rw-r--r-- 1 1453 ene 31 11:31 area.ctl
-rw-r--r-- 1 1816 ene 31 11:31 cambio.ctl
-rw-r--r-- 1 1332 ene 31 11:31 cargo_cambio.ctl
-rw-r--r-- 1 1816 ene 31 11:31 cargo.ctl
-rw-r--r-- 1 1332 ene 31 11:31 cuve.ctl
-rw-r--r-- 1 3873 ene 31 11:31 deposito.ctl
-rw-r--r-- 1 1574 ene 31 11:31 deposito_item_cobrable.ctl
-rw-r--r-- 1 1816 ene 31 11:31 dominio.ctl
-rw-r--r-- 1 1937 ene 31 11:31 egreso.ctl
-rw-r--r-- 1 1695 ene 31 11:31 ejercicio_contable.ctl
-rw-r--r-- 1 1574 ene 31 11:31 especie.ctl
-rw-r--r-- 1 1574 ene 31 11:31 estado.ctl
...
```

A continuación se lista la salida de ejemplo del archivo de control para la tabla área del esquema *stttp*, que luego será utilizado por el comando *sqlldr* para realizar la carga de datos en Oracle.

Código 4.30 area.ctl

```
LOAD DATA
INFILE 'area.dat' "str '|\n'"
REPLACE
INTO TABLE AREA
FIELDS TERMINATED BY '¬'
TRAILING NULLCOLS
(
ID                              DECIMAL EXTERNAL NULLIF (ID=BLANKS)
,NOMBRE                 CHAR(512000) "TRIM(:NOMBRE)"
,VERSION                DECIMAL EXTERNAL NULLIF (VERSION=BLANKS)
)
```

Con los objetos creados y lo archivos de control generados se prosigue a ejecutar el *script* de carga de datos.

Código 4.31 Ejecución del *script load_data.sh* para el esquema stttp

```
$ ./load_data.sh stttp stttp stttp
--Tabla: arancel volcada
----Tabla: arancel cargada
--Tabla: area volcada
----Tabla: area cargada
--Tabla: cambio volcada
----Tabla: cambio cargada
--Tabla: cargo volcada
----Tabla: cargo cargada
--Tabla: cargo_cambio volcada
----Tabla: cargo_cambio cargada
--Tabla: cuve volcada
----Tabla: cuve cargada
--Tabla: deposito volcada
----Tabla: deposito cargada
--Tabla: deposito_item_cobrable volcada
----Tabla: deposito_item_cobrable cargada
--Tabla: dominio volcada
----Tabla: dominio cargada
--Tabla: egreso volcada
----Tabla: egreso cargada
--Tabla: ejercicio_contable volcada
----Tabla: ejercicio_contable cargada
--Tabla: especie volcada
----Tabla: especie cargada
--Tabla: estado volcada
----Tabla: estado cargada
--Tabla: externo_firma_representante volcada
----Tabla: externo_firma_representante cargada
--Tabla: firma_representante volcada
----Tabla: firma_representante cargada
--Tabla: global_oficina_regional volcada
----Tabla: global_oficina_regional cargada
--Tabla: informacion_de_rendiciones volcada
----Tabla: informacion_de_rendiciones cargada
--Tabla: interno_puesto_autorizado volcada
----Tabla: interno_puesto_autorizado cargada
--Tabla: item_cobrable volcada
----Tabla: item_cobrable cargada
--Tabla: movimiento volcada
----Tabla: movimiento cargada
--Tabla: nro_comprobante volcada
----Tabla: nro_comprobante cargada
--Tabla: nro_comprobante_generado volcada
----Tabla: nro_comprobante_generado cargada
...
========================
TIEMPO: 600 SEGUNDOS
========================
```

Vemos una muestra de un fragmento del archivo generado con las sentencias para desactivar las restricciones del esquema *stttp* que se ejecuta en la linea 25 del *script load_data.sh*.

Esto se realiza antes de volcar y cargar las tablas desde PostgreSQL a Oracle.

Código 4.32 disable_constraints.sql

```
$head disable_constraints.sql
ALTER TABLE ARANCEL DISABLE CONSTRAINT SYS_C0061075 CASCADE;
ALTER TABLE ARANCEL DISABLE CONSTRAINT SYS_C0061133 CASCADE;
ALTER TABLE ARANCEL DISABLE CONSTRAINT SYS_C0061140 CASCADE;
ALTER TABLE AREA DISABLE CONSTRAINT SYS_C0061052 CASCADE;
ALTER TABLE AREA DISABLE CONSTRAINT SYS_C0061111 CASCADE;
ALTER TABLE CAMBIO DISABLE CONSTRAINT SYS_C0061062 CASCADE;
ALTER TABLE CAMBIO DISABLE CONSTRAINT SYS_C0061120 CASCADE;
ALTER TABLE CAMBIO DISABLE CONSTRAINT SYS_C0061142 CASCADE;
ALTER TABLE CAMBIO DISABLE CONSTRAINT SYS_C0061196 CASCADE;
ALTER TABLE CARGO DISABLE CONSTRAINT SYS_C0061025 CASCADE;
...
```

La salida del *script* anterior se guarda en el archivo *disable_constraints.log*.

Código 4.33 disable_constraints.log

```
$head disable_constraints.log
Table altered.
Table altered.
Table altered.
Table altered.
Table altered.
Table altered.
Table altered.
Table altered.
Table altered.
Table altered.
...
```

Vemos un ejemplo de un archivo de dato "volcado", realizado en las lineas 31 y 32 del *script load_data.sh*, desde PostgreSQL para la tabla área.

Código 4.34 area.dat

```
id¬nombre¬version|
1¬Animal¬0|
2¬Vegetal¬0|
3¬Bajo Acuerdo¬0|
```

Cuando el archivo *area.dat* es cargado, en la linea 34 del *script load_data.sh*, se genera un archivo de *log* como se muestra a continuación.

En caso de que algún registro no se pueda cargar, este será registrado en un archivo con extensión *.bad*.

Código 4.35 area.log

```
SQL*Loader: Release 11.2.0.3.0 - Production on Wed Mar 7 11:14:59 2012
Copyright (c) 1982, 2011, Oracle and/or its affiliates.  All rights
reserved.

Control File:   area.ctl
Data File:      area.dat
  File processing option string: "str '|'"
  Bad File:     area.bad
  Discard File:  none specified

 (Allow all discards)

Number to load: ALL
Number to skip: 1
Errors allowed: 999
Continuation:    none specified
Path used:      Direct

Table AREA, loaded from every logical record.
Insert option in effect for this table: REPLACE
TRAILING NULLCOLS option in effect

Column Name Position Len   Term Encl Datatype
----------- -------- ----- ---- ---- ---------------------
ID          FIRST    *     ¬         CHARACTER  NULL if ID = BLANKS
NOMBRE      NEXT     ***** ¬         CHARACTER  Maximum field length
is 512000 SQL string for column : "TRIM(:NOMBRE)"
VERSION     NEXT     *     ¬         CHARACTER  NULL if VERSION =
BLANKS

Table AREA:
  3 Rows successfully loaded.
  0 Rows not loaded due to data errors.
  0 Rows not loaded because all WHEN clauses were failed.
  0 Rows not loaded because all fields were null.

Bind array size not used in direct path.
Column array  rows :    4194
Stream buffer bytes:  256000
Read   buffer bytes:10000000

Total logical records skipped:          1
Total logical records read:             3
Total logical records rejected:         0
Total logical records discarded:        0
Total stream buffers loaded by SQL*Loader main thread:        1
Total stream buffers loaded by SQL*Loader load thread:        0

Run began on Wed Mar 07 11:14:59 2012
Run ended on Wed Mar 07 11:14:59 2012

Elapsed time was:     00:00:00.08
CPU time was:         00:00:00.02
```

En la línea 39 del *script load_data.sh* se muestra un fragmento para la tabla área de las sentencias generadas para activar las restricciones del esquema *stttp*.

Código 4.36 enable_constraints.sql

```
$ head enable_constraints.sql
ALTER TABLE EJERCICIO_CONTABLE ENABLE CONSTRAINT SYS_C0061012;
ALTER TABLE RANGO_EJERCICIO ENABLE CONSTRAINT SYS_C0061013;
ALTER TABLE TIPO_RENDICION ENABLE CONSTRAINT SYS_C0061014;
ALTER TABLE CUVE ENABLE CONSTRAINT SYS_C0061015;
ALTER TABLE SECUENCIAS ENABLE CONSTRAINT SYS_C0061016;
ALTER TABLE VARIABLES_GLOBALES ENABLE CONSTRAINT SYS_C0061017;
ALTER TABLE MOVIMIENTO ENABLE CONSTRAINT SYS_C0061018;
ALTER TABLE NRO_DE_RENDICION ENABLE CONSTRAINT SYS_C0061019;
ALTER TABLE CARGO_CAMBIO ENABLE CONSTRAINT SYS_C0061021;
ALTER TABLE CARGO_CAMBIO ENABLE CONSTRAINT SYS_C0061020;
...
```

La salida del *script* anterior se guarda en el archivo *enable_constraints.log*.

Código 4.37 enable_constraints.log

```
$head enaable_constraints.log
Table altered.
Table altered.
Table altered.
Table altered.
Table altered.
Table altered.
Table altered.
Table altered.
Table altered.
Table altered.
...
```

Por último se ejecutan los *scripts* para reacomodar el último número de cada una de las secuencias.

Primero se ejecuta el *script generate_seq.sh* que genera el archivo *seq_join.txt* que se utilizará para crear las mismas.

Código 4.38 Ejecución del *script generate_seq.sh* para el esquema stttp

```
$ ./generate_seq.sh stttp stttp stttp
1:"seq_arancel":seq_arancel:
2:"seq_area":seq_area:
3:"seq_boleta_de_deposito":seq_boleta_de_deposito:
4:"seq_cambio":seq_cambio:
5:"seq_cargo":seq_cargo:
6:"seq_deposito":seq_deposito:
7:"seq_deposito_item_cobrable":seq_deposito_item_cobrable:
8:"seq_dominio":seq_dominio:
9:"seq_egreso":seq_egreso:
10:"seq_ejercicio":seq_ejercicio:
11:"seq_especie":seq_especie:
12:"seq_estado":seq_estado:
13:"seq_exportacion":seq_exportacion:
```

```
14:"seq_firma_representante":seq_firma_representante:
15:"seq_habilitacion":seq_habilitacion:
16:"seq_informacion_de_rendiciones":seq_informacion_de_rendiciones:
17:"seq_item_cobrable":seq_item_cobrable:
18:"seq_movimiento":seq_movimiento:
...
========================
TIEMPO: 10 SEGUNDOS
========================
```

Y por último se ejecuta el *script create_seq.sh* que regenera las secuencias anteriormente listadas.

Código 4.39 Ejecución del *script create_seq.sh* para el esquema sttp

```
$ ./create_seq.sh stttp stttp stttp
========================
TIEMPO: 120 SEGUNDOS
========================
```

La ejecución genera el *script ini_seq.sql* con las sentencias necesarias para crear las secuencias y que es ejecutado en la linea 25 del *script create_seq.sh*

Código 4.40 Fragmento del archivo *ini_seq.sql* para el esquema sttp

```
$ cat ini_seq.sql
--Init seq
DROP SEQUENCE seq_arancel;
CREATE SEQUENCE seq_arancel START WITH 1 INCREMENT BY 1 NOMAXVALUE
NOMINVALUE NOCACHE;
GRANT SELECT ON seq_arancel TO stttp_app;
DROP SEQUENCE seq_area;
CREATE SEQUENCE seq_area START WITH 1 INCREMENT BY 1 NOMAXVALUE
NOMINVALUE NOCACHE;
GRANT SELECT ON seq_area TO stttp_app;
DROP SEQUENCE seq_boleta_de_deposito;
CREATE SEQUENCE seq_boleta_de_deposito START WITH 1 INCREMENT BY 1
NOMAXVALUE NOMINVALUE NOCACHE;
GRANT SELECT ON seq_boleta_de_deposito TO stttp_app;
DROP SEQUENCE seq_cambio;
CREATE SEQUENCE seq_cambio START WITH 178611 INCREMENT BY 1 NOMAXVALUE
 NOMINVALUE NOCACHE;
GRANT SELECT ON seq_cambio TO stttp_app;
DROP SEQUENCE seq_cargo;
CREATE SEQUENCE seq_cargo START WITH 80 INCREMENT BY 1 NOMAXVALUE
NOMINVALUE NOCACHE;
GRANT SELECT ON seq_cargo TO stttp_app;
DROP SEQUENCE seq_deposito;
CREATE SEQUENCE seq_deposito START WITH 231462 INCREMENT BY 1
NOMAXVALUE NOMINVALUE NOCACHE;
GRANT SELECT ON seq_deposito TO stttp_app;
DROP SEQUENCE seq_deposito_item_cobrable;
CREATE SEQUENCE seq_deposito_item_cobrable START WITH 23959 INCREMENT
BY 1 NOMAXVALUE NOMINVALUE NOCACHE;
GRANT SELECT ON seq_deposito_item_cobrable TO stttp_app;
```

```
DROP SEQUENCE seq_dominio;
CREATE SEQUENCE seq_dominio START WITH 2 INCREMENT BY 1 NOMAXVALUE
NOMINVALUE NOCACHE;
GRANT SELECT ON seq_dominio TO stttp_app;
DROP SEQUENCE seq_egreso;
CREATE SEQUENCE seq_egreso START WITH 21027 INCREMENT BY 1 NOMAXVALUE
NOMINVALUE NOCACHE;
GRANT SELECT ON seq_egreso TO stttp_app;
DROP SEQUENCE seq_ejercicio;
CREATE SEQUENCE seq_ejercicio START WITH 1 INCREMENT BY 1 NOMAXVALUE
NOMINVALUE NOCACHE;
GRANT SELECT ON seq_ejercicio TO stttp_app;
DROP SEQUENCE seq_especie;
CREATE SEQUENCE seq_especie START WITH 2792 INCREMENT BY 1 NOMAXVALUE
NOMINVALUE NOCACHE;
GRANT SELECT ON seq_especie TO stttp_app;
...
EXIT
```

Agrupación de aplicaciones

Dada la existencia de esquemas que eran compartidos por varias aplicaciones, se decidió generar un plan de migración basado en niveles y grupos de esquemas.

En las siguientes imágenes se puede observar el desglose realizado de los roles en la base de datos origen que tenían como fin servir de guía para agrupar la migración de las aplicaciones.

Por ejemplo, para migrar el esquema viaticosoficina, primero se debía migrar el esquema registro_unico, pues este lo accedía mediante el rol usaRegistroUnico, que a su vez accedía al esquema compartido mediante el rol usaCompartido.

Este desglose se presentó ante el Senasa y se propuso un plan de migración por etapas en donde se mantendrían ambas bases de datos funcionando de manera productiva hasta tanto se completara la migración.

Este período duró alrededor de un año y se comenzó la migración por las aplicaciones menos utilizadas y con menos dependencia de otras aplicaciones.

Esta idea de migración escalonada nos dio una visión paulatina de como se iría comportando la nueva base de datos, nos permitió realizar ajustes sobre la marcha y nos ayudó a prever inconvenientes que podrían surgir con aplicaciones más demandantes.

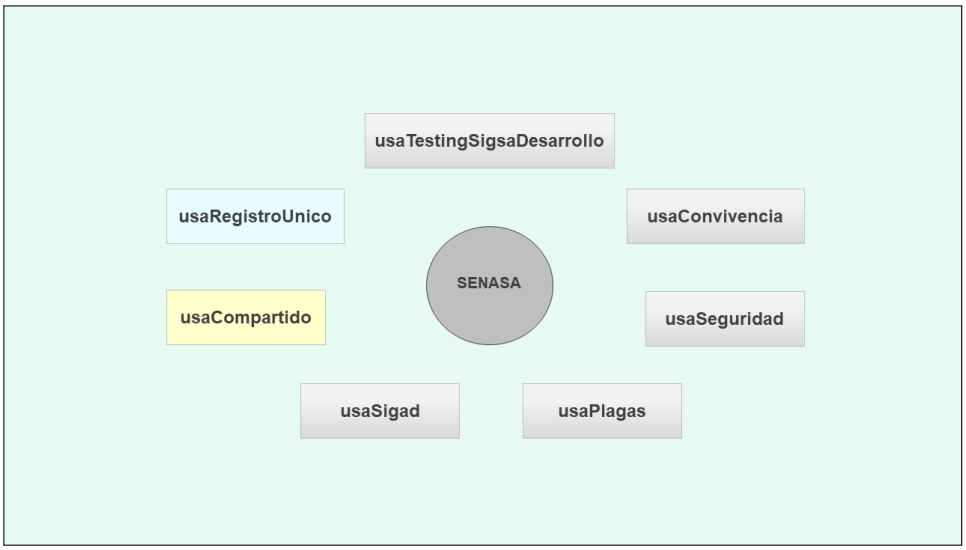

Figura 5.1: Roles existentes en la base de datos PostgreSQL

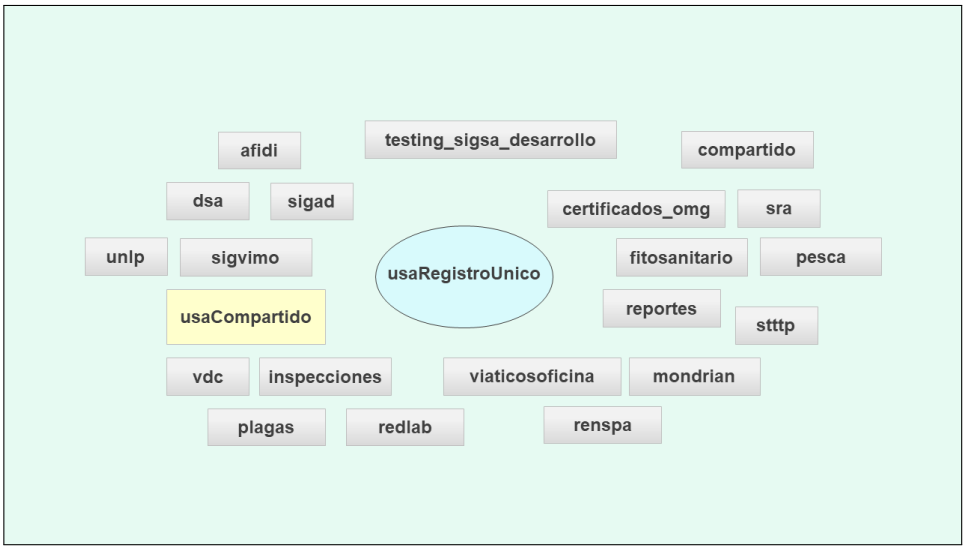

Figura 5.2: Aplicaciones que utilizan el rol usaRegistroUnico

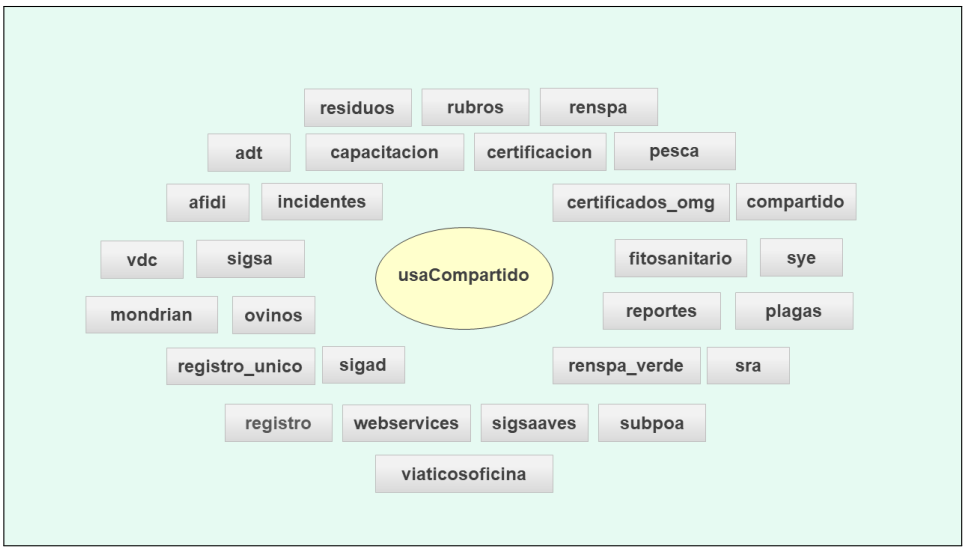

Figura 5.3: Aplicaciones que utilizan el rol usaCompartido

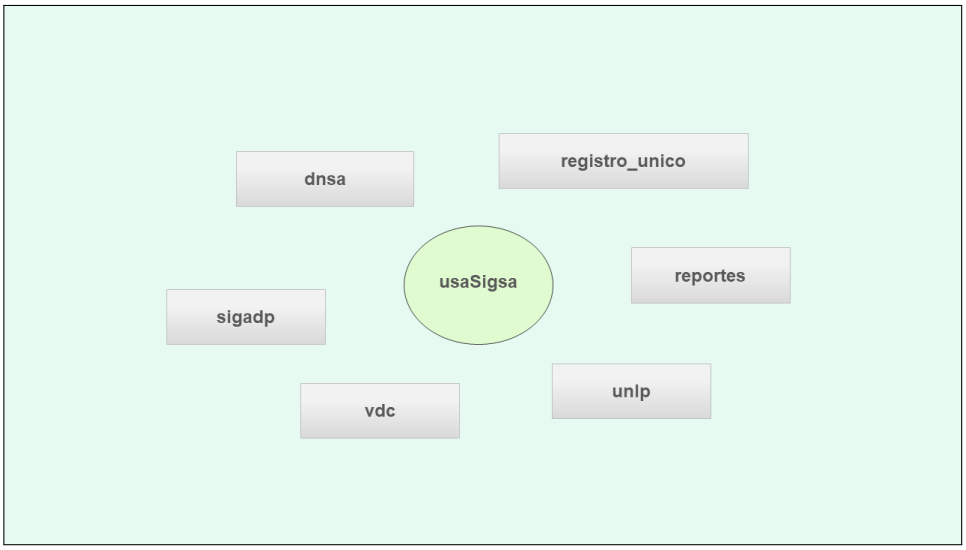

Figura 5.4: Aplicaciones que utilizan el rol usaSigsa

Planillas de control

Una vez creados y probados los *scripts* de migración, se crearon planillas de control para cada aplicación a migrar.

Estas planillas registraban la cantidad de objetos, así como también la cantidad de registros a migrar.

Ejemplo de planilla con recuento de objetos (tablas, vistas, secuencias, funciones, *triggers*) por aplicación:

Aplicación	Cantidad total de Vistas y Tabl	Status Scripts Migración	igración a D	Referencias
adt	107	Solo falta vistas, constraints y triggers /Asig priv terminando		
afidi	61	Solo falta vistas y triggers / Asig priv completos		
agroquimicos	23	Solo falta vistas, constraints y triggers / no tiene asig priv		
auxiliar	15	Solo falta vistas, constraints y triggers / no tiene asig priv		
capacitacion	38	Solo falta vistas y triggers/ Asig priv completos	Migrada	
certificacion	113	Solo falta vistas, constraints y triggers/ Asig priv completos		
certificados_omg	6	Solo falta vistas, constraints y triggers / no tiene asig priv		
compartido	49	Solo falta triggers /Asig priv completos		sigue en prioridad
convivencia_sgs	13	Solo falta vistas y triggers / no tiene asig priv		
dnsa	2	Solo falta vistas, constraints y triggers / no tiene asig priv		
fitosanitario_v4	64	Solo falta vistas y triggers /Asig priv completos		Fueron desarrollados por Cespi
incidentes	22	Solo falta vistas y triggers / no tiene asig priv	Migrar 1	
msigsa	46	Solo falta vistas, constraints y triggers / no tiene asig priv		Sigsa y appl relacionadas
oncca	6	Solo falta vistas, constraints y triggers / no tiene asig priv		
ovinos	16	Solo falta vistas y triggers /no tiene asig priv	Migrar 1	
pesca	38	Solo falta vistas y triggers/no tiene asig priv	Migrar 1	grantear para crear las vistas y las constraints que referencian a otros esquemas
plagas	115	Solo falta vistas, constraints y triggers /Asig priv terminando		
ppapa	18	Solo falta vistas, constraints y triggers / no tiene asig priv		
redlab	51	Solo falta vistas, constraints y triggers / no tiene asig priv		
registro	24	Solo falta vistas, constraints y triggers /Asig priv terminando		
registro_unico	233	Solo falta vistas, constraints y triggers /Asig priv terminando		
renspa	20	Solo falta vistas, constraints y triggers / no tiene asig priv		
renspa_verde	16	Solo falta vistas, constraints y triggers / no tiene asig priv		
reportes	58	Solo falta vistas, constraints y triggers / no tiene asig priv		
requisitos_fitosanitarios	43	Solo falta vistas, constraints y triggers / no tiene asig priv		

Figura 6.1: Planilla de estado de migración

Aplicación	Cantidad total de Vistas y Tablas
adt	107
afidi	61
agroquimicos	23
auxiliar	15
capacitacion	38
certificacion	113
certificados_omg	6
compartido	49
convivencia_sgs	13
dnsa	2
fitosanitario_v4	64
incidentes	22
msigsa	46
oncca	6
ovinos	16
pesca	38
plagas	115
ppapa	18
redlab	51
registro	24
registro_unico	233
renspa	20
renspa_verde	16
reportes	58
requisitos_fitosanitarios	43
residuos	58
rubros	14
seguridad	22
sigad	42
sigadp	15
sigsaaves	42
sra	6
stage	5
stttp	49

Figura 6.2: Planilla con recuento de vistas y tablas por aplicación

Aplicación	Cantidad total de Secuencias
adt	80
afidi	53
agroquimicos	16
auxiliar	1
capacitacion	34
certificacion	64
certificados_omg	5
compartido	36
fitosanitario_v4	51
incidentes	12
ovinos	9
pesca	32
plagas	87
ppapa	11
redlab	38
registro	11
registro_inspectores_apicolas	20
registro_unico	110
renspa	4
renspa_verde	3
requisitos_fitosanitarios	35
residuos	51
rubros	13
seguridad	4
sigad	37
sigsaaves	20
sra	6
stttp	32
subpoa	12
sye	18
testing_sigsa_desarrollo	115
transito	9
viaticosoficina	23
webservices	3
Total	**1055**

Figura 6.3: Planilla con recuento de secuencias por aplicación

ESQUEMA	TABLAS	CANTIDAD de REGISTROS
adt	producto_firma	38442
adt	producto	38400
adt	especie_por_producto	28875
adt	producto_alimento_humano	28873
adt	proceso_por_producto	28869
adt	variante_por_producto	18984
adt	tramite	13039
adt	establecimiento	8631
adt	componente_por_producto	7511
adt	producto_farmaco	7505
ESQUEMA	**TABLAS**	**CANTIDAD**
afidi	afidi	20269
afidi	afidi_importadores	9042
afidi	afidi_impresiones	20457
afidi	afidi_jnl	64296
afidi	afidi_productos	5139
afidi	afidi_productos_jnl	5318
afidi	afidi_productos_vieja	5158
afidi	afidi_reglas	38392
afidi	afidi_reglas_backup	30399
afidi	afidi_reglas_jnl	1988
ESQUEMA	**TABLAS**	**CANTIDAD**
agroqiomicos	clasedeuso	25
agroqiomicos	tipoinfraccion	22
agroqiomicos	codigoacta	14
agroqiomicos	clasetoxicologica	12
agroqiomicos	tipoproducto	4
agroqiomicos	tipoacta	3
agroqiomicos	acta	0
agroqiomicos	acta_inspectores	0
agroqiomicos	acta_productosacta	0

Figura 6.4: Planilla con recuento de registros por tablas

Estas planillas fueron de suma importancia a la hora de realizar el control de los registros migrados, brindaban información concreta y de fácil lectura al momento de realizar los controles, contando con filtros que permiten realizar búsquedas rápidas.

Índices de texto de Oracle

Concluida la migración completa de la base de datos de PostgreSQL a Oracle, y con todas las aplicaciones funcionando en el nuevo motor, comenzó la puesta a punto tratando de aprovechar las herramientas disponibles.

En este sentido se decidió aprovechar una herramienta llamada *Oracle Text* para solventar un inconveniente de una funcionalidad de una aplicación específica.

Dado el diseño lógico de esta funcionalidad, la consulta realizaba un *parseo* mediante el operador *in* de un *string* de 10 caracteres en una colección de -en el peor de los casos- unos 10 000 *strings*, lo cual significaba que los tiempos no sean estables y que gran parte de ellos superaran varios minutos en ejecución.

Con la implementación de los índices de texto de Oracle en dos columnas, puntualmente se logró estabilizar los tiempos, o sea, todas las consultas tardaban lo mismo y el tiempo de ejecución se encontraba por debajo del segundo.

Existen varios tipos de índices de texto, para esta solución se utilizó el índice de texto *CONTEXT*. Este es un índice de dominio de base de datos Oracle y es adecuado para indexar documentos grandes y coherentes en formatos como Microsoft Word, HTML o texto sin formato.

La forma de consultarlo es con el operador *CONTAINS*, por lo tanto se tuvieron que modificar todas las consultas asociadas a dichos índices.

Un índice *CONTEXT* no es transaccional y funciona de la siguiente manera:

- Cuando se elimina un registro, el cambio en el índice es inmediato. Su propia sesión ya no encontrará el registro desde el momento en que realice el cambio, y otros usuarios no encontrarán el registro tan pronto como lo confirme.

- Para inserciones y actualizaciones, la nueva información no será visible para búsquedas de texto hasta que se haya producido una sincronización del índice.

Así las cosas, cuando realiza inserciones o actualizaciones en la tabla base, debe sincronizar explícitamente el índice con *CTX_DDL.SYNC_INDEX*.

En el siguiente *script* se muestra la definición de las preferencias del índice a crear entre los que se encuentran el *tablespace* que va a utilizar, en este caso se decidió utilizar uno dedicado para este tipo de índices.

Código 7.1 create_pref_ctx_mov.sql

```
1  begin
2    ctx_ddl.create_preference('"MOVIMIENTOS_TEXTO_IDX_DST"','
       DIRECT_DATASTORE');
3  end;
4  /
5
6  begin
7    ctx_ddl.create_preference('"MOVIMIENTOS_TEXTO_IDX_FIL"','NULL_FILTER
       ');
8  end;
9  /
10
11 begin
12   ctx_ddl.create_section_group('"MOVIMIENTOS_TEXTO_IDX_SGP"','
       NULL_SECTION_GROUP');
13 end;
14 /
15
16 begin
17   ctx_ddl.create_preference('"MOVIMIENTOS_TEXTO_IDX_LEX"','BASIC_LEXER
       ');
18 end;
19 /
20
21 begin
22   ctx_ddl.create_preference('"MOVIMIENTOS_TEXTO_IDX_WDL"','
       BASIC_WORDLIST');
23   ctx_ddl.set_attribute('"MOVIMIENTOS_TEXTO_IDX_WDL"','STEMMER','
       SPANISH');
24   ctx_ddl.set_attribute('"MOVIMIENTOS_TEXTO_IDX_WDL"','FUZZY_MATCH','
       SPANISH');
25 end;
26 /
27
28
29 begin
30   ctx_ddl.create_preference('"MOVIMIENTOS_TEXTO_IDX_STO"','
       BASIC_STORAGE');
31   ctx_ddl.set_attribute('"MOVIMIENTOS_TEXTO_IDX_STO"','I_TABLE_CLAUSE'
       ,'tablespace SIGSA_IDX_U05');
32   ctx_ddl.set_attribute('"MOVIMIENTOS_TEXTO_IDX_STO"','K_TABLE_CLAUSE'
       ,'tablespace SIGSA_IDX_U05');
33   ctx_ddl.set_attribute('"MOVIMIENTOS_TEXTO_IDX_STO"','R_TABLE_CLAUSE'
       ,'tablespace SIGSA_IDX_U05');
34   ctx_ddl.set_attribute('"MOVIMIENTOS_TEXTO_IDX_STO"','N_TABLE_CLAUSE'
       ,'tablespace SIGSA_IDX_U05');
35   ctx_ddl.set_attribute('"MOVIMIENTOS_TEXTO_IDX_STO"','I_INDEX_CLAUSE'
       ,'tablespace SIGSA_IDX_U05 storage (initial 1M) compress 2');
36   ctx_ddl.set_attribute('"MOVIMIENTOS_TEXTO_IDX_STO"','P_TABLE_CLAUSE'
       ,'tablespace SIGSA_IDX_U05');
37 end;
38 /
```

En este otro *script* se muestra la definición de la creación del índice propiamente dicho utilizando los parámetros anteriormente definidos.

Código 7.2 create_ctx_mov.sql

```
create index "SIGSA"."MOVIMIENTOS_TEXTO_IDX"
  on "SIGSA"."MOVIMIENTOS"
      ("CARAVANAS")
  indextype is ctxsys.context
  parameters('
    datastore       "MOVIMIENTOS_TEXTO_IDX_DST"
    filter          "MOVIMIENTOS_TEXTO_IDX_FIL"
    section group   "MOVIMIENTOS_TEXTO_IDX_SGP"
    lexer           "MOVIMIENTOS_TEXTO_IDX_LEX"
    wordlist        "MOVIMIENTOS_TEXTO_IDX_WDL"
    storage         "MOVIMIENTOS_TEXTO_IDX_STO"
    fast_query
    sync (manual)
  ')
/
```

Esto implicó un análisis exhaustivo de cuándo realizar la sincronización del índice para cumplir con las políticas del negocio de la aplicación, dando como resultado la sincronización de los índices en tres horarios del día, uno por la mañana, otro por la tarde y uno más a la noche.

Código 7.3 sync_ctx_mov.sql

```
BEGIN
CTX_OUTPUT.START_LOG('CTX_SYNC_MOVIMIENTOS_TEXTO_IDX_2015.log',TRUE);
CTX_DDL.SYNC_INDEX('SIGSA.MOVIMIENTOS_TEXTO_IDX',NULL,NULL,
    PARALLEL_DEGREE=>2,MAXTIME=>15,LOCKING=>0);
CTX_OUTPUT.END_LOG;
END;
```

Se creó una tarea de mantenimiento del índice que se ejecuta los fines de semana. Esta tarea realiza una reconstrucción del mismo. Si no se realizaba esta tarea, el índice podía crecer indefinidamente y superar los 30 GB de almacenamiento.

Código 7.4 rebuild_ctx_mov.sql

```
BEGIN
CTX_OUTPUT.START_LOG('CTX_MOVIMIENTOS_TEXTO_IDX.log',TRUE);
EXECUTE IMMEDIATE 'ALTER INDEX SIGSA.MOVIMIENTOS_TEXTO_IDX REBUILD
    ONLINE PARALLEL 2';
CTX_OUTPUT.END_LOG;
END;
```

Conclusiones

Se alcanzaron los objetivos esperados, se logró la migración de datos y una *performance* de la base de datos acorde a los requerimientos del organismo.

A partir de la nueva definición obtenida bajo un motor como Oracle, se estudiaron diferentes contextos para evaluar el rendimiento final del motor en cuanto a la utilización de la base de datos.

Esta migración se realizó entre los años 2010 y 2011, con las herramientas disponibles en aquel momento, por ello, como trabajo futuro se podrían adaptar los *scripts* a un lenguaje de *scripting* más moderno y flexible aprovechando técnicas de concurrencia o paralelismo, disminuyendo los tiempos de migración.

Asimismo, se podría extender la compatibilidad con otros motores de base de datos utilizando la misma técnica que se adoptó en este documento.

Bibliografía

[1] The PostgreSQL Global Development Group®: *Postgresql database documentation*. https://www.postgresql.org/docs/8.4/index.html, 1996-2014. Accedido: 2011.

[2] Oracle®: *Database administrator's guide*. https://docs.oracle.com/cd/E11882_01/server.112/e25494/toc.htm, 2001-2015. Accedido: 2011.

[3] F. Yergeau, The Internet Society®: *Utf-8, a transformation format of iso 10646*. https://datatracker.ietf.org/doc/html/rfc3629#page-4, 2003. Accedido: 2021.

[4] Unicode, Inc®: *The unicode® standardversion 14.0 – core specification*. https://www.unicode.org/versions/Unicode14.0.0/ch02.pdf#G13708, 1991-2021. Accedido: 2021.

[5] Oracle®: *Database 2 day dba*. https://docs.oracle.com/cd/E25178_01/server.1111/e10897/install.htm, 2004-2011. Accedido: 2011.

[6] 2022 Slashdot Media®, Hitachi Vantara LLC 2022®: *Pentaho from hitachi vantara*. https://sourceforge.net/projects/pentaho/, 2022. Accedido: 2022.